ALGORITHM DESIGN AND ANALYSIS

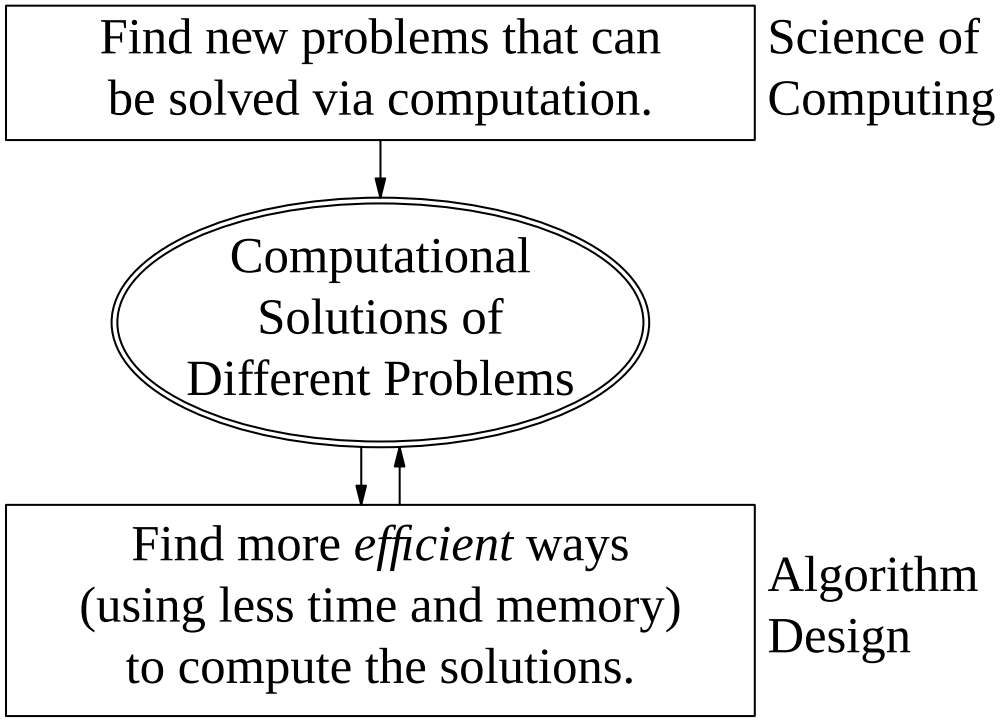
Dr. Sukhamay Kundu

Computer Science Dept, Louisiana state University
Baton Rouge, LA 70803

kundu@csc.lsu.edu

Spring, 2008

ALGORITHM DESIGN: A PART OF "SCIENCE OF COMPUTING"

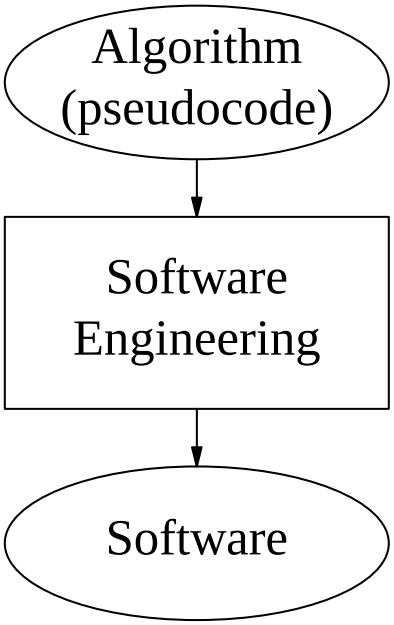


- For very difficult problems, we use *heuristic* algorithms to get an approximate solution and save the computation time.
- Finding better heuristics is also a part of Algorithm Design.

Examples of Computational Solutions:

- Text-formating with tables, equations, diagrams, automatic numbering of figures and sections, etc.
- Medical image analysis for detection of bone-fractures, cancerous growths, bone-loss, etc.
- Design of smart molecules with specific biological or electro-mechanical properties.
- Simulation of engineering designs, environmental impacts of earthquakes and floods, trade policies, etc.

ALGORITHM
vs. SOFTWARE ENGINEERING



Algorithm design (including the related data-structures)

Algorithm implementation, with proper choice of user-interface fi le structures, variables, and functions (and their parameters)

An end-product of Computer Science

Every successful software has in its core an *efficient* algorithm that solves an *important* (useful) problem.

- Focus of Algorithm Design: algorithm's correctness and effi - ciency.
  - An incorrect algorithm is not an algorithm. (We do not consider heuristic algorithms in this course.)
- Focus of Software Engineering: quality of code, user-inter- face, extendibility, etc.
  - An incorrect implementation of an algorithm is not worth much.

FIVE ALGORITHM DESIGN PROBLEMS

Designing effi cient algorithms involve identifying relevant concepts and exploiting their properties for all entities involved.

Two Not-So-Difficult Problems:

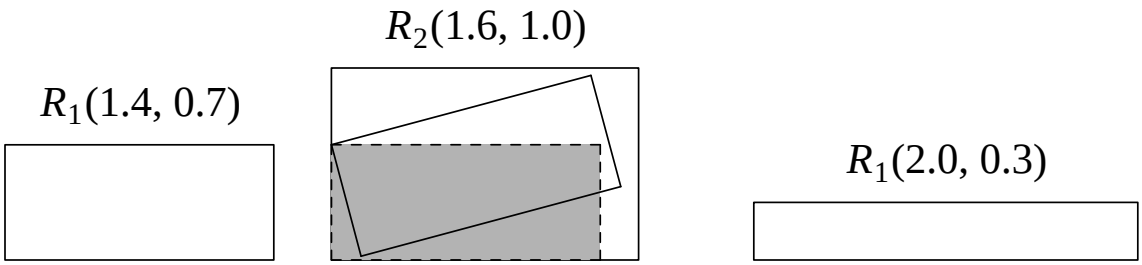
- A geometric problem in Engineering.
  - Look at relevant properties of the geometric objects.
- A problem involving strings (which can represent many different kinds of things).
  - Look at relevant properties of the set of strings.

Three More-Difficult Problems:

- A geometric problem in drawing a program-flowchart.
  - Look at relevant properties for certain substructures in a flowchart and their positions in the drawing. (Solution via depth fi rst processing.)
- A geometric problem in Wireless Communication Networks.
  - Look at relevant connectedness property of graphs. (Solution via minimum spanning tree.)
- A problem of grouping numerical scores into classes.
  - Look at relevant properties of minimum-error grouping. (Solution via shortest-paths.)

A SIMPLE GEOMETRIC COMPUTATION PROBLEM

- Let $R_1 = (W_1, H_1)$ and $R_2 = (W_2, H_2)$ be two rectangles, where W_i = width(R_i) ≥ height(R_i) = H_i. Can R_1 be placed inside R_2, and if so then can we find such a placement?



(i) Two of the infinitely many ways of placing R_1 inside R_2.

(ii) This R_1 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 covers the maximum area of R_1, i.e., maximizes $R_1 \cap R_2$.

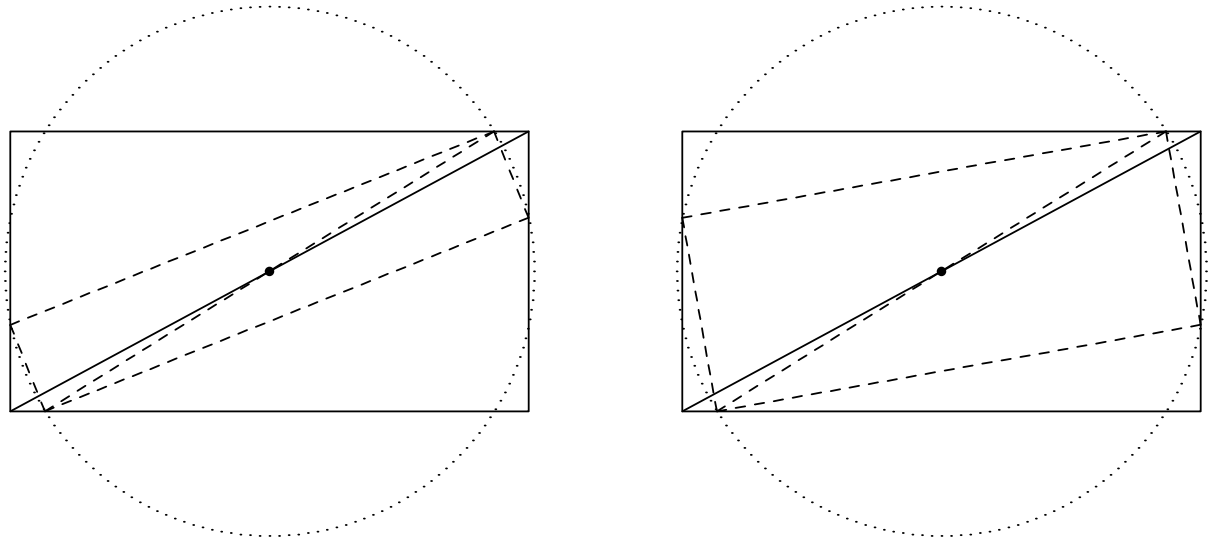
5? Give another generalization of rectangle-placement problem.

