ADVANCED DATA-STRUCTURES
&
ALGORITHM ANALYSIS

Dr. Sukhamay Kundu
Computer Science Dept, Louisiana state University
Baton Rouge, LA 70803
kundu@csc.lsu.edu

Spring 2011
(copyright @2010 , @2011)
ROLE OF DATA-STRUCTURES IN COMPUTATION

Makes Computations Faster:

- Faster is better. (Another way to make computations faster is to use parallel or distributed computation.)

Three Basic Computation Steps:

1. Locate/Access data-values (inputs to a step)
2. Compute a value (output of a step)
3. Store the new value

External Input → Computation = Sequence of Computation Steps → External Output

(1) Locate/Access data-values (inputs to a step)
(2) Compute a value (output of a step)
(3) Store the new value

Program: Algorithm + DataStructure + Implementation.

- Algorithm
  - The basic method; it determines the data-items computed.
  - Also, the order in which those data-items are computed (and hence the order of read/write data-access operations).

- Data structures
  - Supports efficient read/write of data-items used/computed.

Total Time = Time to access/store data + Time to compute data.

Efficient Algorithm = Good method + Good data-structures
(+ Good Implementation)

Question:

- What is an efficient program?
- What determines the speed of an Algorithm?
- A program must also solve a "problem". Which of the three parts algorithm, data-structure, and implementation embodies this?
ALGORITHM OR METHOD
vs. DATA STRUCTURE

Problem: Compute the average of three numbers.

Two Methods:
1. \( \text{aver} = (x + y + z)/3. \)
2. \( \text{aver} = (x/3) + (y/3) + (z/3). \)

- Method (1) superior to Method (2); two less div-operations.
- They access data in the same order: \( \langle x, y, z, \text{aver} \rangle \).
- Any improvement due to data-structure applies equally well to both methods.

Data structures:
(a) Three variables \( x, y, z \).
(b) An array \( \text{nums}[0..2] \).
   - This is inferior to (a) because accessing an array-item takes more time than accessing a simple variable. (To access \( \text{nums}[i] \), the executable code has to compute its address \( \text{addr}(\text{nums}[i]) = \text{addr}(\text{nums}[0]) + i \times \text{sizeof(int)}, \) which involves 1 addition and 1 multiplication.)
   - When there are large number of data-items, naming individual data-items is not practical.
   - Use of individually named data-items is not suitable when a varying number of data-items are involved (in particular, if they are used as parameters to a function).

A Poor Implementation of (1): Using 3 additions and 1 division.
\[
\begin{align*}
a &= x + y; \quad \text{//uses 2 additional assignments} \\
b &= a + z; \\
\text{aver} &= b/3;
\end{align*}
\]
LIMITS OF EFFICIENCY

Hardware limit:

- *Physical* limits of time (speed of electrons) and space (layout of circuits). This limit is computation problem *independent*.

From 5 mips (millions of instructions per sec) to 10 mips is an improvement by the factor of 2.

One nano-second = $10^{-9}$ (one billionth of a second); 10 mips = 100 ns/instruction.

Software limit:

- *Limitless* in a way, except for the inherent nature of the problem. That is, the limit is *problem dependent*.

<table>
<thead>
<tr>
<th>Sorting Algorithm</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>$O(n \cdot \log n)$</td>
</tr>
<tr>
<td>A2</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

$(n = number \ of \ items \ sorted)$

A1 is an improvement over A2 by the factor

$$\frac{n^2}{n \cdot \log n} = \frac{n}{\log n} \rightarrow \infty \ as \ n \rightarrow \infty.$$

- $O(n \cdot \log n)$ is the efficiency-limit for sorting Algorithms.
**MEASURING PERFORMANCE**

**Analytic Method:**
- Theoretical analysis of the Algorithm’s time complexity.

**Empirical Methods:**
- Count the number of times specific operations are performed by executing an *instrumented* version of the program.
- Measure directly the actual program-execution time in a run.

**Example of Instrumentation:**

| Original code: | if (x < y) small = x;  
|               | else small = y;       |
| Instrumented code: | countComparisons++; //initialized elsewhere  
|                   | if (x < y) small = x;  
|                   | else small = y;       |

**Question:**
- What is wrong with the following instrumentation:
  ```c
  if (x < y) { countComparisons++; small = x; }  
  else small = y;  
  ```

- Instrument the code below for readCount and writeCount of x:
  ```c
  if (x < 3) y = x + 5;  
  ```

- Show the new code when updates to loopCount is moved outside the loop:
  ```c
  for (i=j; i<max; i++) {  
    loopCount++;  
    if (x[i] < 0) break;  
  }
  ```
EXERCISE

1. Instrument the code below to count the number of Exchanges (numExchanges) and number of comparisons (numComparisons) of the array data-items. Show the values of numExchanges and numComparisons after each iteration of the outer for-loop for the input items[] = [3, 2, 4, 5, 2, 0].

```c
void crazySort(int *items, int numItems)
{ int i, j, small,
  for (i=0; i<numItems; i++) //put ith smallest item in items[i]
    for (j=i+1; j<numItems; j++)
      if (items[i] > items[j]) { //exchange
        small = items[j]; items[j] = items[i];
        items[i] = small;
      }
}
```

(a) If we use "i < numItems − 1" in place of "i < numItems" in the outer for-loop, do we still get the same final result? Will it affect the execution time?

(b) Is the algorithm in the code more closely related to insertion-sort or to selection-sort? In what way does it differ from that?

2. For numItems = 6, find an input for which crazySort will give maximum numExchanges. When will numExchanges be minimum?

3. Give a pseudocode for deciding whether three given line segments of lengths x, y, and z can form a triangle, and if so whether it is a right-angled, obtuse-angled, or an acute-angled triangle. Make sure that you minimize the total number operations (arithmetic and comparisons of data-items)?

4. Given an array lengths[1..n] of the lengths of n line segments, find a method for testing if they can form a polygon (quadrilateral for n = 4, pentagon for n = 5, etc).
SOLUTION TO SELECTED EXERCISES:

1. void crazySort(int *items, int numItems)
   {
      int i, j, small,
      numComparisons=0, //for two elements in items[]
      numExchanges=0;  //of elements in items[]
   for (i=0; i<numItems; i++) { //put ith smallest item in items[i]
      for (j=i+1; j<numItems; j++) {
         numComparisons++;  //keep it here
         if (items[i] > items[j]) { //exchange
            numExchanges++;
            small = items[j]; items[j] = items[i];
            items[i] = small;
         }
      }
   }
   printf("numComparisons = %d, numExchanges = %d\n",
   numComparisons, numExchanges);
   }

   After the comparison and exchanges (if any) for input items[] = [3, 2, 4, 5, 2, 0].

   | i=0, j=1, items[]: | 2 | 3 | 4 | 5 | 2 | 0 |
   | i=0, j=2, items[]: | 2 | 3 | 4 | 5 | 2 | 0 |
   | i=0, j=3, items[]: | 2 | 3 | 4 | 5 | 2 | 0 |
   | i=0, j=4, items[]: | 2 | 3 | 4 | 5 | 2 | 0 |
   | i=0, j=5, items[]: | 0 | 3 | 4 | 5 | 2 | 2 |
   numComparisons = 5, numExchanges = 2

   | i=1, j=2, items[]: | 0 | 3 | 4 | 5 | 2 | 2 |
   | i=1, j=3, items[]: | 0 | 3 | 4 | 5 | 2 | 2 |
   | i=1, j=4, items[]: | 0 | 2 | 4 | 5 | 3 | 2 |
   | i=1, j=5, items[]: | 0 | 2 | 4 | 5 | 3 | 2 |
   numComparisons = 9, numExchanges = 3

   | i=2, j=3, items[]: | 0 | 2 | 4 | 5 | 3 | 2 |
   | i=2, j=4, items[]: | 0 | 2 | 3 | 5 | 4 | 2 |
   | i=2, j=5, items[]: | 0 | 2 | 2 | 5 | 4 | 3 |
   numComparisons = 12, numExchanges = 5

   | i=3, j=4, items[]: | 0 | 2 | 2 | 4 | 5 | 3 |
   | i=3, j=5, items[]: | 0 | 2 | 2 | 3 | 5 | 4 |
   numComparisons = 14, numExchanges = 7

   | i=4, j=5, items[]: | 0 | 2 | 2 | 3 | 4 | 5 |
   numComparisons = 15, numExchanges = 8

   | i=5, j=6, items[]: | 0 | 2 | 2 | 3 | 4 | 5 |
   numComparisons = 15, numExchanges = 8

   This is more closely related to selection-sort, which involves at most one exchange for each iteration of outer-loop. #(Comparisons) is still $C_2^n$.

2. Triangle classification pseudocode; assume that $0 < x \leq y \leq z$. 
if (z < x + y) {
    zSquare = z*z; xySquareSum = x*x + y*y;
    if (zSquare == xySquareSum)
        right-angled triangle;
    else if (zSquare > xySquareSum)
        obtuse-angled triangle;
    else acute-angled triangle;
}
else not a triangle;

3. Condition for polygon:
   • The largest length is less than the sum of the other lengths.
   • The lengths [2, 4, 5, 20] will not make a quadrilateral because
     $20 \leq 2 + 4 + 5 = 11$, but the lengths [2, 4, 5, 10] will.
ANALYZING NUMBER OF EXCHANGES IN CRAZY-SORT

Pseudocode #1:
1. Create all possible permutations $p$ of $\{0, 1, 2, \ldots, n-1\}$.
2. For each $p$, apply crazySort and determine numExchanges.
3. Collect these data to determine numPermutations[$i$] = #(permutations which has numExchanges = $i$) for $i = 0, 2, \ldots, C_n^2$.
4. Plot numPermutations[$i$] against $i$ to visualize the behavior of numExchanges.

Pseudocode #2: //No need to store all $n!$ permutations.
1. For ($i=0$; $i<C_n^2$; $i++$), initialize numPermutations[$i$] = 0.
2. While (there is a nextPermutation($n$) = $p$) do the following:
   (a) Apply crazySort to $p$ and determine numExchanges.
   (b) Add 1 to numPermutation[numExchanges].
3. Plot numPermutations[$i$] against $i$.

Note: We can use this idea to analyze other sorting algorithms.

Question:
• If $p$ is a permutation of $S = \{0, 1, 2, \ldots, n-1\}$, then how to determine the nextPermutation($p$) in the lexicographic order? Shown below are permutations for $n = 4$ in lexicographic order.

\[
\begin{array}{cccccccc}
0123 & 0312 & 1203 & 2013 & 2301 & 3102 \\
0132 & 0321 & 1230 & 2031 & 2310 & 3120 \\
0213 & 1023 & 1302 & 2103 & 3012 & 3201 \\
0231 & 1032 & 1320 & 2130 & 3021 & 3210 \\
\end{array}
\]
PSEUDOCODE vs. CODE

Characteristics of Good Pseudocode:
+ Shows the key concepts and the key computation steps of the Algorithm, avoiding too much details.
+ Avoids dependency on any specific prog. language.
+ Allows determining the correctness of the Algorithm.
+ Allows choosing a suitable data-structures for an efficient implementation and complexity analysis.

Example. Compute the number of positive and negative items in $nums[0..n-1]$; assume each $nums[i] \neq 0$.

(A) Pseudocode:
1. Initialize positiveCount = negativeCount = 0.
2. Use each $nums[i]$ to increment one of the counts by one.

Code:
1.1 positiveCount = negativeCount = 0;
2.1 for (i=0; i<n; i++) //each $nums[i] \neq 0$
2.2 if (0 < nums[i]) positiveCount++;
2.3 else negativeCount++;

(B) Pseudocode:
1. Initialize positiveCount = 0.
2. Use each $nums[i] > 0$ to increment positiveCount by one.
3. Let negativeCount = $n - positiveCount$.

Code:
1. positiveCount = 0;
2. for (i=0; i<n; i++) //each $nums[i] \neq 0$
3. if (0 < nums[i]) positiveCount++;
4. negativeCount = $n - positiveCount$;

Question:
•? Why is (B) slightly more efficient than (A)?

Writing a pseudocode requires skills to express an Algorithm in a concise and yet clear fashion.
PSEUDOCODE FOR SELECTION-SORT

Idea: Successively find the \( i \)th smallest item, \( i = 0, 1, \ldots \).

Algorithm Selection-Sort:

- **Input:** Array \( \text{items}[\] \) and its size \( \text{numItems} \).
- **Output:** Array \( \text{items}[\] \) sorted in increasing order.

1. For each \( i \) in \{ 0, 1, \ldots, \text{numItems}-1 \}, in some order, do (a)-(b):
   - (a) Find the \( i \)th smallest item in \( \text{items}[\] \).
   - (b) Place it at position \( i \) in \( \text{items}[\] \).

Finding \( i \)th smallest item in \( \text{items}[\] \):

- Finding \( i \)th smallest item directly is difficult, but it is easy if we know all the \( k \)th smallest items for \( k = 0, 1, 2, \ldots, (i - 1) \).
- It is the smallest item among the remaining items.
- If we assume that \( \text{items}[k], 0 \leq k \leq (i - 1) \), are the \( k \)th smallest items, then smallest item in \( \text{items}[i..\text{numItems}-1] = i \)th smallest item. This gives the pseudocode:

  - (a.1) \( \text{smallestItemIndex} = i \);
  - (a.2) for \( j = i + 1; j < \text{numItems}; j++ \)
  - (a.3) if \( \text{items}[j] < \text{items}[\text{smallestItemIndex}] \)
  - (a.4) then \( \text{smallestItemIndex} = j \);

Question: In what way (a.1)-(a.4) is better than step (a)?

Placing \( i \)th smallest item at position \( i \) in \( \text{items}[\] \).

  - (b.1) if \( \text{smallestItemIndex} > i \) // why not \( \text{smallestItemIndex} \neq i \)
  - (b.2) then exchange \( \text{items}[i] \) and \( \text{items}[\text{smallestItemIndex}] \);

"What" comes before "how".
**EXERCISE**

1. Which of "put the items in right places" and "fill the places by right items" best describes the selection-sort Algorithm? Shown below are the steps in the two methods for input [3, 5, 0, 2, 4, 1].

<table>
<thead>
<tr>
<th>Put the items in right places</th>
<th>Fill the places with right items</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. [2, 5, 0, 3, 4, 1]</td>
<td>[0, 5, 3, 2, 4, 1]</td>
</tr>
<tr>
<td>3 moved to right place</td>
<td>1st place is filled by 0</td>
</tr>
</tbody>
</table>

| 2. [0, 5, 2, 3, 4, 1] | [0, 1, 3, 2, 4, 5] |
| 2 moved to right place | 2nd place is filled by 1 |

| 3. [0, 5, 2, 3, 4, 1] | [0, 1, 2, 3, 4, 5] |
| 0 already in right place | 3rd place is filled by 2 |

| 4. [0, 1, 2, 3, 4, 5] | [0, 1, 2, 3, 4, 5] |
| 5 moved to right place | all places filled properly |

| 5. [0, 1, 2, 3, 4, 5] | |
| all items in right places | |

Note that once an item is put in right place, you must not change its position while putting other items in proper places. It is for this reason, we make an exchange (and not an insertion) when we move an item in the right place. The insertion after removing 3 from its current position in [3, 5, 0, 2, 4, 1] would have given [5, 0, 2, 3, 4, 1] but not [2, 5, 0, 3, 4, 1] as we showed above.

2. Which input array for the set numbers {0, 1, 2, 3, 4, 5} requires maximum number of exchanges in the first approach?

3. Give a pseudocode for the first approach.
ANOTHER EXAMPLE OF PSEUDOCODE

**Problem:** Find the position of rightmost "00" in binString[0..(n-1)].

1. Search for 0 right to left upto position 1 (initially, start at position n-1).
2. If (0 is found and the item to its left is 1), then go back to step (1) to start the search for 0 from the left of the current position.

**Three Implementations:** Only the first one fits the pseudocode.

1. \( i = n; \) \(/= \) length of binString
   \[ \text{do } \begin{array}{l} 
   \text{for (} i = i - 1; \ i > 0; \ i --) \\
   \text{if (} 0 \text{ == binString}[i] \text{) break;}
   \end{array} \] 
   \[ \text{while (} 1 \text{ == binString}[--i]); \] \(/= \) has a bug; find it

2. \( \text{for (} i = n - 1; \ i > 0; \ i --) \)
   \[ \text{if (} 0 \text{ == binString}[i]) \text{ && (} 0 \text{ == binString}[i - 1]) \text{ break; \ /= inefficient but works} \]

3. \( \text{for (} i = n - 1; \ i > 0; \ i --) \) \(/= \) bad for-loop; body updates i
   \[ \text{if (} 0 \text{ == binString}[i]) \text{ && (} 0 \text{ == binString}[--i]) \text{ break; \ /= works and efficient} \]

**Question:**

• Show how these implementations work differently using the binString: \( \cdots 00111010101 \). Extend each implementation to return the position of the left 0 of the rightmost "00".

• Instrument each code for readCount of the items in binString[ ].

• Which of (1)-(3) is the least efficient in terms readCount?

• Give a pseudocode to find rightmost "00" without checking all bits from right till "00" is found.

It is not necessary to sacrifice clarity for the sake of efficiency.
EXERCISE

1. BinStrings(n, m) = \{ x: x is a binary string of length n and m ones \}, 0 \leq m \leq n. The strings in BinStrings(4, 2) in lexicographic order are:

   0011, 0101, 0110, 1001, 1010, 1100.

Which of the pseudocodes below for generating the strings in BinStrings(n, m) in lexicographic order is more efficient?

(a) 1. Generate and save all binary strings of length n in lexicographic order.
    2. Throw away the strings which have numOnes \neq m.

(b) 1. Generate the first binary string \(0^{n-m}1^m\) \in BinStrings(n, m).
    2. Successively create the next string in BinStrings(n, m) until the last string \(1^m0^{n-m}\).

Which of the three characteristics of a good pseudocode hold for each of these pseudocodes?

2. Give the pseudocode of a recursive Algorithm for generating the binary strings in BinStrings(n, m) in lexicographic order.

3. Give an efficient pseudocode for finding the position of rightmost "01" in an arbitrary string \(x \in\) BinStrings(n, m). (The underlined portion in 101100\underline{11}100 shows the rightmost "01".) Give enough details so that one can determine the number of times various items \(x[i]\) in the array \(x\) are looked at.

4. Given a string \(x \in\) BinStrings(n, m), give a pseudocode for generating the next string in BinStrings(n, m), if any.
ALWAYS TEST YOUR METHOD
AND YOUR ALGORITHM

• Create a few general examples of input and the corresponding outputs.
  – Select some input-output pairs based on your understanding of the problem and before you design the Algorithm.
  – Select some other input-output pairs after you design the Algorithm, including a few cases that involve special handling of the input or output.

• Use these input-output pairs for testing (but not proving) the correctness of your Algorithm.

• Illustrate the use of data-structures by showing the "state" of the data-structures (lists, trees, etc.) at various stages in the Algorithm’s execution for some of the example inputs.

Always use one or more carefully selected example to illustrate the critical steps in your method/algorithm.
EFFICIENCY OF NESTED IF-THEN-ELSE

• Let $E =$ average #(condition evaluations). We count 1 for evaluation of both $x$ and its negation ($\neg x$).

Example 1. For the code below, $E = 3.5$.

```c
if (x and y) z = 0;
else if ((not x) and y) z = 1;
else if (x and (not y)) z = 2;
else z = 3;
```

<table>
<thead>
<tr>
<th>Value of z</th>
<th>#(condition evaluations)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2 ($x = T$ and $y = T$)</td>
</tr>
<tr>
<td>1</td>
<td>3 ($x = F, \neg x = T,$ and $y = T$)</td>
</tr>
<tr>
<td>2</td>
<td>5 ($x = T, y = F, \neg x = F, x = T,$ and $\neg y = T$)</td>
</tr>
<tr>
<td>3</td>
<td>4 ($x = F, \neg x = T, y = F, x = F$)</td>
</tr>
</tbody>
</table>

Question:

•? Show #(condition evaluations) for each $z$ for the code and also the average $E$:

```c
if (x)
    if (y) z = 0;
    else z = 2;
else if (y) z = 1;
else z = 3;
```

•? Give a code to compute $z$ without using the keyword "else" (or "case") and show #(condition evaluations) for each value of $z$.

•? Show the improved form of the two code-segments below.

(a). if (nums[i] >= max) max = nums[i];
(b). if (x > 0) z = 1;
    if ((x > 0) && (y > 0)) z = 2;
BRIEF REVIEW OF SORTING

Questions:

- What is Sorting? Explain with an example.
- Why do we want to sort data?
- What are some well-known sorting Algorithms?
- Which sorting Algorithm uses the following idea:
  Successively, find the smallest item, the second smallest item, the third smallest items, etc.
- Can we sort a set of pairs of numbers like \{(1,7), (2,7), (5,4), (3,6)\}? What is the result after sorting?
- Can we sort non-numerical objects like the ones shown below?
  Strings: abb, ba, baca, cab.
  Binary trees on 3 nodes (convert them to strings to sort):
  Flowcharts with 2 nodes (convert them to trees or strings to sort):
EXERCISE

1. Give a more detailed pseudocode (not code) for sorting using the idea "put the items in the right places". Determine the number of comparisons of involving data from items[0..numItems-1] based on the pseudocode. Explain the Algorithm in detail for the input items[] = [3, 2, 4, 5, 1, 0].

2. Write a pseudocode for insertion-sort. Determine the number of comparisons of involving data from items[0..numItems-1] based on the pseudocode; also determine the number of data-movements (i.e., movements of items from the items-array) based on the pseudocode. Explain the Algorithm in detail for the input items[] = [3, 2, 4, 5, 1, 0].

3. For each of the sorting Algorithms insertion-sort, selection-sort, bubble-sort, and merge-sort, show the array after each successive exchange operation starting the initial array [3, 2, 4, 5, 1, 0].

4. Some critical thinking questions on selection-sort. Assume that the input is a permutation of \{1, 2, \ldots, n\}.
   (a) Give an example input for which the number of data-movements is maximum (resp., minimum).
   (b) In what sense, selection-sort minimizes data-movements?
   (c) Suppose we have exchanges of the form \(e_1\): items[i1] and items[i2], \(e_2\): items[i2] and items[i3], \ldots, \(e_{k-1}\): items[i(k-1)] and items[ik]. Then argue that the indices \{i1, i2, \ldots, ik\} form a cycle in the permutation. Note that the exchange operations \(e_i\) may be interleaved with other exchanges.

5. Is it true that in bubble-sort if an item moves up, then it never moves down? Explain with the input items[] = [3, 2, 4, 5, 1, 0].
AVERAGE #(COMPARISONS) TO LOCATE A DATA-ITEM IN A SORTED-ARRAY

**Binary Search:** Assume \( N = \) numItems = 15 = \( 2^4 - 1 \).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>A[7]</td>
</tr>
</tbody>
</table>

- Number of comparisons for an item \( x \):
  - Total #(Comparisons) = \( 1 \times 1 + 2 \times 2 + 3 \times 4 + 4 \times 8 = 49 \); Average = \( 49/15 = 3.3 \).

- General case (\( N = 2^n - 1 \)): Total #(Comparisons) = 
  \[
  \sum_{i=0}^{n-1} \text{#(compar. per node at level i)} \times \text{#(nodes at level i)} \\
  = 1 \times 1 + 2 \times 2 + 3 \times 4 + \cdots + n \times 2^{n-1} = 1 + (n - 1)2^n \\
  = 1 + \lfloor \log(N + 1) - 1 \rfloor. (N + 1) = O(N \log N)
  \]
  Average #(Comp.) = \( O(\log N) \)

**A simpler argument:**
- \( \text{Max(#Comp)} = n \) and hence average \( \leq n = O(\log N) \).
**HEAP DATA-STRUCTURE**

**Heap:** A special kind of binary-tree, which gives an efficient $O(N \cdot \log N)$ implementation of selection-sort.

- **Shape constraints:** Nodes are added left to right, level by level.
  - A node has a rightchild only if it has a leftchild.
  - If there is a node at level $m$, then there are no missing nodes at level $m - 1$.

- **Node-Value constraint:** For each node $x$ and its children $y$, $\text{val}(x) \geq \text{val}(y)$, $\text{val}(x) =$ the value associated with node $x$.

**Example:** The shape of heaps with upto 7 nodes.

![heap-diagram](image)

**Questions:** Which of the following is true?

1. Each node has exactly one parent, except the root.
2. Each node has 0 or 2 children, except perhaps one.
3. The leftchild node with no brother has the maximum height.
4. The properties (1)-(3) define a heap.

**Example.** Heaps with upto 4 nodes and small node-values.

![heap-diagram](image)
ARRAY-IMPLEMENTATION OF HEAP

Array-structure for Heap of 12 nodes:

```
A[0]
  /     \
 /     /     \
|       |       |
      |       |
```


$A[0] = \max\{A[0], A[1], \ldots, A[11]\}$


... 

Parent-Child relations in the Array:
- Not dependent on values at the nodes and does not use pointers.


EXERCISE

1. Show all possible heaps with 5 nodes and the node values $\{1, 2, 3, 4, 5\}$. 
HEAP-SORTING METHOD

Two Parts in Heap-Sort: Let $N = \text{numItems}$.

