Name :- karnam Nithin Faculty :- Dr. Senthil Kumar

Reg NO :- 21BAI1091 slot :- E2+TE2

Design and analysis of algorithms

PPS-1

1) **Given problem statement :-**

Consider sorting n numbers in array A by first finding the smallest element of A and exchanging it with the element in A[1]. Then find the second smallest element of A and exchange it with A[2]. continue in this manner for the first n-1 elements of A.

solution :-

The solution for the above mentioned problem statement is "selection sort".

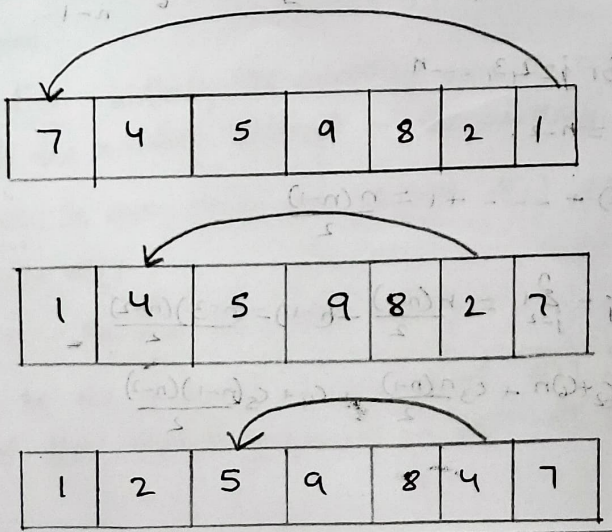
Pseudo code :-

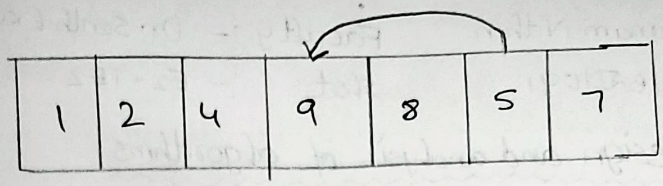
```
for i=0 to A.length-1
    minindex = i
    for j=i+1 to A.length
        if A[j] < A[minindex] and j≠minindex
            minindex = j
    swap A[i] with A[minindex]
```

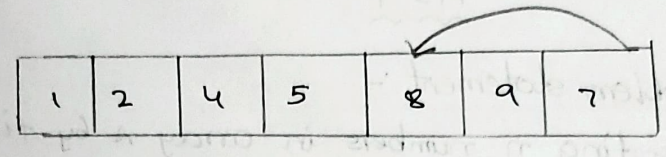
Loop invariant :-

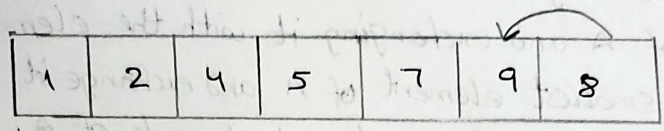
At the start of each iteration of the outer for loop of lines 1-6, the subarray A[1....i-1] consists of i-1 smallest elements of A, sorting in increasing order.

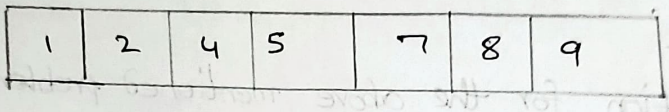
Simulation :-











Proof of correctness :-

→ For every iteration we will treat ith element as min element at first.

→ Later compare that element with elements which are rightside of it.

→ If we found any smaller element compared to it then swap that element with the ith element.

→ Do this iterations for n-1 elements.

→ arranging n-1 elements in their positions will automatically arrange the element in correct position.

