Tutorial-3

1. Write linear search pseudocode to search an element in a sorted array with minimum comparisions.

→
```
while(low<=high)
{
    mid=(low+high)/2;
    if(arr[mid]==key)
        return true;
    else if(arr[mid]>key)
        high=mid-1;
    else
        low=mid+1;
}
return false;
```

2. Write pseudocode for iterative and recursive insertion sort. Insertion sort is called online sorting. Why? What about other sorting algorithms that has been discussed in lectures?

→ Iterative insertion Sort:

```
for(int i=1; i<n; i++) {
    j=i-1;
    x=A[i];
    while(j>-1 && A[j]>n) {
        A[j+1]=A[j];
        j--;
    }
    A[j+1]=n;
}
```

Recursive insertion sort:

```
void insertion_sort(int ar[], int n) {
    if(n<=1)
        return;
    insertion_sort(ar,n-1);
    int last=ar[n-1];
    j=n-2;
```

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

Insertion sort is called online sorting as whenever a new element comes, insertion sort define its right place.

3. Complexity of all sorting algorithms that has been discussed in lectures.

→ Bubble sort - $O(n^2)$
Insertion sort - $O(n^2)$
Selection sort - $O(n^2)$
Merge sort - $O(n \log n)$
Quick sort - $O(n \log n)$
Count sort - $O(n)$
Bucket sort - $O(n)$

4. Divide all the sorting algorithms into inplace/stable/online sorting.

→ Online sorting - Insertion sort
Inplace sorting - Bubble sort, Insertion sort, Selection sort
Stable sorting - Bubble sort, Insertion sort, Merge sort

5. Write recursive/iterative pseudo code for binary search. What is the time and space complexity of linear and binary search (Recursive and Iterative).

→ Iterative binary search:

```
while (low <= high) {
    int mid = (low + high) / 2
    if (arr[mid] == key)
        return true;
    else if (arr[mid] > key)
        high = mid - 1;
```

```
    else
        low = mid + 1;
}
```

Recursive binary search:

```
while (low <= high) {
    int mid = (low + high) / 2
    if (arr[mid] == key)
        return true;
    else if (arr[mid] > key)
        binary-search(arr, low, mid-1);
    else
        binary-search(arr, mid+1, high);
}
return false;
```

- Binary search time complexity = O(log n)
- Binary search space complexity:
  - iterative = O(1) - recursive = O(log n)
- linear search time complexity = O(n)
- linear search space complexity = O(1)

6. Write recurrence relation for binary recursive search.

→ $T(n) = T(n/2) + T(n/2) + C$

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

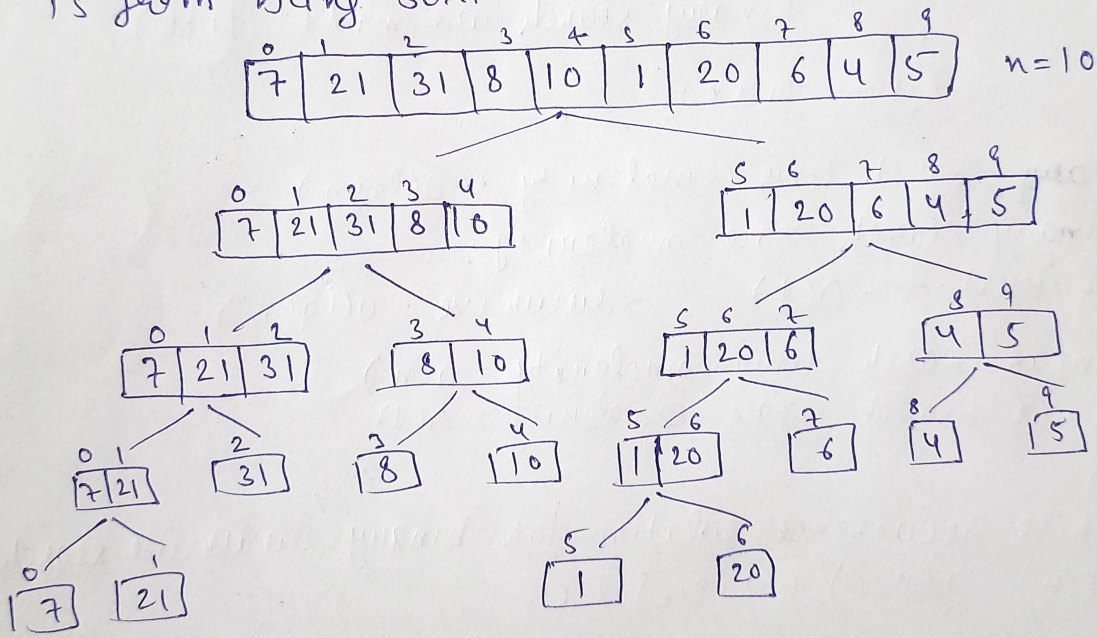
→
```
map<int, int> m;
for (int i=0; i< arr.size(); i++) {
    if (m.find(target - arr[i]) = m.end())
        m[arr[i]] = 1;
    else {
        cout << i << " " << m[arr[i]];
    }
}
```

8. Which sorting is best for practical uses? Explain.

→ Quick sort is fastest general purpose sort. In most practical situation, quick sort is the method of choice. If stability is important and space is available, merge sort might be best.

9. What do you mean by number of inversions in an array? Count the number of inversions in Array arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5} using merge sort.

→ Inversion indicates how far or close the array is from being sorted.



Inversions = 31

10. In which cases Quick sort will give the best and the worst time complexity?

→ Worst case → occurs when the pivot is always an extreme (smallest or largest) element. This happens when i/p array is sorted or reverse sorted and either first or last element is picked as pivot.

$O(n^2)$

Best case → occurs when pivot element is middle or near to middle.

$O(n \log n)$

11. Write recurrence relation of Merge and Quick sort in best and worst case? What are the similarities and differences between complexities of two algorithms and why?

→ Merge sort: $T(n) = 2T(n/2) + O(n)$

Quick sort: $T(n) = 2T(n/2) + n+1$

Basis	Quick sort	Merge sort
• Partition	Splitting is done in any ratio	array is parted into just 2 halves
• Works well on	smaller array	any size of array
• Efficient	inefficient for larger array	more efficient
• Sorting Method	Internal	External
• Stability	Not stable	Stable

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

→ Selection sort can be made stable if instead of swapping, the minimum element is placed in its position without swapping i.e. by placing the number in its position by pushing every element one step forward.

```
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;
}
```

13. 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 and why? Also explain the concept of External & Internal Sorting.

→ The easiest way to do this is to use external sorting (Merge sort). We divide our source file into temporary files of size equal to the size of the RAM & first sort these files.

- External Sorting → If the input data is such that it cannot be adjusted in the m/m entirely at once, it needs to be sorted in a hard disk, floppy disk or any other storage device. This is called external sorting.
- Internal Sorting → If the input data is such that it can be adjusted in the main m/m at once, it is called internal sorting.