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 Iterative 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;
}
```

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

Recursive Insertion sort :-

```
void insertion (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];
  } arr[j+1] = last;
}
```

Ans 3

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

Ans 4. Online sorting - InsertionSort
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;
  }
```

Recursive Binary search :-

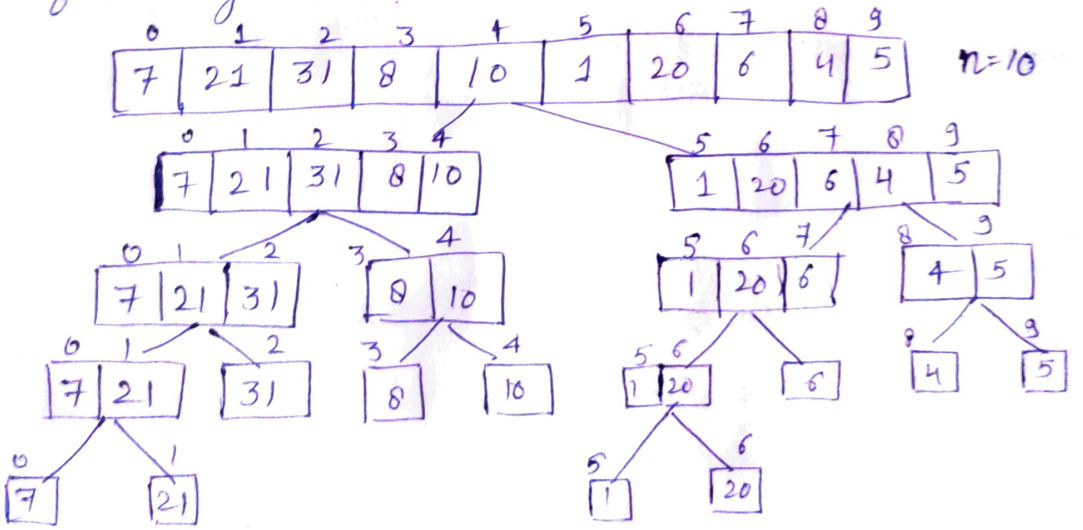
```
while (low <= high)
{ int mid = (low + high)/2;
  if (arr[mid] == key)
      return true;
  else if (arr[mid] > key)
      Binary-search(arr, low, mid-1);
  else
      Binary-search(arr, mid+1, high);
  }
  return false;
  }
```

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]] = 1;
  else
  {
    cout << i << " " << map[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 indicates - how far, as close the array is from being sorted



Ans 10

Worst Case :- This worst case occurs when the picked pivot always an extreme (smallest as largest) element. This happens when input array is sorted as reverse sorted & either first or last element is picked as pivot. $O(n^2)$

Best Case :- Best Case occurs when pivot element is the middle element as 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	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)
Efficient	inefficient for larger array	more efficient
Sorting method	Internal	External
Stability	Not stable	stable

Ans 12

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;
  }
}
```

Ans 13. We will use merge sort because we can divide the 4GB data into 4 packets of 1GB and sort them separately and combine them later.

- External sorting: All the data to set is sorted in memory at all times while sorting is in progress.

- External Sorting: All the data is stored outside memory & only loaded into memory in small chunks.