- Make the input-array into a heap.
- Use the heap to sort as follows:
  - Make $A[0..N – 2]$ into a max-heap: each child-value < parent-value.
  - Make $A[0..N – 3]$ into a heap and so on, each time working with a smaller initial part of the input-array.

Example. Part of the heap-sorting process.
HEAP-Sorting Algorithm

MakeHeap, using the recursive AddToHeap: \( n = \text{numItems} \).
- \( \text{nums}[(n-1)..(n-1)] \) is an heap.
- For \( i = n-2, n-3, \ldots, 1, 0 \), make the tail part \( \text{nums}[i..n-1] \) into an heap by adding \( \text{nums}[i] \) to the heap \( \text{nums}[i+1..n-1] \).

AddToHeap(i, numItems): //call for i=numItems-1, numItems-2, ..., 0
1. If (\( \text{nums}[i] \) have no children) stop. //\( 2i+1 > \text{numItems}-1 \)
2. Otherwise, do the following:
   (a) Find index \( j \) of the largest child-items of \( \text{nums}[i] \).
   (b) If (\( \text{nums}[j] > \text{nums}[i] \)) then exchange(\( \text{nums}[i], \text{nums}[j] \))
       and call AddToHeap(j, numItems).

MakeHeap(numItems): //make \( \text{nums}[0..(\text{numItems}-1)] \) into a heap
1. If (\( \text{numItems} = 1 \)) stop.
   //\( \text{nums}[i] \) has no children if \( i > \text{numItems}/2 - 1 \).
2. Else, for (\( i=\text{numItems}/2 - 1; i \geq 0; i-- \)) AddToHeap(i, numItems).

HeapSort, using recursion and AddToHeap:
- Implements Selection-Sort.
- Uses Heap-structure to successively find the max, the next max, the next next max and so on, filling the places \( \text{nums}[n-1], \text{nums}[n-2], \ldots, \text{nums}[0] \) in that order with the right item.

HeapSort(numItems): //sort \( \text{nums}[0..(\text{numItems}-1)] \) by heap-sort
1. If (\( \text{numItems} = 1 \)) stop.
2. Otherwise, do the following:
   (a) If (this is the top-level call) then MakeHeap(numItems)
   (b) Exchange(\( \text{nums}[0], \text{nums}[\text{numItems}-1] \)),
       AddToHeap(0, numItems-1), and HeapSort(numItems-1).
UNDERSTANDING MakeHeap(numItems)

**Input:** nums[] = [3, 2, 4, 5, 1, 0] is not a heap; \( n = \) numItems = 6.

\[
\begin{array}{c}
a[0] = 3 \\
a[1] = 2 \\
a[2] = 4 \\
a[3] = \\text{a[i] for nums[i], in short.}
\end{array}
\]

\[
\begin{array}{c}
5 \\
\downarrow \\
1 \\
\downarrow \\
0
\end{array}
\]

\[
\begin{array}{c}
\text{MakeHeap(6)} \\
\begin{array}{c}
3 \\
\downarrow \\
2 \\
\downarrow \\
1 \\
\downarrow \\
0
\end{array}
\end{array}
\]

**MakeHeap(6):** Makes 3 calls to AddToHeap as shown below:

1. AddToHeap(2,6): max-child index \( j = 5 \);
   \[ \text{nums}[5] = 0 \neq 4 = \text{nums}[2], \text{ do nothing} \]
2. AddToHeap(1,6): max-child index \( j = 3 \);
   \[ \text{nums}[3] = 5 > 2 = \text{nums}[1], \text{ exchange(2, 5);} \]
   \[ \text{calls AddToHeap(3, 6); //does nothing} \]
3. AddToHeap(0,6): max-child index \( j = 1 \)
   \[ \text{nums}[1] = 5 > 3 = \text{nums}[0], \text{ exchange(3, 5);} \]
   \[ \text{calls AddToHeap(3, 6); //does nothing} \]
   \[ \text{we get the final heap as shown on top.} \]

**Question:** How can you modify AddToHeap(i, numItems) to eliminate some unnecessary calls to AddToHeap?
UNDERSTANDING HeapSort(numItems)

- Shown below are the recursive calls to HeapSort, calls to MakeHeap and AddToHeap, and the exchange-action, for sorting input [3, 2, 4, 5, 1, 0].
- Each node shows the input-array to its action, which is a function-call or the exchange operations.
- We only show the initial part of the array of interest at each point. An item is shown as marked by overstrike (such as 5 for 5 in 3rd child of root-node) before it is hidden away in remaining nodes.
- Calls to AddToHeap resulting from MakeHeap(6) are not shown.
PROGRAMMING EXERCISE

1. Implement the following functions; you can keep nums[0..(numItems-1)] as a global variable.

   void AddToHeap(int itemNum, int numItems)
   void MakeHeap(int numItems)
   void HeapSort(int numItems)

Keep a constant NUM_ITEMS = 10.

(a) First run MakeHeap-function for the input nums[0..9] = [0, 1, ..., 9], and show each pair of numbers (parent, child) exchanged, one pair per line (as shown below), during the initial heap-formation. These outputs will be generated by AddToHeap-function.

   ...

(b) Then, after commenting out this detailed level output-statements, run HeapSort-function. This time you show successively the array after forming the heap and after exchange with the root-item (which puts the current max in the right place). The first few lines of the output may look like:

   Successive heap array and after exchange with root-item:
   [9, 8, 6, 7, 4, 5, 2, 0, 3, 1]
   [1, 8, 6, 7, 4, 5, 2, 0, 3, 9]
   [8, 7, 6, 3, 4, 5, 2, 0, 1]
   [1, 7, 6, 3, 4, 5, 2, 0, 8]
   ...

(c) Repeat (b) also for the input [1, 0, 3, 2, ..., 9, 8].
COMPLEXITY OF INITIAL HEAP FORMATION FOR $n$ ITEMS

Cost of Adding a Node $x$:

- It may cause at most changes to the nodes along the path from $x$ to a terminal node.

```
adding this node
x to the heap
```

- The particular shape of an $n$-node heap means:

```
The shape of a heap
on n = 6 nodes
```

- At least $\lceil n/2 \rceil$ nodes are terminal nodes (no work for these).
- The number of nodes on a path from root to a terminal node is at most $\lceil \log_2(n + 1) \rceil$.

- Each change takes at most a constant time $c$ (finding largest child and exchanging the node with that child).
- Total cost of adding a node $\leq c.\lceil \log_2(n + 1) \rceil - 1 = O(\log n)$.
- Total for all nodes $\leq n. O(\log n) = O(n. \log n)$.

A better bound $O(n)$ for Total Cost: Assume $2^{m-1} \leq n < 2^m$.

- Total cost $\leq 1.(m - 1) + 2.(m - 2) + 4.(m - 3) + \cdots + 2^{(m-2)}.1 = O(n)$. 
COMPLEXITY OF HEAP-SORTING

Computing max, next max, next next max, ⋅⋅⋅:

- Each takes one exchange and one re-heap operation of adding \text{nums}[0] to the heap (of size less than the previous one).
  - This is $O(\log n)$.

- Total of this phase for all nodes: $n \cdot O(\log n) = O(n \cdot \log n)$.

Total for Heap-Sort:

- Initial heap formation: $O(n)$.
- Rest of heap-sort: $O(n \cdot \log n)$.
- Total = $O(n) + O(n \cdot \log n) = O(n \cdot \log n)$. 
APPLICATIONS OF SORTING

Car-Repair Scheduling:
You have a fleet of $N$ cars waiting for repair, with the estimated repair times $r_k$ for the car $C_i$, $1 \leq k \leq N$. What is the best repair-schedule (order of repairs) to minimize the total lost time for being out-of-service.

Example. Let $N = 3$, and $r_1 = 7$, $r_2 = 2$, and $r_3 = 6$. There are $3! = 6$ possible repair-schedules.

<table>
<thead>
<tr>
<th>Repair Schedule</th>
<th>Repair completion times</th>
<th>Total lost service-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle C_1, C_2, C_3 \rangle$</td>
<td>7, 7+2=9, 7+2+6=15</td>
<td>31</td>
</tr>
<tr>
<td>$\langle C_1, C_3, C_2 \rangle$</td>
<td>7, 7+6=13, 7+6+2=15</td>
<td>35</td>
</tr>
<tr>
<td>$\langle C_2, C_1, C_3 \rangle$</td>
<td>2, 2+7=9, 2+7+6=15</td>
<td>26</td>
</tr>
<tr>
<td>$\langle C_2, C_3, C_1 \rangle$</td>
<td>2, 2+6=8, 2+6+7=15</td>
<td>25</td>
</tr>
<tr>
<td>$\langle C_3, C_1, C_2 \rangle$</td>
<td>6, 6+7=13, 6+7+2=15</td>
<td>34</td>
</tr>
<tr>
<td>$\langle C_3, C_2, C_1 \rangle$</td>
<td>6, 6+2=8, 6+2+7=15</td>
<td>29</td>
</tr>
</tbody>
</table>

Best schedule: $\langle C_2, C_3, C_1 \rangle$,
lost service-time = $2 + (2+6) + (2+6+7) = 25$

Worst schedule: $\langle C_1, C_3, C_2 \rangle$,
lost service-time = $7 + (7+6) + (7+6+2) = 35$.

Question:

• Show that the total service-time loss for the repair-order $\langle C_1, C_2, \ldots, C_N \rangle$ is $N \cdot r_1 + (N - 1) \cdot r_2 + (N - 2) \cdot r_3 + \ldots + 1 \cdot r_N$.

• What does this say about the optimal repair-order?

• If $\langle C_1, C_2, \ldots, C_N \rangle$ is an optimal repair-order for all cars, is $\langle C_1, C_2, \ldots, C_m \rangle$ an optimal repair-order for $C_i$, $1 \leq i \leq m < N$?
PSEUDOCODE FOR OPTIMAL CAR REPAIR-SCHEDULE

Algorithm OptimalSchedule:

\[\text{Input: } \text{Repair times } r_i \text{ for car } C_i, 1 \leq i \leq N.\]
\[\text{Output: } \text{Optimal repair schedule } \langle C_{i_1}, C_{i_2}, \ldots, C_{i_N} \rangle.\]

1. Sort the cars in non-decreasing repair-times \( r_{i_1} \leq r_{i_2} \leq \cdots \leq r_{i_N}. \)
2. Optimal repair schedule \( \langle C_{i_1}, C_{i_2}, \ldots, C_{i_N} \rangle, \) with total lost-time = \( N \cdot r_{i_1} + (N - 1) \cdot r_{i_2} + (N - 2) \cdot r_{i_3} + \cdots + 1 \cdot r_{i_N}. \)

EXERCISE

1. Give #(additions and multiplications) needed to compute \( r_1 + (r_1 + r_2) + (r_1 + r_2 + r_3) + \cdots + (r_1 + r_2 + \cdots + r_N). \) (You may want to simplify the expressions first.)

2. How much computation is needed to find the lost service-times for all schedules?

3. What is the optimal car-repair order for the situation below, where a link \((x, y)\) means car \(x\) must be repaired before car \(y\)?

```
A: 3  C: 2  
D: 1    F: 5
B: 4  E: 7  G: 6
```

The number next to each car is its repair time.
ANOTHER APPLICATION: FINDING A CLOSEST PAIR OF POINTS ON A LINE

**Problem:** Given a set of points $P_i$, $1 \leq i \leq N$ ($\geq 2$) on the x-axis, find $P_i$ and $P_j$ such that $|P_i - P_j|$ is minimum.

![Diagram](image)

$\{P_4, P_6\}$ is the closest pair.

**Application:**
If $P_i$’s represent national parks along a freeway, then a closest pair $\{P_i, P_j\}$ means it might be easier to find a camp-site in one of them.

**Brute-force approach:** Complexity $O(N^2)$.
1. For (each $1 \leq i < j \leq N$), compute $d_{ij} = \text{distance}(P_i, P_j)$.
2. Find the pair $(i, j)$ which gives the smallest $d_{ij}$.

**Implementation** (combines steps (1)-(2) to avoid storing $d_{ij}$’s):

```cpp
besti = 0; bestj = 1; minDist = Dist(points[0], points[1]);
for (i=0; i<numPoints; i++)        //numPoints > 1
    for (j=i+1; j<numPoints; j++)
        if ((currDist = Dist(points[i], points[j])) < minDist)
            { besti = i; bestj = j; minDist = currDist;  }
```

**Question:**
• Give a slightly different algorithm (a variant of the above) and its implementation to avoid the repeated assignment "besti = i" in the nested for-loop; it should have fewer computations. Explain the new algorithm using a suitable test-data.
• Restate the pseudocode to reflect the implementation.
A BETTER ALGORITHM FOR CLOSEST PAIR OF POINTS ON A LINE

{P_4, P_6} is the closest pair.

The New Method:
- The point nearest to \( P_i \) is to its immediate left or right.
- Finding immediate neighbors of each \( P_i \) requires sorting the points \( P_i \).

Algorithm NearestPairOfPoints (on a line):

Input: An array \( nums[1: N] \) of \( N \) numbers.
Output: A pair of items \( nums[i] \) and \( nums[j] \) which are nearest to each other.

1. Sort \( nums[1.. N] \) in increasing order.
2. Find \( 1 \leq j < N \) such that \( nums[j + 1] - nums[j] \) is minimum.
3. Output \( nums[j] \) and \( nums[j + 1] \).

Complexity:
- Sorting takes \( O(N \log N) \) time; other computations take \( O(N) \) time.
- Total = \( O(N \log N) \).

A geometric view sometimes leads to a better Algorithm.
A MATCHING PROBLEM

Problem:
- Scores $x_1 < x_2 < \cdots < x_N$ for $N$ male students $M_i$ in a test, and scores $y_1 < y_2 < \cdots < y_N$ for $N$ female students $F_i$.
- Match male and female students $M_i \leftrightarrow F_i'$ in an 1-1 fashion that minimizes $E = \sum (x_i - y_i')^2$ ($1 \leq i \leq N$), the squared sum of differences in scores for the matched-pairs.

\[ \begin{align*}
  & y_1 \quad y_2 \quad x_1 \quad x_2 \\
  \downarrow & \quad \downarrow \quad \downarrow \quad \downarrow \\
  y_1' & \quad x_1' \quad y_2' \quad x_2'
\end{align*} \]

The possible relative positions of $x_i$’s and $y_i$’s except for interchanging $x_i$’s with $y_i$’s.

Brute-force method:
1. For each permutation $(y_1', y_2', \cdots, y_N')$ of $y_i$’s, compute $E$ for the matching-pairs $x_i \leftrightarrow y_i'$.
2. Find the permutation that gives minimum $E$.

Question: How many ways the students can be matched?

Complexity: $O(N \cdot N!)$.
- Computing $N!$ permutations takes at least $N(N!)$ time.
- Computing $E$ for a permutation: $O(N)$; total = $O(N \cdot N!)$.
- Finding minimum takes $O(N!)$.
A BETTER METHOD FOR THE MATCHING PROBLEM

Observation:
(1) The matching \{x_1 \leftrightarrow y_1, x_2 \leftrightarrow y_2\} gives the smallest \(E\) for \(N = 2\) in each of the three cases.
(2) The same holds for all \(N > 2\): matching \(i\)th smallest \(x\) with \(i\)th smallest \(y\) gives the minimum \(E\).

Question:
•? How can you prove (1)?
•? Consider \(N = 3\), and \(y_1 < y_2 < x_1 < y_3 < x_2 < x_3\). Argue that the matching \(x_i \leftrightarrow y_i\) give minimum \(E\). (Your argument should be in a form that generalizes to all \(N\) and to all distributions of \(x_i\’s\) and \(y_i\’s\).)

Pseudocode (exploits output-properties):
1. Sort \(x_i\’s\) and \(y_i\’s\) (if they are not sorted).
2. Match \(M_i\) with \(F_i\) if \(x_i\) and \(y_i\) have the same rank.

Complexity: \(O(N \log N) + O(N) = O(N \log N)\).

EXERCISE
1. Is it possible to solve the problem by recursion (reducing the problem to a smaller size) or by divide-and-conquer?

Every efficient Algorithm exploits some properties of input, output, or input-output relationship.
2-3 TREE: A GENERALIZATION OF SEARCH-TREE

2-3 Tree:

- An ordered rooted tree, whose nodes are labeled by items from a linear ordered set (like numbers) with the following shape constraints (S.1)-(S.2) and value constraints (V.1)-(V.3).

  (S.1) Each node has exactly one parent, except the root, and each non-terminal node has 2 or 3 children.

  (S.2) The tree is height-balanced (all terminal nodes are at the same level).

  (L.1) A node $x$ with 2 children has one label, $label_1(x)$, with the following property, where $T_L(x)$ and $T_R(x)$ are the left and right subtree at $x$.

    $$\text{labels}(T_L(x)) < label_1(x) < \text{labels}(T_R(x))$$

  (L.2) A node $x$ with 3 children has two labels, $label_1(x) < label_2(x)$, with the following property, where $T_M(x)$ is the middle subtree at $x$.

    $$\text{labels}(T_L(x)) < label_1(x) < \text{labels}(T_M(x)) < label_2(x) < \text{labels}(T_R(x))$$

  (L.3) A terminal node may have 1 or 2 labels.

Example. Some small 2-3 trees.

```
1  1,2
  2
  1  3
min number of labels = 3

3
  1,2  4,5
max number of labels = 5

2,4
  1  3  5
min number of labels = 5

3,6
  1,2  4,5  7,8
max number of labels = 8
```
SEARCHING A 2-3 TREE

Searching for a value $k_9 \leq x \leq k_{10}$:
- Compare $x$ and the values at the root: $k_5 < x$; branch right
- Compare $x$ and the values at the right child: $k_8 < x < k_{11}$; branch middle
- Compare $x$ and the values at the middle child: $k_9 \leq x \leq k_{10}$; if $x = k_9$ or $x = k_{10}$, the value is found, else $x$ is not there.

Role of Balancedness Property of 2-3 trees:
- Ensures optimum search efficiency.

B-tree and $B^+$-tree:
- These are more general form of 2-3 trees, which are the main data-structures used in databases to optimize search efficiency for very large data-sets. (We talk about them later.)
BUILDING 2-3 TREES

Shapes of 2-3 Trees (with different $M = \#$(terminal nodes)):

- $M = 1$
- $M = 2$
- $M = 3$
- $M = 4$
- $M = 5$
- $M = 6$

Adding 1 to an empty tree:

1

Adding 2: Find the place for 2, and add if there is space.

1 $\xrightarrow{\text{add 2}}$ 1, 2

Adding 3: Find place for 3, split if no space adding a parent node.

1, 2 $\xrightarrow{\text{add 3}}$ 1, 2, 3 $\xrightarrow{\text{split}}$ 1, 3

Adding 4: Find the place for 4 and add if there is space.

1, 2 $\xrightarrow{\text{add 4}}$ 1, 3, 4
CONTD.

Adding 5: Find place for 5, split if no space adding a parent, and adjust by merging.

Adding 6: Find place for 6, and add it if there is space.

Adding 7: Find place for 7, split if no space adding a parent, adjust by merging, and if no space, then split by adding parent again.

Question: Show the results after adding 1.1, 2.3, and 1.2.
EXERCISE

1. How many ways the 2-3 tree on the left can arise as we build the 2-3 tree by inputting \{1, 2, 3, 4\} in different order. What were the 2-3 trees before the 4th item were added? Show that the two 2-3 trees on the right arise respectively from 48 and 72 (total = 120 = 5!) permutations of \{1, 2, \ldots, 5\}.

![2-3 Trees]

2. Show the minimum and the maximum number data-items that can be stored in 2-3 trees with 5 and 6 terminal nodes. Show the labels in the nodes (using the numbers 1, 2, 3, \ldots) for both cases.

3. What information we can store at the nodes of a 2-3 tree to quickly find the key-value of the \(i\)th smallest item? Explain the use of this information to find the 9th item in the 2-3 tree below.

![2-3 Tree with Values]
TOPOLOGICAL SORTING OR ORDERING
NODES OF A DIGRAPH

Topo. Sorting (ordering):

- List the digraph’s nodes so that each link goes from left to right.
- This can be done if and only if there are no cycles in the digraph.

An acyclic digraph.

A digraph with a cycle.

Two topo. orderings:

- Any linear arrangement of the nodes will have at least link going from right to left.
- No topological ordering.

- The topological orderings = The schedules for the tasks at nodes.

Questions:

- Show all possible topological orderings of the digraph below with 4 nodes \{A, B, C, D\} and two links \{(A, B), (C, D)\}. If we add the link \( (A, D) \), how many of these topo. ordering are eliminated?

- Is it true that each acyclic digraph has at least one source-node and at least one sink-node? Is the converse also true? For each "no" answer, give an examples to illustrate your answer.

- What is the maximum number of links in an acyclic digraph with \( N \) nodes? What is the number if we allow cycles?

- Show all possible acyclic digraphs on 3 nodes (do not label nodes).
PSEUDOCODE FOR TOPOLOGICAL ORDERING

Pseudocode:
1. Choose a node \( x \) which is currently a source-node, i.e., all its preceding nodes (if any) have been output,
2. Repeat step (1) until all nodes are output.

Example. Shown below are possible choice of nodes \( x \) and a particular choice of \( x \) at each iteration of step (1).

<table>
<thead>
<tr>
<th>Choice</th>
<th>Source-Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>{A, B}</td>
<td>A</td>
</tr>
<tr>
<td>{B, C}</td>
<td>B</td>
</tr>
<tr>
<td>{C, D, E}</td>
<td>C</td>
</tr>
<tr>
<td>{D, E}</td>
<td>D</td>
</tr>
<tr>
<td>{E}</td>
<td>E</td>
</tr>
<tr>
<td>{F}</td>
<td>F</td>
</tr>
<tr>
<td>{G}</td>
<td>G</td>
</tr>
</tbody>
</table>

Relevant Data Structures:

- A stack to keep track of current source-nodes.
  - A node \( x \) enters the stack when it becomes a source-node.
  - When we remove \( x \) from the stack, we delete the links from it, add new source-nodes to the stack (if any), and output it.
- Keep track of \( \text{inDegree}(x) = \#(\text{links to } x) \) to determine when it becomes a source-node.
USE OF STACK DATA-STRUCTURE FOR TOPOLOGICAL-SORTING

\[
\begin{align*}
\text{inDegree}(y) &= \text{number of links } (x, y) \text{ to } y \\
\text{outDegree}(y) &= \text{number of links } (y, z) \text{ from } y \\
\text{source-nodes} &= \{ x : \text{inDegree}(x) \text{ is } 0 \} \\
\text{sink-nodes} &= \{ z : \text{outDegree}(z) \text{ is } 0 \} \\
\text{adjList}(x) &= \text{adjacency-list of node } x \\
\end{align*}
\]

Source nodes = \{A, B\}, \quad \text{adjList}(D) = \langle F, G \rangle

Sink nodes = \{C, G\}, \quad \text{adjList}(G) = \text{empty-list}

Stack = \text{nodes with current inDegree}(x) = 0 \text{ and not yet output.}

<table>
<thead>
<tr>
<th>Stack (top on right)</th>
<th>Node ( x ) Selected</th>
<th>Nodes and their initial or reduced inDegrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>\langle A, B \rangle</td>
<td>B</td>
<td>A: 0 B: 0 C: 1 D: 2 E: 1 F: 2 G: 2</td>
</tr>
<tr>
<td>\langle A, E \rangle</td>
<td>E</td>
<td></td>
</tr>
<tr>
<td>\langle A \rangle</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>\langle C, D \rangle</td>
<td>D</td>
<td></td>
</tr>
<tr>
<td>\langle C, F \rangle</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>\langle C, G \rangle</td>
<td>G</td>
<td></td>
</tr>
<tr>
<td>\langle C \rangle</td>
<td>C</td>
<td></td>
</tr>
</tbody>
</table>

EXERCISE

1. Show the processing in the Topo-Sorting algorithm after adding the link \((G, A)\), which creates one or more cycles in the digraph. (Remember the algorithm stops when the stack become empty.)