Analysis :-

	cost	time
for i=0 to A.length-1	C_1	n
minindex = i	C_2	$n-1$
for j=i+1 to A.length	C_3	$\sum_{j=2}^{n} t_j$
if A[j] < A[minindex]	C_4	$\sum_{j=2}^{n} (t_j-1)$
minindex = j	C_5	$\sum_{j=2}^{n} (t_j-1)$
swap A[i] with A[minindex]	C_6	$n-1$

$t_j = (n-j+1)$ for $j = 2, 3, \ldots n$

$t_2 = n-1$, $t_3 = n-2$.

$$\sum_{j=2}^{n} t_j = (n-1)+(n-2)+\cdots+1 = \frac{n(n-1)}{2}$$

$$\sum_{j=2}^{n} (t_j-1) = \sum_{j=2}^{n} t_j - \sum_{j=2}^{n} 1 = \frac{n(n-1)}{2} - (n-1) = \frac{(n-3)(n-2)}{2}$$

$$T(n) = C_1(n+1) + (C_2+C_3)n + C_3\frac{n(n-1)}{2} + C_4 + C_5\frac{(n-1)(n-2)}{2}$$

For the best case, as the array is already sorted. So $c_5=0$. even with that the expression of $T(n)$ will be reduced to the form an^2+bn+c, i.e the algorithm will run at $\theta(n^2)$ time.

Comparing with insertion sort :-

→ For both the best case is sorted array, worst case is reverse sorted array but each of n element will be compared with rest of n-1 elements. so both same the similar time complexity i.e $\theta(n^2)$

2) Problem statement :-

Sorting n numbers stored in an array A, by repeatedly swapping the adjacent elements that are not in an increasing order, If A[i] and A[i+1] are such that A[i] > A[i+1], then A[i] and A[i+1] shall be swapped. Based on the approach described above, write an algorithm for arranging the given n numbers in an increasing order of numbers.

Solution :-

Bubble sort is the solution for the above mentioned problem statement,

Pseudocode :-

```
for i=0 to A.length-1
    for j=0 to A.length-1-i
        if A[j+1] < A[j]
            swap A[j] with A[j+1]
```

Proof of correctness :-

At the start of each iteration of the for loop of 2-4, the subarray A[j---n] consists of the elements originally in A[j---n] before entering the loop but possibly in a different order and the first element A[i] is the smallest among them.

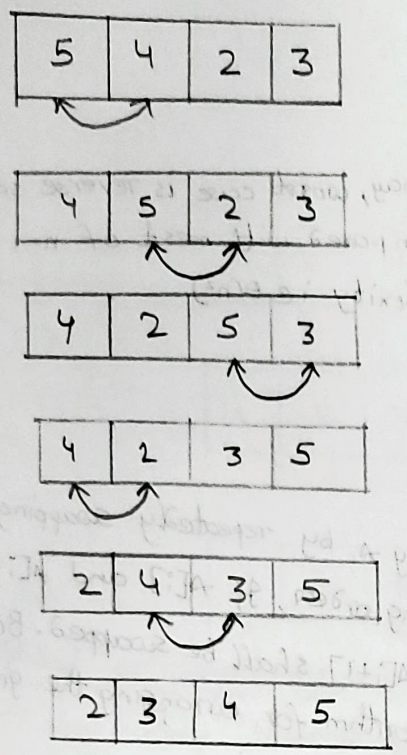
Initialization : initially the subarray contains only the last element A[n], which is trivially the smallest element of the subarray.

Maintainance : In every step we compare A[i] with A[j+1]. And we will swap if A[j+1] < A[i]

Termination :- The loop terminates when j = i

According to the statement of loop variant, A[i] is the smallest among A[i---n] consists of the elements originally in A[i---n] before entering the loop.

simulation :-



Analysis :-

	cost	time
for i=0 to A.length-1	c_1	n
for j=0 to A.length-i-1	c_2	n-i
if A[j+1] < A[j]	c_3	n-i-1
swap(A[j], A[j+1])	c_4	n-i-1

$T(n) = c_1 n + c_2(n-i) + c_3(n-i-1) + c_4(n-i-1)$

* This sorting algorithm contains two loops: inner, outer loops
* The inner loop performs O(n) comparisions deterministically

Best case :- The array is already sorted therefore c_4 doesn't exist or in other terms $c_4(n-i-1) = 0$. Therefore $T(n) = O(n^2)$

worst case :-

In worst case, the outer loop runs O(n) times, as a result the time complexity is $T(n) = O(n^2)$.