Placing a triangle Δ_1 inside another triangle Δ_2:

- Triangles are more complex objects than rectangles (why?). This makes the triangle-placement problem more difficult.

6? Name a simpler geometric object than a rectangle for which the placement problem is simpler (explain).

SOME EXTREME CASES OF THE RECTANGLE PLACEMENT PROBLEM



For the case on left, R_1 can be placed inside R_2 basically in just one way.

Question:

- In which of the two figures the dashed rectangle (R_1) can be slightly rotated and still keep it inside the solid rectangle (R_2)?
- Which of the dashed rectangles has the larger area? Can one of them be placed inside the other? Justify your answer.
- Consider the following cases in determining the necessary and sufficient condition for placing R_1 inside R_2, and in each case derive the required conditions:
  - R_1 can be placed inside R_2 with its sides parallel to those of R_2.
  - R_1 can be placed inside R_2 in essentially only one way.
  - R_1 must be tilted to place inside R_2.

A BINARY-STRING GENERATION PROBLEM

Problem: Generate all (n, m)-binary strings, with $n - m$ zeros and m ones. There are six $(4, 2)$-binary strings:

Binary strings:	0011	0101	0110	1001	1010	1100
Associated integers:	3	5	6	9	10	12

Algorithm AllBinaryStrings(n, m): //n=length, m = numOnes

1. For (i = 0, 1, 2, ···, $2^n - 1$) 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 even 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).

1. Always look at some example situations to get a clear understanding of the algorithm-design problem.
2. Test your algorithm's correctness/efficiency using them.

Question:

1? How can we use (n, m)-binary strings to find all increasing subsequences of size, say, $m = 3$ in $\langle 4, 5, 2, 9, 7, 6, 8, 1\rangle$?

2? How to generate (n, m)-binary strings using (a) or (b): (a) Starting with $0^{n-m}1^m$, successively generate the next string. (b) Starting with 1^m, keep adding one 0 in all possible ways until all zero's are added (e.g., 11 $\rightarrow$ {011, 101, 110}).

NEXT (n, m)-BINARY-STRING GENERATION

Examples of Successive (10, 6)-Binary Strings:

A (10,6)-binary string:	0100111100
Next (10,6)-binary string:	0101<u>00</u>0111
Next (10,6)-binary string:	010100<u>10</u>11
Next (10,6)-binary string:	0101001<u>10</u>1
...	...
The last (10,6)-binary string:	1111000000

A necessary-and-sufficient condition for string y = next(x):

(1) The rigthmost "01" in x is changed to "10" in y.

(2) The 1's to the right of "01" in x is 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.

Questions:

1? What happens when there is no "01"?

2? Give a pseudocode for creating the previous (n,m)-binary string (if any).

3? 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 $x_1 x_2 \cdots x_n$? 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 locate the right-most ’1’.
2. Continue right-to-left scan till you find the first ’0’.

Question:

1? What is the right-to-left scan is better than left-to-right scan to locate the rightmost "01" (for our application)?

2? Give a program code for the pseudocode above, assuming that the binary string is represented as binString[0..length–1].

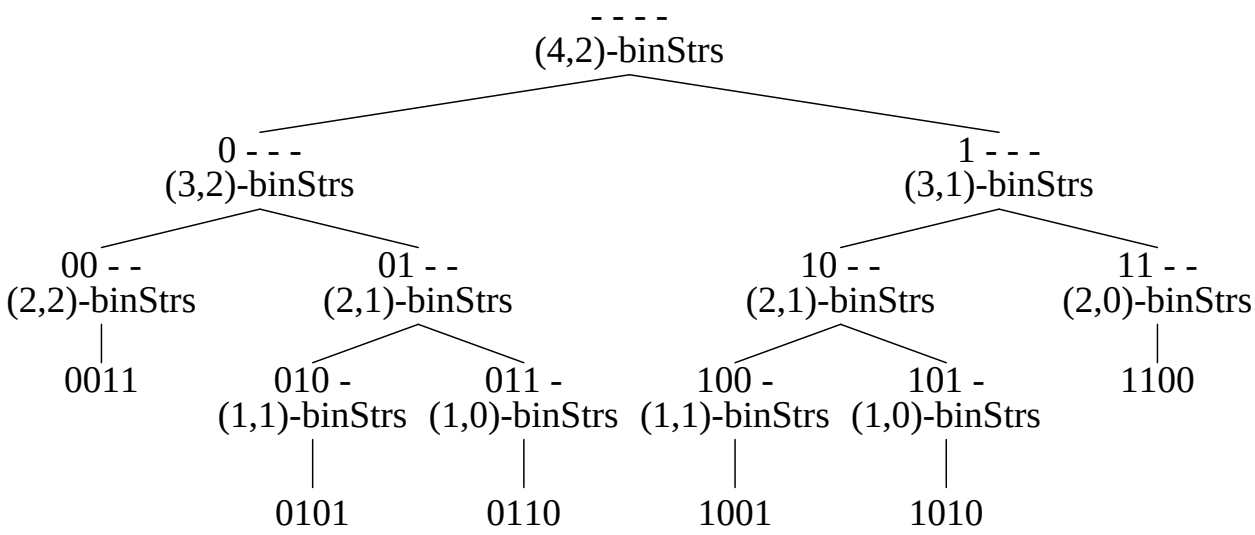
Does the following code find the rightmost "01"?

```
for (i=length-1; i>=1; i--)
    if ((1 == binString[i]) && (0 == binString[i-1]))
       break;
```

Question:

•? 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.

A RECURSIVE APPROACH FOR GENERATING ALL (n, m)-BINARY STRINGS



Pseudocode for RecGenBinStrings(n, m):

1. If top-level call, then create the array binString[0..$n-1$] and let strLength = n.
2. If ($n = m$) or ($m = 0$), then fill the last n positions in binString with 1's or 0's resp., print binString, and return;

   otherwise, do the following:

   (a) Let binString[strLength – n] = '0' and call RecGenBinStrings($n-1$, m).

   (b) Let binString[strLength – n] = '1' and call RecGenBinStrings($n-1$, $m-1$).

Question:

1? Let $W(n, m)$ = #(total write-operations into 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 $1 \le n \le 6$ and $0 \le m \le n$ in Pascal-triangle form.

ANOTHER STRING PROBLEM

Substring: Given a string $x = a_1 a_2 \cdots a_n$, 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 substring.

Question:

•? How many ways can we form k-substrings of $a_1 a_2 \cdots a_n$? When do we get all k-substrings ($0 < k < n$) 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.

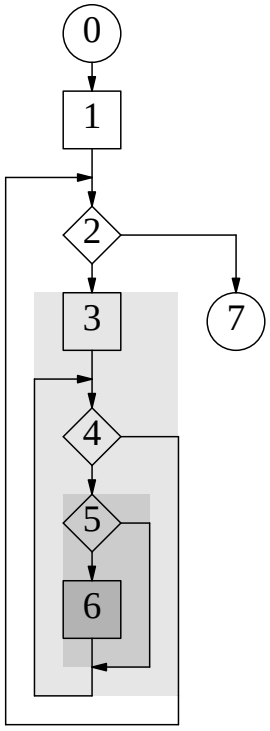
$$\begin{aligned} x_{ab} &= aababba, & x_{bc} &= bccbb, \\ x_{ac} &= aacaca, & x_{bd} &= bbbdd, \\ x_{ad} &= aaaadd, & x_{cd} &= ccdd. \end{aligned}$$

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 $y_{cd} = x_{cd}$.

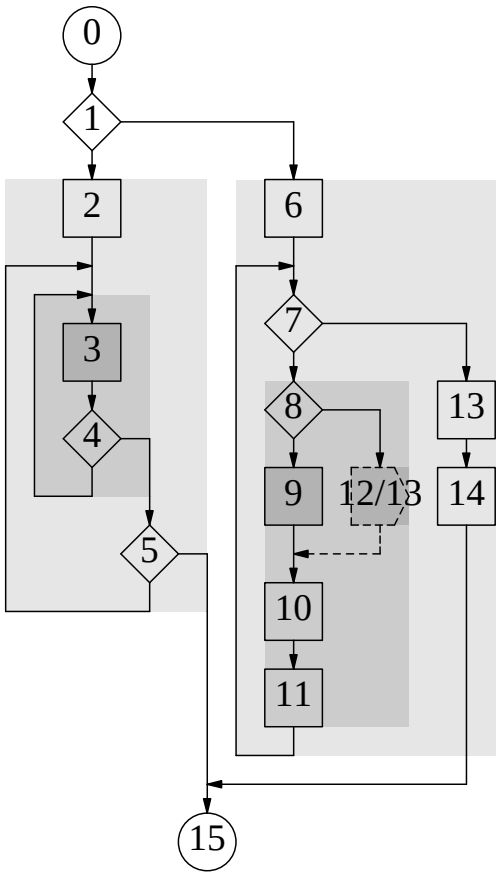
•? Give an algorithm to determine the string x from its 2-projections. Explain the algorithm using $x = aabcacbbadd$.

