TUTORIAL-3

Q1] Write linear search pseudocode to search an element in a sorted array with minimum comparisons.

Ans

```
int Lsearch_mincomparsion(int a[], int n, int k)
{
    mid = (0+n-1)/2;
    if(k > a[mid])
      i= mid+1;
    else
      i= mid-1;

    for( ; i<n; i++)
    {
        count++;
        if (a[i]==k)
        {
          cout<<"Yes found" << i << "comparsions="
                    << count;
        } ~~break~~ return;
}

cout<<" NOT found";
}
```

Q2] Write pseudo code for iterative and recursive insertion sort. Insertion sort is called online sorting why? what about other sorting algorithm that has been discussed in lectures?

Ans Iterative Approach

```
void insertionSort (int arr[], int n)
{
    int i, j, tmp;
    for (i=1; i<n; i++)
    {
        tmp = arr[i];
        j = i-1;
        while (j>=0 && arr[j] > tmp)
        {
            arr[j+1] = arr[j];
            j = j-1;
        }
        arr[j+1] = tmp;
    }
```

⟹

Recursive Approach

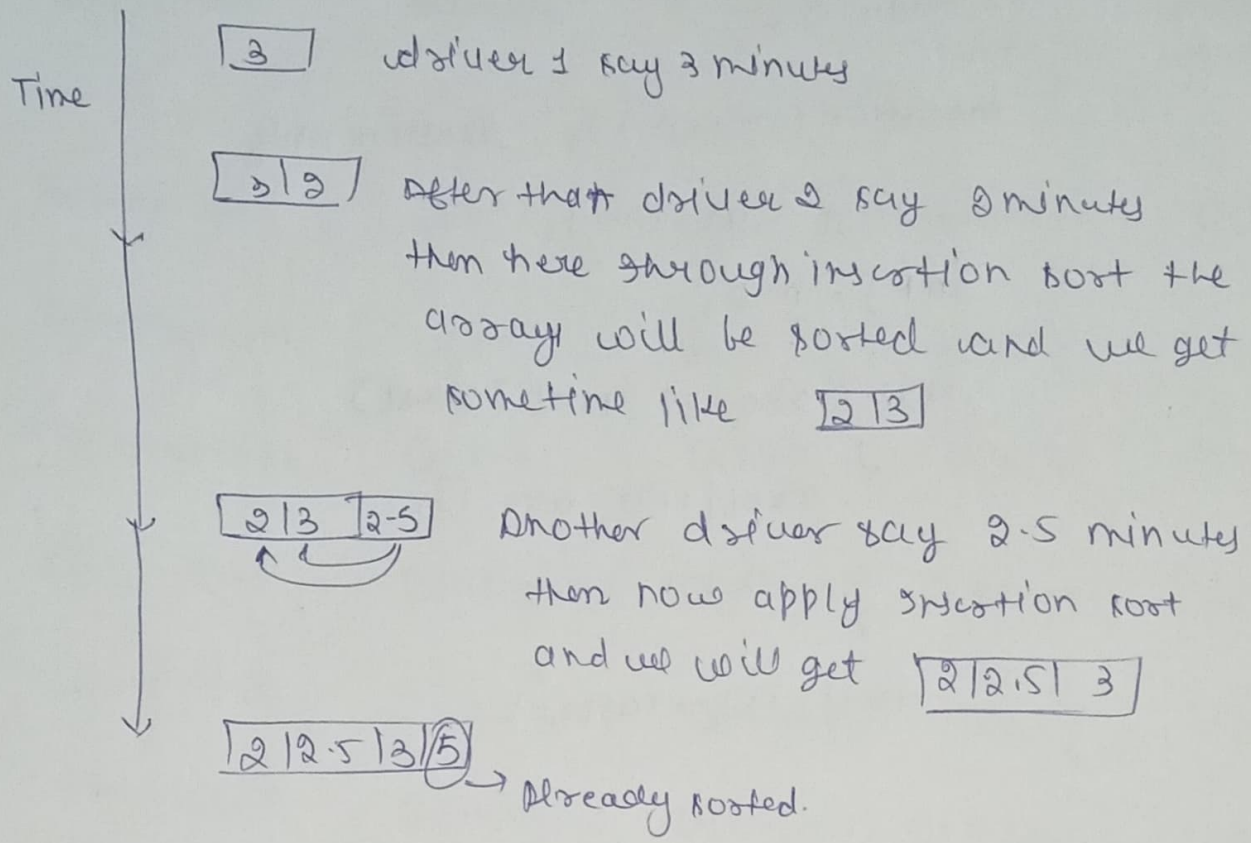
```
void insertion (int arr[], int n)
{
    if(n <= 1)
        return;

    insertion(arr, n-1);   //calling itself
    int last = arr[n-1];
    int j = n-2;

    while(j>=0 && arr[j] > last)
    {
        arr[j+1] = arr[j];
        j--;
    }
    arr[j+1] = last;
}
```

