Data Structure
Assignment (Sorting and BST)

Manoj Kumar
11911671
CSE

Q1) Algorithm

```
for (i=1; i<n ; i++)
{   temp = a[i];
      j = i-1;
    while (j >=0 && a[j] > temp)
    {     a[j+1]= a[j];      }
              j--;           }  — C1
    }
    a[j+1]= temp;          — C2
}
```

In worst case

i	while
1	1
2	2
⋮	⋮
n-1	n-1

total iteration = $1 + 2 + \dots (n-1)$

$= \frac{n(n-1)}{2}$

Total time = $C1 \times \frac{n(n-1)}{2} + C2 \times (n-1)$

$= O\left(C1\left(\frac{n(n-1)}{2}\right) + C2(n-1)\right)$

$\approx O(n^2)$

In best case
All the elements are sorted
Inner ~~loop~~ while condition will not be executed

$T = O(c_2 \times (n-1)) \approx O(n)$

Insertion is good for small set of data but for large data it times complexity is much bigger.

We can decrease time complexity to some extent by ~~t~~ increasing space complexity i.e. we can use bigger array or take an extra array for the task.

Q2) Bubble sort
It is an in-place sort algorithm

Algorithm

```
Step1 : start
step2 : ~~for~~ FOR I=0 to Array.size
step3 :     FOR J=0 to Array.size
Step4 :         IF Array [j+1] > Array [j]
                    swap ( Array [j+1] + Array [j] )
step 5:
Step6 :     ~~END~~ ~~INNER LOOP~~ STOP
```

Time complexity for Bubble sort in worst case is $O(n^2)$ as there are two ~~to nest~~ loops iterating from 0 to Array size

Quick sort
It is an in place sorting algorithm

Algorithm

Step 1: Start
Step 2: Choose any variable as pivot
Step 3: Take two variable to point left & right of list excluding pivot
Step 4: left point to low index
Step 5: Right point to right index
Step 6: while value at left is less than pivot move right
Step 7: while value at right is greater than pivot move left
Step 8: if both step 6 & 7 do not match swap left & right
Step 9: if left ≥ right, then point where they met is new pivot

Time complexity: worst case = $O(n^2)$
Average case = $O(n \log n)$

Merge sort is out place sort algorith
it's time complexity : $O(n \log n)$

Bubble sort & insertion sort have complexities $O(n^2)$

& quick sort has $O(n^2)$ for worst & for average - $O(n \log n)$

but Merge sort has $O(n \log n)$ for ~~both~~ every case
so it's complexity is better than above sorting algorithms

2) Merge sort Algorithm

Step 1: Find mid position of array & divide the array from mid point

Step 2: Divide until each sublist has 1 element

Step 3: Then merge sublist in sorted order

Q3)

```
Step 1:   create a structure of a node & create root node
Step 2:   create temp & temp= root
Step 3:   Take data from user
Step 4:   if ( temp = = NULL ) root= temp = newnode ~~[illegible] temp~~;
              FOR n inputs
Step 5:   while ( temp~~[illegible]~~ ! = NULL )
Step 6:   if ( newnode. data > ~~root~~ temp data )
Step 7               temp = temp → right;
Step 8:   else       temp = temp → left
Step 7:   end of while
Step 8:   END of FOR
Step 9:   END
```