A FLOWCHART-DRAWING PROBLEM



- The while-do loop at node 4 checks characters in a word for counting.
- The while-do loop at node 2 reads one-word at-a-time from an input fi le.

(i) The fbwchart of an Word-AndCharCount program.



(ii) A more complex fbwchart; node 12 is a break-statement.

Question (Flowchart-Drawing):

1? What are some of the decisions involved in "drawing" a fbw-chart like the ones shown above?

2? What is involved in making those decisions and in which order can we decide them in drawing the fbwchart?

Question (Flowchart-Analysis):

3? How to verify that a fbwchart corresponds to a structured-program (no ’goto’s, but may have returns = ’goto end’)?

4? Can we combine the do-while loops at nodes 4 and 5 in the rightside fbwchart into a single do-while loop?

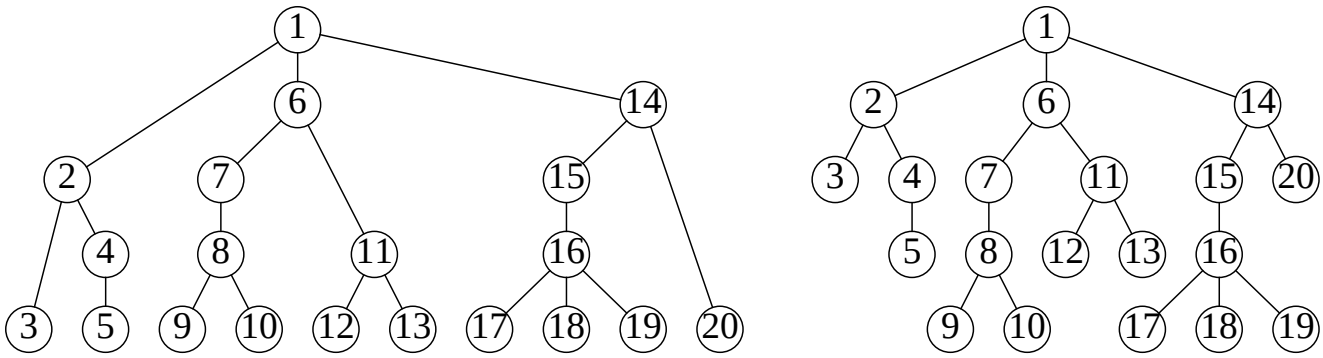
A TREE-DRAWING PROBLEM

Two Different Drawings of A Tree:

(1) All terminal nodes at the same level and with the horizontal separation δ_x between two consecutive nodes.

- The horizontal position of a parent node equals that of the middle child or the midpoint of the middle two children (resp. for odd and even number of children).
- The vertical position of the parent is at a fixed vertical separation δ_y of that of the highest children.

(2) Nodes with the same distance from root are at the same level and with at least δ_x horizontal separation between them.



The drawing on the right is more compact and yet it clearly shows the subtree-structure.

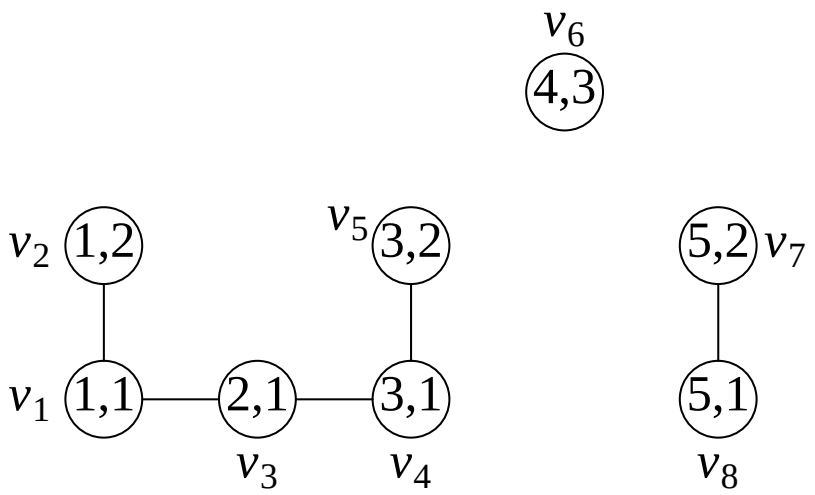
Questions:

1? If y and z are two consecutive children of a node x and we know the drawing of the subtrees T_y and T_z by themselves, how do we determine the horizontal separation between y and z in the drawing of T_x (e.g., take $y = 6$ and $z = 14$)?

A PROBLEM IN WIRELESS NETWORK

Problem: Given the coordinates (x_i, y_i) of the nodes v_i, $1 \le i \le N$, 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 $r = c.\sqrt{P}$ from it ($c > 0$ is a constant).
- Let $r_{\min}$ be the minimum r for which the links $E(r) = \{(v_i, v_j): d(v_i, v_j) \le r\}$ form a connected graph on the nodes. Then, $P_{\min} = (r_{\min}/c)^2$ gives the minimum transmission power to be used by each node.



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

Question:

1? What is $r_{\min}$ for the set of nodes above? Give an example to show that $r_{\min} \ne \max$ {distance of a node nearest to v_i: $1 \le i \le N$}. (If $r_{\min}$ were always equal to the maximum , then what would be an algorithm to determine $r_{\min}$?)

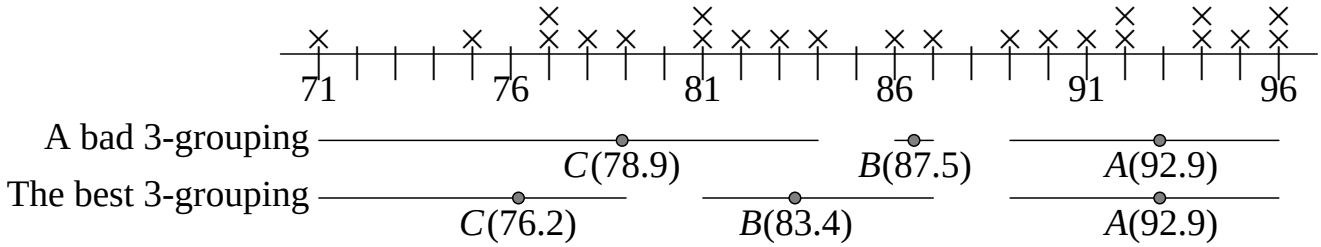
GROUPING NUMERICAL SCORES INTO CLASSES

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

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 goup.
- Thus, we only need to find the group boundaries.

Example. Scores of 23 students in a test (one '×' per student).



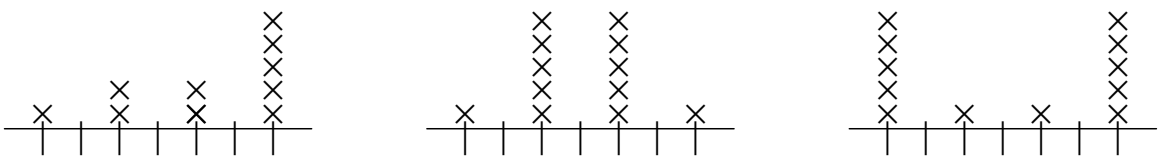
Closest-Neighbor Property (CNP) for Optimal Grouping:

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

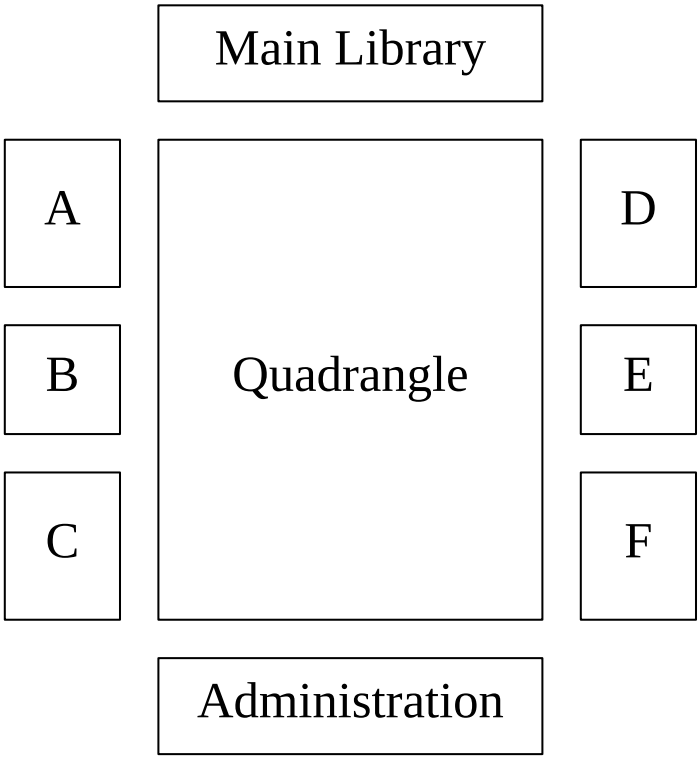
Question:

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 effi cient)?

