21L-7512 Abdullah Dar
DAA An no 1

Question no 1:

Loop Invariant

Code:
Find min

```
Minimum (A)
min = A[1]
for i=2 to A.length
  if min > A[i]
  min = A[i]
return min
```

Solution:

Loop invariant:

Before the execution of i^{th} iteration, the variable "min" has smallest element in sub Array from A[1 to i-1].

Initillization:

At initillization the value of i=2 and min has A[1] element within it from the sub array A[1 to 1]. So it stands to be correct.

Maintenance:

Before the execution of any i^{th} iteration our loop invariant holds true. and we have the minimum value from sub array A[1 to $(i-1)$]

During execution of i^{th} iteration, it will check if min variable has bigger value than A[i] then it will update the value of min to A[i] else nothing happens. and i gets incremented as $i = i+1$

So, before execution of $(i+1)^{th}$ iteration, there is smallest value in the min from sub Assay A[1 to i].

Termination:

At Termination $i = size+1$ where minimum contains the smallest element from subarray A[1 to size].

Question no 2:

Code:

Bubble Sort

```
Bubble Sort(A)
for i = 1 to A.length-1
    for j = A.length downto i+1
        if A[j] < A[j-1]
            exchange A[j] with A[j-1]
```

Loop invariant: (inner loop)

Before execution of j^{th} iteration the smallest element from sub Array A[j+1 to n] is at $(j+1)^{th}$ index

Initillization:

At initillization $j = n$, so smallest element from A[n+1 to n] is true.

Maintanence:

Before execution of j^{th} ~~cutiet~~ iteration our loop invariant holds.

During execution of j^{th} iteration. It will check if $A[j] < A[j-1]$ and swap the elements if it hold, true else nothing happens. After that j is changed into $j = j - 1$

So before execution of $(j-1)^{th}$ iteration, the min element from sub array A[j to n] is at j^{th} index.

Termination:

At end $j = i$ the smallest element from sub array A[j+1 to n] is at $(j+1)^{th}$ index.

Loop invariant: (outer loop)

Before the execution of i^{th} iteration All the elements with in array from A[1 to i-1] are sorted in ascending order.

Initillization:

At initillization $i = 1$ So all elements from A[1 to 1] are sorted.

Maintenance:

Before execution of any particular i^{th} iteration our loop invariant holds true.

During execution of i^{th} iteration the inner loop executes and after that it sorts 1 more element and array gets sorted from A[1 to i] in ascending order. After that i get updated to $i = i+1$

So, before the execution of $[i+1]^{th}$ iteration. All the elements in the sub array A[1 to i] are sorted in ascending order.

Termination:

At end $i = n$ are element are sorted in Ascending order from

A[1 to n].

Question no 3:

Code:

Selection Sort

```
SelectionSort(A)
for j = 1 to A.length - 1
    smallest_index = j
    for i = j+1 to A.length
        if A[i] < A[smallest_index]
            smallest_index = i
    exchange A[j] with A[smallest_index]
```

Loop invariant: (inner loop)

Before the execution of i^{th} iteration the smallest element from sub array A[j to i-1] is at A[smallest_index].
i.e smallest_index = j (outer loop)

Initillization:

At initillization i = j+1 and smallest_index = j, so the smallest from sub array A[j to j] is A[j] as only single element.

Maintenance:

Before the execution of any particular i^{th} iteration our loop invariant holds.

During the execution i^{th} iteration we first check A[k] is smaller than A[smallest-index] if its true then we update smallest_index to i else nothing happens. Now, A[smallest index] has smallest element from the sub array A[j to i]. After that, the value of i gets updated $i = i+1$.

So, Before the execulation of $(i+1)^{th}$ iteration, A[smallest_index] has the smallest element from the subarray A[j to i]

Termination:

At end, $i = size + 1$ so smallest element from sub array A[j to n] will be at A[smallest_index].

Loop invariant: (outer loop)

Before the execution of j^{th} iteration all the elements in the array from A[1 to j-1] are sorted via Ascending order.

Initillization:

At initillization $j = 1$, so all element of the array $A[1 \text{ to } 1]$ are sorted.

Maintenance:

Before the execution of j^{th} iteration our loop invariant hold true.

During the execution of j^{th} iteration when inner loop executes $A[smallest\_index]$ has smallest element from sub array $A[j \text{ to } size]$. Swapping it with $(j+1)^{th}$ element which gives us a sorted sub array to became $A[1 \text{ to } j]$ and $A[j+2 \ldots j+1]$

Before execution of $(j+1)^{th}$ iteration All the elements in the array $A[1 \text{ to } j]$ are sorted in ascending order.

Termination:

At termination $j = i$ the smallest element from array $A[i+1 \text{ to } n]$ is at $i+1^{th}$ index

Designing Algorithms

Question no 1:

→ Product of every other element

```
fun (arr1[], size)
{
1   arr2[size];
2   for  i = 1  to  size
3        arr2[i] = 1
4   for  j = 1  to  size
5        if ( i != j)
6        { arr2[i]  = arr1[j] * arr2[i]
         }
7        j = j+1
         }
8        i = i+1 }

9   return arr2;
}
```

Time complexity = $O(n^2)$

Question no2

Remove Duplicates in nlogn

```
Remove_Duplicates (A, size)
{
1   MergeSort (A, 1, size)

2   count = 0
3   for i = 1 to size - 1
4   { if (A[i] != A[i+1]) // not equal
5       count ++
    }
```

```
6       count ++
7    arr2[count]
8                    k = 1
9    arr2[1] = A[1]
10 for i = 2 to size.
11   if (A[i] != arr2[k]) // not equal
12   { arr2[k+1] = A[i]
13     k = k+1 }

14  return arr2
```

} T.C = n.logn + n ≈ nlogn

Time Complexity O(n logn)

Question no:3

Divide & Conquer in O(logn)

```
find same index (A , size)
{
1    left = 1
2    right = size
3    flag = false
```

```
4   while (right >= left)
5   {   mid = (left + right) / 2

6       if (mid == A[mid])
7           flag = true
8       else if (mid > A[mid])
9           left = mid + 1
10      else if (mid < A[mid])
11          right = mid - 1
12  return flag
}
```

Time Complexity = O(log n)

Question no 4

O(log m + log n) k^th smallest ele

```
Smallestele (A, B, k)  // k <= (m, n)
{
1   if (A.length == 0)
2       return B[k]
3   if (B.length == 0)
4       return A[k]

5   i = k/2;
6   j = k - i
```

```
7       check = k/4
8   while (check > 0)
    {
9       if (A[i-1] > B[j-1])
10      {   i = i - check
11          j = j + check
        }
12      else
        {
13          i = i + check
14          j = j - check
15      check = check/2
        }
    }
```

Time complexity $O(\log m + \log n)$
