Page No.
Date

Assignment 2

Q-1) Perform Analysis on time complexity of insertion sort algorithm in best case. Can you suggest few modification in order to reduce time complexity of this algorithm from $O(n^2)$ to some lower order term.

Sol^n. Insertion sort performs two operations.

a) It scans through the list, comparing each pair of elements.

b) and it swaps elements if they are out of order. ~~e~~

Each operation contributes to the running time of the algorithm. If the input is already in sorted order, insertion sort compares $O(n)$ elements and performs no swaps. Therefore, in the best case, insertion sort runs in $O(n)$ time.

The worst and average case analysis, the insertion sort algorithm first scan through the list, comparing each pair of elements and it swaps elements if they are out of order, Therefore the time complexity in worst case will be $O(n^2)$

$2 \times (1+2+ --- +n-1)$

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

Page No.
Date

We can use binary search to reduce the number of comparisons in normal insertion sort. Binary insertion sort uses binary search to find proper location to insert the selected item at its right position.

In normal sort it takes $O(n^2)$ time complexity But using binary search in insertion sort it reduces its time complexity to $O(\log n)$.

If the array is already sorted

2	5	8	7	9

then the insertion sort algorithm only compares the elements and do not swap so only O(n) time complexity is there.

Q-2) Giving suitable example discuss Quick sort and Bubble sort algorith and its category (in place or out of place sorting algorithm). Discuss its time complexity of and compare it with merge and insertion sort. Also make program of each algorithms.

Sol^n) **Quick sort :-**

It is a divide and conquer algorithm that uses recursion to perform its sorting. The main feature of quick sort is the selection of a pivot point. It is used to partition the array. The purpose of partitioning process is to move items that are on the wrong side of the pivot value.

a) Take two variables to point left and right of the list excluding pivot.

b) left points to low index and right to high index.

Page No.
Date

c) while value at left is less than pivot move right.
d) while value at right is greater than pivot move left.
e) if both step (c) and (d) does not match swap left and right
f) If left ≥ right, the point where they met is new pivot.

<u>Quick sort</u> is in-place sorting algorithm, because it takes O(logn) space in the stack to keep track of the subarrays in its divide and conquer strategy.

<u>example</u> :- In array {52, 37, 63, 14, 17, 8, 6, 25}

we take 25 as pivot. So after first pass, the list will be
{6, 8, 17, 14, 25, 63, 37, 52}

Now 6, 8, 17, 14 and 63, 37, 52 are considered as two separate subarrays and same logic is called recursively.

<u>Bubble sort</u> :- It starts by comparing two elements to each other to see which is larger. If the first element is larger than the second, the swap the elements. This continues until all items in the array have been inspected and the largest value is moved to the last. The algorithm repeats the entire process until the array is fully sorted.

example :- original list [90, 50, 10, 20, 70, 60]

Iteration 1 : [50, 10, 20, 70, 60, 90]

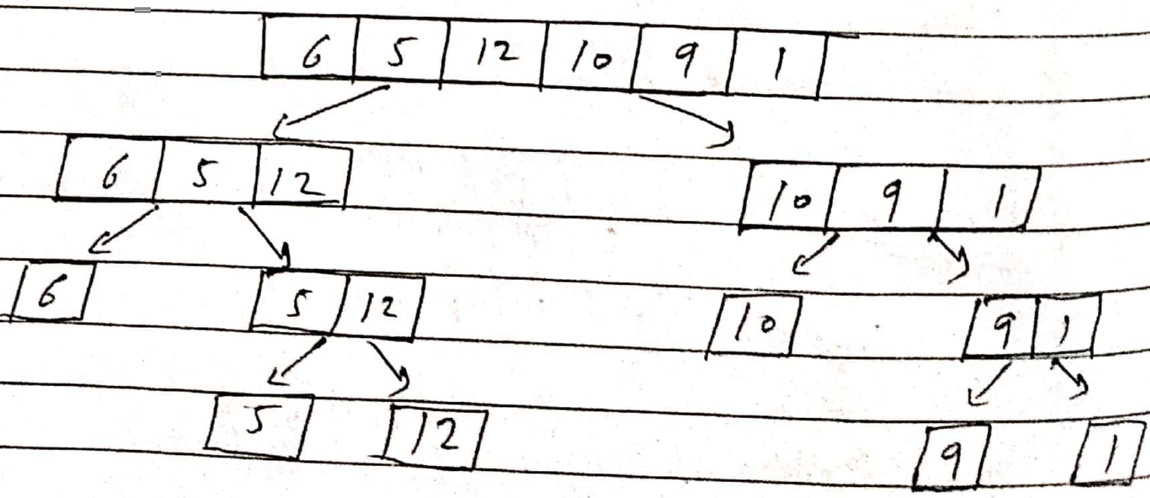
Iteration 2 : [10, 20, 50, 70, 60, 90]