2. Show in a table form the processing of the digraph above using a queue instead of a stack in the topological-sorting Algorithm. Use the notation \langle A, B, C \rangle for a queue with C as the head and A as the tail. If we add \( D \), the queue becomes \langle D, A, B, C \rangle; if we now remove an item, the queue becomes \langle D, A, B \rangle.
**ADJACENCY-LIST REPRESENTATION OF A DIGRAPh**

<table>
<thead>
<tr>
<th>Array Index</th>
<th>Node name &amp; outDegree</th>
<th>Adjacency-list of node indices; array-size = outDegree(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A; 2</td>
<td>[2, 3]</td>
</tr>
<tr>
<td>1</td>
<td>B; 2</td>
<td>[3, 4]</td>
</tr>
<tr>
<td>2</td>
<td>C; 0</td>
<td>null</td>
</tr>
<tr>
<td>3</td>
<td>D; 3</td>
<td>[5, 6, 7]</td>
</tr>
<tr>
<td>4</td>
<td>E; 1</td>
<td>[5]</td>
</tr>
<tr>
<td>5</td>
<td>F; 1</td>
<td>[6]</td>
</tr>
<tr>
<td>6</td>
<td>G; 0</td>
<td>null</td>
</tr>
<tr>
<td>7</td>
<td>H; 0</td>
<td>null</td>
</tr>
</tbody>
</table>

```c
typedef struct {
    char nodeName[MAX_LENGTH];
    int outDegree,
    *adjList;  //array size = outDegree
    //*linkCosts; array size = outDegree
} st_graphNode;
```

**Adjacency Matrix Representation:**

- This is not suitable for some of our algorithms.

```
   A    B    C    D    E    F    G    H
   A    0    0    1    1    0    0    0    0
   B    0    0    0    1    1    0    0    0
   C    0    0    0    0    0    0    0    0
   D    0    0    0    0    0    1    1    1
   E    0    0    0    0    0    1    0    0
   F    0    0    0    0    0    0    1    0
   G    0    0    0    0    0    0    0    0
   H    0    0    0    0    0    0    0    0
```
TOPOLOGICAL SORTING ALGORITHM

Computation of inDegrees:
1. For (each node $i$) initialize $\text{inDegree}(i) = 0$;
2. For (each node $i$ and for each $j$ in adjList($i$))
   add 1 to $\text{inDegree}(j)$;

Initialization of stack: (stack = array of size numNodes)
1. Initialize stack with nodes of indegree zero;

Selection of a node to process:
1. Select top(stack) and delete it from the stack;

Processing node $i$:
1. Add node $i$ to output;
2. For (each node $j$ in adjList($i$)) do the following:
   (a) reduce $\text{inDegree}(j)$ by one;
   (b) if ($\text{inDegree}(j) = 0$) add $j$ to stack;

Algorithm TopSort():

Input: An acyclic digraph, with adjLists representation.
Output: A topological ordering of its nodes.

1. Compute indegrees of all nodes.
2. Initialize the stack.
3. While (stack is not empty) do the following:
   (a) Let $i = \text{top(stack)}$, delete it from stack, and add it to
       topOrder-array;
   (b) Process node $i$;
COMPLEXITY ANALYSIS OF TOPOLOGICAL-SORT ALGORITHM

Observations:

- Each link \((x, y)\) of the digraph is processed exactly twice.
  - All links are looked at once in computing the indegrees.
  - All links are looked at the second time in course of the stack updates; specifically, when we remove \(x\) from the stack, we look at all links \((x, y)\) from \(x\) the second time.

- We look at also each node \(x\) exactly \(2 \times \text{inDegree}(x) + 2\) times.
  - First time, in initializing \(\text{inDegree}(x) = 0\).
  - Then, exactly \(\text{inDegree}(x)\) many times as it is successively updated by adding 1 till it reaches the value \(\text{inDegree}(x)\).
  - Then, another \(\text{inDegree}(x)\) many times as it is successively updated by subtracting 1 till it becomes 0.
  - Finally, when it is taken out of the stack.

\[
\text{Fact: } \sum_{all \ x} \text{inDegree}(x) = \sum_{all \ x} \text{outDegree}(x) = \#(\text{links in the digraph}).
\]

Example. For the digraph on page 1.43, the two sums are

\[
0 + 0 + 1 + 2 + 1 + 1 + 2 + 2 = 9 \quad \text{and} \quad 2 + 2 + 0 + 3 + 1 + 1 + 0 = 9.
\]

Complexity:

- Since each of the operations listed above takes a constant time, total computation time is \(O(\#(\text{nodes}) + \#(\text{links}))\).
PROGRAMMING EXERCISE

1. Implement a function topologicalSort() based on the algorithm TopSort. It should produce one line of output as shown below.

   stack=[0 1], node selected = 1, topOrder-array = [1]
   stack=[0 4], node selected = 4, topOrder-array = [1 4]

   * Use a function readDigraph() to read an input file digraph.dat and build the adjacency-list representation of the digraph. File digraph.dat for the digraph on page 1.43 is shown below.

   8 //numNodes; next lines give: node (outdegree) adjacent-nodes
   0 (2) 2 3
   1 (2) 3 4
   2 (0)
   3 (3) 5 6 7
   4 (1) 5
   5 (1) 6
   6 (0)
   7 (0)

   * In topologicalSort(), use a dynamically allocated local array inDegree[0..numNodes-1]. Compute inDegrees by

   ```
   for (i=0; i<numNodes; i++) {
       outDegree = nodes[i].outdegree;
       adjList = nodes[i].adjList;
       for (j=0; j<outdegree; j++)
           inDegrees[adjList[j]]++;
   }
   ```

   or

   ```
   for (i=0; i<numNodes; i++)
       for (j=0; j<nodes[i].outDegree; j++)
           inDegrees[nodes[i].adjList[j]]++;  
   ```
EXERCISE

1. Given an ordering of the nodes of an acyclic digraph, how will you check if it is a topo. ordering? Give a pseudocode and explain your algorithm using the acyclic digraph on page 1.43.

2. How can you compute a topo. ordering without using inDegrees? (Hint: If outDegree(x) = 0, can we place x in a topo. ordering?)

3. Modify topological-sorting algorithm to compute for all nodes y, numPathsTo(y) = #(paths to y starting at some source-node). State clearly the key ideas. Shown below are numPathsTo(y) and also the paths for the digraph $\vec{G}$ on page 1.42.

<table>
<thead>
<tr>
<th>x</th>
<th>num-PathsTo(x)</th>
<th>Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>$\langle A \rangle$ // trivial path from A to A, with no links.</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>$\langle B \rangle$</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>$\langle A, C \rangle$</td>
</tr>
<tr>
<td>D</td>
<td>2</td>
<td>$\langle A, D \rangle, \langle B, D \rangle$</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>$\langle B, E \rangle$</td>
</tr>
<tr>
<td>F</td>
<td>3</td>
<td>$\langle A, D, F \rangle, \langle B, D, F \rangle, \langle B, E, F \rangle,$</td>
</tr>
<tr>
<td>G</td>
<td>5</td>
<td>$\langle A, D, G \rangle, \langle A, D, F, G \rangle, \ldots, \langle B, E, F, G \rangle,$</td>
</tr>
</tbody>
</table>

Hints:

(a) If $(x, y)$ is a link, what is the relation between numPathsTo(x) and numPathsTo(y). What does it suggest about which of them should be computed first?

(b) How will you compute numPathsTo(y) in terms of all numPathsTo(x) for $\{x: (x, y) is a link to y\}$?

4. Modify your algorithm to compute numPathsFromTo(x, y) = #(paths to node y from node x) for all nodes y to which there is $\geq 1$ path from x (which may not be a source-node). Explain the algorithm for $x = A$ and $y = F$ using the digraph shown earlier.
TOPOLOGICAL ORDERING AND TASK SCHEDULING

Precedence Constraint on Repairs:
- Each link \((x, y)\) means car \(x\) must be repaired before car \(y\).

The number next to each car is its repair time.

Possible Repair Schedules:
- These are exactly all the topological orderings.
- Two repair-schedules and their lost service-times:
  \[
  \langle A, B, C, D, E, F, G \rangle: \quad 3.7 + 4.6 + \cdots + 6.1 = 96
  
  \langle B, A, C, D, E, F, G \rangle: \quad 4.7 + 3.6 + \cdots + 6.1 = 95
  \]

Question:
- What is the optimal schedule?
- What is the algorithm for creating optimal schedule?
ALL POSSIBLE SCHEDULES

An Acyclic Digraph of Task Precedence Constraints:

![Digraph Diagram]

The Acyclic Digraph for Representing Schedules:

- Each node represents the tasks completed.
- Each path from the source-node $\emptyset$ to the sink-node $ABCDEFG$ gives a schedule.

![Schedules Diagram]

- The number of these paths gives $\#$(schedules) = $\#$(topological orderings).

SOME OTHER APPLICATIONS OF STACK DATA-STRUCTURE

Expression-Tree: It is an ordered tree (not a binary tree).

\[
x*3 + 2 - \sqrt{x^2 + 9} = ((x*3) + 2) - \sqrt{x^2 + 9}
\]

- Each non-terminal node gives an operator; also, associated with each node is the expression corresponding to the subtree at it.
- The children of a non-terminal node give the operands of the operator at the node.
- The terminal nodes are the basic operands.

Evaluation Method:
- The children of a non-terminal node are evaluated before evaluating the expression at a node.
- This requires the post-order traversal of the tree:
  Visit the children from left to right, and then the node.

Post-fix form (corresponds to post-order traversal):

\[
x 3 * 2 + x 2 ^ 9 + \sqrt{\_ -}
\]
POST-FIX EXPRESSION EVALUATION USING A STACK

Processing Method: Stack is initially empty.

- Processing an operand: add its value to stack.
- Processing an operator: remove the operands of the operator from the stack, apply the operator to those values, and add the new value to stack.
- The final value of the expression is the only item in the stack at the end of processing.

Example. If $x = 4$, then $x^3 * 2 + x^2 + 9 + \sqrt{} -$ equals 9.

Top of stack is the right in the notation $\langle \ldots \rangle$.

<table>
<thead>
<tr>
<th>Stack</th>
<th>After item processed</th>
<th>Stack</th>
<th>After item processed</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\langle 4 \rangle$</td>
<td>$x$</td>
<td>$\langle 14, 4, 2 \rangle$</td>
<td>2</td>
</tr>
<tr>
<td>$\langle 4, 3 \rangle$</td>
<td>3</td>
<td>$\langle 14, 16 \rangle$</td>
<td>~</td>
</tr>
<tr>
<td>$\langle 12 \rangle$</td>
<td>*</td>
<td>$\langle 14, 16, 9 \rangle$</td>
<td>9</td>
</tr>
<tr>
<td>$\langle 12, 2 \rangle$</td>
<td>2</td>
<td>$\langle 14, 25 \rangle$</td>
<td>+</td>
</tr>
<tr>
<td>$\langle 14 \rangle$</td>
<td>+</td>
<td>$\langle 14, 5 \rangle$</td>
<td>\sqrt{}</td>
</tr>
<tr>
<td>$\langle 14, 4 \rangle$</td>
<td>$x$</td>
<td>$\langle 9 \rangle$</td>
<td>–</td>
</tr>
</tbody>
</table>

EXERCISE

1. Show an in-fix expression that give rise to the post-fi x expression "$x^2 3 x * * + 2 / 15 +$"; make sure that you use proper parentheses as needed, but no unnecessary ones. Show the stacks in evaluating this post-fi x expression for $x = 5$.

2. Show the stacks in converting your in-fix expression in Problem #1 to the post-fi x form (using the method on next page).
CONVERTING ARITHMETIC EXPRESSIONS TO POST-FIX FORM

Input: \( x \times 3 + 2 - \sqrt{x^2 + 9} \)  \( (\times^2 = \text{exponentiation}) \)
Output: \( x 3 \times 2 + x 2^9 + \sqrt{\} \)

- Stack has only operators, including function-symbols and \( '(\text{') \).
- Operator priority: \( \{+, -\} < \{\times, /\} < \times^2 < \text{function-names} \).

Conversion Method: Initially, stack is empty.
- Processing an operand: Output it.
- Processing \( '(' \) or a function-symbol: add it to stack.
- Processing \( ')' \): remove everything from stack up to the first \( '(' \) and a function-symbol below it, if any; \( '(' \) is not added to output.
- Processing an operator \( 'op' \):
  - While ((stack \( \neq \emptyset \)) and (top(stack) \( \geq \) \( 'op' \)), remove top(stack) and output it. (See next page.)
  - Then add \( 'op' \) to stack.
- If end of input, output everything in stack.

<table>
<thead>
<tr>
<th>Stack proc.</th>
<th>Item</th>
<th>Output</th>
<th>Stack proc.</th>
<th>Item</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \langle \rangle )</td>
<td>( x \times )</td>
<td>( x \times )</td>
<td>( \langle -, \sqrt{}, \rangle )</td>
<td>( x \times )</td>
<td>( x \times )</td>
</tr>
<tr>
<td>( \langle \times \rangle )</td>
<td>( * )</td>
<td>( 3 \times 3 )</td>
<td>( \langle -, \sqrt{}, (, \times^2 \rangle )</td>
<td>( x \times )</td>
<td>( x \times )</td>
</tr>
<tr>
<td>( \langle + \rangle )</td>
<td>( + )</td>
<td>( 2 + 2 )</td>
<td>( \langle -, \sqrt{}, (, +\rangle )</td>
<td>( + )</td>
<td>( + )</td>
</tr>
<tr>
<td>( \langle - \rangle )</td>
<td>( - )</td>
<td>( 9 - 9 )</td>
<td>( \langle -, \sqrt{}, (, +\rangle )</td>
<td>( + )</td>
<td>( + )</td>
</tr>
<tr>
<td>( \langle -, \sqrt{} \rangle )</td>
<td>( \sqrt{} )</td>
<td>( \langle -, \rangle )</td>
<td>( +, \sqrt{} )</td>
<td>( \langle \rangle )</td>
<td>( - )</td>
</tr>
</tbody>
</table>
RIGHT-ASSOCIATIVE OPERATIONS AND ITS IMPACT ON POST-FIX CONVERSION

Left Association:
- \( x - y - z \) means \((x - y) - z\) but not \(x - (y - z)\).
- Post-fi x form of \( x - y - z \) is \(x y - z -\).
  Post-fi x form of \( x - (y - z) \) is \(x y z - -\).

Right Association:
- \( x \land y \land z \) means \(x \land (y \land z)\) and not \((x \land y) \land z\), where "\land" is the exponentiation operation.
  The post-fi x form of \( x \land y \land z \) is therefore \(xyz \nand\) instead of \(xy \land z \nand\).
- \( x = y = 3 \) means \(x = (y = 3)\), i.e., \(\{y = 3; x = y;\}\) instead of \(\{x = y; y = 3;\}\).
  Likewise, \( x += y += 3 \) means \(x += (y += 3)\), i.e., \(\{y += 3; x += y;\}\) instead of \(\{x += y; y += 3;\}\). Here, ’+=’ is the operator.
- Post-fi x form of \( x = y = 3: x y 3 ==\).

Processing Right Associative Operator ’op’:
- For conversion to post-fi x form, we replace the test \((\text{top}(\text{stack}) \geq \text{'op'})\) by \((\text{top}(\text{stack}) > \text{'op'})\).

Processing Assignment Operator "=" in Post-fix Form:
- In processing the post-fi x form "\(y 3 ==\)", we do not put the value of \(y\) in stack (as in the case of processing "\(y 3 +\)").
- Other special indicators (called ’lvalue’ are added).
TREE OF A STRUCTURE-DEFINITION
AND THE ADDRESS ASSIGNMENT PROBLEM

typedef struct {
    int id;
    char flag, name[14];
    double val;
} IdName;
typedef struct ListNodeDummy {
    IdName idName;
    struct ListNodeDummy *next, *prev;
} ListNode;
ListNode x;

Number of Bytes for Basic Types:

• size(int) = 4, size(char) = 1, size(double) = 8.
• size(x) = 40, not 4 + 1 + 14 + 8 + 4 + 4 = 35.

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>name[0..13]</td>
<td>5 bytes wasted</td>
<td>val</td>
<td>next</td>
</tr>
</tbody>
</table>

• An actual address allocation of the components of x:

    x = 268439696
    x.idName = 268439696
    x.idName.id = 268439696
    x.idName.flag = 268439700
    x.idName.name = 268439701
    x.idName.name[0] = 268439701
    x.idName.name[1] = 268439702
    x.idName.name[13] = 268439714
    x.idName.val = 268439720
    x.next = 268439728
    x.prev = 268439732

• Start-address(x) is a multiple of 8; because displacement(val) = 24 within x, start-address(val) is a multiple of 8.
• It makes start-address of id, next, and prev multiples of 4.
typedef struct {
    int id;
    char flag, name[14];
    double val;
} IdName;
typedef struct ListNodeDummy {
    IdName idName;
    struct ListNodeDummy *next, *prev;
} ListNode;

ListNode x;

EXERCISE
1. Give a pseudocode for determining start-address, end-address, and numBytes for all nodes of an arbitrary structure-tree. Assume you know the type of each terminal node and you have the structure-tree. (Hint: Your pseudocode must indicate: (1) the order in which the start, end, and numBytes at each node of the structure-tree are computed. and (2) how each of these is computed based on values of various quantities at some other nodes.)
LONGEST-PATHS
IN AN ACYCLIC DIGRAPH

Paths from A to E and their lengths
(1) \langle A, C, E \rangle; length = 2+3 = 5
(2) \langle A, C, D, E \rangle; length = 2+1+1 = 4
(3) \langle A, C, G, E \rangle; length = 2+5−1 = 6

• \( w(x, y) \) = length (cost or weight) of link \((x, y)\); it can be negative.
• Length of a path = sum of the lengths of its links.
• LongestPathFromTo\((A, E)\): \( \langle A, C, G, E \rangle \); length = 6.

Application:
• Critical-path/critical-task analysis in project scheduling.
• Assume unlimited resources for work on tasks in parallel.
• The new acyclic digraph for critical-path analysis:
  – Add a new "end"-node and connect each sink node to it.
  – The length of each link \((x, y)\) = time to complete task \(x\).

The number next to each car is its repair time.
The digraph for critical-path analysis.
The longest-path: \langle B, E, F, G, "end" \rangle.
Tree of Longest Paths From startNode = A:

- First, we can reduce the digraph so that the only source-node is the startNode.

- The tree contains *one* longest path from startNode to each node $x$ which can be *reached* from startNode. (It is not a binary tree or an ordered tree.)

- To obtain the reduced digraph (which is a must for the algorithm given later to work properly) we can successively delete source-nodes $x \neq$ startNode and links from those $x$.

**Question:**

- Show the reduced digraph to compute longest paths from node $B$; also show a tree of longest paths from node $B$. 

DIGRAPH REDUCTION

- We actually don’t delete any nodes/links or modify adjacency-lists.
- We pretend deletion of a link \((x, y)\) by reducing inDegree of \(y\).

\[
\begin{align*}
\text{Reductions for startNode} &= A: \\
\text{inDegree}(D) &= 2 - 1 = 1 \\
\text{inDegree}(E) &= 5 - 2 = 3 \\
\text{inDegree}(F) &= 2 - 1 = 1
\end{align*}
\]

Algorithm ReduceAcyclicDigraph(startNode):

- **Input:** An acyclic digraph in adjacency-list form
- **Output:** Reduced indegrees.

1. Compute indegrees of all nodes.
2. While (there is a node \(x \neq \text{startNode} \) and inDegree\((x) = 0\)) do:
   - if \((x \text{ is not processed})\)
   - then for each \(y \in \text{adjList}(x)\) deduce inDegree\((y)\) by 1.

Notes:

- Use a stack to hold the nodes \(x\) with inDegree\((x) = 0\) and which have not been processed yet. Initialize stack with all \(x \neq \text{startNode}\) and inDegree\((x) = 0\).
- We do not modify the adjList\((x)\) of any node, and thus the digraph is actually not changed.
- The longest-path algorithm works with the reduced indegrees.
LONGEST-PATH COMPUTATION

Array Data-Structures Used:

\[ d(x) = \text{current longest path to } x \text{ from startNode.} \]
\[ \text{parent}(x) = \text{the node previous to } x \text{ on the current longest}\]
\[ \text{path to } x; \text{parent(startNode)} = \text{startNode.} \]
\[ \text{inDegree}(x) = \text{number of links to } x \text{ yet to be looked at.} \]

Stack Data-structure Used:

\- Stack holds all nodes to which the longest-path is known, but links from which have not been processed yet.

Algorithm LongestPathsFrom(startNode):

Input: An acyclic digraph in adjacency-list form and startNode.
Output: A tree of longest paths to each \( x \) reachable from startNode.

1. Apply ReduceAcyclicDigraph(startNode).
2. Initialize a stack with startNode, let \( d(x) = -\infty \) and \( \text{parent}(x) = -1 \)
   for each node \( x \) with \( \text{indegree}(x) > 0 \), and finally let \( d(\text{startNode}) = 0 \) and \( \text{parent}(\text{startNode}) = \text{startNode} \).
3. While (stack \( \neq \) empty) do the following:
   (a) Let \( x = \text{top(stack)} \); remove \( x \) from stack.
   (b) For (each \( y \in \text{adjList}(x) \)) do:
       (i) If \( (d(x) + w(x, y) > d(y)) \), then let \( d(y) = d(x) + w(x, y) \) and \( \text{parent}(y) = x \).
       (ii) Reduce \( \text{inDegree}(y) \) by 1 and if it equals 0 then add \( y \) to stack and print the longest-path to \( y \) from startNode (using the successive parent-links) and \( d(y) \).
ILLUSTRATION OF LONGEST-PATH COMPUTATION

StartNode = A.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Node x</th>
<th>For each node y, inDegree(y) and ((d(y), \text{parent}(y)))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\langle A \rangle)</td>
<td>(A)</td>
<td>(\langle 0, A \rangle ) ((-\infty, ?))</td>
</tr>
<tr>
<td>(\langle C \rangle)</td>
<td>(C)</td>
<td>(\langle 2, C \rangle ) ((-\infty, ?))</td>
</tr>
<tr>
<td>(\langle D, G \rangle)</td>
<td>(G)</td>
<td>(\langle 6, G \rangle ) ((-\infty, ?))</td>
</tr>
<tr>
<td>(\langle D \rangle)</td>
<td>(D)</td>
<td>(\langle 3, D \rangle ) ((-\infty, ?))</td>
</tr>
<tr>
<td>(\langle E \rangle)</td>
<td>(E)</td>
<td>(\langle 7, E \rangle ) ((-\infty, ?))</td>
</tr>
<tr>
<td>(\langle F, H \rangle)</td>
<td>(H)</td>
<td>(\langle 8, H \rangle ) ((-\infty, ?))</td>
</tr>
<tr>
<td>(\langle F \rangle)</td>
<td>(F)</td>
<td>(\langle 9, F \rangle ) ((-\infty, ?))</td>
</tr>
</tbody>
</table>

- We can use minus the sum of all positive link-weights as \(-\infty\).
**EXERCISE**

1. Show the complete executions of \( \text{RreduceAcyclicDigraph}(B) \) and \( \text{LongestPathsFrom}(B) \) in the suitable table forms.

2. How many times a link \((x, y)\) is processed during the longest-path computation and when?

3. What can change as we process a link \((x, y)\) and how long does it take to all those computations?

4. Why is it that the longest-path to a node \(y\) cannot be computed until all remaining links to \(y\) (after the digraph reduction) have been processed? (For example, we must look at the links \((C, E)\), \((D, E)\), and \((G, E)\) before we can compute the longest-path to \(C\)?)
PROGRAMMING EXERCISE

1. Develop a function void longestPathsFrom(int startNode). (Use \(-\sum |w(x, y)|\), summed over all links \((x, y)\), instead of \(-\infty\).) Show the following outputs for startNode \(B\) using the example digraph discussed.

(a) Print the input digraph, with node name, nodeIndex, node’s outDegree in parenthesis, adjacency-list (with weight of the link in parenthesis) in the form:

\[ C, 2 (3): 3(1), 4(3), 6(5) \]

Put the information for each node on a separate line. There should be an appropriate header-line (like "Acyclic digraph: node name, nodeIndex, outdegree, and adjList with link-costs").

(b) Show the successive stacks (one per line) every time it is changed during the digraph reduction process. As usual give an appropriate heading before printing the stacks. Use the node names when you print the stack.

(c) Next, when the longest-paths are computed, for each link \((x, y)\) processed, show the link \((x, y)\); also, if there is a change in \(d(y)\) then shown the new \(d(y)\) and parent(y), and when inDegree(y) becomes 0 show the final values of \(d(y)\) and parent(y). For example, for startNode = A, the processing of the links \((C, E)\), \((G, E)\), and \((D, E)\) should generate output lines

\[ \text{link } (C, E): d(E) = 5, \text{ parent}(E) = C \]
\[ \text{link } (G, E): d(E) = 6, \text{ parent}(E) = G \]
\[ \text{link } (D, E): d(E) = 6, \text{ parent}(E) = G, \text{ final value} \]
CALL-RETURN TREE
OF FUNCTION-CALLS

Example.

```c
int factorial(int n) //n >= 0
{
    if ((n == 0) || (n == 1))
        return(1);
    else return(n*factorial(n-1));
}
```

```
fact(3) 3*2 = 6
    fact(2) 2*1 = 2
        fact(1) 1
```

EXERCISE

1. Show the call-return tree for the initial call Fibonacci(4), given the definition below; also show the return values from each call. Is the resulting tree a binary tree? If not, what kind of tree is it?

```c
int Fibonacci(int n) //n >= 0
{
    if ((n == 0) || (n == 1))
        return(1);
    else return(Fibonacci(n-2) + Fibonacci(n-1));
}
```
A PROBLEM IN WIRELESS NETWORK

**Problem:** Given the coordinates \((x_i, y_i)\) of the nodes \(v_i, 1 \leq i \leq N\), find the minimum transmission-power that will suffice to form a connected graph on the nodes.

- A node with transmission power \(P\) can communicate with all nodes within distance \(r = c \cdot \sqrt{P}\) from it \((c > 0\) is a constant).
- Let \(r_{\text{min}}\) be the minimum \(r\) for which the links \(E(r) = \{(v_i, v_j): d(v_i, v_j) \leq r\}\) form a connected graph on the nodes. Then, \(P_{\text{min}} = (r_{\text{min}}/c)^2\) gives the minimum transmission power to be used by each node.