3) Given,

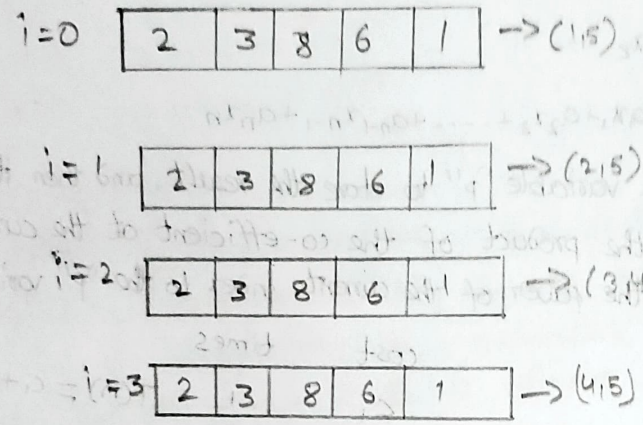
Let A[1,...n] be an array of n distinct numbers. If i<j and A[i]>A[j] then the pair is called an inversion of n,

eg :- Inversions of the array A=[2,3,8,6,1] are (1,5) (2,5) (3,4) (4,5) (3,5)

Pseudocode :-

```
count = 0
for i=0 to A.length-1
    j = n-1
    while j>i
        if A[i] > A[j]
            count += 1
        j = j-1
```

Simulation :-



Proof of correctness :-

→ Outer for loop runs n-1 times

→ In each iteration j starts from n-1 and while loop executes until j>i.

→ while in each "while" loop A[i] and A[j] elements are compared if A[i] > A[j] then we increase the count by 1.

→ Outer for loop terminates when i=n.

Analysis :-

	cost	times
for i=0 to A.length-1	c_1	n
j=n-1	c_2	$n-1$
while j>i	c_3	$\sum_{i=0}^{n-1}(n-i-1)$
if A[i] > A[j]	c_4	$\sum_{i=0}^{n-1}(n-i-1)$
count+=1	c_5	$\sum_{i=0}^{n-1}(n-i-1)$
j=j-1	c_6	$\sum_{i=0}^{n-1}(n-i-1)$

$$T(n) = c_1 n + c_2 n-1 + c_3 \sum_{i=0}^{n-1}(n-i-1) + c_4 \sum_{i=0}^{n-1}(n-1-i) + c_5 \sum_{i=0}^{n-1}(n-i-1) + c_6 \sum_{i=0}^{n-1}(n-i-1)$$

Best case :-

Best case is when the array is already sorted then line c_5 doesnot executes but eventhen two loops are executed so the order of complexity is $O(n^2)$.

Worst case:

Worst case is when the array is reverse sorted then every lines executes. The order of time complexity is $O(n^2)$.

comparision with insertion sort:-

In both inversion algorithm and insertion sort algorithm the best case is when the array is already sorted and the worst case is when the array is in reverse sorted order.

4) Given, $P(x) = \sum_{k=0}^{n} a_k x_k$

$= a_0x_0 + a_1x_1 + a_2x_2 + \cdots + a_{n-1}x_{n-1} + a_nx_n$

⇒ First initialize a variable "p" to store the result, and then iterates from 0 to n, adding the product of the co-efficient at the current index and x raised to the power of the current index to the "p" variable.

Pseudocode:-

```
                                cost    times
P ← 0                           c1      1
for i=0 to n                    c2      n+1
    P = p+a[i]*(x**i)           c3      n
return p                        c4      1
```

$T(n) = c_1 + c_2(n+1) + c_3n + c_4$

Simulation/Illustration:-

Let $P(x) = x^0 + 2x^1 + 3x^2 + 4x^3$

$x = 2$

initially $p = 0$

$i=0$: $p = 0 + 1 \times (x^0) \Rightarrow p = 1 \times 1 \Rightarrow p = 1$

$i=1$: $p = 1 + 2 \times (x^1) \Rightarrow p = 1 + 2 \times 2 \Rightarrow p = 5$

$i=2$: $p = 5 + 3 \times (x^2) \Rightarrow p = 5 + 12 \Rightarrow p = 17$

$i=3$: $p = 17 + 4 \times (x^3) \Rightarrow p = 17 + 32 \Rightarrow p = 49$

Horner's rule:-

$$P(x) = \sum_{k=0}^{n} a_k x^k = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} + a_nx^n$$

$$= a_0 + x(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n) \cdots))$$

It starts with the last co-efficient and multiplies it with x, and then adds the next coefficient from the end and so on.