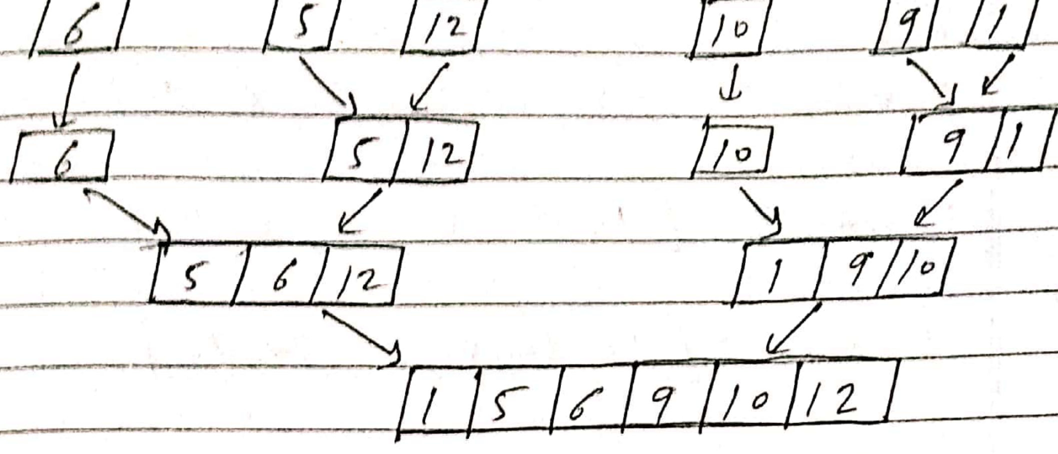
Iteration 3 : [10, 20, 50, 60, 70, 90]

The array is sorted.

Bubble sort is also in-place sorting algorithm because no extra space is required.

Merge sort :- In merge sort the array is repeatedly divided into two halves until we reach a stage where we try to perform merge sort on a subarray of size 1. After that, the merge function comes into play and combines the sorted arrays into larger arrays until the whole array is merged.



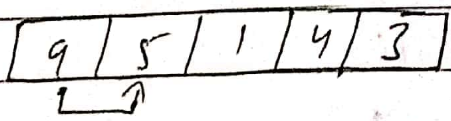


Since, it requires O(n) additional space to perform merge sort it is an out of place algorithm.

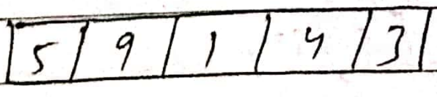
Insertion sort :-

In this sorting, the first element in the array is assumed to be sorted. Take the second element and store it separately in key. Compare key with the first element. If the first element is greater than key, then key is placed in front of the first element. Then repeat this algorith on third, fourth and upto ~~second~~ last element of array.

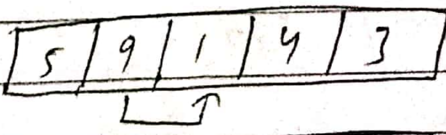
example :-

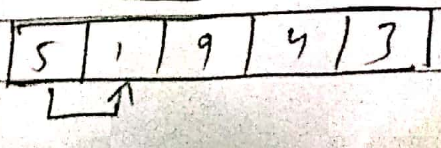


Step 1:-



Step2:- ~~Step 2~~





1	5	9	4	3

Step 3 :-

1	5	9	4	3

1	5	4	9	3

1	4	5	9	3

Step 4 :-

1	4	5	9	3

1	4	5	3	9

1	4	3	5	9

1	3	4	5	9

Sorted array.

