1. Illustrate the operation of Insertion-sort algorithm on array A=<31,41,59,26,41,58>

```
i = 0
A[0] is 31
A: [31,41,59,26,41,58]
i <- 1

i = 1
A[1] = 41
A: [31,41,59,26,41,58]
i <- 2

i = 2
A[2] = 59
A: [31,41,59,26,41,58]
i <- 3

i = 3
A[3] = 26
A[3] < A[2]
A: [31,41,26,59,41,58]
A[2] < A[1]
A: [31,26,41,59,41,58]
A[1] < A[0]
A: [26,31,41,59,41,58]
i <- 4

i = 4
A[4] = 41
A[4] < A[3]
A: [26,31,41,41,59,58]
i <- 5

i = 5
A[5] = 58
A[5] < A[4]
A: [26,31,41,41,58,59]
```

2. The input to the algorithm Unknown illustrated below is an array A of N numbers. (1) what is the output of the algorithm? (2) using big-O notation to show the running time of the algorithm.

(1) it output the max of array A

(2) O(n)

3. What is the worst case and best case running time of selection-sort? How does it compare to insertion sort?

For selection-sort, the running time complexity is O(n^2) for the worst and best case because there is not early break in those two loops.

For insertion sort. The best case is a sorted array, which is O(n), and the worst case is reversed sorted array, which is O(n^2).

4. An array A[1..n-1] contains allintegers from 0 to n except twonumbers. It would be easy to determine to thetwomissing numbersby using an auxiliary array B[0..n] to record which numbers appear in A. Here, we want to avoid the additional storage of B with the size O(n). Devise an algorithm to determine the two missing integersin O(n) time under this constraint. (Note, you can still use additional constant memory as temporary storage.)

```
let sumA = sum of A
let productA = product of A, except 0
let sumAll = sum from 0 to n
let productAll = product from 1 to n

Know we know for two missing number a and b,

a+b = sumAll - sumA
a*b = productAll / productA

a - b = sqrt( (a+b)^2 - 4*a*b );
Then we know a and b
```

5. Let A[1..n] be an array of n distinct numbers. If i < j and A[i] > A[j], then the pair (i, j) is called an inversion of A. a) List the five inversions of the array <2, 3, 8, 6, 1>. b) What array with elements from the set {1, 2, ..., n} has the most inversions? How many does it have? c) What is the relationship between the running time of insertion sort and the number inversions in the input array? Justify your answer.

a. (8, 6) (2, 1) (3, 1) (8, 1) (6, 1)

b. The most inversions happen when array is reversed sorted. In that case, select any two items in the array will be a inversions. Therefore, there is `n*(n-1)/2` inversions

c. The number of inversions in the array is proportional to the running time for insertion sort to sort that array, because the insertion sort swap a inversions in the array once a time in inner loop. If there is no inversions in array, then insertion sort will neglect inner loop. However, If the array is reversed sorted, then swap in the inner loop will be need to remove all inversions in array.

6. You are given two sorted lists of size m and n. Devise an O(log m + log n) time algorithm for computing the kth smallest element in the union of the two list.

```
assume Array M has size m, and Array N has size n, search for kth smallest element

while true
  m <- length of M
  n <- length of N
  if m == 0 return N[k]
  if n == 0 return M[k]

  half_m = m/2
  half_n = n/2

  if (half_m + half_n) < k
    if M[half_m] > N[half_n]
```

```
      N <- N[(half_n+1) : n]
      k <- k - half_n - 1
    else
      M <- M[(half_m+1) : m]
      k <- k - half_m - 1
  else
    if M[half_m] > N[half_n]
        M <- M[0 : half_m]
    else
        N <- N[0 : half_n]
```

7. Given an array of numbers as the input, devise an algorithm to generate a random permutation of the array, such that each number has equal probability to be placed in each position in the output array.

```
for i from 0 to n-2
  swap n[i] & n[randomRange(i, n)]
```

randomRange(a, b) is a function which random generate integer between a and b-1.

For position 0, any element can be there,

For position 1, any element can be there execept the element which is seated at position 0.

For position 2, any element can be there execept elements which are in position 0 & 1.

etc...