Pseudocode :

```
p ← a[n]
for i = n-1 to 0
    p = p*x + a[i]
return p
```

cost	time
c_1	1
c_2	n
c_3	n-1
c_4	1

$T(n) = c_1 + c_2 n + c_3(n-1) + c_4$

Illustration :

Let $P(x) = x^0 + 2x^1 + 3x^2 + 4x^3$

$x = 2$

initially $p = 4$

$i = 3$; $p = 4 \times 2 + 3 = 11$

$i = 2$; $p = 11 \times 2 + 2 = 24$

$i = 1$; $p = 24 + 2 + 1 = 49$

Time analysis :-

Both codes took n iterations so time complexity of two codes are O(n). But Horner's method is more efficient than the iterative approach as it reduces the number of multiplications. while calculating power in iterative it took much time. Therefore Horners method is efficient compared to iterative approach.

6) Problem statement :-

Given a<b if last digit of a is less than last digit of b. we need to compare last digit of every number and sort the numbers.

Solution :-

we will use insertion sort to arrange n numbers in decreasing order.

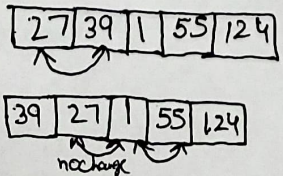
Pseudocode :-

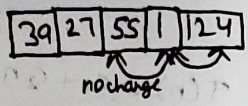
```
for j = 1 to A.length - 1
    key = A[j] % 10
    i = j - 1
    while i ≥ 0 and A[i] % 10 < key
        A[i+1] = A[i]
        i = i - 1
    A[i+1] = key
```

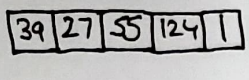
Illustration / simulation :-

input : 27 39 1 55 124

output : 39 27 55 124 1







got the output

Proof of correctness :-

→ The index j indicates the current card being inserted into the hand.

→ At the begining of each iteration of the for loop, which is indexed by j, the subarray consisting of elements A[1...j-1] constitutes the currently sorted hand, and the remaining subarray A[j+1,...n] corresponds to the pile of cards still on the table.

→ In fact, elements A[1...j-1] are the elements originally in positions 1 through j-1, but now in sorted order.

→ we state these properties of A[1...j-1] formally as a loop variant.

→ we use loop variations to help us understand why an algorithm is correct

Analysis :-

	cost	times
for j = 1 to A.length - 1	c_1	n
key = A[j] % 10	c_2	$n-1$
i = j - 1	c_3	$n-1$
while i ≥ 0 and A[i] % 10 < key	c_4	$\sum_{j=2}^{n} t_j$
A[i+1] = A[i]	c_5	$\sum_{j=2}^{n} (t_j - 1)$
i = i - 1	c_6	$\sum_{j=2}^{n} (t_j - 1)$
A[i+1] = key	c_7	$n-1$

Best case :-

→ Array is already sorted in decreasing order.

→ $T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$

$= (c_1 + c_2 + c_3 + c_4 + c_7)n - (c_2 + c_3 + c_4 + c_7)$

Worst case :-

→ Array sorted in increasing order.

→ $T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n-1)}{2} - 1\right) + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7(n-1)$

$= \left(\frac{c_4 + c_5 + c_6}{2}\right)n^2 - \left(c_1 + c_2 + c_3 + \frac{c_4 - c_5 - c_6}{2} + c_7\right)n - (c_2 + c_3 + c_4 + c_7)$

$T(n) = O(n^2)$