3? What would be a better algorithm?

WHAT IS WRONG IN THIS ALGORITHM

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

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

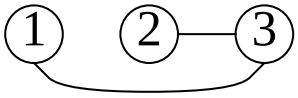
1. For (each $j = 2, 3, \cdots, n$}, choose a random $i \in \{1, 2, \cdots, j - 1)$ and output the edge (i, j).

Successive Edges Produced for n = 3:

- j = 2: the only possible i = 1 and the edge is (1, 2).

- j = 3; i can be 1 or 2, giving the edge (1, 3) or (2, 3).

Cannot generate the tree:



Always test your algorithm.

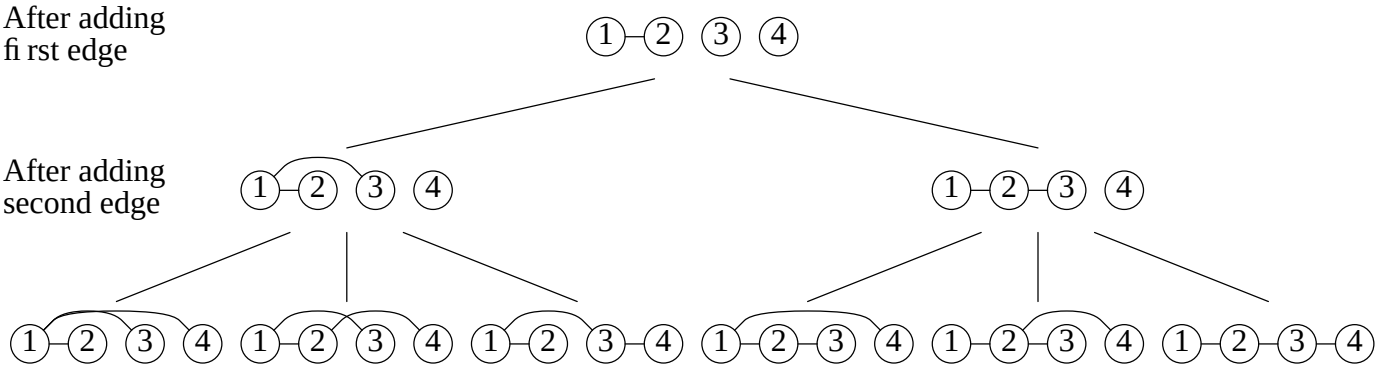
Question:

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)



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, ···, n – 1), do the following:
   (a) Choose random i and j, $1 \le i < j \le n$ 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 $\{1, 2, \cdots, n\}$. 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 $\{1, 2, \cdots, n\}$. 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.

A NON-COMPUTABLE PROBLEM

Halting Problem: Given a program P and an input I, how to determine if P will halt for the input I.

- There is no algorithm to solve the Halting Problem for all (P, I) pairs.
- One might be able to solve specific cases of (P, I) using known properties of P and I.

Question: Why can't we just execute program P for the given input I to see if the program halts?

Example. Does the following program halt for a given input x?

```
void DoesItHalt(double x)
{ double y, sum;
  for (sum = y = 1.0; sum >= 1.0; sum += y)
      y = y*x;
  printf("x=%5.3f, sum=%5.3f\n", x, sum);
}
```

Vars	Initial values	After nth iteration of for-loop n=1	n=2	n=3	n=4
sum	1.0	$1+x$	$1+x+x^2$	$1+x+x^2+x^3$	$1+x+\cdots+x^4$
y	1.0	x	x^2	x^3	x^4

The program will halt if and only if $x < 0$.

GOALS IN THIS COURSE

Goals: Become proficient in algorithm design, complexity-analysis, and application.

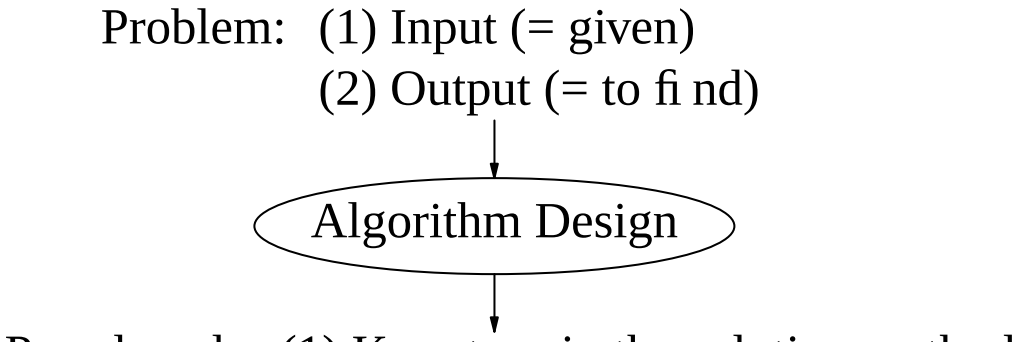
- Become expert in some fundamental graph and combinatorial algorithms, which have many applications.
- Develop new ways of thinking about algorithms.
- Develop ability to
  - convert new problems to known forms
  - create new methods/algorithms
  - determine efficient implementations
  - express algorithms in a pseudocode form

Some Fundamental Algorithms Covered:

- Shortest-paths/longest-paths in a directed graph
- Depth-first search for connected/biconnected components
- Depth-first search for strong components
- String matching
- Maximum flow in a network
- Minimum weight spanning tree in a graph
- Optimal binary-tree for expression-evaluation

They illustrate three main algorithm design techniques:
greedy, dynamic programming, depth-first and general search.

TYPES OF ALGORITHMS



- 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 $partialSums[i] = nums[1] + nums[2] + \cdots + nums[i]$ for $1 \le i \le N$.

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.

```
partialSums[1] = nums[1];
for (i=2 to N) do the following:
    partialSums[i] = nums[1];
    for (j=2 to i) add nums[j] to partialSums[i];
```

#(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.

```
partialSums[1] = nums[1];
for (i=2 to N)
    partialSums[i] = partialSums[i – 1] + nums[i];
```

#(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 in $vals[.,.]$ in the range $1 \le i' \le i$ and $1 \le j' \le j$) for all i and j.

Example.

$$\text{vals} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{bmatrix} \qquad \text{counts} = \begin{bmatrix} 1 & 1 & 1 & 2 \\ 1 & 2 & 2 & 4 \\ 1 & 3 & 4 & 7 \end{bmatrix}$$

- 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 $counts(i, j)$.

$$counts(i, j) \le \begin{cases} counts(i, j+1) \\ counts(i+1, j) \end{cases}$$

$$counts(1, j+1) = counts(1, j) + vals[1, j+1]$$

$$counts(i+1, 1) = counts(i, 1) + vals[i+1, 1]$$

$$counts(i+1, j+1) = counts(i+1, j) + counts(i, j+1) - counts(i, j) + vals[i+1, j+1]$$

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

Algorithm:

1. Let *counts*(1, 1) = *vals*[1, 1]; compute the remainder of first row *counts*(1, *j*), $2 \le j \le N$, using *counts*(1, $j+1$) = *counts*(1, *j*) + *vals*[1, $j+1$].
2. Compute the first column *counts*(*i*, 1), $1 \le i \le M$, similarly.
3. Compute the remainder of each row ($i+1 = 2, 3, \cdots, M$), from left to right, using the formula for *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 *counts*(*i*, *j*) and *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 *vals*[*i*, *j*] at least once.

Brute-force method::

1. For each $1 \le i \le M$ and $1 \le j \le N$, start with *counts*(*i*, *j*) = 0 and add to it all *vals*[*i*′, *j*′] for $1 \le i' \le i$ and $1 \le j' \le j$.

Complexity: #(additions) = $\sum_{i=1}^{M} \sum_{j=1}^{N} ij = (\sum_{i=1}^{M} i)(\sum_{j=1}^{N} j) = O(M^2 N^2)$

MAXIMIZING THE SUM OF CONSECUTIVE ITEMS IN A LIST

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

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

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].

(2) *nums*[1] is combined with others to form M. Then we can replace *nums*[1..3] by *nums*[1]+*nums*[2]+*nums*[3].

(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.

1. Let $M_1 = nums[1]$.
2. If ($N >= 3$) then do the following:
   (a) Let $nums[3] = nums[1] + nums[2] + nums[3]$ and let M_2 be the solution obtained by applying the algorithm to $nums[i]$, $3 \le i \le N$.
   (b) Let M_3 be the solution obtained by applying the algorithm to $nums[i]$, $3 \le i \le N$. (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 $M = \max\{M_1, M_2, M_3\}$.