Online Sorting - If Sorting algorithm can sort the data as it arrives over period of time and not at the very begining then Sorting algorithm is known as Online Sorting algorithm.

Yes Insertion Sort is known as Online Sort Now let us take a example and understand how it do the task.

For example- I need a cab say uber and I open up the uber app and sent the request for the cab

So the drivers that are near by me and are free accepts my request and set a confirmation to me and also they provides me the time-line that how much time they will take to reach me.



Hence by this example we surley say that as soon as data comes insertion sort is able to sort that & that's why insertion sort is known as online sort.

Q3| Complexity of all the sorting algorithm that has been discussed in lectures.

Ans

Algorithm	Best case	Average case	worst case	space worst case
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Quick sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(n)$
Heap sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$

Q4| Divide all the sorting algorithm in inplace / stable / online sorting.

Ans

Algorithm	stable	online	inplace
Bubble sort	✓	✓	✓
Selection sort	×	✓	✓
Insertion sort	✓	✓	✓
Merge sort	✓	×	×
Quick sort	×	×	✓
Heap sort	×	✓	✓

Q5| Write ~~recurrence relation~~ pseudo code for binary search. what is Time and space complexity of Linear search and Binary search (Iterative/ Recursive)

Ans Iterative search

```
Bsearch (int s, int e, int a[], int key)
{
    while (s<=e)
    {
        int mid = (s+e)/2;
        if (a[mid] == key)
            return mid;
        if (arr[mid] < key)
            s = mid+1;
        else
            e = mid-1;
    }
    return -1;
}
```

Recursive Approach

```
Bsearch(int s, int e, int arr[], int key)
{
    if (s>=e)
        return -1;
    int mid = (s+e)/2;
    if (arr[mid] == key)
        return mid;
    if (arr[mid] < key)
    return Bsearch(mid+1, e, arr, key);
    else
      Bsearch(s, mid-1, arr, key);
}
```

Linear Search

Time complexity - O(n) Space complexity - O(1)

Binary Search

Time complexity - O(logn) Space complexity - O(1)

Recursive approach

Space complexity - O(logN) Time complexity - O(logn)

Q6| Write occurrence relation for binary recursive search.

Time complexity is $O(\log n)$

Recurrence relation $T(n) = T(n/2) + 1$

Derivation

$T(n) = T(n/2) + 1$

$T(n/2) = T(n/4) + 1$

$T(n) = T(n/4) + 2$

$T(n/4) = T(n/8) + 1$

$T(n) = T(n/8) + 2 + 1$

!

$T(n) = T\left(\frac{n}{2^k}\right) + k$ times.

$\frac{n}{2^k} = 1 \rightarrow n = 2^k$

$\log n = k \log 2$

$\log n = k$

$T(n) = T\left(\frac{n}{2^{\log n}}\right) + \log n$

$= T(1) + \log n$

$= 1 + \log n$

$= \log n$ Ans.

Q7) Find two indexes such that A[i] + A[j] = K in minimum time complexity.

Ans

```
void find-indexes(int a[], int n, int k)
{
    int l, r;
    sort(a.begin(), a.end());
    l = 0;
    r = n-1;
    while(l<r)
    {
      if (a[l] + a[r] == k)
        {
          cout<< a[l] << " " << a[r];
          cout<< "indexes" << l << " " << r;
            return;
        }
      else if (a[l] + a[r] < sum)
            l++;
      else
            r--;
    }
    cout<< "No two elements sum equal to
            third";
}
```

Q8| which sorting is best for practical use? explain

Ans Quick sort is the fastest general-purpose sort. In most practical situations, quick sort is the method of choice. If stability is important & space is available

Q9| what do you mean by no. of inversions in an array? count the no. of inversions in Array arr[]: {7, 21, 31, 8, 10, 1, 20, 6, 4, 5} using merge sort.