![Diagram of the network](image)

The links \(E(1)\) corresponding to \(P = 1/c^2\)

**Question:**

1. What is \(r_{\text{min}}\) for the set of nodes above? Give an example to show that \(r_{\text{min}} \neq \max \{\text{distance of a node nearest to } v_i: 1 \leq i \leq N\}\). (If \(r_{\text{min}}\) were always equal to the maximum , then what would be an Algorithm to determine \(r_{\text{min}}\)?)
GROUPING NUMERICAL SCORES INTO CLASSES

Problem: Find the best grade-assignment $A$, $B$, $C$, etc to the student-scores $x_i$, $1 \leq i \leq N$, on a test. That is, find the best grouping of the scores into classes $A$, $B$, $\ldots$.

Interval-property of a group:

- If $x_i < x_k$ are two scores in the same group, then all in-between scores $x_j$ ($x_i < x_j < x_k$) are in the same group.
- Thus, we only need to find the group boundaries.

Example. Scores of 23 students in a test (one ’$\times$’ per student).

A bad 3-grouping

The best 3-grouping

Closest-Neighbor Property (CNP) for Optimal Grouping:

- Each $x_i$ is closest to the average of the particular group containing it compared to the average of other groups.

Question:

1? Give an application of such grouping for weather-data, say.

2? Find the best 2-grouping using CNP for each data-set below. Do these groupings match your intuition?
TWO EXAMPLES OF BAD ALGORITHMS

Algorithm#1 FindBuildingA:
1. Go to Main Library.
2. When you come out of the library, it is on your right.

Algorithm#2 FindBuildingA:
1. Go to the north-west corner of Quadrangle.

Questions:
1? Which Algorithm has more clarity?
2? Which one is better (more efficient)?
3? What would be a better Algorithm?
WHAT IS WRONG IN THIS ALGORITHM

Algorithm GenerateRandomTree(n): //nodes = {1, 2, ..., n}

Input: $n = \#(\text{nodes}); n \geq 2.$
Output: The edges $(i, j), i < j,$ of a random tree.

1. For (each $j = 2, 3, \ldots, n$), choose a random $i \in \{1, 2, \ldots, j - 1\}$ and output the edge $(i, j)$.

Successive Edges Produced for $n = 3$:

- $j = 2$: the only possible $i = 1$ and the edge is $(1, 2)$.
  \[ \begin{array}{c} \text{1} \text{2} \text{3} \end{array} \]

- $j = 3; i$ can be 1 or 2, giving the edge $(1, 3)$ or $(2, 3)$.
  \[ \begin{array}{c} \text{1} \text{2} \text{3} \end{array} \]

Cannot generate the tree: \[ \begin{array}{c} \text{1} \text{2} \text{3} \end{array} \]

Always test your Algorithm.

Question:

1? Does the above Algorithm always generate a tree (i.e., a connected acyclic graph)? Show all graphs generated for $n = 4$.

2? How do you modify GenerateRandomTree($n$) so that all trees with $n$ nodes can be generated (i.e., no one is excluded)?

3? Why would we want to generate the trees (randomly or all of them in some order) - what would be an application?
TREES GENERATED BY GenerateRandomTree(4)

After adding first edge

After adding second edge

Only 6 different trees are generated, each with degree(4) = 1.

Question:
1? Does the following Algorithm generate all trees on \( n \) nodes? What is the main inefficiency in this Algorithm?

1. Let \( E = \emptyset \) (empty set).
2. For \( (k = 1, 2, \ldots, n - 1) \), do the following:
   (a) Choose random \( i \) and \( j \), \( 1 \leq i < j \leq n \) and \( (i, j) \notin E \).
   (b) If \( \{(i, j)\} \cup E \) does not contain a cycle (how do you test it?), then add \( (i, j) \) to \( E \); else goto step (a).

2? Give a recursive Algorithm for generating random trees on nodes \( \{1, 2, \ldots, n\} \). Does it generating each of \( n^{n-2} \) trees with the same probability?

3? Do we get a random tree (each tree with the same probability) by applying a random permutation to the nodes of a tree obtained by GenerateRandomTree(4)?

4? Give a pseudocode for generating a random permutation of \( \{1, 2, \ldots, n\} \). Create a program and show the output for \( n = 3 \) for 10 runs and the time for 10 runs for \( n = 100,000 \).
PSEUDOCODES ARE SERIOUS THINGS

Pseudocode is a High-Level Algorithm Description:

- It must be unambiguous (clear) and concise, with sufficient details to allow
  - correctness proof and
  - performance efficiency estimation
- It is not a "work-in-progress" or a "rough" description.

Describing Algorithms in pseudocode forms requires substantial skill and practice.
TYPES OF ALGORITHMS

Problem: (1) Input (= given)  
(2) Output (= to find)

Algorithm Design

Pseudocode: (1) Key steps in the solution method  
(2) Key data-structures

- Choose a proper solution method first and then select a data-structure to fit the solution method.

Exploit Input/Output Properties:
- Exploit properties/structures among the different parts of the problem-input.
- Exploit properties/structures of the solution-outputs, which indirectly involves properties of input-output relationship.

Method of Extension (problem size $N$ to size $N + 1$, recursion)  
Successive Approximation (numerical Algorithms)  
Greedy Method (a special kind of search)  
Dynamic Programming (a special kind of search)  
Depth-first and other search methods

Programming tricks alone are not sufficient for efficient solutions.
USE OF OUTPUT-STRUCTURE

Problem: Given an array of \( N \) numbers \( nums[1..N] \), compute
\[
partialSums[i] = \sum_{j=1}^{i} nums[j]
\]
for \( 1 \leq i \leq N \).

Example. \( nums[1..5] \): 2, -1, 5, 3, 3
\( partialSums[1..5] \): 2, 1, 6, 9, 12

• There is no input-structure to exploit here.

Two Solutions. Both can be considered method of extension.

(1) A brute-force method.
\[
\begin{align*}
partialSums[1] &= nums[1]; \\
\text{for (} i=2 \text{ to } N \text{) do the following:} \\
partialSums[i] &= nums[1]; \\
\text{for (} j=2 \text{ to } i \text{) add } nums[j] \text{ to } partialSums[i]; \\
\end{align*}
\]

\[\#(\text{additions involving } nums[.]) = 0 + 1 + \cdots + (N-1) = N(N-1)/2 = O(N^2).\]

(2) Use the property \"partialSums[i + 1] = partialSums[i] + nums[i + 1]\" among output items.
\[
\begin{align*}
partialSums[1] &= nums[1]; \\
\text{for (} i=2 \text{ to } N \text{)} \\
partialSums[i] &= partialSums[i-1] + nums[i]; \\
\end{align*}
\]

\[\#(\text{additions involving } nums[.]) = N - 1 = O(N).\]

• The \( O(N) \) Algorithm is optimal because we must look at each \( nums[i] \) at least once.
ANOTHER EXAMPLE OF
THE USE OF OUTPUT-STRUCTURE

Problem: Given a binary-matrix \( vals[1..M, 1..N] \) of 0’s and 1’s, obtain \( counts(i, j) = \#(1’s \text{ in } vals[., .] \text{ in the range } 1 \leq i' \leq i \text{ and } 1 \leq j' \leq j) \) for all \( i \) and \( j \).

Example.

\[
vals = \begin{bmatrix}
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 1 & 1
\end{bmatrix} \quad \text{and} \quad \text{counts} = \begin{bmatrix}
1 & 1 & 1 & 2 \\
1 & 2 & 2 & 4 \\
1 & 3 & 4 & 7
\end{bmatrix}
\]

- Since \( vals[i, j]'s \) can be arbitrary, there is no relevant input property/structure.
- The outputs \( counts(i, j) \) have many properties as shown below; the first one does not help in computing \( counts(i, j) \).

\[
\begin{align*}
counts(i, j) &\leq \begin{cases} 
counts(i, j + 1) \\
counts(i + 1, j) 
\end{cases} \\
counts(1, j + 1) &= counts(1, j) + vals[1, j + 1] \\
counts(i + 1, 1) &= counts(i, 1) + vals[i + 1, 1] \\
counts(i + 1, j + 1) &= counts(i + 1, j) + counts(i, j + 1) \\
&\quad - counts(i, j) + vals[i + 1, j + 1]
\end{align*}
\]

Not all input/output properties may be equally exploitable in a given computation.
Algorithm:

1. Let \( \text{counts}(1, 1) = \text{vals}[1, 1] \); compute the remainder of first row \( \text{counts}(1, j), 2 \leq j \leq N \), using \( \text{counts}(1, j + 1) = \text{counts}(1, j) + \text{vals}[1, j + 1] \).

2. Compute the first column \( \text{counts}(i, 1), 1 \leq i \leq M \), similarly.

3. Compute the remainder of each row \( (i + 1 = 2, 3, \ldots, M) \), from left to right, using the formula for \( \text{counts}(i + 1, j + 1) \) above.

Exploiting the output-properties includes choosing a proper order of computing different parts of output.

Complexity Analysis:

We look at the number of additions/subtractions involving \( \text{counts}(i, j) \) and \( \text{vals}[i', j'] \).

- **Step 1:** \( N - 1 = O(N) \)
- **Step 2:** \( M - 1 = O(M) \)
- **Step 3:** \( 3(M - 1)(N - 1) = O(MN) \)
- **Total:** \( O(MN) \); this is optimal since we must look at each item \( \text{vals}[i, j] \) at least once.

**Brute-force method:**

1. For each \( 1 \leq i \leq M \) and \( 1 \leq j \leq N \), start with \( \text{counts}(i, j) = 0 \) and add to it all \( \text{vals}[i', j'] \) for \( 1 \leq i' \leq i \) and \( 1 \leq j' \leq j \).

**Complexity:** \( \#(\text{additions}) = \sum_{i=1}^{M} \sum_{j=1}^{N} ij = \left( \sum_{i=1}^{M} i \right) \left( \sum_{j=1}^{N} j \right) = O(M^2 N^2) \)
MAXIMIZING THE SUM OF CONSECUTIVE ITEMS IN A LIST

Problem: Given an array of numbers $nums[1..N]$, find the maximum $M$ of all $S_{ij} = \sum_{k} nums[k]$ for $i \leq k \leq j$.

Example: For the input $nums[1..15] = [-2, 7, 3, -1, -4, 3, -4, 9, -5, 3, 1, -20, 11, -3, -1]$, the maximum is $7 + 3 - 1 - 4 + 3 - 4 + 9 = 13$.

Brute-Force Method:

- For $(j = 1$ to $N)$, compute $S_{ij}$, $1 \leq i \leq j$, using the method of partial-sums and let $M(j) = \max \{ S_{ij} : 1 \leq i \leq j \}$.
- $M = \max \{ M(j) : 1 \leq j \leq N \}$.

Question: What is the complexity?

Observations (assume that at least one $nums[i] > 0$):

- Eliminate items equal to 0.
- The initial (terminal) −ve items are not used in a solution.
- If a solution $S_{ij}$ uses a +ve item, then $S_{ij}$ also uses the immediate +ve neighbors of it. This means we can replace each group of consecutive +ve items by their sum.
- If a solution $S_{ij}$ uses a −ve item, then $S_{ij}$ uses the whole group of consecutive −ve items containing it and also the group of +ve items on immediate left and right sides. This means we can replace consecutive −ve items by their sum.

Simplify Input: It is an array of alternate +ve and −ve items.

(nums[1..9] = [10, −5, 3, −4, 9, −5, 4, −20, 11].)
ADDITIONAL OBSERVATIONS

Another Observation: There are three possibilities:

1. \( M = nums[1] \).


3. \( nums[1] \) is not part of an optimal solution. Then we can throw away \( nums[1..2] \).

- A similar consideration applies to \( nums[N] \).

Search For a Solution for \( nums[] = [10, -5, 3, -4, 9, -5, 4, -20, 11] \):

(a) 10 or solution from \( [8, -4, 9, -5, 4, -20, 11] \) or solution from \( [3, -4, 9, -5, 4, -20, 11] \), i.e., 10 or solution from \( [8, -4, 9, -5, 4, -20, 11] \).

(b) 10 or 8 or solution from \( [13, -5, 4, -20, 11] \) or solution from \( [9, -5, 4, -20, 11] \), i.e., 10 or solution from \( [13, -5, 4, -20, 11] \).

(c) 10 or 13 or solution from \( [12, -20, 11] \) or solution from \( [4, -20, 11] \), i.e., 13 or solution from \( [12, -20, 11] \).

(d) 13 or 12 or solution from \( [3] \) or solution from \( [11] \).

(e) Final solution: \( M = 13 = 8 - 4 + 9 = 10 - 5 + 3 - 4 + 9 \).

Question:

- Is this a method of extension (explain)?

- Can we formulate a solution method by starting at the middle +ve item (divide and conquer method)?
A RECURSIVE ALGORITHM

Algorithm MAX_CONSECUTIVE_SUM: //initial version

**Input:** An array nums[1..N] of alternative +ve/-ve numbers, with nums[1] and nums[N] > 0.

**Output:** Maximum sum $M$ for a set of consecutive items.

2. If ($N >= 3$) then do the following:
   (b) Let $M_3$ be the solution obtained by applying the Algorithm to $nums[i]$, $3 \leq i \leq N$. ($M_3$ is the best solution when none of $nums[1]$ and $nums[2]$ are used.)
   else let $M_2 = M_3 = M_1$.
3. Let $M = \max \{ M_1, M_2, M_3 \}$.

**Question:**

- Characterize the solution $M_2$ (in a way similar to that of $M_3$).
- How does this show that the Algorithm is correct?
- How do you show that we make $2^{(N+1)/2} - 1$ recursive-calls for an input $nums[1..N]$?
AN EXAMPLE OF THE CALL-TREE IN THE RECURSION

\[ [10, -5, 3, -4, 9, -5, 4, -20, 11] \]
solution = \( M = \max \{10, 13, 11\} = 13 \)

\( M_1 = 10 \)

\[ [8, -4, 9, -5, 4, -20, 11] \]
solution = \( M_2 = 13 \)

\( M_1 = 8 \)

\[ [13, -5, 4, -20, 11] \]
\( M_2 = 13 \)

\[ [9, -5, 4, -20, 11] \]
\( M_3 = 11 \)

Question:

• Complete the above call-tree, examine it carefully, identify the redundant computations, and then restate the simplified and improved form of MAX_CONSECUTIVE_SUM. How many recursive-calls are made in the simplified Algorithm for \textit{nums}[1..N]?

• Let \( T(N) = \#(\text{additions involving } \textit{nums}[i] \text{ in the new Algorithm for an input array of size } N) \). Show that \( T(N) = T(N - 2) + 2 \) and \( T(1) = 0 \). (This gives \( T(N) = N - 1 = O(N) \).)

• Let \( T(N) = \#(\text{comparisons involving } \textit{nums}[i] \text{ in the new Algorithm for an input array of size } N) \), Show the relationship between \( T(N) \) and \( T(N - 1) \).
A DYNAMIC PROGRAMMING SOLUTION

Let \( M(j) = \max \{ S_{ij}: 1 \leq i \leq j \}; \) here, both \( i, j \in \{1, 3, \ldots, N\}. \)

Example. For \( \text{nums}[] = [10, -5, 3, -4, 9, -5, 4, -20, 11], \)

<table>
<thead>
<tr>
<th></th>
<th>( j = 1 )</th>
<th>( j = 3 )</th>
<th>( j = 5 )</th>
<th>( j = 7 )</th>
<th>( j = 9 )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( S_{11} = 10 )</td>
<td>( S_{13} = 8 )</td>
<td>( S_{15} = 13 )</td>
<td>( S_{17} = 12 )</td>
<td>( S_{19} = 3 )</td>
<td></td>
</tr>
<tr>
<td>( S_{33} = 3 )</td>
<td>( S_{35} = 8 )</td>
<td>( S_{37} = 7 )</td>
<td>( S_{39} = -2 )</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( S_{55} = 9 )</td>
<td>( S_{57} = 8 )</td>
<td>( S_{59} = -1 )</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( S_{77} = 4 )</td>
<td>( S_{79} = -5 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( S_{99} = 11 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( M(j) \)           | 10   | 8    | 13   | 12   | 11   |

Observations:

\[
M(1) = \text{nums}[1].
\]

\[
M(j + 2) = \max \{ M(j) + \text{nums}[j + 1] + \text{nums}[j + 2], \text{nums}[j + 2] \}.
\]

\[
M = \max \{ M(j): j = 1, 3, \ldots, N \}.
\]

Pseudocode (it does not "extend a solution" - why?):

1. \( M = M(1) = \text{nums}[1]. \)
2. For \( (j = 3, 5, \ldots, N) \) let \( M(j) = \max \{ \text{nums}[j], M(j - 2) + \text{nums}[j - 1] + \text{nums}[j] \} \) and finally \( M = \max \{ M, M(j) \}. \)

Complexity: \( O(N). \)

\[
\#(\text{additions involving \text{nums}[]}) = N - 1
\]

\[
\#(\text{comparisons in computing } M(j)'s) = (N - 1)/2
\]

\[
\#(\text{comparisons in computing } M) = (N - 1)/2
\]
ANOTHER $O(N)$ METHOD

Observation:
- For $1 \leq i \leq j \leq N$, $S_{i,j} = S_{1,j} - S_{1,(i-1)}$; here $S_{1,0} = 0$ for $i = 1$.
- If $S_{ij} = M$, then $S_{1,(i-1)} = \min \{ S_{1,i'}: i' \leq j \}$.

Solution Method: There are three steps.

1. Find $(i - 1)$’s which can possibly give maximum $S_{ij}$.
   - Find the successive decreasing items $m_0 > m_1 > m_2 > \ldots > m_n$ among $S_{1,i-1}$, $i = 1, 3, \ldots, N$. (That is, $m_k$ is the first partial-sum $< m_{k-1}$ to the right of $m_{k-1}$; $m_0 = 0 = S_{1,0}$.)
   - For each $m_k$, let $i_k$ be corresponding $i$, i.e., $m_k = S_{1,(i_k-1)}$.

2. For each $i = i_k$, find the associated $j = j_k$.
   - Let $M_{k-1} = \max \{ S_{1,j}: i_{k-1} \leq j < i_k \} = S_{1,j_k}$ for $1 \leq k \leq n$; let $M_n = \max \{ S_{1,j}: j \geq i_n \}$.

3. Let $M = \max \{ M_k - m_k: 0 \leq k \leq n \}$. 
(CONTD.)

A Slightly Larger Example.

<table>
<thead>
<tr>
<th>nums[i]</th>
<th>10</th>
<th>-5</th>
<th>3</th>
<th>-4</th>
<th>9</th>
<th>-5</th>
<th>4</th>
<th>-20</th>
<th>11</th>
<th>-6</th>
<th>10</th>
<th>-17</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>i, j</td>
<td>1</td>
<td>3</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>13</td>
<td>17</td>
<td>14</td>
<td>10</td>
<td>9</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

| S_{i,j-1} | 0  | 5  | 4 | 8  | -8 | -3 | -10|
| m_k       | m_0 | 5  | 4 | 8  | -8 | -3 | -10|
| i_k       | 1   | 9  | 13| 13 | 13 | 13 | 13|

| S_{1,j}   | 10 | 8  | 13| 12 | 3  | 7  | 4  |
| M_k       | M_0=13 | 5  | M_1=7 | 11 | M_2=5 | 13 |
| j_k       |     | j_0=9, i_2=13 |     | j_3=13, i_3=13 |     | j_3=13, i_3=13 |     |

\[ M = \max \{ 13 - 0, 7 - (-8), 4 - (-10) \} = 15 = S_{i_2,j_2} = S_{9,11}. \]

Question:

• Why can’t we call this method a "method of extension"?
**PSEUDOCODE vs. CODE**

**Characteristics of Pseudocode:**

± Shows key concepts and computation steps of the Algorithm, avoiding details as much as possible.

− Avoids dependency on any specific programming language.

+ Allows determining correctness of the Algorithm.

+ Allows choice of proper data-structures for efficient implementation and complexity analysis.

**Example.** The pseudocodes below for computing the number of positive and negative items in \( \text{nums}[1..N] \), where each \( \text{nums}[i] \neq 0 \), do not use the array-bounds. The pseudocode in \((B)\) is slightly more efficient than the one in \((A)\).

\[
\begin{align*}
(A) & \quad 1. & \text{positiveCount} &= \text{negativeCount} = 0; \\
& \quad 2. & \text{for} \ (i=0; \ i<n; \ i++) \ &//\text{each nums}[i] \ > \ 0 \ or \ < \ 0 \\
& \quad 3. & \text{if} \ (0 < \text{nums}[i]) \ &\text{positiveCount}++; \\
& \quad 4. & \text{else} \ \text{negativeCount}++; \\
\end{align*}
\]

1. Initialize positiveCount = negativeCount = 0.
2. Use each \( \text{nums}[i] \) to increment one of the counts by one.

\[
\begin{align*}
(B) & \quad 1. & \text{positiveCount} &= 0; \\
& \quad 2. & \text{for} \ (i=0; \ i<n; \ i++) \ &//\text{each nums}[i] \ > \ 0 \ or \ < \ 0 \\
& \quad 3. & \text{if} \ (0 < \text{nums}[i]) \ &\text{positiveCount}++; \\
& \quad 4. & \text{negativeCount} = n - \text{positiveCount}; \\
\end{align*}
\]

1. Initialize positiveCount = 0.
2. Use each \( \text{nums}[i] > 0 \) to increment positiveCount by one.
3. Let negativeCount = numItems – positiveCount.

Writing a pseudocode requires skills to express an Algorithm in a concise and yet clear fashion.
ANOTHER EXAMPLE OF PSEUDOCODE

Problem. Compute the size of the largest block of non-zero items in \( nums[1..N] \).

Pseudocode:

1. Initialize \( \text{maxNonZeroBlockSize} = 0 \).
2. while (there are more array-items to look at) do:
   a. skip zero’s. //keep this
   b. find the size of next non-zero block and update \( \text{maxNonZeroBlockSize} \).

Code:

```c
i = 1; maxNonZeroBlockSize = 0;
while (i <= N) {
    for (; (i<=N) && (nums[i]==0); i++); //skip 0’s
    for (blockStart=i; (i<=N) && (nums[i]!=0); i++);
    if (i - blockStart > maxNonZeroBlockSize)
        maxNonZeroBlockSize = i - blockStart;
}
```

Question:

• If there are \( m \) non-zero blocks, then what is the maximum and minimum number of tests involving the items \( nums[i] \)?
• Rewrite the code to reduce the number of such comparisons. What is reduction achieved?
• Generalize the code and the pseudocode to compute the largest size same-sign block of items.
ALWAYS TEST YOUR METHOD
AND YOUR ALGORITHM

(a) Create a few general examples of input and the corresponding outputs.
   – Select some input-output pairs based on your understanding of the problem and before you design the Algorithm.
   – Select some other input-output pairs based on your Algorithm.

   Include a few cases of input that require special handling in terms of specific steps in the Algorithm.

(b) Use these input-output pairs for testing (but not proving) correctness of your Algorithm.

(c) Illustrate the use of data-structures by showing the "state" of the data-structures (lists, trees, etc.) at various stages in the Algorithm’s execution for some of the example inputs.

   Always use one or more carefully selected example to illustrate the critical steps in your method/Algorithm.
A DATA-STRUCTURE DESIGN PROBLEM

Problem:

- We have \( N \) switches[1..\( N \)]; initially, they are all "on".
- They are turned "off" and "on" in a random fashion, one at a time and based on the last-off-first-on policy: if switches[\( i \)] changed from "on" to "off" before switches[\( j \)], then switches[\( j \)] is turned "on" before switches[\( i \)].
- Design a data-structure to support following operations:
  
  \[ M = \# \text{ switches that are "on"}. \]
  
  - Print: print the "on"-switches (in the order 1, 2, ..., \( N \)) in time proportional to \( M \).
  
  - Off(\( k \)): turn switches[\( k \)] from "on" to "off"; if switches[\( k \)] is already "off", nothing happens. It should take a constant time (independent of \( M \) and \( N \)).
  
  - On: turn "on" the most recent switch that was turned "off"; if all switches are currently "on", then nothing happens. It should take a constant time.

Example: Shown below are some on/off-operations (1 = on and 0 = off).

\[
\begin{array}{cccccccccc}
\text{Switches[1..9]:} & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\
0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
\text{Off(3):} & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
\text{Off(5):} & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
\text{On:} & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 \\
\end{array}
\]
AVERAGE-TIME ANALYSIS
FOR ALL SWITCHES TO BECOME OFF

Assume: If #(on-switches) = m and 0 < m < N, then there are m+1 switches that can change their on-off status. One of them is arbitrarily chosen with equal probability to change its on-off status.

State-diagram for N = 4: state = #(on-switches).

```
off: 1  off: 3/4  off: 2/3  off: 1/2
4    3    2    1    0

on: 1/4  on: 1/3  on: 1/2  on: 1
```

At state m = 2:
Prob(a switch going from "on" to "off") = 2/(1+2) = 2/3.
Prob(a switch going from "off" to "on") = 1/(1+2) = 1/3.

Analysis: Let $E_k =$ Expected time to reach state 0 from state $k$.

