Design and Analysis of Algorithm

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;
```

Answer-2) Iterative Insertion Sort.

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

Recursive insertion sort

```
void insertion_sort(int arr[], int n){
        if (n<=1)
              return;
        insertion_sort(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 called as online sorting because whenever a new element came, insertion sort defines its right place.

Ans-3)

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 → $O(n)$

Ans-4)

Online Sorting ⇒ Insertion sort

Stable sorting ⇒ Merge sort, Insertion sort, Bubble sort

Inplace Sorting ⇒ Bubble sort, Insertion sort, selection sort.

Ans-5) 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 - Binary_Search(~~int~~ arr[], int low, int high){

```
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)

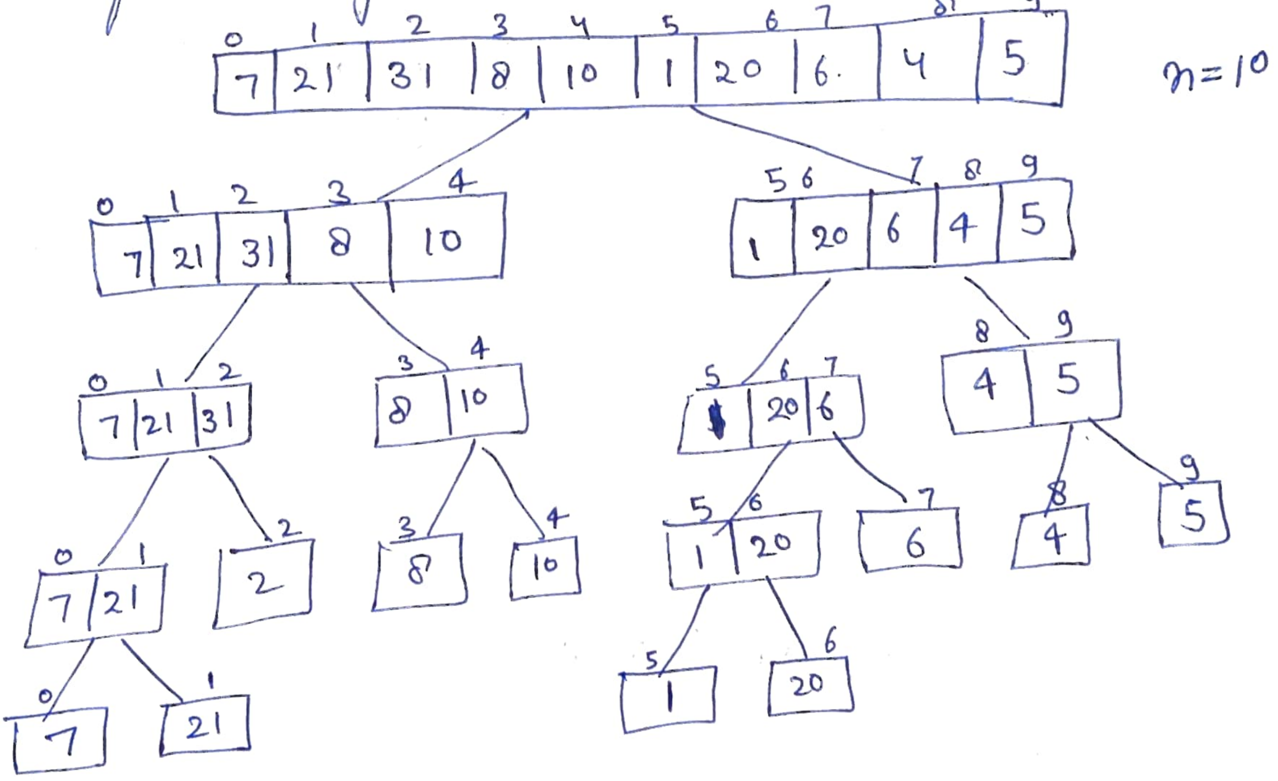
Ans-6)

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

Ans-7)
```
map < int, int> m;
for (int i=0 ; i< ar.size(); i++) {
        if (m.find(target - arr[i]) = m.end())
               m[arr[i]] = i;
        else {
           cout << i <<"  "<< mp[arr[i]];
        }
}
```

Ans-8) 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, merge sort might be best.

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



Inversions = 31

Ans-10) 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 as reverse sorted and either first or last element is picked as pivot.

$O(n^2)$

Best Case- Best case occurs when pivot element is the middle element or near to the middle element.

$O(n \log n)$

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

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

Basis	Quick Sort	Merge Sort
• Partition	Splitting is done in any ratio	Array is parted into just 2 halves.
• Works well on	Smaller array	fine on any size of array.
• Additional Space	Less (in-place)	More (Not in-place)
• Sorting Method	Internal	External
• Stability	Not-Stable	Stable.

Ans-12) Stable Selection sort.

```
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;
  }
```

Ans-13) Optimized Bubble Sort

```
int i, j ;
bool swapped ;
for(i=0; i<n-1; i++) {
    swapped = false ;
    for (j=0 ; j<n-i-1; j++){
         if (arr[j] > arr[j +1]){
             swap(~~&~~arr[j], arr[j+1]);
             swapped= true ;
         }
    }
    if (swapped = =false)
        break;
}
```

Ans-14) We will use Merge Sort because we can divide the 4 GiB data into 4 packets of 1 GiB and sort them separately 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.