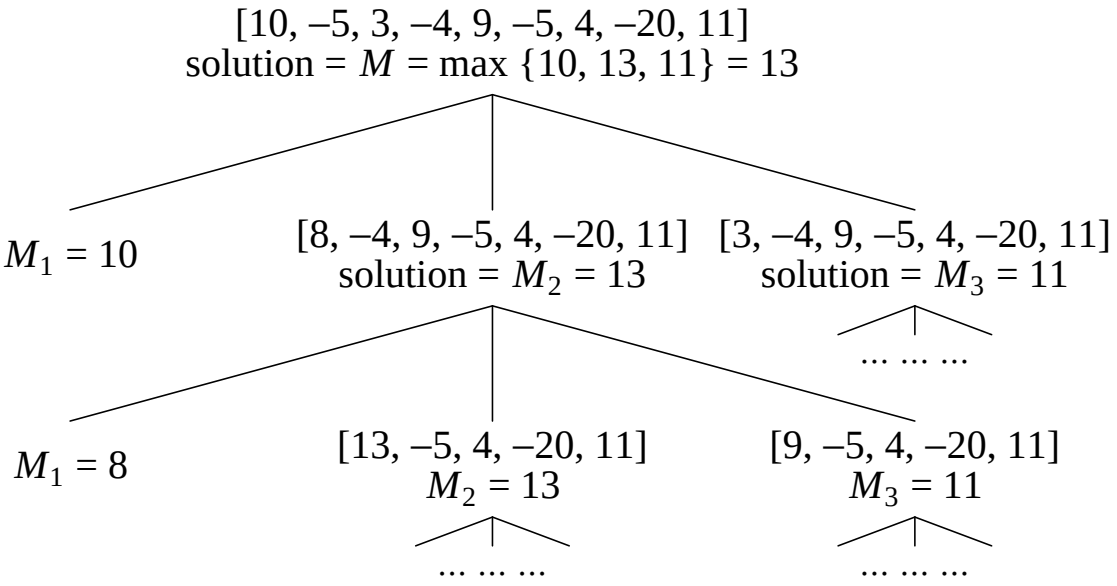
Question:

•? 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



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 *nums*[1. . *N*]?
- Let $T(N)$ = #(additions involving *nums*[*i*] 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 $T(N)$ = #(comparisons involving *nums*[*i*] 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 $M(j) = \max\{S_{ij}: 1 \le i \le j\}$; here, both $i, j \in \{1, 3, \cdots, N\}$.

Example. For *nums*[] = [10, –5, 3, –4, 9, –5, 4, –20, 11],

	$j = 1$	$j = 3$	$j = 5$	$j = 7$	$j = 9$
	$S_{11} = 10$	$S_{13} = 8$	$S_{15} = 13$	$S_{17} = 12$	$S_{19} = 3$
		$S_{33} = 3$	$S_{35} = 8$	$S_{37} = 7$	$S_{39} = -2$
			$S_{55} = 9$	$S_{57} = 8$	$S_{59} = -1$
				$S_{77} = 4$	$S_{79} = -5$
					$S_{99} = 11$
$M(j)$	10	8	13	12	11

Observations:

$$M(1) = nums[1].$$
$$M(j+2) = \max\{M(j) + nums[j+1] + nums[j+2], nums[j+2]\}.$$
$$M = \max\{M(j): j = 1, 3, \cdots, N\}.$$

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

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

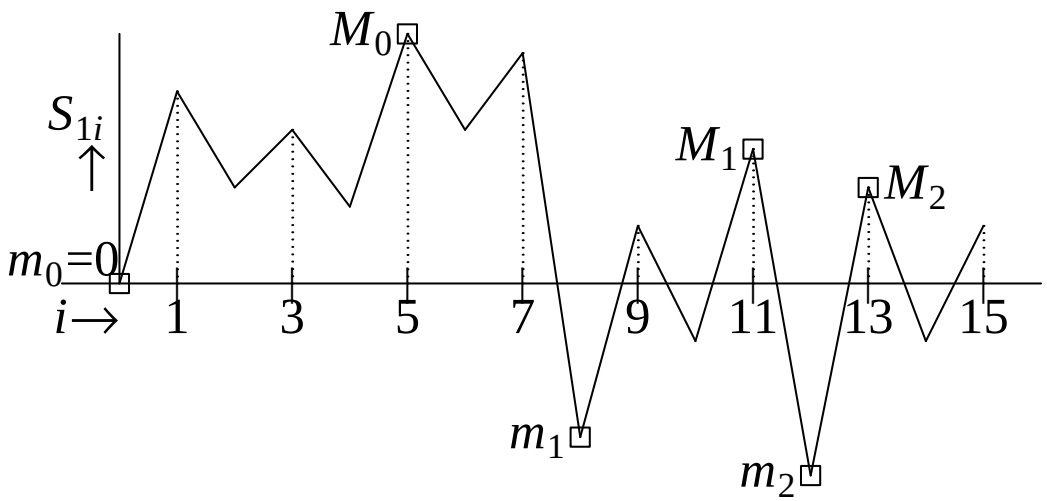
Complexity: $O(N)$.

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

ANOTHER $O(N)$ METHOD

Observation:

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



Solution Method: There are three steps.

1. Find $(i - 1)$'s which can possibly give maximum S_{ij}.
   - Find the successive decreasing items $m_0 > m_1 > m_2 > \cdots > m_n$ among $S_{1,i-1}$, $i = 1, 3, \cdots, N$. (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 $M_{k-1} = \max \{S_{1,j}: i_{k-1} \le j < i_k\} = S_{1,j_k}$ for $1 \le k \le n$; let $M_n = \max \{S_{1,j}: j \ge i_n\}$.
3. Let $M = \max \{M_k - m_k: 0 \le k \le n\}$.

(CONTD.)

A Slightly Larger Example.

$nums[i]$:	10	–5	3	–4	9	–5	4	–20	11	–6	10	–17	14
i, j:	1		3		5		7		9		11		13
$S_{1,i-1}$:	0		5		4		8		–8		–3		–10
m_k:	m_0								m_1				m_2
i_k:	1								9				13
$S_{1,j}$:	10		8		13		12		3		7		4
M_k:					M_0=13						M_1=7		M_2=5
j_k:					5						11		13

$$i_1 = 1, \quad i_2 = 9, \quad i_3 = 13$$
$$j_1 = 5, \quad j_2 = 7, \quad j_3 = 13$$

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

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 specifi c programming language.

+ Allows determining correctness of the algorithm.

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

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

(A)
```
1. positiveCount = negativeCount = 0;
2. for (i=1; i<=N; i++) //each nums[i] > 0 or < 0
3.     if (0 < nums[i]) positiveCount++;
4.     else negativeCount++;
```

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

(B)
```
1. positiveCount = 0;
2. for (i=1; i<=N; i++) //each nums[i] > 0 or < 0
3.     if (0 < nums[i]) positiveCount++;
4. negativeCount = N - positiveCount;
```

1. Initialize positiveCount = 0.
2. Use each $nums[i] > 0$ to increment positiveCount by one.
3. 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 maxNonZeroBlockSize = 0.
2. while (there are more array-items to look at) do:
   (a) skip zero's. //keep this
   (b) fi nd the size of next non-zero block
   and update maxNonZeroBlockSize.