- The following equations follow from the state-diagram:

1. $E_4 = 1 + E_3$
2. $E_3 = (1 + E_2).3/4 + (1 + E_4).1/4 = 1 + 3. E_2/4 + (1+E_3)/4$
   i.e., $E_3 = 1 + 2/3 + E_2$
3. $E_2 = (1 + E_1).2/3 + (1 + E_3).1/3 = 1 + 2. E_1/3 + E_3/3$
   i.e., $E_2 = 1 + 2/2 + 2/(2.3) + E_1$
4. $E_1 = 1 + 2/1 + 2/(1.2) + 2/(1.2.3) + E_0$
   i.e., $E_1 = 1 + 2/1 + 2/(1.2) + 2/(1.2.3)$ because $E_0 = 0$

- Thus, $E_4 = 1 + (1+2/3) + (1+2/2+2/6) + (1+2/1+2/2+2/6) = 9 \frac{1}{3}$. 

OPTIMUM PAGE-INDEX SET FOR A KEYWORD IN A DOCUMENT

A Covering-Problem: $D$ is a document with $N$ pages.

- $D[i] = 1$ means page $i$ of the document contains one or more occurrences of a keyword; we say page $i$ is non-empty. Otherwise $D[i] = 0$ and we say page $i$ is empty.

- $m =$ Maximum number of references allowed in the index for the keyword. Each reference is an interval of consecutive pages; the interval $[k, k]$ is equivalent to the single page $k$.

- We want to find an optimal set of reference page-intervals $PI = \{I_1, I_2, \ldots, I_k\}$, $k \leq m$, where $I_j$’s are disjoint, $\bigcup I_j$, $1 \leq j \leq k$, covers all non-empty pages, and $|\bigcup I_j|$ is minimum.

Example. The solid dots below correspond to non-empty pages. For $m = 3$, the optimal $PI = \{2-6, 12-12, 15-20\}$. There are two optimal solutions for $m = 4$ (what are they?) and one for $m \geq 5$.

Solution by Greedy Elimination:
1. Scan $D[1..N]$ to determine all 0-blocks.
2. If ($D[1] = 0$), throw away the 0-block containing $D[1]$.
3. If ($D[N] = 0$), throw away the 0-block containing $D[N]$.
4. Successively throw away the largest size 0-blocks until we are left with $\leq m$ blocks.
A VARIATION OF PAGE-INDEX SET PROBLEM

- $\bigcup I_j$ need not cover all non-empty pages.
- Maximize $\text{Val}(PI) = \#(\text{non-empty pages covered by } \bigcup I_j) - \#(\text{empty pages covered by } \bigcup I_j) = |\bigcup I_j| - 2.\#(\text{empty pages covered by } \bigcup I_j)$.

**Example.** Let $D[1..20]$ be as before.

- For $m = 1$, the optimal $PI = \{15-20\}$, with value $6 - 2.1 = 4$. (For the original problem and $m = 1$, optimal $PI = \{2-20\}$.)
- For $m = 2$, there are two optimal solutions: $PI = \{2-6, 15-20\}$ or $PI = \{4-6, 15-20\}$, both with value $3+4 = 7$.

**Algorithm?**

- Finding an optimal $PI$ is now considerably more difficult and requires a substantially different approach. (This problem can be reduced to a shortest-path problem in a digraph.)

A slight variation in the problem-statement may require a very different solution method.

**Question:**

- What is the connection between this modified keyword-index problem and the consecutive-sum problem when $m = 1$?
- What are some possible approaches to modify the solution method for $m = 1$ for the case of $m = 2$?
AN EXAMPLE OF THE USE OF INPUT-STRUCTURE

**Problem:** Find minimum and maximum items in an array $nums[1..N]$ of distinct numbers where the numbers are initially increasing and then decreasing. (For $nums[] = [10, 9, 3, 2]$, the increasing part is just 10.)

**Example.** For $nums[] = [1, 6, 18, 15, 10, 9, 3, 2]$, minimum = 1 and maximum = 18.

**Algorithm:**
1. minimum = min \{nums[1], nums[N]\}.
2. If ($nums[N - 1] < nums[N]$) then maximum = $nums[N]$.
3. Otherwise, starting with the initial range 1..N and position 1, do a binary search. In each step, we move to the mid-point $i$ of the current range and then select the right-half of the range if the numbers are increasing ($nums[i] < nums[i + 1]$) at $i$ and otherwise select the left-half, until $nums[i]$ is larger than its each neighbor.

**Complexity:**
#(comparisons involving $nums[]$) = $O(1)$ for minimum and $O(\log N)$ for maximum.

- This is better than $O(N)$, if we do not use the input structure.

**Question:** How will you use the input structure to sort the numbers $nums[1..N]$? How long will it take?
ILLUSTRATION OF BINARY SEARCH

Test for "increasing" at $i$: $nums[i] < nums[i + 1]$

- Strictly speaking, this is not a successive approximations because at $(i + 1)$th iteration we may be further away from the maximum than at $k$th (though we are closer to the maximum at $(k + 2)$th iteration than at $k$th iteration).

- To compute maximum by the principle of extending the solution from the case $N$ to $N + 1$, we would proceed as:
  1. If $(nums[N + 1] > nums[N])$ then max = $nums[N + 1]$.
  2. Otherwise, apply the same method to $nums[1..N]$.

This can take $N - 1 = O(N)$ comparisons for $nums[1..N]$. 
**BALANCED be-STRINGS**

**Balanced be-string:** $b =$ begin or ’(’ and $e =$ end or ’)

- The unique matching of each $b$ to an $e$ on its right without crossing

A matching with crossing

- For each initial part (prefix) $x'$ of $x$, $(b, x') \geq (e, x')$, with equality for $x' = x$. In particular, $x$ starts with $b$ and ends with $e$. This means every $b$ has a matching $e$ to its right, and conversely every $e$ has a matching $b$ to its left. (Why?)

**Two basic structural properties:**

(1) **Nesting:**

If $x$ is balanced, then $bxe$ (with the additional starting $b$ and ending $e$) is balanced.

(2) **Sequencing:**

If both $x$ and $y$ are balanced, then $xy$ is balanced.

All balanced be-strings are obtained in this way starting from $\lambda$ (empty string of length 0).

**Question:** If $x_1$ and $x_2$ are balanced be-strings, $x = x_1x_2$, and $n(x) = #(matchings with or without crossing for x)$, then how do you show that $n(x_1x_2) = n(x_1)n(x_2)$?
ORDERED ROOTED TREES

- The children of each node are ordered from left to right.

Two different ordered rooted trees; as unordered rooted trees, they are considered the same.

- The ordered rooted trees have the same two structural characteristics of *nesting* and *sequencing* as the balanced *be*-strings:
  - The subtrees correspond to nesting, and
  - The left to right ordering of children of a node (or, equivalently, the subtrees at the child nodes) corresponds to sequencing.
MAPPING ORDERED ROOTED TREES TO BALANCED \textit{be}-STRINGS

\begin{itemize}
\item balString(T) = \lambda
\end{itemize}

\begin{center}
\begin{tikzpicture}
  \node[shape=circle,draw] (T1) at (0,0) {$T_1$};
  \node[shape=circle,draw] (T2) at (1,0) {$T_2$};
  \node[shape=circle,draw] (Tk) at (2,0) {$T_k$};
  \node[shape=circle,draw] (root) at (1,1) {$\cdots$};
  \draw (T1) -- (root);
  \draw (T2) -- (root);
  \draw (Tk) -- (root);
\end{tikzpicture}
\end{center}

balString(T) = bx_1e \cdot bx_2e \cdot \ldots \cdot bx_ke

where \( x_i = \text{balString}(T_i) \)

\textbf{Example.} Build the string \textit{balString}(T) bottom-up.

\begin{center}
\begin{tikzpicture}
  \node[shape=circle,draw] (T) at (1,0) {$\lambda$};
  \node[shape=circle,draw] (be) at (0,0) {$be = b\cdot \lambda \cdot e$};
  \draw (T) -- (be);
  \node[shape=circle,draw] (be1) at (-1,0) {$b$};
  \node[shape=circle,draw] (be2) at (0,0) {$be$};
  \node[shape=circle,draw] (be3) at (1,0) {$b\lambda e$};
  \draw (be1) -- (be2);
  \draw (be2) -- (be3);
\end{tikzpicture}
\end{center}

\textbf{Question:}

\begin{itemize}
\item What would be wrong if for the one-node tree we take \textit{beString}(T) = \textit{be} (instead of \( \lambda \))? \\
\item How will you show that \textit{balString}(T_1) \neq \textit{balString}(T_2) for \( T_1 \neq T_2 \), and that \textit{balString}(T) is always balanced? \\
\item How will you show that for every balanced \textit{be}-string \( x \) there is a tree \( T \) with \textit{balString}(T) = x? 
\end{itemize}

\begin{center}
\begin{tabular}{|c|}
\hline
\#(ordered rooted trees with \((n + 1)\) nodes) \\
= \#(balanced \textit{be}-strings of length \(2n\)) = \frac{1 \cdot 3 \cdots (2n - 1)}{(n!)} \cdot \frac{2^n}{(n + 1)} \\
\hline
\end{tabular}
\end{center}

\begin{itemize}
\item For length = \(2n\), \( \frac{\#(\text{balanced } \textit{be}-\text{strings})}{\#(\text{all } \textit{be}-\text{strings})} \to 0 \text{ as } n \to \infty. \)
\end{itemize}
MAPPING BINARY TREES TO BALANCED be-STRINGS

(i) A binary tree $T$. (ii) After adding a child "$e" for each null-pointer (or missing child) and labeling each original node as "$b".

**beString($T$):** Delete the rightmost $e$ of the pre-order listing of the labels $b$ and $e$ in the extended tree.

For the above $T$, the pre-order listing gives $bbeebbeee$ and $\text{beString}(T) = bbeebbebee$.

**Question:**

- If $n = \#($nodes in $T$), then how many new nodes are added?
- What is the special property of the new binary tree?
- In what sense the pre-order listing $bbeebbebee$ is almost balanced? How will you prove it?
- How is $\text{beString}(T)$ related to $\text{beString}(T_1)$ and $\text{beString}(T_2)$, where $T_1$ and $T_2$ are the left and right subtrees of $T$?
- How is the notion of nesting and sequencing accounted in $\text{beString}(T)$?
GENERATING BALANCED be-STRINGS

Problem: Compute all balanced be-strings of length $N = 2k \geq 2$.

Example: Input: $N = 4$; Output: \{bbee, bebe\}.

\[
\begin{array}{cccc}
  bbb & bbe & bbe & bbee \\
  bbe & bebe & bbee & bbee \\
  bbe & bbe & bbe & bbe \\
  bbe & bbe & bbe & bbe \\
\end{array}
\]

Only 2 out of $2^N = 16$ strings of \{b, e\} are balanced.

Idea: Generate all $2^N$ be-strings of length $N$ and eliminate the unbalanced ones.

Algorithm BRUTE-FORCE:

Input: $N \geq 2$ and even.

Output: All balanced be-strings of length $N$.

1. Generate all strings of \{b, e\} of length $N$.
2. Eliminate the be-strings that are not balanced.

Complexity:

- $O(N \cdot 2^N)$ for step (1).
- $O(N)$ to verify balancedness of each be-string in step (2).
- Total = $O(N \cdot 2^N)$.
A BETTER METHOD BY USING THE OUTPUT-STRUCTURE

Idea: Generate only the balanced be-strings using their structure.

(1) Structure within a balanced be-string
(2) Structure among balanced be-strings of a given length \( N \).

Ordered-Tree of Balanced be-strings: For \( N = 6 \).

This structure is suitable to compute all balanced be-strings of a given length by recursion, where the recursive call-tree follows the above tree-structure.

- The string at a non-terminal node is the part common to all balanced be-strings below it.
- The children of a non-terminal node correspond to filling the left-most empty position by \( b \) or \( e \).
- A node has a single child = \( b \) if number of \( b \)'s and \( e \)'s to the left of the position are equal; a node has a single child = \( e \) if all \( b \)'s are used up.
- Otherwise, it has two children (one for \( b \) and one for \( e \)).
- Terminal nodes are balanced be-strings in the lexicographic (dictionary) order from left to right.
DEVELOPING THE PSEUDOCODE

General Idea:

(1) Recursive Algorithm; each call generates a subtree of the balanced be-strings and prints those at its terminal nodes.

(2) The initial call starts with the be-string having its first position = ’b’ and the last position = ’e’.

Data-structure: beString[1..N]
Initial Parameters: beString

Initial Pseudocode for GenBalStrings(beString):

1. If (no child exist, i.e., no blanks in beString), then print beString and stop.

2. Otherwise, create each childString of beString and call GenBalStrings(childString).

Additional Parameters: firstBlankPosn (= 2 in initial call)

First refinement for GenBalStrings(beString, firstBlankPosn):

1. If (firstBlankPosn = N), then print beString and stop.

2.1. Let numPrevBs = #(b’s before firstBlankPosn) and numPrevEs = #(e’s before firstBlankPosn).

2.2. If (numPrevBs < N/2), then beString[firstBlankPosn] = ’b’ and call GenBalStrings(beString, firstBlankPosn+1).

2.3. If (numPrevBs > numPrevEs), then beString[firstBlankPosn] = ’e’ and call GenBalStrings(beString, firstBlankPosn+1).
FURTHER REFINEMENT

Additional Parameters: numPrevBs

Second refinement:
GenBalStrings(beString, rstBlankPosn, numPrevBs):

1. If (rstBlankPosn = N), then print beString and stop.

2.1. Let numPrevEs = #(e’s before rstBlankPosn).

2.2. If (2*numPrevBs < N) then beString[firstBlankPosn] = ’b’ and call GenBalStrings(beString, rstBlankPosn+1, numPrevBs+1).

2.3. If (numPrevBs > numPrevEs), then beString[firstBlankPosn] = ’e’ and call GenBalStrings(beString, rstBlankPosn+1, numPrevBs).

Implementation Notes:

• Make beString a static-variable in the function instead of passing as a parameter.

• Eliminate the parameters rstBlankPosn and numPrevB by making them static variable in the function, and use the single parameter length.

• Eliminate the variable numPrevEs (how?).

• Update rstBlankPosn and numPrevBs before and after each recursive call as needed. Initialize the array beString when rstBlankPosn = 1 and free the memory for beString before returning from the rst call.
/cc genBalBeStrings.c (contact kundu@csc.lsu.edu for
//comments/questions)
//This program generates all balanced be-strings of a given
//length using recursion. One can improve it slightly to
//eliminate the recursive calls when "length == 2*numPrevBs".

01. #include <stdio.h>

02. void GenBalBeStrings(int length) //length > 0 and even
03. { static char *beString;
04.  static int firstBlankPosn, numPrevBs;
05.  if (NULL == beString) {
06.   beString = (char *)malloc(length+1, sizeof(char));
07.   beString[0] = 'b'; beString[length-1] = 'e';
08.   beString[length] = '\0'; //helps printing
09.   firstBlankPosn = numPrevBs = 1;
10.  }
11.  if (length-1 == firstBlankPosn)
12.      printf("beString = \%s\n", beString);
13.  else { if (2*numPrevBs < length) {
14.    beString[firstBlankPosn++] = 'b';
15.    numPrevBs++;
16.    GenBalBeStrings(length);
17.  }
18.  }
19.  if (2*numPrevBs > firstBlankPosn) {
20.    beString[firstBlankPosn++] = 'e';
21.    GenBalBeStrings(length);
22.  }
23.  if (1 == firstBlankPosn)
24.  { free(beString); beString = NULL; } }

25. int main()
26. { int n;
27.  printf("Type the length n (even and positive) ");
28.  printf("of balanced be-strings: ");
29.  scanf("%d", &n);
30.  if ((n > 0) && (0 == n%2))
31.     { GenBalBeStrings(n); GenBalBeStrings(n+2); }
32. }
FINDING A BEST RECTANGULAR APPROXIMATION TO A BINARY IMAGE

Example. Black pixels belong to objects; others belong to background. Let $B = \text{Set of black pixels}.$

(i) An image $I.$

(ii) An approximation $R.$

- $R$ covers $|R - B| = 18$ white pixels (shown in grey).
- $R$ fails to cover $|B - R| = 29$ black pixels.
- $Val(R) = 29 + 18 = 47.$

$R$ = The rectangular approximation.

$B \Delta R = (B - R) \cup (R - B)$, the symmetric-difference.

$Val(R) = |(B \Delta R)|$, Value of $R$.

$Val(\emptyset) = |B| = 65$; $Val(I) = \# \text{(white pixels)} = 115$

Question: Is there a better $R$ (with smaller $Val(R)$)?

EXERCISE

1. Suppose we fix the top-row $r_t$ and the bottom-row $r_b \geq r_t$ of $R$. How do you convert the problem of finding an optimal $R$ to a maximum consecutive-sum problem?
FINDING THE BINARY IMAGE OF A CIRCLE

Problem: Find the pixels in the first quadrant belonging to the circular arc of radius \( N \) centered at \((0, 0)\).

Example. Shown below are the binary images for \( N = 6 \) to 8.

![Binary Images](image)

Each circular arc is entirely contained in the pixels representing the circle.

Some Properties of Output:

1. The lower and upper halves of the quadrant are symmetric.
2. The lower-half has at most 2 pixels in a row (why?).
3. For radius \( N \), there are at most \((2N - 1)\) pixels in the first quadrant.

Notes on Designing An Algorithm:

- Exploit the output-properties (1)-(2) to find the required pixels; we need to use only integer operations.
- Some pixels that are not in the final set will be examined.

Complexity: \( O(N) \);

Brute-Force Method: Complexity \( O(N^2) \).
THE $O$-NOTATION FOR ASYMPTOTIC UPPER BOUND

Meaning of $O(n)$:

- The class of all functions $g(n)$ which are *asymptotically bounded above* by $f(n) = n$, i.e.,

  \[ O(n) = \{ g(n): g(n) \leq c \cdot n \text{ for some constant } c \text{ and all large } n \} \]

- $c$ may depend on $g(n) ; c > 0$.
- "all large $n"$ means "all $n \geq N$ for some $N > 0"$; $N$ may depend on both $c$ and $g(n)$.

Example. We show $g(n) = 7 + 3n \in O(n)$.

We find appropriate $c$ and $N$, which are not unique.

(1) For $c = 4$, $7 + 3n \leq 4 \cdot n$ holds for $n \geq 7 = N$.

(2) For $c = 10$, $7 + 3n \leq 10 \cdot n$ or $7 \leq 7n$ holds for $n \geq 1 = N$.

A smaller $c$ typically requires larger $N$;
if $c$ is too small, there may not exist a suitable $N$.

(3) For $c = 2$, $7 + 3n \leq 2 \cdot n$ holds only for $n \leq -7$, i.e., there is no $N$. This does not say $7 + 3n \not\in O(n)$.

Each linear function $g(n) = A + Bn \in O(n)$.

Example. We show $g(n) = A \cdot n^2 \not\in O(n)$.

For any $c > 0$, $A \cdot n^2 < c \cdot n$ is false for all $n > c/A$ and hence there is no $N$. 
MEANING OF $O(n^2)$

- The class of all functions $g(n)$ which are asymptotically bounded above by $f(n) = n^2$, i.e.,

$O(n^2) = \{ g(n) : g(n) \leq c \cdot n^2 \text{ for some constant } c \text{ and all large } n \}$

- As before, $c$ may depend on $g(n)$ and $N$ may depend on both $c$ and $g(n)$.

**Example.** We show $g(n) = 7 + 3n \in O(n^2)$.

We find appropriate $c$ and $N$; again, they are not unique.

1. For $c = 1$, $7 + 3n \leq n^2$, i.e., $n^2 - 3n - 7 \geq 0$ holds for $n \geq (3 + \sqrt{9 + 28})/2$ or for $n \geq 5 = N$.

2. In this case, there is an $N$ for each $c > 0$.

**Example.** We show $g(n) = 7 + 3n + 5n^2 \in O(n^2)$.

We find appropriate $c$ and $N$.

1. For $c = 6$, $7 + 3n + 5n^2 \leq 6n^2$, i.e., $n^2 - 3n - 7 \geq 0$ holds for $n \geq 5 = N$.

2. For $c = 4$, $7 + 3n + 5n^2 \leq 4n^2$, i.e., $-n^2 - 3n - 7 \geq 0$ does not hold for any $n \geq 1$. This does not say $7 + 3n + 5n^2 \notin O(n^2)$.

Each quadratic function $g(n) = A + Bn + Cn^2 \in O(n^2)$; $g(n) = n^3 \notin O(n^2)$. 
SOME GENERAL RULES FOR $O(\cdot)$

(O1) The constant function $g(n) = C \in O(n^0) = O(1)$.

(O2) If $g(n) \in O(n^p)$ and $c$ is a constant, then $c \cdot g(n) \in O(n^p)$.

(O3) If $g(n) \in O(n^p)$ and $p < q$, then $g(n) \in O(n^q)$.

The pair $(c, N)$ that works for $g(n)$ and $n^p$ also works for $g(n)$ and $n^q$.

(O4) If $g_1(n), g_2(n) \in O(n^p)$, then $g_1(n) + g_2(n) \in O(n^p)$.

This can be proved as follows. Suppose that $g_1(n) \leq c_1 \cdot n^p$ for all $n \geq N_1$ and $g_2(n) \leq c_2 \cdot n^p$ for all $n \geq N_2$.

Then, $g_1(n) + g_2(n) \leq (c_1 + c_2) \cdot n^p$ for all $n \geq \max \{N_1, N_2\}$. So, we take $c = c_1 + c_2$ and $N = \max\{N_1, N_2\}$.

A similar argument proves the following.

(O5) If $g_1(n) \in O(n^p)$ and $g_2(n) \in O(n^q)$, then $g_1(n)g_2(n) \in O(n^{p+q})$.

Also, max $\{g_1(n), g_2(n)\} \in O(n^q)$ assuming $p \leq q$.

Question: If $g_1(n) \leq g_2(n)$ and $g_2(n) \in O(n^p)$, then is it true $g_1(n) \in O(n^p)$?
MEANING OF \( g(n) \in O(f(n)) \)

\[ O(f(n)) = \{ g(n) : g(n) \leq cf(n) \text{ for some constant } c \text{ and all large } n \} \]
\[ = \{ g(n) : \limsup_{n \to \infty} \frac{g(n)}{f(n)} = U < \infty \}. \]

All other \( \frac{g(n)}{f(n)} \) are on left \( \iff \) Only finitely many \( \frac{g(n)}{f(n)} \) are on right

\[ U \]
\[ c = U + \varepsilon, \varepsilon > 0 \]

- We sometimes write \( g(n) \) is \( O(f(n)) \) or \( g(n) = O(f(n)) \), by abuse of notation.

Examples:

(1) \( 7 + 3n = O(n) \) since \( \limsup \frac{g(n)}{n} = \limsup \frac{7 + 3n}{n} = 3 < \infty \).

(2) If \( g(n) \leq 7 + 3\log_2 n \), then \( g(n) = O(\log_2 n) \) since \( \limsup \frac{g(n)}{\log_2 n} \leq \limsup \left[ \frac{7}{\log_2 n} + 3 \right] = 3 < \infty \).

(3) If \( g(n) = 7 + 3n + 5n^2 \), then \( g(n) = O(n^2) \) since \( \limsup \frac{g(n)}{n^2} = \limsup \left[ \frac{7}{n^2} + \frac{3}{n} + 5 \right] = 5 < \infty \).

(4) \( g(n) = 2^n \not\in O(n^p) \) for any \( p = 1, 2, \ldots \).
ASYMPTOTIC LOWER BOUND $\Omega(f(n))$

- We say $g(n) \in \Omega(f(n))$ if
  \[
  \liminf_{n \to \infty} \frac{g(n)}{f(n)} = L > 0 \quad (L \text{ maybe } +\infty)
  \]
  i.e., $\frac{g(n)}{f(n)} > L - \varepsilon$ or $g(n) > (L - \varepsilon)f(n)$ for all large $n$
  i.e., $g(n) \geq cf(n)$ for some constant $c > 0$ for all large $n$.
- We also write in that case
  
  $g(n)$ is $\Omega(f(n))$ or $g(n) = \Omega(f(n))$.

Examples.

(1) $g(n) = 7 + 3n \in \Omega(n) \cap \Omega(1)$, but $g(n) \notin \Omega(n^2)$.
(2) $g(n) = 7 + 3n + 5n^2 \in \Omega(n^2) \cap \Omega(n) \cap \Omega(1)$, but $g(n) \notin \Omega(n^3)$.
(3) $g(n) = \log_2 n \in \Omega(1)$ but $g(n) \notin \Omega(n)$.

Question:

- If $g(n) \in O(f(n))$, then which of the following is true: $f(n) \in O(g(n))$, $f(n) \in \Omega(g(n))$, and $g(n) \in \Omega(f(n))$?
- If $g(n) \in \Omega(f(n))$, can we say $f(n) \in O(g(n))$?
- State appropriate rules ($\Omega1$)-(\$\Omega5$) similar to (O1)-(O5).
ASYMPTOTIC EXACT ORDER $\Theta(f(n))$

- We say $g(n) \in \Theta(f(n))$ if $g(n) \in O(f(n)) \cap \Omega(f(n))$

**Question:** Why does $g(n) \in \Theta(f(n))$ imply $f(n) \in \Theta(g(n))$?

**Example.**

1. $g(n) = 7 + 3n + 5n^2 \in \Theta(n^2)$, but not in $\Theta(n)$ or $\Theta(n^3)$.
2. If $\log_2(1 + n) \leq g(n) \leq 1 + \log_2 n$, then $g(n) = \Theta(\log_2 n)$.

