Data Structure Assignment
Sorting and BST

Q1 Insertion sort time complexity for best case :

For best case the input is already sorted.
Only the outer loop is executed and no element is moved.
So the complexity is $O(n)$.

Modification to reduce the time complexity :

→ In normal insertion sort algorithm,

Time complexity of shifting elements : $O(n^2)$
Time complexity of comparing : $O(n \log n)$

Hence, complexity is $O(n^2)$
To improve complexity for shifting elements, the following can be done :

⇒ $2n$ spaces is taken in an array where 'n' is the total number of elements.
⇒ The insertion begins from $(n-1)^{th}$ position of the array
⇒ Finding the position of the elements to be inserted will be done using binary search.

Reference : https://www.ukessays.com/essays/computer-science/increasing-time-efficiency-insertion-6036.php

This reduces worst case complexity to $O(n \log n)$ from $O(n^2)$ at the cost of space.

Q2 Quick sort Algo :-

It a divide an conquer algo.

→ A pivot is selected.

→ Then the elements are divided into two parts around the pivot : the smaller half and the greater half.

→ This is repeated recursively with each part till only single elements are left.

→ Thus the smaller and greater halves are sorted and merged around the pivot.

Bubble Sort Algo :-

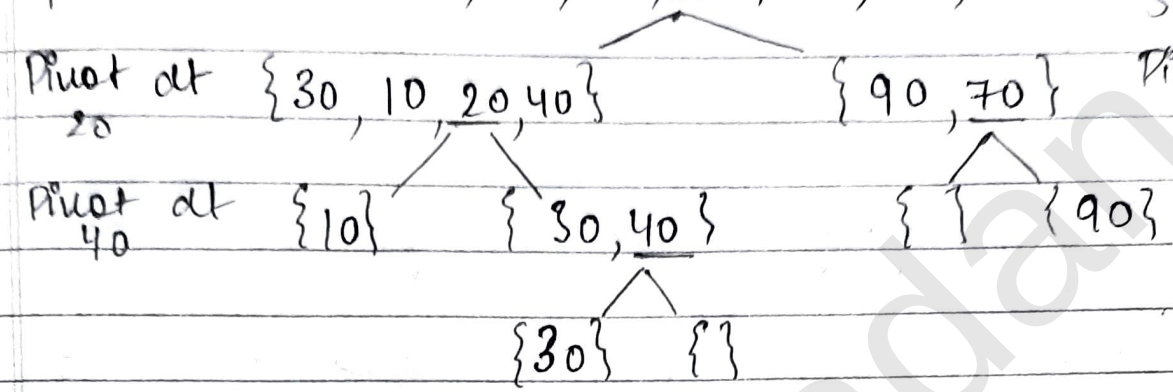
This algo works by swapping adjacent elements.

→ Outer loop runs (n-1) time with every iteration resulting in the largest element of the unsorted part getting placed at the end of the unsorted part.

→ The inner loop iterates over the unsorted part and swaps element by comparing each one with the next one. Thus the largest one reaches the end.

Example :

i) Quicksort : {30, 10, 90, 50, 40, 20, 70} Pivot at 50



ii) Bubble Sort : (5 1 4 2 8)

First pass :

(5 1 4 2 8) swap (1 5 4 2 8)
(1 5 4 2 8) swap (1 4 5 2 8)
(1 4 5 2 8) swap (1 4 2 5 8)
(1 4 2 5 8) no swap (1 4 2 5 8)

Second pass :

(1 4 2 5 8) no swap (1 4 2 5 8)
(1 4 2 5 8) swap (1 2 4 5 8)
(1 2 4 5 8) no swap (1 2 4 5 8)

Third pass :

(1 2 4 5 8) no swap (1 2 4 5 8)

Although the array is sorted, the loop continues.

~~Complexity~~

(i) Category :- (In place vs out of place)

(1) Quicksort :- As per the broad definition, quicksort qualifies as an in place algo as the space required for input and sorting does not vary. But it is not 'strictly in place' because extra space is used for recursive calls.

(ii) Bubble Sort :- It is strictly in place as the elements are just swapped and no extra space is utilized.

Complexity :-

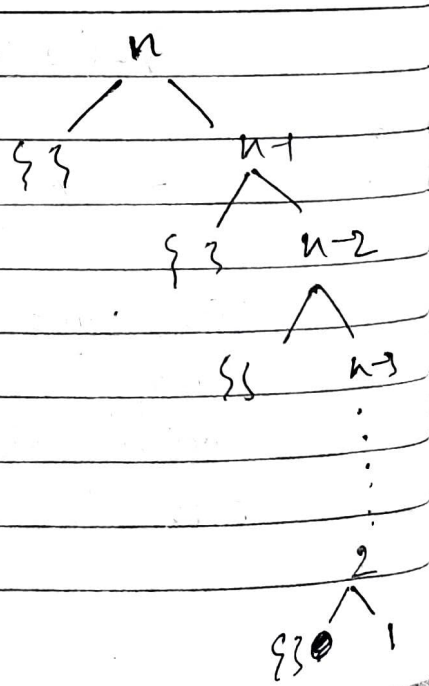
(i) Quick sort

→ Worst case :- when the pivot is either largest or smallest in every recursive call.

$$n+(n-1)+(n-2)+(n-3)+\ldots+2$$

$$\Rightarrow \frac{n(n+1)}{2} - 1$$

$$O(n^2)$$



Average case :- For the average case and best case the pivot is in between the elements and divides the greater and smaller half equally.

Complexity for Partition - O(n)

Complexity : $\frac{n}{} + \frac{n}{2} + \frac{n}{4} \ldots\ldots 2+1$

$: n\left(1 + \frac{1}{2} + \frac{1}{4} \ldots - \frac{1}{n}\right)$

$\Rightarrow O(n \log n)$

(ii) Bubble Sort :

→ For bubble sort the Worst case, Averge case and Best case complexity is same because the algo is non-adaptive and there is no change in running time with change in input.

$(n-1) + (n-2) + (n-3) \ldots + 2 + 1$

$\Rightarrow \frac{n(n-1)}{2} \rightarrow O(n^2)$

This could be modified by placing a counter for swapping in the inner loop. If the counter doesn't increase for one iteration of outer loop then the array is already sorted. So the complexity

for best case becomes, $O(n)$. And for Average case it "tends" to $O(n^2)$ rather that $O(n^2)$ strictly.

(iii) Merge Sort

→ Similar to quick sort it is an divide and conquer Algorithm. complexity is,

→ $n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots 2$

⇒ $n\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots \frac{2}{n}\right)$

⇒ $O(n \log n)$ for Best, Averge and Worst Case.

(iv) Insertion Sort.

Worst case : When the array is in reverse order. So there are $(n-1)$ comparisons and $(n-1)$ swaps in the first iteration followed by $(n-2)$ comparisons and swaps and so on, Complexity is :

⇒ $2((n-1)+(n-2)+(n-3)+(n-4)\dots+2+1)$

⇒ $2\left(\frac{n(n-1)}{2}\right)$

⇒ $O(n^2)$