Code:

```
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 specifi c 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:

  Print: print the "on"-switches (in the order 1, 2, ..., N) in time proportional to M = #(switches that are "on").

  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.

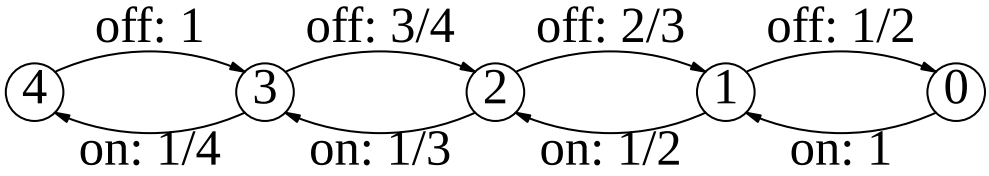
Example: Shown below are some on/off-operations (1 = on and 0 = off).

Switches[1..9]:	1	2	3	4	5	6	7	8	9
	0	1	1	0	1	0	1	1	1
Off(3):	0	1	0	0	1	0	1	1	1
Off(5):	0	1	0	0	0	0	1	1	1
On:	0	1	0	0	1	0	1	1	1

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).



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 $E_0 = 0$

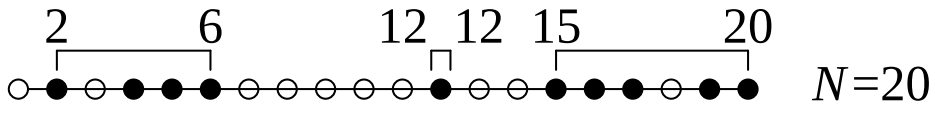
- Thus, $E_4 = 1 + (1+2/3) + (1+2/2+2/6) + (1+2/1+2/2+2/6) = 9\frac{1}{3}$.

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 $PI = \{I_1, I_2, \cdots, I_k\}$, $k \leq m$, where I_j's are disjoint, $\cup I_j$, $1 \leq j \leq k$, covers all non-empty pages, and $|\cup I_j|$ is minimum.

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



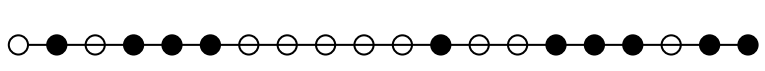
Solution by Greedy Elimination:

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

- $\cup I_j$ need not cover all non-empty pages.
- Maximize Val(PI) = #(non-empty pages covered by $\cup I_j$) – #(empty pages covered by $\cup I_j$) = $|\cup I_j|$ – 2.#(empty pages covered by $\cup I_j$).

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:

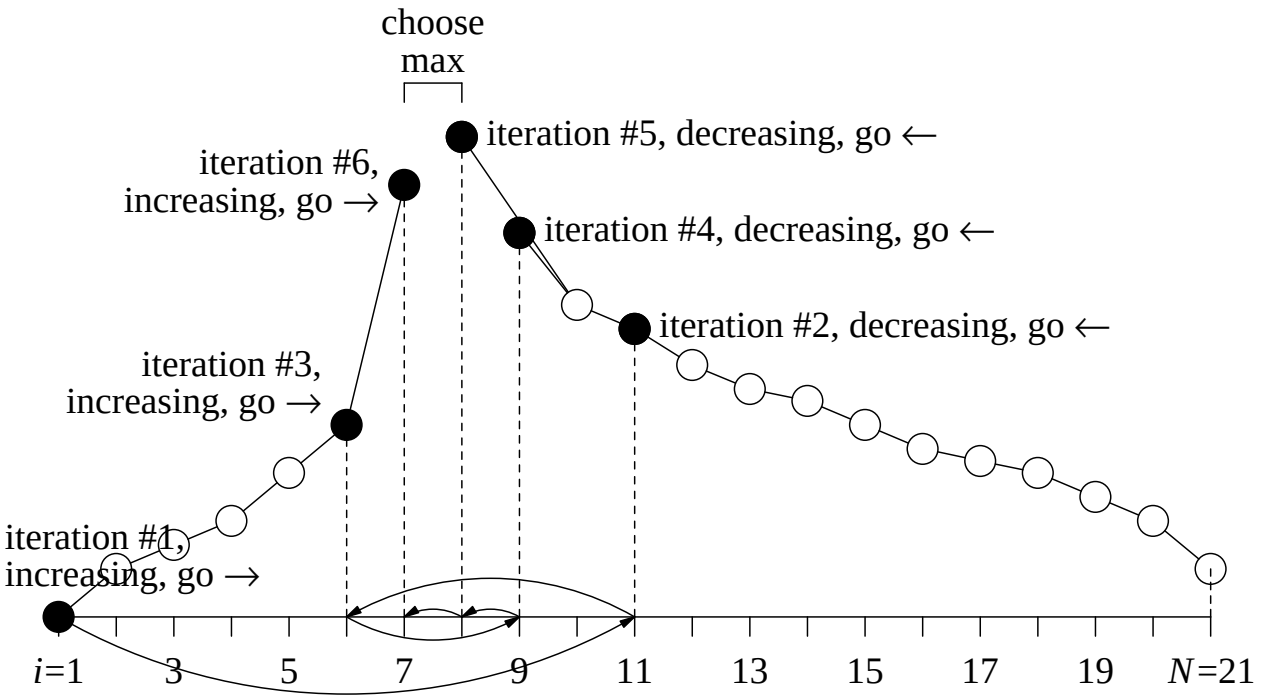
1. minimum = min $\{nums[1], nums[N]\}$.
2. If $(nums[N-1] < nums[N])$ then maximum = $nums[N]$.
3. Otherwise, starting with the initial range $1..N$ and position 1, do a binary search. In each step, we move to the mid-point i of the current range and then select the right-half of the range if the numbers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

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 kth (though we are closer to the maximum at $(k+2)$th iteration than at kth iteration).
- To compute maximum by the principle of *extending* the solution from the case N to $N+1$, we would proceed as:

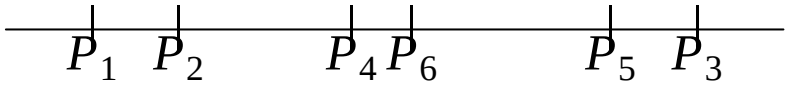
  (1) If $(nums[N+1] > nums[N])$ then max = $nums[N+1]$.

  (2) Otherwise, apply the same method to $nums[1..N]$.

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

FINDING A CLOSEST PAIR OF POINTS ON A LINE

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



$\{P_4, P_6\}$ 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 $1 \le i < j \le N$), compute d_{ij} = distance(P_i, P_j).
2. Find the pair (i, j) which gives the smallest d_{ij}.

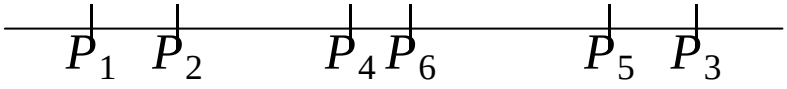
Implementation (combines steps (1)-(2) to avoid storing d_{ij}'s):

```
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:

- ? Restate the pseudo-code to reflect the implementation?
- ? Give an alternate implementation to avoid the repeated assignment "besti = i" without increasing the complexity.

(CONTD.)



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

A better approach:

- The point nearest to P_i is to its immediate left or right.
- Finding immediate neighbors of each P_i require sorting.

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 $1 \le j < N$ such that *nums*[$j+1$] – *nums*[j] is minimum.
3. Output *nums*[j] and *nums*[$j+1$].

Complexity:

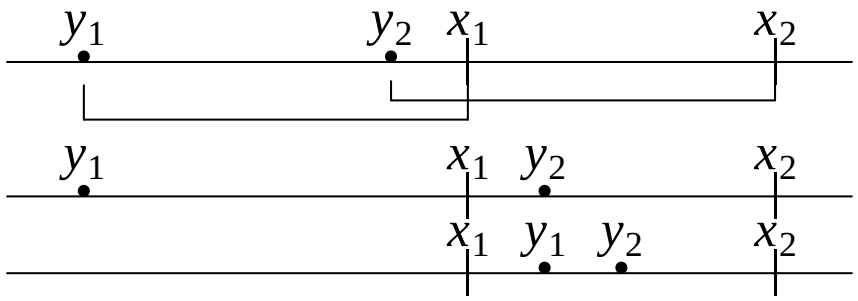
- Sorting takes $O(N \log N)$ time; other computations take $O(N)$ time.
- Total = $O(N \log N)$.

A geometric view often leads to a better solution.

A MATCHING PROBLEM

Problem:

We have the scores $x_1 < x_2 < \cdots < x_N$ for N male students M_i in a test, and the 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 $E = \sum(x_i - y_{i'})^2$ $(1 \le i \le N)$, the squared sum of differences in scores for matched-pairs.



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.

Complexity: $O(N.N!)$.

- Computing $N!$ permutations takes at least $N(N!)$ time.
- Computing E for each permutation takes: $O(N)$.

  Total = $O(N(N!))$
- Finding minimum takes $O(N!)$.

A BETTER METHOD FOR MATCHING BOYS AND GIRLS

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 ith smallest x with ith smallest y gives the minimum E.

Question:

•? How can you prove (1)?

•? Consider now $N = 3$, the first distribution above, and $x_1 < y_3 < x_2 < x_3$. 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 all distributions.)

Pseudocode (exploiting output-properties/criteria):

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

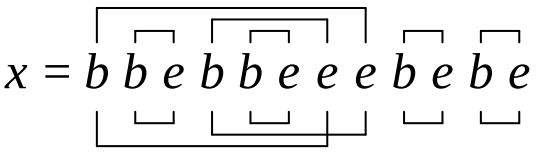
EXERCISE

1. How can you generate all permutations of $\{1, 2, \cdots, N\}$?
2. 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.

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:*

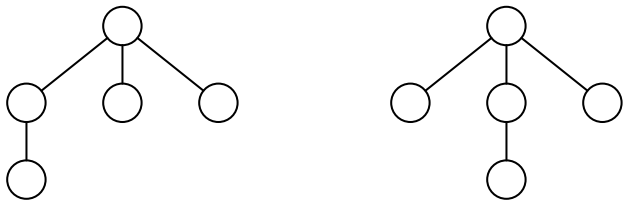
If both x and y are balanced, then xy is balanced.

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

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

ORDERED ROOTED TREES

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

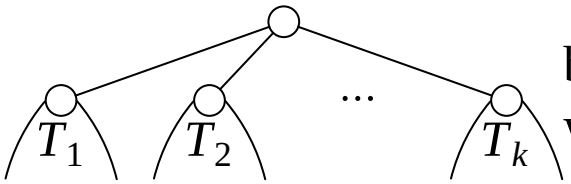


Two different ordered rooted trees; as unordered rooted trees, they are considered the same.