**Question:** If $g_1(n) = \Theta(n^p)$, $g_2(n) = \Theta(n^q)$, and $p \leq q$, then what can you say for $g_1(n) + g_2(n)$ and $g_1(n)g_2(n)$?
COMPARISON OF VARIOUS ASYMPTOTIC CLASSES

\[ g_1(n) = \begin{cases} 
\log_2 n, & \text{for } n \text{ even} \\
n, & \text{for } n \text{ odd}
\end{cases} \]

\[ g_2(n) = \begin{cases} 
\log_2 n, & \text{for } n \text{ even} \\
n^2, & \text{for } n \text{ odd}
\end{cases} \]

\[ g_3(n) = \begin{cases} 
\log_2 n, & \text{for } n \text{ even} \\
n^3, & \text{for } n \text{ odd}
\end{cases} \]

Question:

• Place the boxes for \( \Omega(n^2) \) and \( \Theta(n^2) \) in the diagram above.

• Now, place the function \( g_4(n) = \begin{cases} 
n^{1.5}, & \text{for } n \text{ even} \\
n^{2.5}, & \text{for } n \text{ odd}
\end{cases} \)

Always give the best possible bound using \( O \) or \( \Omega \) notation as appropriate, or give the exact order using \( \Theta \).
There are infinitely many $\Theta(f(n))$ between $\Theta(1)$ and $\Theta(n)$ above; for example, we can have

$$f(n) = n^p, \ 0 < p < 1$$
$$f(n) = (\log n)^p, \ 0 < p$$
$$f(n) = \log^m(n), \ m = 1, 2, \ldots$$

For each $\Theta(f(n))$ between $\Theta(1)$ and $\Theta(n)$, $\Theta(n^k \cdot f(n))$ is between $\Theta(n^k)$ and $\Theta(n^{k+1})$ and vice-versa.

$O(f(n)) = \bigcup_{g(n) \in O(f(n))} \Theta(g(n))$

$\Omega(f(n)) = \bigcup_{g(n) \in \Omega(f(n))} \Theta(g(n))$

**Question:** Why don’t we talk of $O(1/n)$?
**ALGORITHM DESIGN vs. ANALYSIS**

![Diagram showing input $x$ to Algorithm A and output $f(x)$]

**Four (3+1) Basic Questions on an Algorithm:**

1. What does $A$ do – inputs, outputs, and their relationship?
2. How does $A$ do it – the method for computing $f(x)$.
3. Any special data-structures used in implementing the method?
4. What is its performance?
   - Time $T(n)$ required for an input of size $n$ (measured in some way).
     
     If different inputs of size $n$ require different computation times, then we can consider:

     $T_w(n)$: the worst case (maximum) time
     $T_b(n)$: the best case (minimum) time
     $T_a(n)$: the average case time

   - Similar questions on the use of memory-space.

     Since the amount of memory in use during the time $T(n)$ may vary, one can also talk about the maximum (and similarly, the minimum and the average) memory over the period $T(n)$.
1. Show the first quadrant for $N = 9$.

2. Is it true that the circles obtained in this way for various $N \geq 1$ have no pixels in common?

3. Is it true that they fill-up all the pixels?

4. Give an efficient Algorithm in a pseudocode form using the properties/structures identified above to determine the pixels on the circle of radius $N$. It should use, in particular, only integer arithmetic. How many pixels do you test (not all of which may be part of your answer) in determining the first quadrant of the circle?

5. Show that the number of pixels on the perimeter of the circle in the first quadrant is $2N - 1$. (Hint: if there are many pixels in a column as is the case on the right side of the first quadrant, then there are many columns with few pixels as is the case on the left of the first quadrant. Note that if we bent the line $i + j = N$ slightly, then it takes $2N - 1$ pixels to cover it.)

6. How will you create the three dimensional image of the surface of the sphere of radius $N$ in a similar way? (Each pixel is now a small cube.)
IMPROVE THE LOGIC/EFFICIENCY IN THE FOLLOWING CODE SEGMENTS

Ignore language-specific issues (such as "and" vs. "&&").

1. if (nums[i] >= max) max = nums[i];

2. if (x and y) z = 0;
   else if ((not x) and y) z = 1;
   else if (x and (not y)) z = 2;
   else z = 3;

3. if (x > 0) z = 1;
   if ((x > 0) && (y > 0)) z = 2;

4. for (i=1; i<n; i++)
   if (i < j) sum = sum + nums[i];  //sum += nums[i]

5. for (i=0; i<n; i++)
   if (i == j) items[i] = 0;
   else items[i] = 1;

6. for (i=1; i<n; i++)
   for (j=1; j<n; j++) {
     diff = nums[i] - nums[j];
     if (i ≠ j) sumOfSquares += diff*diff;
   }

7. for (i=1; i<n; i++)
   for (j=1; j<n; j++) {
     if (i == j) A[i][j] = -1;
     else if (M[i][j] >= M[j][i]) A[i][j] = 1;
     else A[i][j] = 0;
   }

8. for (i=0; i<3*length; i++)
   printf(" ");

9. for (i=0; i<10; i++) {
    char stringOfBlanks[3*10+1] = ""
    for (j=0; j<i; j++)
      strcat(stringOfBlanks, " ");
    if (...) printf("%s: %d\n", stringOfBlanks, i);
    else printf("%s: ...", stringOfBlanks, ...);
TOPICS TO BE COVERED

Introductory Material:
• (1) Solution method before Algorithm - necessary & sufficient condition in rectangle inclusion

Sorting:
• (1) Review and close look at some sorting Algorithms.
• (1) Sorting non-numerical things (strings, trees, flowcharts, digraphs)
• (1) Some non-trivial application of sorting.
• (2) Heap-data structure for efficient implementation of selection-sort.

---------------------- Quiz #1 ----------------------------------
• (1) 2-3 trees: a generalization of heap.

Application of Stack: Topological Sorting:
• (1) Sorting nodes of an acyclic digraph and finding all topological sorting.
• (1) Counting the number of topological sorting.
• (1) Converting an infix expression to a postfix expression using a stack and evaluating a postfix expression using stack.
• (1) Finding longest paths

---------------------- Quiz #2 ----------------------------------
• (1) Longest increasing subsequence
• (2) Depth first search and depth first tree
Minimum Weight Spanning Tree:
- (2) Finding minimum weight spanning tree

Shortest and Longest Paths:
- (1) Find all acyclic paths and cycles from a node (undirected graph)
- (2) Finding shortest paths - Dijkstra; connection between shortest and longest paths

------------- Quiz #3 ------------------------

- (2) Finding shortest paths - Floyd

String Matching:
- (2) String matching

Huffman tree:
- (1) Prefix free coding and Huffman tree

------------- Quiz #4 ------------------------
Jan 12

- I am Kundu. I want this course to be a rewarding and enjoyable experience for you so that you have a renewed sense of confidence in and love for computer science. This also means that I expect you to put a lot of effort, a full 120%.

- One of your goals for being here, I believe, is that by the end of the semester you want to become a good/expert programmer in terms of using proper data-structures and Algorithms, and you are ready to compete with other CS graduates from any other University in US or elsewhere.

- Good programmers write good (efficient and clear, not just programs that somehow produce the right output) programs, but what goes into a good program?

**Good Implementation:**

- Good choice of names for variables, functions, parameters, and files.
- Good choice of local and global variables.
- Good choice of conditions for branch-point and loops.
• To do all these good selections, you need to know some example of good Algorithms and their implementations. (We indeed learn from experience.) In this course, we are going to: (1) learn a number of interesting Algorithms and (2) practice solving some new problems using those Algorithms and their variations.

Difference between a good program/software and a good product: solves a useful problem and good interface.

• Give some example problems that the students will be able to solve by the end of semester
  – Take them from MUM-lectures; minimum energy nodes to form a connected sensor network
    Let \( r_{\text{min}} \) be the minimum \( r \) where the links \( E(r) = \{(v_i, v_j): d(v_i, v_j) \leq r\} \) form a connected graph on the nodes.

    $\begin{align*}
    &4,3 \quad v_6 \\
    &v_2 \quad v_3 \quad v_5 \quad v_7 \\
    &v_1 \quad v_4 \quad v_8
    \end{align*}$

    The links \( E(1) \).

    – Question: What is \( r_{\text{min}} \) for the set of nodes above? Give an example where \( r_{\text{min}} \neq \max \{\text{distance of a node nearest to } v_i: 1 \leq i \leq N\} \). (If \( r_{\text{min}} \) always equals the maximum, then what would be an Algorithm to determine \( r_{\text{min}} \)?)

    – Find the largest number of points \( P_i = (x_i, y_i) \) that can be roped in with a rope of length \( L \).
Some Critical-Thinking Questions On Selection Sort:

For the questions below, it suffices to consider the input to be a permutation of \( \{1, 2, \ldots, numItems\} \).

- Is it true that the number of upward data-movements are always the same as the number of downward data-movements?

- If we know that \( n \) of the data-items are out of order, what is the maximum and minimum number of data-movements? Show the example inputs in which this maximum and minimum are achieved.

- In what sense the Selection Sort minimizes data-movement?

- How many data-comparisons are made in finding the \( i \)th smallest item? What is the total number of data-comparisons? Does it depend on the input?

- Suppose a series of related exchanges are of the form \( \text{items}[i_1] \) and \( \text{items}[i_2] \), \( \text{items}[i_2] \) and \( \text{items}[i_3] \), \ldots , \( \text{items}[i(k-1)] \) and \( \text{items}[ik] \). Then argue that the indices \( \{i_1, i_2, \ldots, ik\} \) form a cycle in the permutation. Note that the exchange operations in the different cycles may be interleaved.

An Example of Creative Thinking Related to Selection Sort:

- If we view Selection Sort as a way of "filling the places by the right items", then give a high level pseudocode of an Algorithm that fits the description "finding and putting each item in the right place".

- Can you think of another variant of selection-sort?

In bubble sort is it true that if a data-item moved up, then it is never moved down? How about if we interchange "up" and "down" in the
above sentence?
• Concept of Sorting
  
  − An example: \( \langle 7, 2, 6, 1 \rangle \) becomes \( \langle 1, 2, 6, 7 \rangle \) after sorting in increasing order. Lexicographic ordering of \{bat, but, cap, happy, life\}.

  Sort names in a printed voter/airline-passenger list to quickly locate if a given name is in the list. (For electronic copy, it is not necessary to sort it; a binary search list is more suitable.) The words in a dictionary are sorted as are index-words at the end of a book.

  − How do you define the sorting problem?

    Given a set of \( n \) things \( t_j, 1 \leq j \leq n \) \( \cdots \), which are mutually comparable in some way (i.e., there is a linear order among them), find the arrangement as in: \( t_1 < t_2 < \cdots < t_n \), i.e., find the smallest item, the second smallest item, and so on.

  − Strings have linear ordering among them (the lexicographic ordering), they can be sorted: but \(<\) cat \(<\) cup \(<\) heavy \(<\) life.

  − What kinds of things cannot be sorted? If there is no linear ordering as in the case of subsets of a set. For \( S_1 = \{a, b\} \) and \( S_2 = \{b, c\} \), we have both \( S_1 \subset S = \{a, b, c\} \) but \( S_1 \not\subset S_2 \) and \( S_2 \not\subset S_1 \). Thus, \( \{S_1, S_2\} \) cannot be sorted under the subset-relation. (Indeed, we can simply declare that \( S_1 \subset S_2 \) is the sorting, but others need not accept this.)

  − What is an application (distinction between "use" and "application").

Jan 14:
• How do we compute the partial sums $d_1$, $(d_1 + d_2)$, $(d_1 + d_2 + d_3)$, ..., $(d_1 + d_2 + ... + d_n)$ most efficiently?

• How would we modify the code below to count the number of time the condition $C$ is evaluated and likewise read and write counts of $x$ and $y$ (use variables xReadCount, xWriteCount, etc)?

```c
... ...
if (C) z = 0;
else z = 1;
... ...
if (x < 3)
    y = x + 5;
... ...
```

• Discussion on the program below for generating successive binary string and its variations with numOnes (see the other file binString-prog.t).

  – The successive calls to NextBinString(3) produces 000, 001, 010, 011, 100, 101, 110, 111, and NULL.

  – The next binary string of 0110001011 is 0110001100, and its next is 0110001101.

  – Pseudocode:

    1. Find the rightmost 0 (finding from right is faster since most change take place on the rightside).
    2. If (0 is found) then make that 0 to 1 and all 1’s to its right 0.
    3. Otherwise stop.

    – The two key issues needed to develop the Algorithm are (this is true for this case, and the case where the number of 1’s is fixed and also in the case generating next permutation):

      (1) where do we start making the change, and
      (2) what is the change

This abstraction ties together all three next-item generation Algorithms.
NextBinString program

//use this function with same length repeatedly to generate all binary strings of that length
//until the return value is NULL; only then use a different length, if desired, or use the same
//length to repeat the cycle.

char *NextBinString(int length) //length > 0
{ static char *binString=NULL; //arraySize=length+1; 1 for end-of-string to help print binString

int i;
if (!binString) {
    binString = (char *)malloc((length+1) * sizeof(char));
    for (i=0; i<length; i++)
        binString[i] = '0';
    binString[length] = '\0';
}
else {
    for (i=length-1; i>=0; i--) //find position of rightmost 0
        if ('0' == binString[i]) break;
    if (i >= 0) {
        //update binString
        binString[i] = '1';
        for (i=i+1; i<length; i++) binString[i] = '0';
    }
    else binString = NULL; //reset for next call of NextBinString
}
if (binString)
    printf("binString: %s\n", binString);
return(binString);
}

Pseudocode for finding the next binary string of given length and number of ones.

1. Find the rightmost 01 (finding from right is faster since most change take place on the rightside).
2. If (found) then make that 01 to 10 and all move 1’s to its right to rightmost places.
3. Otherwise stop.

• Show a pseudocode and a piece of C/Java-code for finding the rightmost "00" in a binaryString[0..(length-1)]. Keep things as clean and efficient as possible.

1. Find rightmost 0.
2. If (the previous item is 1), then go back to step (1) and start the search from the left of the current position.

The implementation below, is cleaner than the one following it in terms of logic and is equally efficient.

```c
i = length;
do { for (i=i-1 ; i>0; i--)  
        if (0 == binString[i]) break;  
    } while (1 == binString[--i]);
for (i=length-1; i>0; i--) //warning: body of for-loop updates i
    if (0 == binString[i]) & & (0 == binString[--i]) break;
```

1. **Bonus:** Let $R(W, H)$, where $W \geq H > 0$, denote a rectangle with width $W$ and height $H$. How will you determine if a rectangle $R_1(W_1, H_1)$ can be placed completely inside another rectangle $R_2(W_2, H_2)$, and if so how can you find at least one an actual placement (there can be more than one ways to place $R_1$ inside $R_2$). (Note that the problems of placing a circle inside a rectangle and of placing a rectangle inside a circle are easy.) First, show that if $D_1 = D_2$, where $D_i$ is the length of the diagonal of $R_i$, then the only way $R_1$ can be placed inside $R_2$ is $R_1 = R_2$, i.e., $W_1 = W_2$ (and hence $H_1 = H_2$).

2. **Homework:** Consider again the car-repair problem, where now we have two repair-men. Suppose we have four cars $C_1, C_2, C_3,$ and $C_4$ with the repair-times 7, 2, 6, and 1 respectively. Show all possible repair-schedules (who repairs which cars and in what order) which has the minimum total lost-service time; the person who repairs $C_1$, call him $A$ and call the other person $B$.

- What do you think (guess) is the general rule for creating the best repair-schedule?
- If there are $2n$ cars and two repair men, what is the number of optimal repair-schedules?
3. **Homework:** How to compute the successive permutations of \(\{1, 2, \ldots, n\}\) in the lexicographic order?

Given two permutations \(p = (p_1, p_2, \ldots, p_n)\) and \(q = (q_1, q_2, \ldots, q_n)\), we say \(p < q\) if for the leftmost position \(i\) where \(p_i \neq q_i\), we have \(p_i < q_i\). The lexicographic ordering of the permutations for \(n = 3\) is

\[(1, 2, 3) < (1, 3, 2) < (2, 1, 3) < (2, 3, 1) < (3, 1, 2) < (3, 2, 1)\]

For \(n = 9\), what is the first permutation \(p\) that starts with \((4, 3, 1, 9, 6, \ldots)\) and what is the one next to it, and the one next to that? Also, what is the one previous to \(p\)? Show the pseudocode for computing the permutation which is next to a given permutation \((p_1, p_2, \ldots, p_n)\).

**Jan 21**

- Discuss homework problems for NextPermutation(numItems), two-person car repair scheduling, rectangle placement, and programming of NextBinString(length, numOnes).

- The Algorithms for NextBinString(length), NextBinString(length, numOnes), and NextPermutation(numItems) have the following common form although they differ in the details of each of the three steps.

  1. Find the rightmost place where a change occurs.
  2. Make the change at that place
  3. Make the change to its right.

- Problem random generation of a binary string of length \(n\):

  1. Save all the strings in a file.
  2. Create a random number \(0 \leq k < \text{numStrings}\).
  3. Select \(k\)th string.

Problem too much time to compute all of them and too much storage to save. Better approach

Compute successive bits of the string with suitable probability.

- Algorithm for random permutation;

  1. For (each \(0 \leq i < \text{numItems}\)) choose randomly an item from \(\{0, 1, 2, \ldots, n-1\}\) which is different from previous items.

An implementation (very inefficient):

```c
1. permutation[0] = random()%numItems;
2. for (i=1; i<numItems; i++) {
3.    do { item = random()%numItems;
4.        for (j=0; j<i; j++)
5.            if (permutation[j] == item) break;
6.        } while (j < i);
7.    permutation[i] = item;
8. }
```

Better idea: keep track of remaining items and choose one at random from the remaining items.

- **Homework+Program:** Find a better way and compare the average number of times random() is called for generating \(10^6\) cases of random permutations for numItems = 50. Also, show the details for numItems = 4 and 5 different runs of RandomPermutation(4), show the sequence of random items generated by the brute-force method as each new permutation[i] is determined, the final permutation, and the counts of random() in each case.
A variation of car-repair problem that can be solved in the same way: we have customers lined up in a shop to get some service, and we want to serve them in a way that reduce their total weight time.

Now we can introduce some probability that a customer may leave at any time based on an (say) exponential distribution, i.e., a customer leaves within a time period \( t \) with probability \( 1 - e^{-t} \) and the probability \( e^{-t} \) that he does not leave (where \( x = e^{-l} \) for some \( l > 0 \), i.e., \( 0 < x < 1 \)). Then what is the best order-of-service to maximize the profit, i.e., the amount of service that can be provided.

- If we have just two customers with \( d_1 = 2 \) and \( d_2 = 6 \), then the processing order \( \langle C_2, C_1 \rangle \) is optimal with the expected extra return \( [8x^6 + 6.(1 - x^6)] - [8x^2 + 2.(1 - x^2)] \geq 0 \) for all \( 0 < x = e^{-l} < 1 \).

- If you have two repair-men, then what is the optimal distribution of the work between them for the \( d_i \)-values \{2, 6, 7, 11, 13\}?

- A generalization to the case of a precedence constraints among the tasks.

Suppose I have 6 pieces of tools \{A, B, ..., F\} in my machine shops which need repair. Also, some of the tools themselves are needed to repair some of the other tools as shown below; here, tool A is needed to repair both the tools C and D (as indicated by the links \( (A, C) \) and \( (A, D) \) respectively). The number next to each node is the time needed to repair that tool.

Here two of the many possible repair-sequence are: \( \langle A, B, C, D, E, F \rangle \) and \( \langle B, A, C, D, E, F \rangle \).

Here, the best repair-sequence is: \( \langle A, C, B, D, E, F \rangle \).

You always repair the tool which has no precedence constraint (i.e., is not waiting for some other tool to be repaired) and which has the smallest repair time.

<table>
<thead>
<tr>
<th>Set of tools ready for repair</th>
<th>Best choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 3, B: 4</td>
<td>A</td>
</tr>
<tr>
<td>B: 4, C: 2</td>
<td>C</td>
</tr>
<tr>
<td>B: 4</td>
<td>B</td>
</tr>
<tr>
<td>D: 1, E: 7</td>
<td>D</td>
</tr>
<tr>
<td>E: 7</td>
<td>E</td>
</tr>
<tr>
<td>F: 5</td>
<td>F</td>
</tr>
</tbody>
</table>

- **Homework:** Find 5 different repair-sequences and the associated total lost-time for each of them. How many repair-sequences are there?

  - How do you compute the number of possible repair-sequences for a general precedence digraph;

  - We can use a shortest-path computation on the digraph below to get the best repair-sequence. The link \( (S_i, S_j) \) connecting node \( S_i \) to \( S_j \) corresponds to the repair job for tool \( T_k \in S_j - S_i \), and the cost of the link is \( d_k(N - |S_j|) \), which is the total contribution to the delay for repair of the remaining \( N - |S_j| \) tools.
Below each node we show the shortest-path length from the node $\emptyset$.

- What is the basic assumption in sorting: there is a linear order among the items to be sorted.
  - We have seen linear ordering of numbers, strings, and permutations.
  - Can we use the linear order of binary strings of length 3 to provide a linear order on subsets of $\{a, b, c\}$? What happens if we associate $a$ with the leftmost bit, $b$ with middle bit, and $c$ with rightmost bit and map $010 \rightarrow \{b\}$, $101 \rightarrow \{a, c\}$, and so on giving
    \[
    \{c\} < \{b\} < \{a\} < \{b, c\} < \{a, c\} < \{a, b\} < \{a, b, c\}.
    \]

- Following is a pseudocode for Insertion-sort Algorithm, where we have used recursion; here, numItems = #(items to be sorted) = size(input array). Here, you know nothing of the final result until the very end.

1. If (numItems = 1) then stop.
2. Otherwise, sort the first (numItems-1) items from the input and insert the last item.

For the initial input array $[7, 2, 6, 1]$, the recursion proceeds as follows:

```
[7, 2, 6, 1] → insert 1 in [2, 6, 7]: [2, 6, 7, 1] → [2, 6, 7] → [2, 1, 6, 7] → [1, 2, 6, 7]
```

Lots of data-movements: $[7, 2, 6, 1] \rightarrow [2, 7, 6, 1] \rightarrow [2, 6, 7, 1] \rightarrow [2, 6, 1, 7] \rightarrow [2, 1, 6, 7] \rightarrow [1, 2, 6, 7]$. 

Worst case: $1 + 2 + 3 + \cdots + (n - 1) = \frac{n(n-1)}{2}$, arising for input $[7, 6, 2, 1]$; same for the number of comparisons. Best case: #(data movements) = 0 and #(comparisons) = $n - 1$.

Indeed, you can use a for loop:

1. For ($i = 1$ to numItems-1)
   insert $\text{nums}[i]$ among $\text{nums}[0..i-1]$ so that $\text{nums}[0..i]$ are sorted.
Insertion: pseudocode and implementation (where steps (1)-(2) are combined):

Pseudocode:
1. Find the position \(0 \leq j \leq i\) for \(nums[i]\).
2. If \((j < i)\) then move items in \(nums[j..(i-1)]\) one position right (save \(nums[i]\) before this) and place \(nums[i]\) in position \(j\).

Implementation:
1. \(\text{for } (j=i-1; j>=0; j--)\)
2. \(\text{if } (nums[j+1] > nums[j]) \text{ break; } /\!/>=\)
3. \(\text{else interchange } nums[j+1] \text{ and } nums[j];\)

- Selection Sort: Here, you do know part of the final output at the intermediate phases (unlike insertion-sort). This is iterative from the output point of view while insertion-sort iterative from an approximation viewpoint. The recursive form below applies recursion after some preliminary computation (cf. insertion-sort)
  1. If \((\text{numItems} = 1)\) do nothing.
  2. Otherwise, Find the largest item and interchange it with the items[\text{numItems-1}], if necessary, and then apply the method recursively to items[0..\text{numItems-2}].

For input array [2, 7, 1, 6], the recursion proceeds as shown below.

Few data-movements here: maximum of 1 per each recursion’s own direct computation. Worst case: \(n - 1\).
The number of comparisons is always \((n - 1) + (n - 2) + \cdots + 3 = 2 + 1 = \frac{n(n-1)}{2}\).

- Merge sort:
  1. If \((\text{numItems} == 1)\) do nothing.
  2. Otherwise divide input into two equal (or close to equal) halves (first half size \(\leq\) second half size) and sort each part.
  3. Merge the two sorted part.

Show with an example of 8 items that merging may take longer if we divide into 2/3 and 1/3 parts instead of into 1/2 and 1/2.

An extreme case of this division into first \(n - 1\) and the last item gives insertion sort.