<u>Time Complexity</u> :-

<u>Bubble sort</u> :-

Worst Case = $O(n^2)$
Best Case = $O(n)$
Average Case = $O(n^2)$

<u>Merge sort</u> :-

Worst case = $O(n \log(n))$
Best case = $O(n \log(n))$
Average Case = $O(n \log(n))$

Page No. ______
Date

Quick sort :-

Worst Case = $O(n^2)$
Best Case = $O(n \log(n))$
Average Case = $O(n \log(n))$

Insertion sort :-

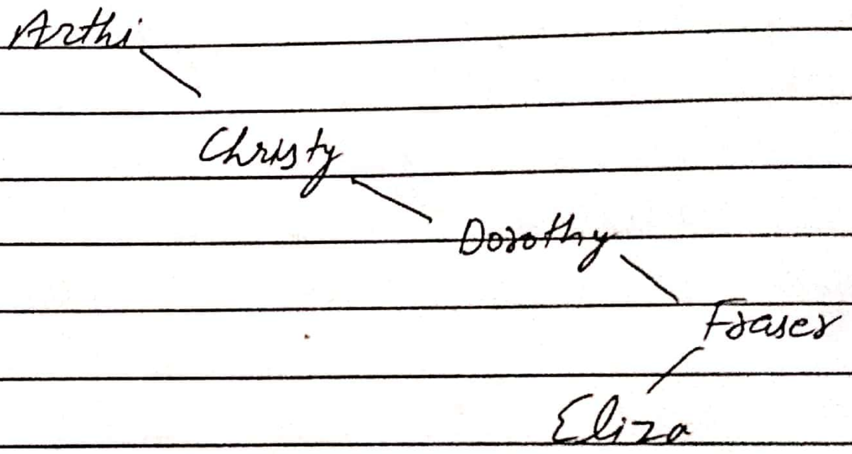
Worst Case = $O(n^2)$
Best Case = $O(n)$
Average case = $O(n^2)$

Basis for comparison	Quick sort	Merge sort	Bubble sort	Insertion sort
Worst case complex.	$O(n^2)$	$O(n \log(n))$	$O(n^2)$	$O(n^2)$
Works well on	Small array	operates fine on any size	small array	small array
Speed of execution	It works faster than other for small data set	Consistent speed on any size	slow	It works faster for small array
type	In-Place	out of place	In Place	In-Place

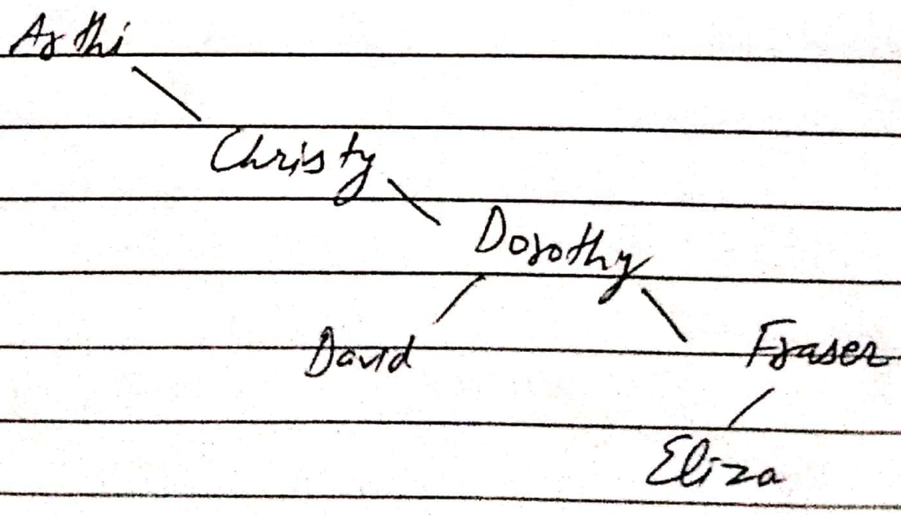
Q-3.

Binary Search Tree of string.

Arthi , Christy, Dorothy , Fraser, Eliza



After inserting 'David'


Arthi
Christy
Dorothy
David
Fraser
Eliza