- 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 *be*-STRINGS

○ balString(T) = λ

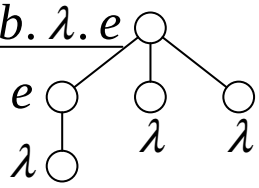


balString(T) = $bx_1e.\, bx_2e.\, \cdots .\, bx_ke$
where x_i = balString(T_i)

Example. Build the string *balString(T)* bottom-up.

bbeebebe = $\underline{b.\, be.\, e}.\underline{b.\, \lambda.\, e}.\underline{b.\, \lambda.\, e}$

$be = b.\, \lambda.\, e$



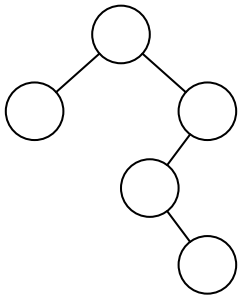
Question:

- •? What would be wrong if for the one-node tree we take beString(T) = *be* (instead of λ)?
- •? How will you show that balString(T_1) ≠ balString(T_2) for $T_1 \neq T_2$, and that balString(T) is always balanced?
- •? How will you show that for every balanced *be*-string x there is a tree T with balString(T) = x?

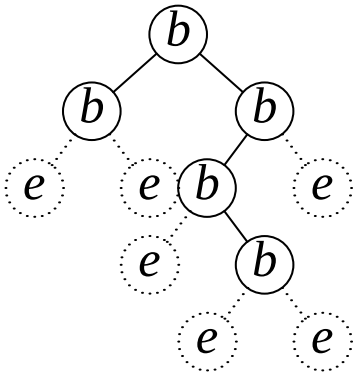
#(ordered rooted trees with $(n+1)$ nodes)
= #(balanced *be*-strings of length $2n$) = $\dfrac{1.3\cdots(2n-1)}{(n!)} \cdot \dfrac{2^n}{(n+1)}$

- For length = $2n$, $\dfrac{\#(\text{balanced } be\text{-strings})}{\#(\text{all } be\text{-strings})} \to 0$ as $n \to \infty$.

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 *bbeebbebeee* and beString(T) = *bbeebbebee*.

Question:

- •? If n = #(nodes in T), then how many news nodes are added?
- •? What is the special property of the new binary tree?
- •? In what sense the pre-order listing *bbeebbebeee* is almost balanced? How will you prove it?
- •? How is beString(T) related to beString(T_1) and 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 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}.

bbbb	bbbe	bbeb	bbee
bebb	bebe	beeb	beee
ebbb	ebbe	ebeb	ebee
eebb	eebe	eeeb	eeee

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:

- $O(N.2^N)$ for step (1).
- $O(N)$ to verify balancedness of each *be*-string in step (2).
- Total = $O(N.2^N)$.

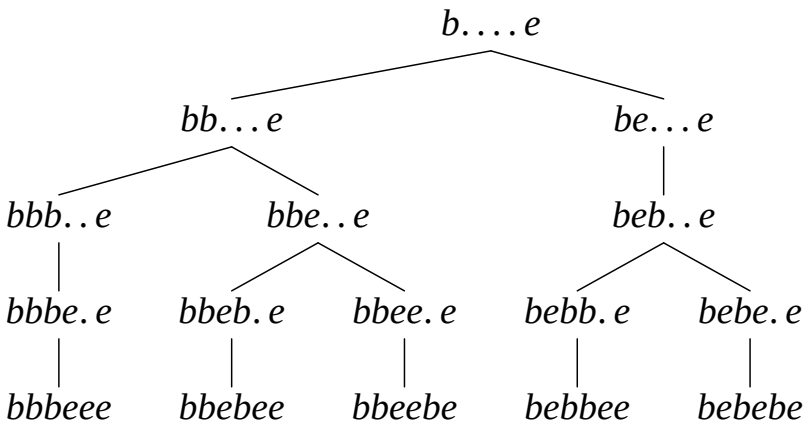
A BETTER METHOD BY USING THE OUTPUT-STRUCTURE

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 fi lling the leftmost 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*, firstBlankPosn, numPrevBs):

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

2.1. Let numPrevEs = #(*e*'s before firstBlankPosn).

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

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

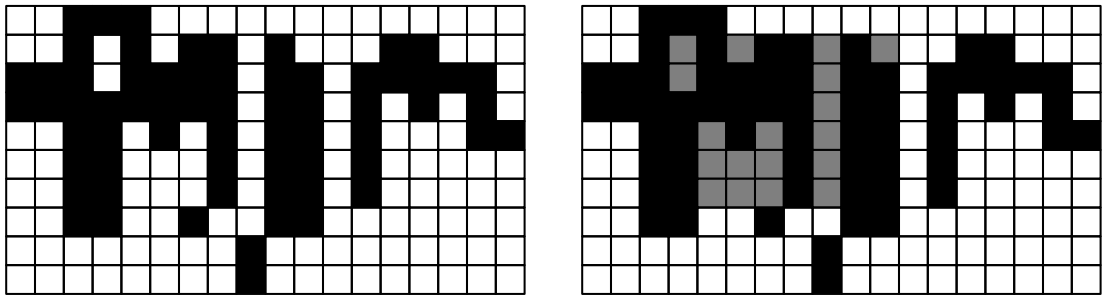
Implementation Notes:

- Make *beString* a static-variable in the function instead of passing as a parameter.
- Eliminate the parameters firstBlankPosn and numPrevB by making them static variable in the function, and use the single parameter length.
- Eliminate the variable numPrevEs (how?).
- Update firstBlankPosn and numPrevBs before and after each recursive call as needed. Initialize the array *beString* when firstBlankPosn = 1 and free the memory for *beString* before returning from the first 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';
         beString[length] = '\0'; //helps printing
08.      firstBlankPosn = numPrevBs = 1;
09.   }
10.   if (length-1 == firstBlankPosn)
         printf("beString = %s\n", beString);
11.   else { if (2*numPrevBs < length) {
12.             beString[firstBlankPosn++] = 'b';
                numPrevBs++;
13.             GenBalBeStrings(length);
14.             firstBlankPosn--; numPrevBs--;
15.          }
16.          if (2*numPrevBs > firstBlankPosn) {
17.             beString[firstBlankPosn++] = 'e';
18.             GenBalBeStrings(length);
19.             firstBlankPosn--;
20.          }
21.        }
22.   if (1 == firstBlankPosn)
         { free(beString); beString = NULL; }
23. }

