Tutorial-3

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

Ans1)
```
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;
```

Q2) Write pseudo code for iterative and recursive insertion sort. Insertion Sort is called online sorting. Why ? What about other sorting algorithms that has been discussed in lectures.

Ans2) Iterative Insertion Sort -
```
for (int i=1 ; j< n; i++)
{
   j= i-1;
   x = A[i];
   while(j> -1 && A[j] > n)
   {
     A[j+1] = A[j];
     j--;
   }
   A[j+1] = n;
}
```

Insertion Sort is online sorting because whenever a new element come, insertion sort define its right place.

Recursive Insertion Sort -
```
Void insertion Sort (int arr[], int n)
{ if (n <=1)
    return;
  insertionsort(arr, n-1);
  int last = arr[n-1];
  j = n-2;
  while(j>=0 && arr[j] > last)
  { arr[j+1] = arr[j];
  {  j--;
```

Ans3) 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 - $O(n)$

Q4) Divide all the sorting algorithms into inplace/stable/online sorting.

Ans4) Online Sorting - Insertion Sort

Stable Sorting - Merge Sort, Insertion Sort, Bubble Sort.

Inplace Sorting - Bubble Sort, Insertion Sort, Selection Sort.

Ans5) Recursive ~~Iterative~~ Binary Search -

$O(\log n)$

```
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;
```

Iterative Binary Search -

$O(\log n)$

```
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;
}
```

Ans6) $T(n) = T(n/2) + T(n/2) + C$

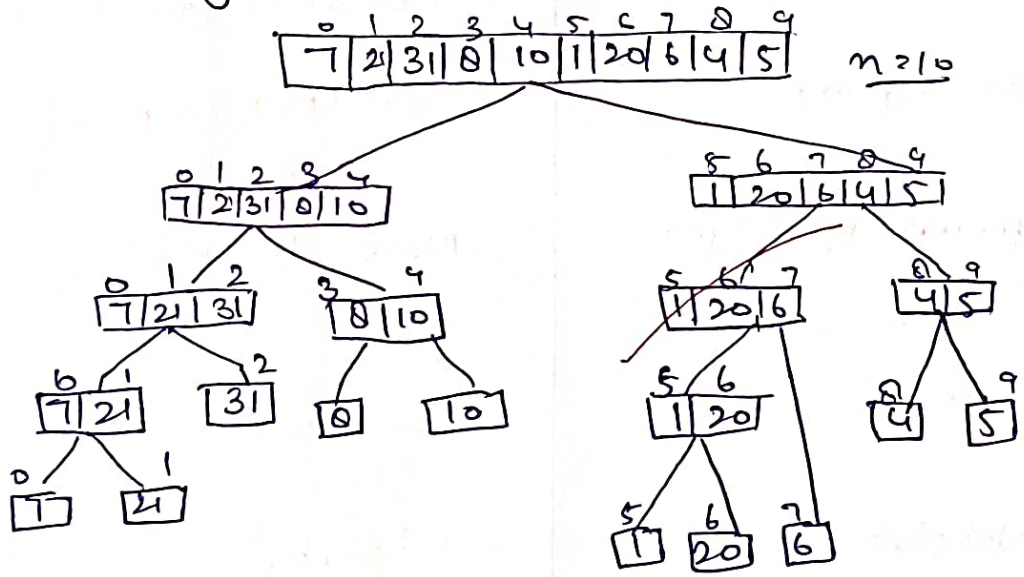
Ans 7)

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

Q8) Which sorting is best for practical uses? Explain

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

Ans 9) Inversion indicates - how far or close the array is from being sorted.



Worst case - The worst case occurs when the picked pivot is always an extreme (smallest or largest) element. This happens when input 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 the middle element or near to the middle element.

$O(n \log n)$.

Ans. 11 Merge Sort - $T(n) = 2T(n/2) + O(n)$

Quick Sort - $T(n) = 2T(n/2) + n + 1$

Basis	QuickSort	Merge Sort
Partition	Splitting is done in any ratio.	array is parted into just 2 halves.
Works well on	Smaller array	Fine on any size of array.
Additional space	Less (in place)	More (Not in-place)
Efficient	Inefficient for larger array	More efficient.
Sorting method	Internal	External
Stability	Not stable	Stable

Ans 14) We will use Merge Sort because we can divide the 4 GB data into 4 Packets of 1 GB and Sort them seperately and combine them later.

* Internal Sorting - all the data to Sort is stored in memory at all times while sorting is in progress.

* External Sorting - all the data is stored outside memory and only loaded into memory in small chunks.