- Homework. For the input \(\text{nums}[0..3] = [7, 2, 6, 1]\), show the sequence of successive value-pairs compared in the insertion-sort Algorithm (instead of writing the pair as \((\text{nums}[0], \text{nums}[1])\), write \((7,2)\) and not \((2, 7)\)). Also, show the whole \(\text{nums-array}\) every time some data-movement takes place in the array. In what input situation, we have the maximum number of data-movements (give an example for an array of 5 items)? In what input situation, we have the maximum number of comparisons (give example)?

- Homework. Give a recursion-based pseudocode (not C-code) for insertion-sort. Imagine that you are doing this to develop a program later for the function \text{InsertionSort}(\text{int } *\text{nums}, \text{int } \text{numItems}). Show the successive calls that will be made for the initial input \(\text{nums}[0..3] = [7, 2, 6, 1]\).

- ONUUS. Use the above piece of code to create a function \text{GenRandomPermutation}(\text{int } \text{numItems}), which prints all the successive random items generated and putting a '*' next to an item when it becomes part of the permutation (you can put all the values of item in a line). It should also count the total number of
random numbers generated in creating a random permutation. Show the detailed output for 5 calls to the function for numItems = 4. Finally, show the average value of count for 5 calls to the function for numItems = 100000 (don’t show the details of random items generated for these permutations).

- **Homework:** Show a similar pseudocode for a recursive form of Selection-sort Algorithm and show its call-return tree and the computations for the input [7, 2, 6, 1].

**Feb 09**

- 2-3 tree: An ordered rooted tree, whose nodes are labeled by items from a linear ordered set (like numbers) with the following properties (T.1)-(T.3) and (L.1)-(L.3). Shown below are few small 2-3 trees.

(T.1) Each node has exactly one parent, except the root

(T.2) It is height balanced: all terminal nodes are at the same distance from the root.

(T.3) Each non-terminal node has either 2 children or 3 children.

(L.1) A node $x$ with 2 children has one label, $label_1(x)$, with the properties:

\[
\text{labels}(T_L(x)) < label_1(x) \quad \text{where} \quad T_L(x) \text{ is left-subtree at } x,
\]

\[
label_1(x) < \text{labels}(T_R(x)) \quad \text{where} \quad T_R(x) \text{ is right-subtree at } x
\]

(L.2) A node $x$ with 3 children has two labels, $label_1(x) < label_2(x)$, with the properties:

\[
\text{labels}(T_L(x)) < label_1(x) \quad \text{where} \quad T_L(x) \text{ is left-subtree at } x,
\]

\[
label_1(x) < \text{labels}(T_M(x)) < label_2(x) \quad \text{where} \quad T_M(x) \text{ is middle-subtree at } x
\]

\[
label_2(x) < \text{labels}(T_R(x)) \quad \text{where} \quad T_R(x) \text{ is right-subtree at } x
\]

(L.3) A terminal node may have one label or two labels.

- Example of 2-3 trees with different number of terminal nodes:
Feb 11

- How many ways can the 2-3 tree on left can arise? There are 12 ways, i.e., 12 possible input sequences (permutations of \(\{1, 2, 3, 4\}\)) that gives this 2-3 tree. The only other 2-3 tree with the labels \(\{1, 2, 3, 4\}\) is also obtained in 12 ways, covering \(12 + 12 = 24 = 4!\) permutations of \(\{1, 2, 3, 4\}\).

- It came from a 3 node 2-3 tree (of the same shape) – why? The 3-node 2-3 tree can be only one of the following, and by adding 2 to the first tree and 1 to the second tree we get the above tree.

- How many ways we get the first 2-3 tree above? There are 6 ways, i.e, from 6 different permutations of \(\{1, 3, 4\}\) and they all come from 3 different one-node 2-3 tree.

- **Homework:** Show all possible structure of 2-3 tree with 5 terminal nodes and 6 terminal nodes. Also, label the nodes of each with the numbers 1, 2, 3, … for the case of minimum number of data items in the nodes and also for the case of maximum number of data items in the nodes.

- **Homework.** Show that the following 2-3 trees arise from 48 and 72 (total = 120 = 5!) permutations of \(\{1, 2, \cdots, 5\}\). In each case, they come from a 3-node 2-3 tree.

- **Homework.** What additional information we could at each node of 2-3 tree if we want to quickly find the key-value of the \(i\)th smallest item? Show how you will use that to determine the 9th item in the following 2-3 tree \((k_1 < k_2 < \cdots)\).

- How to choose the probability for successive bits in the binary string of length \(n\) and numOnes \(m\)?

<table>
<thead>
<tr>
<th>Probability</th>
<th>Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
1. \(\text{Prob}(0) = \frac{1}{2}\) for each position

2. \(\text{Prob}(0)\) depends on position \(n' = \text{remainingLength}\), and \(m' = \text{remainingNumOnes}\) (\(\text{prob}(0) = \frac{C_{n'-1}^{m'}}{C_{n'}^{m'}}\))

3. Depends on position \(n' = \#(\text{remaining symbols})\)
   \(\text{prob}(s) = \frac{1}{n'}\) for each remaining symbols

The case of length \(n = 4\), numOnes \(m = 2\), and numStrings \(N = 6\):

```
string: 0 0 1 1
prob = 1/2
```

```
string: 0 1
prob = 2/3
```

```
string: 0 0
prob = 1/2
```

```
string: 1...
prob = 1/2
```

```
string: ...
```

Feb 18

CA: circle at \((0,0)\)
CB: circle at \(CA+(x,0)\); line -> from CA to CB
CC: circle at \(CA+(x/2,-y)\); line -> from CA to CC
(ii) The acyclic digraphs on "\(n = 3\) nodes and maximum number links 3." at CC.s-(0,z)

- Given an acyclic digraph, finding #(paths from \(x\) to \(y\)).

  Method #1: Assume that we have computed indegree of each node.
  (1) Initialize the stack by adding each source-node to it.
  (2) For each node \(z\), initialize \(p(z) = \#(\text{paths from source-nodes to } z) = 0\). Also, initialize \(p(x) = 1\).
  (3) Do the following until indegree\((y) = 0:\)
      (a) Let \(z = \text{top(stack)}\); remove \(z\) from stack.
      (b) For each node \(w\) in adjList\((z)\), reduce indegree\((w)\) by 1 and if indegree\((w) = 0\) then add it to stack. Also, add \(p(z)\) to \(p(w)\).

- **Homework.** Show in the table form how the topological sorting would proceed on the same digraph with the nodes \{A, B, ..., G\} (which we looked at before Mardi Gras holidays) when we use a queue instead of a stack to keep the current nodes of indegree 0 that have not been processed yet. (This might give a different topological sorting/ordering than the one using a stack.)

  Suppose we write a queue in the form <A, B, C>, where C is the head of the queue and A is the tail. Then adding D to the queue would give <D, A, B, C>, D being the new tail. If we want to take an item of the queue out, then we have to take the head-item C out and this would make the new queue <D, A, B>.

  Your table should show the queue (with head on right and tail on left), the node selected, the updated indegrees, and the new topological ordering. This is similar to the table we made using the stack for topological ordering.

**Depth-First Search**
**Depth-first search of a graph and its applications:**

1. Finding an \(xy\)-path,
2. Finding if the graph is connected,
3. Finding a cut-vertex,
4. Finding a bicomponent, etc.

**Given any spanning tree of a connected graph and having chosen any node as the root,** the non-tree edges can be classified as back-edges and cross-edges.

- If there are no cross-edges then we can think of the tree as a depth-first tree.
- If there are no back-edges then we can think of the tree as a breadth-first tree. (This is also the tree of shortest paths from the root, with 0/1 weights for the edges; some of the cross edges may represent alternative shortest paths.)
- If we disregard the ordering of the children of a node, then there is just one df-tree and one bf-tree for each choice of root node.
- Thus, all but \(n + n\) spanning trees are neither df-trees and nor bf-trees.
- A df-tree is a bf-tree if and only if the graph has no cycles.

**Connected graph:** there is a path between any pair of nodes \(x\) and \(y\) \((y \neq x)\).

(i) A connected graph on nodes \{A, B, ..., E\}.

(ii) A disconnected graph on nodes \{A, B, ..., F\}.

**Homework.** Is it true that "if there is path from some node \(z\) to every other node, then there is a path between every pair of nodes"? Why is this result important (in determining connectivity of a graph)?

**Cut-vertex \(x\):**

removal of \(x\) and its adjacent edges destroys all paths (one or more) between some pair of nodes \(y\) and \(z\); we say \(x\) separates \(y\) and \(z\).

In this case every path from \(y\) to \(z\) has to go through \(x\), and thus \(#(\text{acyclic path from } y \text{ to } z) = #(\text{acyclic paths from } y \text{ to } x) \times #(\text{acyclic paths from } x \text{ to } z)\).

- \(B\) and \(C\) are the only cut-vertices in the first graph; the other graph has no cut-vertex.

**Homework.** What is the minimum edges that need to be added to the first graph so that it has no cut-vertex.

**Depth-first search of a connected graph:**

1. Depends on the start-vertex and the ordering of nodes in the adjacency-list of nodes.
2. Produces an ordered rooted tree, with root = start-vertex; it is called the depth-first tree. The children of a node are ordered from left to right in the order they are visited.
3. Each non-tree edge creates a cycle in the graph.
4. Each edge \((x, y)\) of the graph is visited twice: once in the direction \(x\) to \(y\) and once in the direction \(y\) to \(x\).
Cross-edge and back-edge:

There are no cross-edges in the df-tree; each edge joins a node with a parent or with an ancestor. 

\((x, y)\) is a back edge if \(\text{dfLabel}(x) > \text{dfLabel}(y)\) and \(y \neq \text{parent}(x)\)

The start-vertex is a cut-vertex if and only if it has more than one child.

**Homework.** Show in a similar table form the result of depth-first processing when each adjacency-list is ordered in the reverse of alphabetical-list.

**Homework.** For the graph below, show all possible depth-first trees that may arise if we change the start-vertex and order the adjacency list in different ways.

**BONUS** Consider the depth-first tree shown above. Show the maximum possible number of back edges. Is there any cut-vertices if all those edges are present in the graph?

Mar 09

**Algorithm DepthFirstTraverse:**

Use the following local data structures and variables in the function. (You could add parent-information to the structure GraphNode if the depth-first tree is to be used later for some other purpose.)

- `lastDfLabel`: \(0\) initially; it is incremented by one before assigning to a node.
- `dfLabels[0..numNodes-1]`: each \(\text{dfLabels}[i] = 0\) initially.
- `nextToVisit[0..numNodes-1]`: each `nextToVisit[i] = 0` initially; `nextToVisit[i]` gives the position of the item in `adjList` of node \(i\) that is to be visited next from node \(i\), i.e., the next link to visit from node \(i\) is link \((i, j)\), where \(j = \text{nodes}[i].\text{adjList}[\text{nextToVisit}[i]]\).
- `stack[0..numNodes-1]`: initialized with the startNode; recall that this gives the path in the depth-first tree from the root to the current node.
- `parents[0..numNodes-1]`: \(\text{parents}[i]\) is the parent of node \(i\).

Pseudocode: //It has a little bug; find this out as you create the program and test it, and then fix the bug.

1. Initialize `lastDfLabel`, `dfLabels-array`, `parents-array`, `nextToVisit-array`, the stack; also, let `parent[currentNode] = currentNode` (or `-1`).
2. While (stack \(\neq\) empty) do the following:
   (a) Let `currentNode = top(stack)`; update `lastDfLabel` and let `dfLabels[currentNode] = lastDfLabel`.
   (b) If `nextToVisit[currentNode] = degree[currentNode]` then backtrack by throwing away top of stack and go back to step (2).
(c) Otherwise, let nextNode = the node in position nextToVisit[currentNode] in adjList of currentNode, and update nextToVisit[currentNode].

(c) Classify the type of the link (currentNode, nextNode) as follows

1. tree-edge: if dfLabels[nextNode] = 0; in this case, let parent[nextNode] = currentNode and add nextNode to stack.

2. back-edge: if (dfLabels[nextNode] < dfLabels[currentNode]) and (nextNode ≠ parents[currentNode])

3. second visit: otherwise.

• Program. Create the function DepthFirstTraverse(int startNode) and show the output for the graph considered in the class with startNode 0 = A and startNode 1 = B. Create your datafile using the format we used for digraph, except that now node 𝑖 will appear in the adjacency list of 𝑗 if 𝑖 appears in the adjacency list of 𝑗; keep the adjacency lists sorted in increasing order. For a graph, inDegree(𝑖) = outDegree(𝑖) = degree(𝑖) for each node 𝑖. The function DepthFirstTraverse should produce one line of output for each link processed, and a separate line from backtracking and every time stack is modified. A possible output may look like:

stack = [0], node 0, dfLabel = 1
link = (0, 1), tree-edge
stack = [0 1], node = 1, dfLabel = 2
link = (1, 0), 2nd-visit
link = (1, 2), tree-edge
stack = [0 1 2], node = 2, dfLabel = 3
link = (2, 0), back-edge
link = (2, 1), 2nd-visit
backtrack from 2 to parent(2) = 1
stack = [0 1]

Mar 11

• 3rd quiz.

• Breadth first traversal of a connected graph

<table>
<thead>
<tr>
<th>Breadth first</th>
<th>Depth first</th>
</tr>
</thead>
<tbody>
<tr>
<td>breadth-fi rst spanning tree (BFT)</td>
<td>depth-fi rst spanning tree (DFT)</td>
</tr>
<tr>
<td>rooted ordered tree</td>
<td>rooted ordered tree</td>
</tr>
<tr>
<td>tree-edges and cross-edges</td>
<td>tree-edges and back-edges</td>
</tr>
<tr>
<td>cross-edges limited to levels differing by ≤ 1</td>
<td>back-edges between levels differing by ≥ 2</td>
</tr>
<tr>
<td>no backtracking</td>
<td>backtracking</td>
</tr>
<tr>
<td>whole tree need to be maintained</td>
<td>backtrack nodes can be deleted from the tree</td>
</tr>
<tr>
<td>BFT tree tends to be &quot;wide&quot;</td>
<td>DFT tends to be &quot;tall&quot;</td>
</tr>
<tr>
<td>each edge visited twice</td>
<td>each edge visited twice</td>
</tr>
<tr>
<td>(O(</td>
<td>E</td>
</tr>
</tbody>
</table>

Mar 16

• Computing all paths in a graph from a start-node (reset dfLabel(𝑥) = 0 when you backtrack from 𝑥 ≠ start-node and reset the nextItemSeenFromAdjListToProcess(𝑥) at the beginning of adjList(𝑥)).

(1) For 𝑥 ≠ start-node, \(\#(\text{occurrences of } 𝑥 \text{ in the new dfTree}) = \#(\text{acyclic paths from start-node to } 𝑥)\).
\[ P = \#(\text{path from } i \text{ to } j \text{ in } K_n) = (n - 2)! \left[ \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \cdots + \frac{1}{(n-2)!} \right] = e(n - 2)! . \]

\[ \#(\text{occurrences of a node } i \text{ in the new dTree(1)}) = P, \text{ except for } i = 1 = \text{root}. \]

\[ \#(\text{tree edges in the new dTree(1)}) = T(n) = (n - 1)P = (n - 1)T(n - 1) + (n - 1), \text{ with } T(1) = 0 \text{ and } T(2) = 1. \]

This gives, \( T(n) = (n - 1)! + n(n - 1)/2 = O((n - 1)! ) \text{ for } n \geq 2. \)

- Check if there is a hamiltonian cycle by depth first search.
- Compute the number of topological sorting.
- Minimum spanning tree by **Prim’s Algorithm**.

**Mar 18**

- Minimum weight spanning tree of a weighted graph.
  - Number of trees on \( n \) nodes is \( n^{n-2} \), too large to create them, find their weights, and choose the minimum.
  - Need a more direct way.
    + Start with a spanning tree and keep modifying it when its weight cannot be reduced any more.
    + Build a spanning tree slowly by adding a edge to an existing tree so that it ends up with a MST.
- The first approach:
  1. Build a spanning tree \( T \) (start at any node and do a depth-first traversal).
  2. Sort the edges in increasing (non-decreasing) link weights: \( e_1, e_2, \ldots, e_m \).
  3. For each edge \( e_1, e_2, \ldots \) do the following:
    a. If \( e_i \) is not in the current spanning tree \( T \) and its weight is the not least weight in the cycle \( C \) in \( T + e_i \), then add \( e_i \) and remove the maximum weight link in \( C \).

**Problem:** Takes too much computation for detecting the cycles for various \( e_i \) (although each time we can detect the cycle in \( T + e_i \)).

- **Homework.** If \( e_i = (x_i, y_i) \) where will you begin depth-first search of \( T + e_i \) to detect the cycle?

- **Pseudocode for second approach: Prim’s Algorithm.**
  1. Choose a start-node \( x_0 \) and let \( T \) consists of just this node.
  2. Repeat the following \( n - 1 \) times:
    a. Add a new node \( x_i \) (\( i = 1, 2, \ldots, n - 1 \)) and connect it to \( T \) via an edge \( (x_i, y_i) \), where \( y_i \in T \) such that this is the least cost edge connecting \( T \) to the outside.

Selecting \( x_i \) and \( (x_i, y_i) \):
  1. For each \( x_i \in T \), find the best link \( (x_i, y_i) \) connecting \( x_i \) to \( T \).
  2. Find the link with minimum weight among all \( (x_i, y_i) \). This gives both \( x_i \) and \( (x_i, y_i) \).

**Mar 23**

- **Homeworks.**
  1. Show in a table form (as indicated below) the steps and the trees in Prim’s Algorithm; here, the second column shows the starting node. Note that once a node is added to \( T \) the column for that node for the remainder of the table will not have any entry (indicated by ‘-‘ below). Use the following input graph.
2. What effects do we have on an MST (minimum weight spanning tree) when we reduce each link-weight by some constant \( c \) (which might make some link-weights < 0)?

**Program:**

1. Write a function PrimMinimumSpanningTree(startNode) to construct an MST for a weighted graph.

   The output should show the following, with \( \#(\text{output lines}) = \#(\text{nodes in the connected input graph}) \).

   (a) The start-node.

   (b) For each successive line, a list of the triplets of the form \((x_i, y_i, w(x_i, y_i))\) for each node \( x_i \) not in the current tree \( T \), where \((x_i, y_i)\) is the current best link connecting \( x_i \) to \( T \).

   Follow this by the node selected for adding to \( T \).

   **Pseudocode** for processing the links from the node \( x \) added to \( T \):

   1. For each \( y \) in adjList\((x)\) do the following:

      (a) If \( y \) is not in \( T \), then update \( \text{bestLinkFrom}(y) = x \) if \( w(y, \text{bestLinkFrom}(y)) > w(y, x) \).

   **Notes:**

   (a) Use an array \( \text{bestLinkFrom}[0..(n-1)] \), where \( n = \#(\text{nodes}) \), and initialize each \( \text{bestLinkFrom}[i] = -1 \) to indicate that the best link is not known. For the start-node, let \( \text{bestLinkFrom}[\text{startNode}] = \text{startNode} \).

      This is the array that is returned by the function.

   (b) Use another array \( \text{inTree}[0..(n-1)] \), with \( \text{inTree}[i] = 1 \) meaning that \( i \) is in \( T \) and 0 otherwise.

   (c) The input-file graph.dat now should give the link weights as indicated below, where each item in the adjacency-list is followed by the link-weight in parentheses.

\[
0 \ (3): \ 1(1) \ 2(4) \ 4(1) \ /\text{for node } A = 0 \text{ in the graph shown above}
\]
• Questions on Prim’s Algorithm:
  − When do we process a link \((x, y)\)?
  − What does the processing of \((x, y)\) involve?
  − What is the complexity of processing \((x, y)\)?
  − What is the complexity of Prim’s Algorithm?
  − What is the main data structures needed for implementing Prim’s Algorithm?

• Shortest paths in a weighted digraph, with \(w(x, y) \geq 0\) for Dijkstra’s Algorithm.

**Apr 01**

• Longest path in a acyclic weighted digraph (weights can be –ve):
  − Comparison with Dijkstra’s shortest-path algo.
    + Unlike Dijkstra’s algo, we need to look at all incoming links to \(y\) before we can find a longest-path to \(y\).
    + It process a link \((x, y)\) only after it finds a longest path to \(x\)
    + Subpath of a longest-path is also a longest-path between its end points.
  − It has complexity \(O(|E|)\), similar to topological sorting Algorithm.
  − It is in many ways similar (with some variation) to topological sorting.
• Pseudocode for longestPath(startNode).

It use following array data-structures:

- $d(x)$ = current longest path to $x$ from startNode
- $\text{parent}(x)$ = the node previous to $x$ on the current longest path to $x$; $\text{parent}(\text{startNode}) = \text{startNode}$
- indegree($x$) = number of links to $x$ not yet looked at; it changes during the Algorithm

1. Preprocess the input digraph to make the startNode the only source-node:
   (a) Compute indegree($x$) for each node $x$.
   (b) Initialize a stack with all source-nodes, if any, which are different from startNode (which may or may not be a source-node).
   (c) While (stack $\neq$ empty) do the following:
      (i) Let $x$ = top(stack); remove $x$ from stack.
      (ii) For (each $y \in \text{adjList}(x)$) reduce indegree($y$) by 1 and if it equals 0 then add $y$ to stack.

2. Initialize a stack with startNode, let $d(x) = -\infty$ and $\text{parent}(x) = -1$ for each node $x$ with indegree($x$) $> 0$, and finally let $d(\text{startNode}) = 0$ and $\text{parent}(\text{startNode}) = \text{startNode}$. (You can take $-\infty$ to be a number which is minus of the sum of absolute values of all link-costs.)

3. While (stack $\neq$ empty) do the following:
   (a) Let $x$ = top(stack); remove $x$ from stack.
   (b) For (each $y \in \text{adjList}(x)$) do:
      (i) If ($d(x) + w(x, y) > d(y)$), then let $d(y) = d(x) + w(x, y)$ and parent($y$) = $x$.
      (ii) Reduce indegree($y$) by 1 and if it equals 0 then add $y$ to stack and also print the longest-path to $y$ from startNode using the successive parent-links and print the cost of this path.

• Program. Develop a function LongestPath(int startNode) and test it with the digraph below. Show the output in a reasonable form (you have seen enough examples of proper outputs) for startNode = A. In particular, every time $d(y)$ for some node $y$ is updated, print a separate line of the form "$d(3) = 2$, parent(2) = 0" to show the new $d(y)$ and its parent. (You can start with your topological sorting program and modify it appropriately.)

![](image)

• Homework. Show the details (in the table form) the computations in Prim’s Algorithm to construct an MST for the graph on the nodes shown below (given next to each node $v_i$ are its $x$ and $y$ coordinates in the plane), where the link $(v_i, v_j)$ has cost equal to the Euclidean distance between $v_i$ and $v_j$. Assume the start-node is $v_1$. (Most of you did not do this problem right in the Quiz.)
Find a suitable acyclic weighted digraph so that if we compute the longest between some pairs of nodes of this digraph then we will get the longest increasing subsequence (LIS) for the input sequence <4, 1, 3, 8, 5, 7, 13, 6>. Your method for constructing the digraph must be general enough that it will can be used for any input sequence for finding an LIS. Show your digraph, the longest path in your digraph, and the associated longest increasing subsequence.

Apr 15

Huffman tree/Huffman code: assigning prex-free codes to a set of symbols with given probabilities.

- Alphabet \( \Sigma \) = a non-empty finite set of symbols; word is a finite non-empty string of symbols in \( \Sigma \).
- Code(\( x \)) = code of symbol \( x \in \Sigma \) = a binary string; code(\( x_1 x_2 \cdots x_n \)) = code(\( x_1 \)).code(\( x_2 \))\( \cdots \).code(\( x_n \)).
- Example. Let \( \Sigma = \{ A, B, C, D, E \} \).

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>Pref x-property</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>001</td>
<td>010</td>
<td>011</td>
<td>100</td>
<td>code(AAB) = 000000001; easy to decode</td>
</tr>
<tr>
<td>0</td>
<td>01</td>
<td>001</td>
<td>0001</td>
<td>0001</td>
<td>code(C) = code(AB) = 001; not always possible to uniquely decode</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>001</td>
<td>0001</td>
<td>0001</td>
<td>yes</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>100</td>
<td>1000</td>
<td>10000</td>
<td>no</td>
</tr>
</tbody>
</table>

- Some requirements:

1. Each binary string has at most one possible decoding.
2. It should be possible to do the decoding from the left, i.e. as the symbols are received.

- A sufficient condition for both (1)-(2) the that the codes satisfy prefix property:

  No code(\( x \)) is the prefix of another code(\( y \)) for \( x \) and \( y \in \Sigma \).
  In particular, code(\( x \)) \( \neq \) code(\( y \)).

- A code with prefix x-property can be represented as the terminal nodes of a binary tree with 0 = label(left branch) and 1 = label(right branch).
**Homework.** Consider the codes shown below.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>000</td>
<td>001</td>
<td>011</td>
<td>10</td>
<td>110</td>
</tr>
</tbody>
</table>

(a) Arrange the codes in a binary tree form, with 0 = label(leftbranch) and 1 = label(rightbranch).

(b) Is it true that the codes has the prefix-property? How do you decode the string 10110001000?

(c) Modify the above code (keeping the prefix property) so that the new code will have less average length no matter what the probabilities of the symbols are. Show the binary tree for the new code.