Ans Inversion count for an array indicates - how far (or close) the array is from being sorted. If the array is already sorted, then the inversion count is 0, but if array is in the reverse ordering the inversion count is the maximum.

Q.

Two elements a[i] & a[j] form an inversion if a[i] > a[j] & i < j.

7	21	31	8	10	1	20	6	4	5
0	1	2	3	4	5	6	7	8	9

7	21	31	8	10

1	20	6	4	5

7	8	10	21	31
0	1	2	3	4

1	4	5	6	20
5	6	7	8	6

i = ~~0~~ ~~1~~ 2

Mid = 5

j = ~~5~~ ~~6~~ ~~7~~ 8

inversion count = ~~0~~ ~~5~~ ~~10~~ ~~15~~ ~~20~~ 22

when a[i] compared with a[j] then we find that a[i] > a[j] so we increase the inversion count by mid-i then we will increment j by 1 Now again check still same then increase inversion count by mid-i & inc. j still same now j = 7 Now again check still same j=8 after them j=9 and now a[i] is not less than a[j] then increase a[i] by 1 Now a[i] will become I at i = 3 ~~will~~ we will get the condition where a[i] > a[j] then increase inversion count.

So inversion count value = 22

Q10) In which case Quick sort will give the best and worst case complexity?

Ans) The worst case complexity of Quick sort is $O(n^2)$ & it occurs when the picked pivot element which is an extreme (smallest or largest) element. This happens when i/p array is sorted or reverse sorted & either first or last element is picked as pivot.

The Best case time complexity of Quick sort is $O(n \log n)$. And it occurs when the partition process always picks the middle element as pivot.

Q11) Write Recurrence Relation of merge and Quick sort in best & worst case? what are the similarities & differences blw complexities of 2 algo and why?

Quick sort:

Best :- $T(n) = 2T(n/2) + n$

Worst :- $T(n) = T(n-1) + n$

Merge sort

$T(n) = 2T(n/2) + n$

In merge sort the array is divided into 2 equal parts

n times T.C. = O(nlogn)

In quick sort the array is divided int any ratio depending on the partion of pivot element

∴ Time complexity varies from O(n²) to O(nlogn).

Q12) Selection sort is not stable by default but can you write a version of stable selection sort.

Ans

```
void stableSelectionSort(int a[], int n)
{
    for(int i=0; i<n-1; i++)
    {
        int min = i;
        for(int j=i+1; j<n; j++)
            if(a[min] > a[j])
                min = j;
        int key = a[min];
        while(min>i)
        {
            a[min] = a[min-1]
            min--;
        }
        a[i] = key;
    }
}
```

Q13) Bubble sort scans whole array even when array is sorted. can you modify the bubble sort so that it doesn't scan the whole array is once sorted.

Ans)

```
void BubbleSort(int arr[], int n)
{
    int swap = 0;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n-i-1; j++)
        {
            if (arr[j] > arr[j+1])
            {
                swap(arr[j], arr[j+1])
                swap = 1;
            }
        }
        if (swap == 0)
        break;
    }
}
```

Q (4) Your computer has a RAM (Physical memory) of 2 GB and you are given an array of 4 GB for sorting. Which algorithm you are going to use for this purpose & why? Also explain the concept of external & internal sorting.

Ans We can do this by external sorting.
We will divide our source file into temporary file of size equal to the size of the RAM and first sort these files.

If the data sorting process takes place entirely within the Random-Access memory (RAM) of a computer, it's called internal sorting. This is possible whenever the size of the data set to be sorted is small enough to be held in RAM.

For sorting larger datasets, it may be necessary to hold only a smaller chuck of data in memory at a time, since it won't all fit in the RAM. The rest of the data is normally held on some larger, but slower medium, like a hard disk. The sorting of these large datasets will requires different set of algorithms which is called external sorting.