24. int main()
25. { int n;
26.   printf("Type the length n (even and positive) ");
      printf("of balanced be-strings: ");
27.   scanf("%d", &n);
28.   if ((n > 0) && (0 == n%2))
         { GenBalBeStrings(n); GenBalBeStrings(n+2); }
29. }
```

FINDING A BEST RECTANGULAR APPROXIMATION TO A BINARY IMAGE

Example. Black pixels belong to objects; others belong to background. Let B = 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.
$Val(\varnothing)$ = $|B| = 65$; $Val(\text{I})$ = #(white pixels) = 115

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

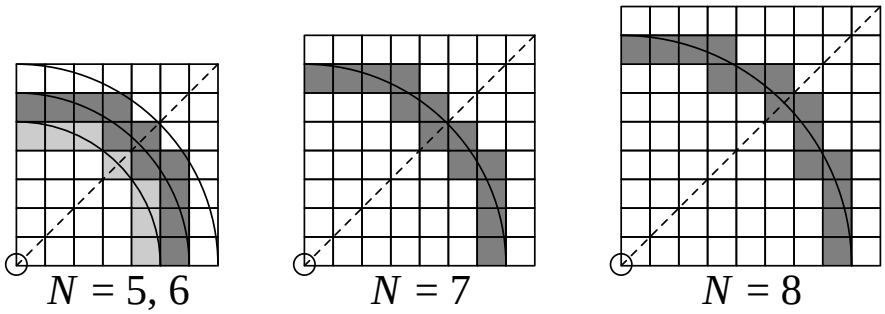
EXERCISE

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.



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.,

  $O(n) = \{g(n)$: $g(n) \le c.\, n$ for *some* constant c and *all large* $n\}$

  – c may depend on $g(n)$; $c > 0$.

  – "all large n" means "all $n \ge N$ 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 $c = 4$, $7 + 3n \le 4.\, n$ holds for $n \ge 7 = N$.

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

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

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

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

Example. We show $g(n) = A.\, n^2 \notin O(n)$.

For any $c > 0$, $A.\, n^2 < c.\, n$ 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.,

$$O(n^2) = \{g(n): g(n) \le c.\, n^2 \text{ for } \textit{some} \text{ constant } c \text{ and } \textit{all large } n\}$$

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

(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 $c = 6$, $7 + 3n + 5n^2 \le 6.\, n^2$, i.e., $n^2 - 3n - 7 \ge 0$ holds for $n \ge 5 = N$.

(2) For $c = 4$, $7 + 3n + 5n^2 \le 4.\, n^2$, i.e., $-n^2 - 3n - 7 \ge 0$ does not hold for any $n \ge 1$. 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 $c.\, g(n) \in O(n^p)$.

(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 $g_1(n) \leq c_1.\, n^p$ for all $n \geq N_1$ and $g_2(n) \leq c_2.\, n^p$ for all $n \geq N_2$.

Then, $g_1(n) + g_2(n) \leq (c_1 + c_2).\, n^p$ 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 $g_1(n)g_2(n) \in O(n^{p+q})$.

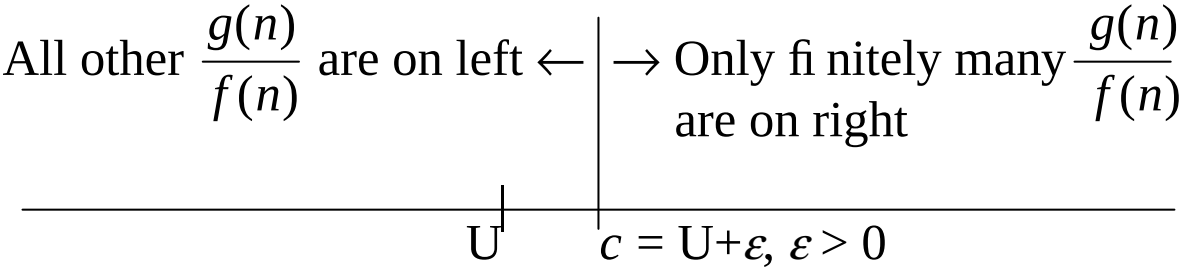
Also, $\max\{g_1(n), g_2(n)\} \in O(n^q)$ assuming $p \leq q$.

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 $g(n) \in O(f(n))$

$O(f(n)) = \{g(n)$: $g(n) \le cf(n)$ for some *constant* c and all large $n\}$

$= \{g(n)$: $\limsup_{n \to \infty} \frac{g(n)}{f(n)} = U < \infty\}$.



- 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 lim sup $\frac{g(n)}{n}$ = lim sup $\frac{7+3n}{n}$ = 3 < ∞.

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

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

(4) $g(n) = 2^n \notin O(n^p)$ for any $p = 1, 2, \cdots$.

ASYMPTOTIC LOWER BOUND $\Omega(f(n))$

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

$$\liminf_{n \to \infty} \frac{g(n)}{f(n)} = L > 0 \ (L \text{ maybe } +\infty)$$

  i.e, $\dfrac{g(n)}{f(n)} > L - \varepsilon$ 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

$$g(n) \text{ is } \Omega(f(n)) \quad \text{or} \quad g(n) = \Omega(f(n)).$$

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 (Ω1)-(Ω5) 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 $f(n) \in \Theta(g(n))$?

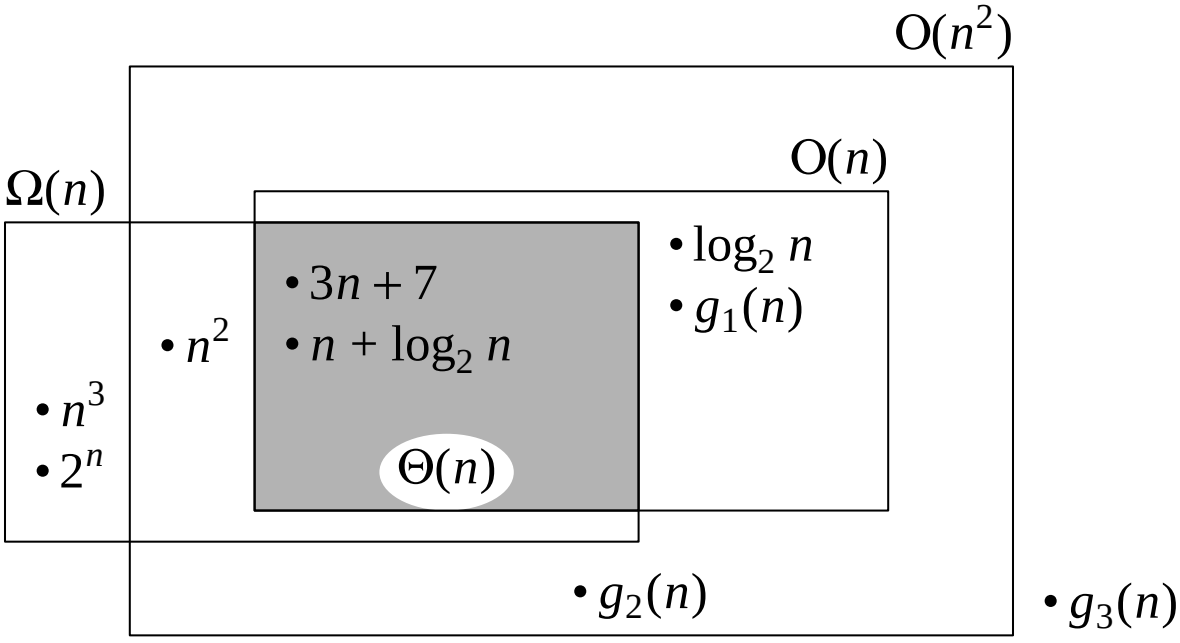
Example.

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

(2) If $\log_2(1+n) \le g(n) \le 1 + \log_2 n$, then $g(n) = \Theta(\log_2 n)$.

Question: If $g_1(n) = \Theta(n^p)$, $g_2(n) = \Theta(n^q)$, and $p \le q$, then what can you say for $g_1(n) + g_2(n)$ and $g_1(n)g_2(n)$?

COMPARISON OF VARIOUS ASYMPTOTIC CLASSES



$$g_1(n) = \begin{cases} \log_2 n, \text{ for } n \text{ even} \\ n, \text{ for } n \text{ odd} \end{cases} \qquad g_2(n) = \begin{cases} \log_2 n, \text{ for } n \text{ even} \\ n^2, \text{ for } n \text{ odd} \end{cases}$$

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

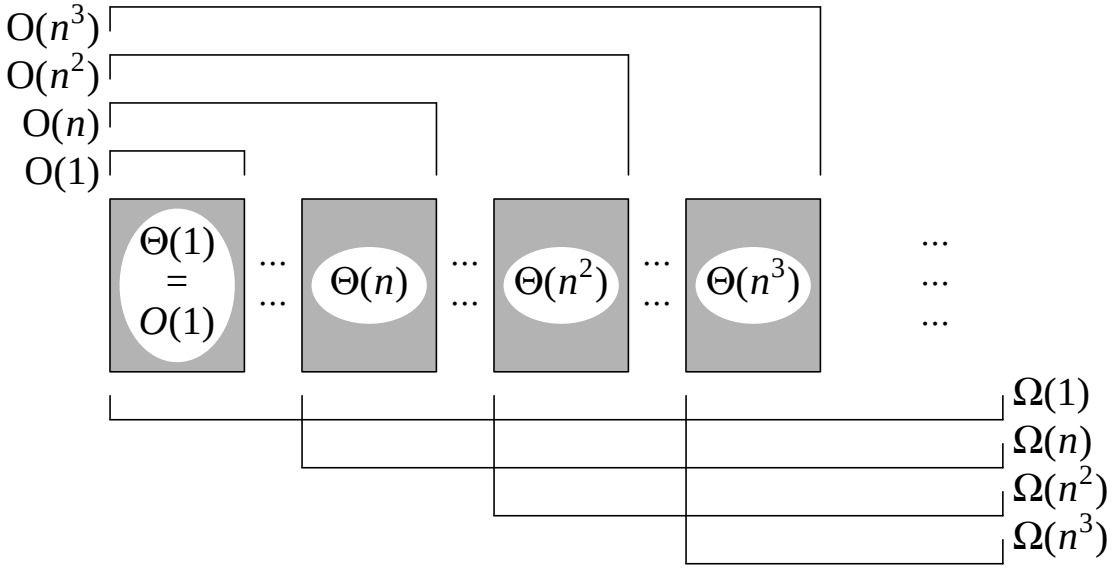
Question:

•? Place the boxes for Ω(n^2) and Θ(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 Ω notation as appropriate, or give the exact order using Θ.

(CONTD.)



- There are infinitely many $\Theta(f(n))$ between $\Theta(1)$ and $\Theta(n)$ above; for example, we can have

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

- For each $\Theta(f(n))$ between $\Theta(1)$ and $\Theta(n)$, $\Theta(n^k . f(n))$ 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

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

6.
```
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;
    }
```

7.
```
for (i=0; i<3*length; i++)
    printf(" ");
```

8.
```
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, ...);
}
```

COMPARING EFFICIENCY OF ALTERNATIVE NESTED IF-ELSE COMPOSITIONS

- Let effi ciency E = average #(condition evaluations).

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

```
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;
```

Value of z	#(condition evaluations)
0	2 ($x = T$ and $y = T$)
1	3 ($x = F$, $\neg x = T$, and $y = T$)
2	5 ($x = T$, $y = F$, $\neg x = F$, $x = T$, and $\neg y = T$)
3	4 ($x = F$, $\neg x = T$, $y = F$, $x = F$)

Example 2. For the code below, $E = 2.0 < 3.5$.

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

Value of z	#(condition evaluations)
0	2 ($x = T$ and $y = T$)
1	2 ($x = F$ and $y = T$)
2	2 ($x = T$ and $y = F$)
3	2 ($x = F$ and $y = F$)