(d) What are the two key properties of the new binary tree (hint: compare with your answer for part (a))?

(e) Give a suitable probability for the symbols such that prob(A) < prob(B) < prob(C) < prob(D) < prob(E) and the new code in part (c) is optimal (minimum average length) for those probabilities.

**Apr 20**

- Floyd’s Algorithm for shortest-path computation for all \((x_i, x_j)\) node pairs.
  - The digraph may have -ve link costs; in that case, Dijkstra’s Algorithm cannot be used.
    - If there is a cycle with -ve cost, then shortest-paths between nodes in the cycle are not defined.
  - Total complexity is \(O(N^3)\) for all node-pairs, which is comparable to \(O(N^2)\) for shortest-path from a fixed start-point to all other nodes in Dijkstra’s Algorithm.
  - Number of path-lengths computed = \(O(N^3)\), one corresponding to the computation of \(F^k(i, j)\) for each \(1 \leq i, j \leq N\) and \(0 \leq k \leq N\).
    - Per node pair \((i, j)\), we compute \(O(N) = N + 1\) path lengths including the path \(\langle x_i, x_j \rangle\).
    - This means most of the loop-free \(e(N - 2)\) \(x_i x_j\)-paths are not looked at.

- \(F^k(i, j)\) = the shortest \(x_i x_j\)-path length where only intermediate nodes are \(\{x_1, x_2, \ldots, x_k\}\).
  - \(F^0(i, j) = c(x_i, x_j)\)
  - \(F^k(i, j) = \min \{F^{k-1}(i, j), F^{k-1}(i, k) + F^{k-1}(k, j)\}\)
  - \(F^N(i, j)\) = the final shortest \(x_i x_j\)-path length.
• How will you create a sorted list of the key in a 2-3 tree? Preorder traversal where at a node with one label you do

\[ \text{list-left-subtree, list-node-label, list-right-subtree} \]

and for a node with two labels do

\[ \text{list-left-subtree, list-first-node-label, list-middle-tree, list-second-node-label, list-right-subtree} \]

• What is the connection between variance and the sum \((a_i - a_j)^2\), summed over all \(1 \leq i, j \leq n\) for a given collection of numbers \(a_i\)?

• Find the next binary string of a given length \(n\).

• **Homework** Find the smallest pair of numbers from \(\text{nums}[1..n]\) whose average is closest to 0.

• **Homework** Find three numbers from \(\text{nums}[1..n]\) whose standard deviation is minimum.

• Syntactic and semantic organization of data and operations.

- Lists and arrays are of homogeneous data-units, where that data-unit can be any thing (homogeneous or not).

  This covers the case of lists of pointers to different classes in a common hierarchy in C++ because all those pointers are in a sense considered of the same type, namely, a pointer for the top record in the hierarchy.
• What does the following equal to
  \[ 247801 \times 7125 - 247801 \times 7025 \]
• How do you represent an arithmetic expression like \( a - b \times 3 \) and \((a - b) \times 3\), how do you build the tree, and how do you systematically simplify (bottom-up) it for given values of the variables \( a \) and \( b \)?

• What do you call a tree of the type shown below?

• Why do we call it binary? What is a non-binary tree – have we seen any yet? Why do we call it a search-tree?
• So how would you define a binary search tree?
• What is the main use of such a tree?
• Can you label the nodes of the binary tree below with the numbers 1, 2, ..., 8 to make it a binary search-tree? Is the labeling unique?
• Show two different inputs that can give rise to this tree? How many inputs are there?

• What are the most basic elements that we compute?
  numbers, strings, images (colors and positions of dots), other displays (strings and images).

  Each of them may have different meanings; number = age, weight, salary, temperature, height of a binary tree, length of a string.

• What is an Algorithm?
  A finite sequence of basic computation-steps and three other operations:
    inputs, outputs, and control-flow.

• What are the steps in computing the average of three input numbers \(a, b,\) and \(c\).

• Are there different ways (Algorithms, methods) of the computing average?

• In how many ways can one method be better than the other?
  time-wise, memory-wise, simplicity-wise.

• Algorithm Design: organizing computations for maximum efficiency and the best solution.

• In-Class: Give an Algorithm for new International Students to go to Allen Hall from Student Union.

• Since computation needs data, organization of data for efficient access becomes important.

• Consider a program \(P\) using the data-organization on the left below. If we replace the data-organization by the one on the right, do we have to make any change in \(P\)? Is there then any reason to prefer one to the other? (Yes, the left one takes \(4 + 3 \times 8 = 28\) and right one takes \(3 \times (4+8) = 36\) Why?

  typedef struct {
    char grade, grade2, grade3;
    double score, score2, score3;
  } First;

  typedef struct {
    char grade;
    double score;
    char grade2;
    double score2;
    char grade3;
    double score3;
  } Second;

• How many different structure definitions are there involving three chars and three doubles that would give different memory mappings? How many of them give total size 36 bytes (note that every structure address begins at a multiple of 4 bytes and is of size a multiple of 4 bytes)?

• This course will emphasize data-structure concepts and their applications in efficient program development.
  – Data Structure for better efficiency (linked lists of different kinds, trees) and better organization of data for visibility and naming (struct-construction).
  – Want clear program, with pseudocodes; main-functions is to primarily call other functions and set values of global variables.
  – Use for-loop when the control variable is updated in a regular fashion.

• Write the code for \(\text{firstPositiveItemIndex}(\text{int } *\text{items, int numItems});\) if there are no positive items then it returns -1.

  1. look at items[0], items[1], ... and stop as soon as a positive item is found.
  2. if found then return index of the item else return -1.

    for (i=0; i<\text{numItems}; i++)
      if (items[i] > 0) break
    if (i < \text{numItems}) return(i);
else return(-1);

What is an alternate way of writing the if-then-else statement? (replace "break" by "return(i)"

- Modify it so that each call will find the successive positive item’s index, and call the new function nextPositiveItemIndex; if we call it after it returns -1, then it should again restart the cycle by finding the first positive item’s index. Note that if there is any change in items or numItems, then the search will start with items[0]. Should we find all the positive items and save it in a separate array?

  - The complexity of computing partial sums of items[.] and items[.][.]

- Measuring efficiency via instrumentation of InsertionSort.

  - Need to generate random permutation or all permutations. How to do it?
    1. Find the term to be increased, find the new value, and adjust values to its right.
    2. Repeat the above till the sequence is \( <n, n-1, \ldots, 3, 2, 1> \)

  - Measure average number of comparisons and data-movements

- Finding a subset of \( m \leq n \) items from a list of \( n \) (distinct) items which are most closely packed, i.e., have smallest variance.

Jan 14

- Acyclic digraphs, source-nodes, sink-nodes, and topological sorting, pseudocode.

**Homeworks:** how many ways can you top-sort; tree of all possibilities (not a binary tree); draw the tree with all terminal nodes placed on a line with equal spacing between them.

  - each node of the tree shows the nodes that can be laid off (including the the most recent child to be created).
  - each link of the tree shows what is being laid off.

![Diagram of a tree](image)

- Input file design.

- Program: Write a program to obtain topological sort.

Jan 19

- Comparison of tree and digraph (digraph instead of graph because direction of links being a common feature between them).

<table>
<thead>
<tr>
<th>Rooted Tree ( T )</th>
<th>Digraph ( G )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Made of nodes and directed links</td>
<td>Made of nodes and directed links</td>
</tr>
<tr>
<td>2. For ( n ) nodes, #(links) = ( n - 1 )</td>
<td>For ( n ) nodes, ( 0 \leq #(links) \leq n(n-1) )</td>
</tr>
<tr>
<td>3. Children ( C(x) ) of node ( x )</td>
<td>Nodes ( N^+(x) ) that are adjacent from ( x )</td>
</tr>
<tr>
<td>( C(x) \cap C(y) = \emptyset ) for ( x \neq y )</td>
<td>( \text{this need not hold} )</td>
</tr>
<tr>
<td>Terminal node ( x ) has ( C(x) = \emptyset )</td>
<td>( \text{Sink node } x \text{ has } N^+(x) = \emptyset )</td>
</tr>
<tr>
<td>4. Unique parent ( \text{par}(x) ), except for root</td>
<td>(</td>
</tr>
<tr>
<td>( \text{Root-node } x \text{ has no-parent} )</td>
<td>( \text{Source nodes } x \text{ has } N^-(x) = \emptyset )</td>
</tr>
<tr>
<td>5. Has no cycle</td>
<td>For acyclic digraph, #(links) \leq n(n-1)/2</td>
</tr>
<tr>
<td>( \text{Unique path from root to all nodes} )</td>
<td>( #(\text{paths between two nodes}) \leq e(n-2)! ) for acyclic digraphs</td>
</tr>
</tbody>
</table>
– Minimum connectivity from root to all nodes

Subtree $T(x)$ at a node $x$

Subdigraph $G(x)$ of nodes reachable from $x$

7. $S(x) = \{x\}$, strong component of $x$

Strong component $S(x)$ of $x$ can be as large as $G$

– Merging each $S(x)$ into a node gives an acyclic digraph

8. Already transitively reduced

Need not be already transitively reduced.

Jan 21

Jan 26

• **Iterative solution:** When the solution has many parts, and we compute each part in the same way on a slightly different part of the original input-data, part of which might be modified in the computation of previous parts.

  – Sorting by iteration:
    1. Find $i$th smallest items among $S\setminus\{1st, 2nd, \ldots, (i-1)th$ smallest element\}
    2. Repeat (1) for $n-1$ times, where $|S| = n$.

  **Bubble-sort** is an iterative method, which finds successive largest number, where on completion of the $i$th iteration, more than $i$ items might have properly placed.

  It is a refined implementation of the above pseudocode in some sense, but it may perform too many exchanges for some inputs.

  **Insertion-sort** can be thought of as an iterative (but more appropriately as a recursion) based on the size of the input-data:

  1. Successively sort first $i$ items, $1 \leq i \leq n$.

  Iterative-approximation is a technique common numerical analysis (such as finding roots), where iterations are performed until some error limit is obtained.

• **Recursion** is different in that the computation of $i$th call may not be over before starting the $(i+1)$th call, and each call might compute more than one part of the final solution.

• In depth-first, shortest-path, and longest-path, the basic unit of processing is a link $(x, y)$.

  **Depth-first**:
  $(x, y)$ is process after processing all $(x, y')$ where $y' < y$ in adj-list($x$).

  **Shortest-path**:
  Same as above, with the additional restriction that process all links at $x$ before processing links at another $x$.

  **Longest-path**:
  Same as above, but the selection of successive $x$ is different.

• Consider static and dynamic features for comparing Algorithms, unlike comparing concepts (using only static features).

  **Static features:**
  1. Concepts used, basic computations performed in different iterations (recursions).
  2. Conditions for selecting a unit input element for processing
  3. Complexity
  4. Structure of outputs produced: tree, lists, paths, etc.
  5. Structure of and constraints on input (Floyd vs. Dijkstra).
  6. Presence of pre-processing (simplifying input to a standard form, as in longest path)

  **Dynamic features:**
  1. Iterative vs. recursive,
  2. In which order, certain elements are processed.
  3. Finite-state model and their comparisons

• Computing Science is part of Computer Sc, the latter could include both software and hardware. Data-structure is part of Algorithms, which is part of Software and the latter includes also programming skills.
Each student introduces him/her-self by stating the name, year, major, where are you from?

**In-Class:** Describe in \((\leq 10)\) lines a program that you had written and are proud (were excited) about it.

- Did you state what the input is? How about the output?
- A name for your program? How long is the program?
- What language was used?

**Homework:** Give a short description (< 5 lines) of a programming problem that you would like to be able to solve by the end of this semester? Maybe you have seen something in action and you wondered how to do that sort of things?
ANOTHER EXAMPLE OF PSEUDOCODE

Problem. Compute the size of the largest block of non-zero items in \( \text{nums}[0..n - 1] \).

Example. The underlined part is the largest block.

\[
[2, 0, -1, 3, 1, 0, 0, 5].
\]

Pseudocode:

1. Initialize maxNonZeroBlockSize = 0.
2. while (there are more array-items to look at) do:
   (a) skip zero’s. //keep this
   (b) find the size of next non-zero block and update maxNonZeroBlockSize.

Code:

```c
i = maxNonZeroBlockSize = 0;
while (i < n) {
    for (; (i<n) && (nums[i]==0); i++); //skip 0’s
    for (blockStart=i; (i<n) && (nums[i]!=0); i++);
    if (i - blockStart > maxNonZeroBlockSize)
        maxNonZeroBlockSize = i - blockStart;
}
```

Question:

•? If there are \( m \) non-zero blocks, then what is the maximum and minimum number of tests involving the items nums[i]?

•? Rewrite the code to reduce the number of such comparisons. How much reduction is achieved?

•? Generalize the code and the pseudocode to compute the largest size same-sign block of items.
A GEOMETRIC COMPUTATION PROBLEM

Problem: If $C_1$ and $C_2$ are two circles of radii $r_1$ and $r_2$, then when can we place $C_1$ inside $C_2$?

If $C_1$ can be placed inside $C_2$, then can we place it so that the centers of $C_1$ and $C_2$ coincide?

Question:

• If $S_1$ and $S_2$ are two squares with sides of length $r_1$ and $r_2$, then when can we place $S_1$ inside $S_2$?

If $S_1$ can be placed inside $S_2$, then can we place it so that the centers of $S_1$ and $S_2$ coincide?

• If we have a square and a circle, then when can we place one inside the other? (Can we make their centers coincide in that case?)
PLACING ONE RECTANGLE INSIDE ANOTHER

- Let $R_1 = (W_1, H_1)$ and $R_2 = (W_2, H_2)$ be two rectangles, where $W_i = \text{width}(R_i) \geq \text{height}(R_i) = H_i$. When can we place $R_1$ inside $R_2$, and if so then how can we find an actual placement?

\[
R_1(1.4, 0.7) \quad R_2(1.6, 1.0) \quad R_3(2.0, 0.3)
\]

(i) Two of the infinitely many ways of placing $R_1$ inside $R_2$.
(ii) $R_3$ cannot be placed inside $R_2$.

Question:
1? What is an application of the rectangle-placement problem?
2? What is a necessary condition for placing $R_1$ inside $R_2$?
3? What is a sufficient condition for placing $R_1$ inside $R_2$?
4? Do these conditions lead to a placement-Algorithm (how)?

Generalization of Rectangle-Placement Problem:
- Find a placement that maximizes $R_1 \cap R_2$.

Placing a triangle $\Delta_1$ inside another triangle $\Delta_2$:
- Triangles are more complex objects than rectangles (why?). This makes the triangle-placement problem more difficult.
- What are some special classes of triangles for which the placement problem is easy? Find the placement condition and a particular way of placing.
NECESSARY vs. SUFFICIENT CONDITION

- If a property $P$ implies a property $Q$, then
  - $Q$ is a necessary condition for $P$, and
  - $P$ is a sufficient condition for $Q$.

Example. Let $P =$ "The integer $n$ is divisible by 4".
- Consider the two conditions below, where $n_1 n_2 \cdots n_k = n$:
  
  $Q_1$: "The last digit $n_k$ of $n$ is 0, 2, 4, 6, or 8".
  $Q_2$: "The integer $n' = n_{k-1} n_k$ comprising the last two digits of $n$ is divisible by 4". (Thus, $n' = n$ if $n < 100$.)

- Clearly, $P$ implies $Q_1$ and $P$ implies $Q_2$; so, each of $Q_1$ and $Q_2$ is a necessary condition for $P$.

- However, only $Q_2$ implies $P$; $Q_1$ does not imply $P$ (for example, let $n = 6 = n_k$, which makes $Q_1$ true and $P$ false).
  Thus, only $Q_2$ is a sufficient condition for $P$.

If $Q$ is both necessary and sufficient for $P$, then $P$ is both necessary and sufficient for $Q$.
($P$ and $Q$ are equivalent.)

Question: Are $Q_1$ and $Q_2$ above equivalent?
AN EXTREME CASE OF RECTANGLE PLACEMENT PROBLEM

For the case on right, the dashed rectangle $R_1$ can be slightly rotated and still kept inside the solid rectangle $R_2$.

**Question:**

1. Which of the dashed rectangles has the larger area? Can one of them be placed inside the other? Justify your answer.

2. Derive the necessary and sufficient condition for placing $R_1$ inside $R_2$ for the following cases:
   (a) $R_1$ can be placed inside $R_2$ without tilting.
   (b) $R_1$ must be tilted to place inside $R_2$.
   (c) $R_1$ can be placed inside $R_2$ in essentially only one way as in the lefthand case in the figure (a special case of (a)-(b)).

3. If $R_1$ can be placed inside $R_2$, is it true that we can make the placement so that their centers coincide? Explain your answer.
HINT FOR SOLVING THE CASE (c)

For the case on the right, the dashed rectangle $R_1$ can be slightly rotated and still kept inside the solid rectangle $R_2$.

From similarity of triangles, we get
\[
\frac{x}{H_1} = \frac{H_2 - y}{W_1} \quad \text{and} \quad \frac{y}{H_1} = \frac{W_2 - x}{W_1}.
\]

By comparing the length of the diagonals, we get
\[
W_1^2 + H_1^2 \leq W_2^2 + H_2^2.
\]

We also have $H_1^2 = x^2 + y^2$.

EXERCISE
1. Show that the largest square inside $R_2(W, H)$ is $R_1(H, H)$.
2. If we know that $D_1 = D_2$, where $D_i$ is the length of the diagonal of $R_i$, then what is a necessary and sufficient condition that $R_1$ can be placed inside $R_2$.
3. Give an example of $R_1$ and $R_2$ such that $D_1 < D_2$ and still $R_1$ cannot be placed inside $R_2$. 
A STRING PROBLEM

Substring: Given a string $x = a_1a_2\cdots a_n$, each $x' = a_{i_1}a_{i_2}\cdots a_{i_k}$, where $i_1 < i_2 < \cdots < i_k$, is a $k$-substring of $x$.

For $x = abbacd$, $x' = bcd$ is a 3-substring but $x' = dc$ is not a 2-substring.

Question:

•? How many ways can we form $k$-substrings of $a_1a_2\cdots a_n$? When does all $k$-substrings ($0 < k < n$) become the same?
•? When do we get the maximum number of distinct substrings?

Projection: If we keep all occurrences of some $k$-subset of the symbols in $x$ (in the order they appear in $x$), then the resulting substring is a $k$-projection of $x$.

Example. For $x = aabcacbbadd$, which is made of four symbols \{a, b, c, d\}, we get $6 = C(4, 2)$ many 2-projections as shown below. Note that $x_{ab} = x_{ba}$, $x_{ac} = x_{ca}$, etc.

$$
x_{ab} = aababba, \quad x_{bc} = bccbb,
$$
$$
x_{ac} = aacaca, \quad x_{bd} = bbadd,
$$
$$
x_{ad} = aaaaadd, \quad x_{cd} = ccdd.
$$

Question:

•? Give the string $y$ made of the symbols \{b, c, d\} which has the same 2-projection as $x$ above, i.e., $y_{bc} = x_{bc}$, $y_{bd} = x_{bd}$, and $y_{cd} = x_{cd}$.

•? Give an Algorithm to determine the string $x$ from its 2-projections. Explain the Algorithm using $x = aabcacbbadd$. 

GENERATING \((n, m)\)-BINARY STRINGS

**Problem:** Generate all \((n, m)\)-binary strings, with \(n - m\) zeros and \(m\) ones. There are six \((4, 2)\)-binary strings:

<table>
<thead>
<tr>
<th>Binary strings</th>
<th>0011</th>
<th>0101</th>
<th>0110</th>
<th>1001</th>
<th>1010</th>
<th>1100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Associated integers</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>9</td>
<td>10</td>
<td>12</td>
</tr>
</tbody>
</table>

**An Algorithm**  
AllBinaryStrings\((n, m)\): //\(n=\)length, \(m=\)numOnes  
1. For \((i = 0, 1, 2, \ldots, 2^n - 1)\) do the following:  
   (a) Convert \(i\) to its binary-string form \(s(i)\) of length \(n\).  
   (b) Print \(s(i)\) if it has exactly \(m\) ones.

**Problems with** AllBinaryStrings\((n, m)\):

- It is very inefficient when \(m = n/2\). For \(n = 4\) and \(m = 2\), it generates 16 strings and throws away 16-6 = 10 of them.
- It does not work for \(n > 32\) (= word-size in most computers).

**Question:**

1? What are some difficulties with the following approach \((0 < m < n)\) and how can you get around them:

Start with the string \(1^m\), then add one 0 in all possible ways, then for each of those strings add one 0 in all possible ways, and so on until each string has \(n - m\) zeroes. until all zero’s are added (e.g., \(11 \rightarrow \{011, 101, 110\}\) ).
NEXT \((n, m)\)-BINARY-STRING GENERATION

Examples of Successive \((10, 5)\)-Binary Strings:

A \((10, 5)\)-binary string: \(0100111100\)
Next \((10, 5)\)-binary string: \(0101000111\)
Next \((10, 5)\)-binary string: \(0101001011\)
Next \((10, 5)\)-binary string: \(0101001101\)
... ...  
The last \((10, 5)\)-binary string: \(1111100000\)

A necessary-and-sufficient condition for string \(y = \text{next}(x)\):
(1) The rightmost "01" in \(x\) is changed to "10" in \(y\).
(2) All 1’s to the right of that "01" in \(x\) are moved to the extreme right in \(y\).

Algorithm for Generating next\((x)\) from \(x\):
(1) Locate the rightmost "01" in \(x\) and change it to "10".
(2) Move all 1’s to the right of that "10" to the extreme right.

Moving 1’s To Right: \(\cdots 01\overline{11111000} \rightarrow \cdots \overline{1000011111}\)
• numOnesToMove = \(\min\)\(\text{numEndingZeros}, \text{NumPrevOnes} - 1\)

Questions:
1? What happens when there is no "01"?
2? How will you generate a random \((n, m)\)-binary string, i.e., with what probabilities will you successively determine the bits \(x_i\) of a random binary-string \(x_1x_2\cdots x_n\)? Give the probabilities for successive bits in 01101 (\(n = 5\) and \(m = 3\)).
FINDING THE RIGHTMOST "01" IN A BINARY STRING

Pseudocode:
1. Scan the binary string from right-to-left to find the rightmost '1'.
2. Continue right-to-left scan till you find the first '0'.

Question:
1? Why is right-to-left scan is better than left-to-right scan to locate the rightmost "01" (for our application)?
2? Does the following code find the rightmost "01"?

```plaintext
for (i=length-1; i>=1; i--)
    if ((1 == binString[i]) &&
        (0 == binString[i-1]))
        break;
```

Explain with an example binary string how the above code wastes unnecessary comparisons of the items in binString[]. Describe the situation that makes the performance of the second code worst.

3? Give a piece of code corresponding to the pseudocode above and which does not have the inefficiencies of the code above.
PROGRAMMING EXERCISE

1. Write a function `nextBinString(int length, int numOnes)` that can be called again and again to create all binary strings in the lexicographic order with the given length and number of ones. Choose a suitable return value to indicate when the last binary string is created. Use an array `binString` for the binary-string, and use dynamic memory allocation.

Your main-function should call `nextBinString`-function again and again. It should run for large values of `length` (= 100, say) and all `0 ≤ numOnes ≤ length`.

First, test your program for `length = 6` and `numOnes = 2` and `3`.

Now modify `nextBinString`-function to count #(reads) from and #(writes) into the `binString`-array as you generate each binary string. Call these counts `numReads` and `numWrites`. The output should look like the following; show the average `numReads` and average `numWrites` upto 2 digits after the decimal point.

<table>
<thead>
<tr>
<th>binString</th>
<th>numReads</th>
<th>numWrites</th>
</tr>
</thead>
<tbody>
<tr>
<td>000111</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>001011</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111000</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>averNumReads</th>
<th>averNumWrites</th>
</tr>
</thead>
<tbody>
<tr>
<td>= ...</td>
<td>= ...</td>
</tr>
</tbody>
</table>

Submit the paper copy of your code and the outputs for `length = 6` and `numOnes = 2` and `3`..
A RECURSIVE APPROACH FOR
GENERATING ALL \((n, m)\)-BINARY STRINGS

Pseudocode for \(\text{RecAllBinStrings}(n, m)\):
1. If top-level call, then create the array \(\text{binString}[0..n−1]\) and let \(\text{strLength} = n\).
2. If \((n = m)\) or \((m = 0)\), then fill the last \(n\) positions in \(\text{binString}\) with 1’s or 0’s resp., print \(\text{binString}\), and return;
   otherwise, do the following:
   (a) Let \(\text{binString}[\text{strLength}−n] = '0'\) and call \(\text{RecAllBinStrings}(n−1, m)\).
   (b) Let \(\text{binString}[\text{strLength}−n] = '1'\) and call \(\text{RecAllBinStrings}(n−1, m−1)\).

Question:
1? Let \(W(n, m) = \#(\text{total write-operations into } \text{binString}[])\) for generating all \((n, m)\)-binary strings. Give the equation connecting \(W(n, m), W(n−1, m), \) and \(W(n−1, m−1)\). Show \(W(n, m)\) for \(1 \leq n \leq 6\) and \(0 \leq m \leq n\) in Pascal-triangle form.