SOLUTIONS

1. **Pseudocode for sorting by putting "the items in the right places".** (There is an error in the pseudocode below; correct it.)

Algorithm sort()

*Input:* An array nums[0.. (n – 1)], n ≥ 2.

*Output:* The rearrangement of the items in nums[] in increasing order.

1. For each \(i = 0, 1, \ldots, (n – 2)\) do the following:

   (a) Find the right place \(0 \leq j \leq n – 1\) for nums\([i]\). (This means there are exactly \(j\) items in nums[] which are smaller than nums\([i]\); we sometimes call \(j = \text{rank}(\text{nums}[i])\).)

   (b) If \(i \neq j\), then exchange nums\([i]\) and nums\([j]\).

**Example.**

\[
\begin{align*}
[3, 2, 5, 4, 6, 0, 1] \\
[4, 2, 5, 3, 6, 0, 1] \text{ after putting 3 in right place}
\end{align*}
\]

...
2. **Pseudocode for next permutation.**

Algorithm nextPermutation(p) //p[] integer-array

*Input:* A permutation p of \{0, 1, 2, ..., (n−1)\}, n ≥ 2.

*Output:* The next permutation after p in lexicographic order, if any.

1. Find the rightmost index 0 ≤ i < n − 1 such that \( p[i] < p[i + 1] \). (The items in \( p[(i + 1). . (n − 1)] \) are decreasing to the right.)

2. If there is no such i, then there is no next permutation.

3. Otherwise, do the following:
   
   (a) find the smallest item \( p[j] \) in \( p[(i + 1). . (n − 1)] \) such that \( p[j] > p[i] \). (This is the same as finding the largest index \( i < j ≤ (n−1) \) such that \( p[j] > p[i] \); we can search for \( j \) from right to left or from left to right.)

   (b) Interchange \( p[j] \) and \( p[i] \).

   (c) Rearrange the items \( p[(i + 1). . (n − 1)] \), which are still in decreasing order, in increasing order.

**Example.**

\[ p = [3, 8, 0, 2, 9, 7, 6, 5, 4, 1] \]

step (1): \( i = 3, \ p[i] = 2 \)

step (3.a): \( j = 8, \ p[j] = 4 \)

step (3.b): \[3, 8, 0, 4, 9, 7, 6, 5, 2, 1]\n
step (3.c): \[3, 8, 0, 4, 1, 2, 5, 6, 7, 9]\n
3. **Correcting the bug in the code below.**

```c
i = n; // = length of binString
do { for (i=i-1 ; i>0; i--)
    if (0 == binString[i]) break;
} while (1 == binString[--i]);
```

**Problem:** for binary string 111011 while-loop goes out of bound.

- First time we enter while-loop body, the for-loop processes the tail part "011" and exits with \( i = 3 \).
- The next time we enter while-loop body, the for-loop starts with \( i = 2 \) and processes the initial part "11" (not "111") and exits with \( i = 0 \).
- Then, binString[--i] in while-test breaks down.

**Corrected code:**

```c
i = n; // = length of binString
do { for (i=i-1 ; i>0; i--)
    if (0 == binString[i]) break;
    if (0 == i) break;
} while (1 == binString[--i]);
```

**A slightly more compact form:**

```c
i = n; // = length of binString
do { for (i=i-1 ; i>0; i--)
    if (0 == binString[i]) break;
} while ((i > 0) && (1 == binString[--i]));
```
A CODE FOR nextPermutation(numItems)

- Returns 0 on generating the last permutation; otherwise, returns 1. This lets us generating all permutations by

```c
void testNextPermutation(n) //n > 1
{
    int i;
    do { i = nextPermutation(n);
    } while (i == 1);
}
```

- Can change numItem at any point.

```c
void test2NextPermutation()
{
    nextPermutation(5); //= [0, 1, 2, 3, 4]
    nextPermutation(5); //= [0, 1, 2, 4, 3]
    nextPermutation(5); //= [0, 1, 3, 2, 4]
    nextPermutation(4); //= [0, 1, 2, 3]
    nextPermutation(4); //= [0, 1, 3, 2]
    nextPermutation(4); //= [0, 2, 1, 3]
    nextPermutation(5); //= [0, 1, 2, 3, 4]
}
```

This is done by using a static variable oldNumItems.

- For the rightmost $p[i-1] < p[i]$, it shows the position $j \geq i$ with which $p[i-1]$ is exchanged and also the number of other exchanges in resoring $p[i..(n-1)]$ in increasing order.

```
[2,4,7,3,9,8,6,5,1,0]
|     |
i=4   j=7
[2,4,7,5,9,8,6,3,1,0] after exchange($p[i-1], p[j]$)
[2,4,7,5,0,1,3,6,8,9] after ordering $p[i..(n-1)]$
numOtherExchngs = 3
```
#include <stdio.h>
int *permtn; //short for permutation

//returns 0 on generating the last permtn, and otherwise returns 1
int nextPermutation(int numItems) //numItems > 1
{ int static oldNumItems=0, firstCall = 1,
    startDecreasingIndx; //numItems-1 or numItems-2
int i, j, temp, numExchngs;
if (numItems != oldNumItems)
{ free(permtn);
  permtn = (int *) malloc(numItems + sizeof(int));
  oldNumItems = numItems; firstCall = 1; }
if (firstCall) {
  firstCall = 0;
  for (i=0; i<numItems; i++)
    permtn[i] = i;
  startDecreasingIndx = numItems-1;
  PrintIntVector(permtn, numItems); //prints permutation
  return(1); }
else { for (i=startDecreasingIndx; i>0; i--)
  if (permtn[i-1] < permtn[i]) break;
  printf("startDecreasingIndx = %d and newIndex = %d\n",
        startDecreasingIndx, i);
  if (i == 0) {
    firstCall = 1; PrintIntVector(permtn, numItems);
    return(0); }
  else { for (j=numItems-1; j>0; j--)
    if (permtn[i-1] < permtn[j]) break;
    temp = permtn[i-1]; permtn[i-1] = permtn[j];
    permtn[j] = temp; //permtn[i-1] increases;
    numExchngs = 1 + (numItems - i)/2 ;
    printf("exchange posn %d with %d; numExchngs = %d\n",
            i-1, j, numExchngs);
    if (numExchngs > 1) {
      startDecreasingIndx = numItems-1;
      for (j=0; j<numExchngs; j++) {
        temp = permtn[i+j];
        permtn[i+j] = permtn[numItems-1-j];
        permtn[numItems-1-j] = temp; }
    }
  } startDecreasingIndx = numItems - 2;
  PrintIntVector(permtn, numItems); 
  return(1);
} }

Question:

•? Instrument this code to count # (comparisons of items in nums[]) to locate rightmost \(p[i-1] < p[i]\). For numItems = 7, print this count for each permutation and also the total for all permutations.
•? Modify the code (algorithm) to reduce this as much as possible.