Neeraj Singh Bhandari 03 Sec - H

Tutorial-3

Ans 1

```
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;
```

Ans 2 Interative Insertion Sort

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

Recursive Insertion Sort

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

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

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 - $O(n)$

Ans4 Online sorting :- Insertion Sort
Stable Sorting :- Merge Sort, Insertion sort, Bubble sort
Inplace sorting :- Bubble sort, Insertion sort, selection sort

Ans5 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;
}
```

T.C => $O(\log n)$

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
```

T.C = $O(\log n)$

Ans 6 $T(n) = T(n/2) + T(n/2) + C$

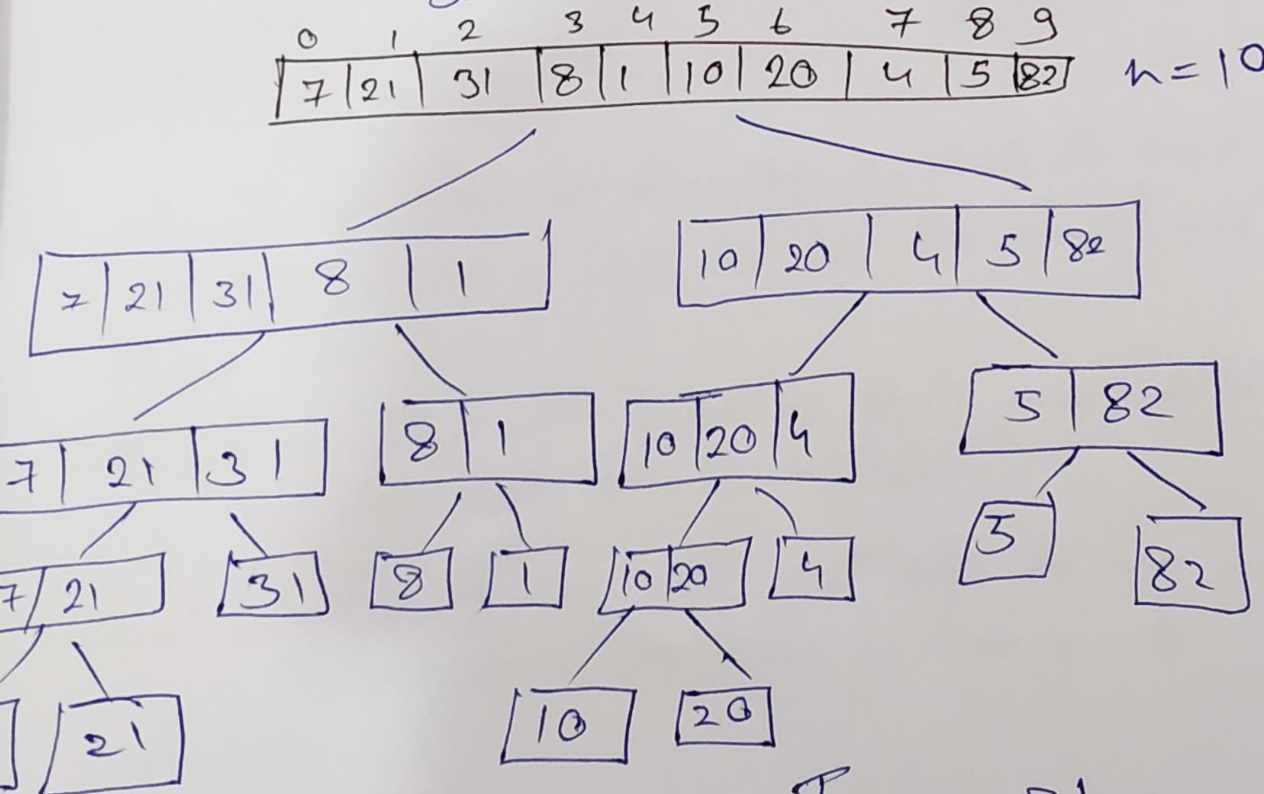
Ans 7

```
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]];
    }

  }
```

Ans 8 Quicksort is the fastest general purpose sort. In most practical situation, quicksort is the method of choice. If stability is important and space is available, merge sort might be best

Ans 9

Inversion Indicators - How far or close the array is from being sorted



Inversion = 31

Ans 10 <u>Worst Case</u>:- The worst case occurs when the picked pivot is always an extreme (smallest or largest) element. i.e. the array is already ~~the~~ sorted or reverse order, and either first or last element is picked as pivot. Then the Complexity is $O(n^2)$

<u>Best Case</u>:- Best Case occurs when pivot element is the middle element or near to the middle element

$TC = O(n \log n)$

Ans 11 Merge Sort $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$

Quick Sort = $T(n) = 2T\left(\frac{n}{2}\right) + n + 1$

Basis	Quick sort	Merge Sort
Partition	spliting is done in any ration	array is Partition into just 2 half by calcuting middle element
Works well on	smaller array	Fine on any size of array.
Additional space	Less (inplace)	More (Not-Inplace
Efficient	inefficient for large array	More efficient
Sorting Method	External	External
Stablility	Not Stable	Stable

Ans 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 time while sorting is in progress
- External Sorting:- all the data is ~~~~ stored outside the memory and only loaded into memory by dividing into Smaller part.