Tutorial -3

Q1: write linear search pseudocode to search an element in a sorted array with minimum comparisons.

→
```
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.

→ Iterative insertion sort -
```
for (int i = 1; i < n; i++)
{ j = i - 1;
  x = A[j];
while (j > -1 && A[j] > 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;
  insertionsort (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.

Q3. complexity of all sorting algorithms.

→ Bubblesort - $O(n^2)$
Insertionsort - $O(n^2)$
selection sort - $O(n^2)$
Merge sort - $O(n * \log n)$
Quik sort - $O(n \log n)$
count sort - $O(n)$
Bucket sort - $O(n)$

Q4. Divide all sorting algorithm into inplace / stable / online sorting.

→ Online sorting → Insertion sort
Stable sorting → Merge sort, Insertion sort, Bubble sort.

Inplace sorting → Bubble sort, Insertion sort, Selection sort.

Q.5. Write recursive / iterative pseudo code for binary search. What is time & space complexity of linear & binary search.

→ Iterative binary search - complexity - O(log n)

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

Recursive binary search - complexity - O(log n)

```
while (low <= high)
{  int mid = (low + high)/2;
   if (a[mid] == key)
      return True;
   else if (a[mid] > key)
   binarysearch (a, low, mid - 1);
   else
     binarysearch (a, mid + 1, high);
 }
 return False; }
```

Q.6. Write ~~recursive~~ recurrence relation for binary recursive search.

→ $T(n) = T(n/2) + T(n/2) + C$

Q7. Find two indexes such that A[i] + A[j] = K in minimum time complexity.

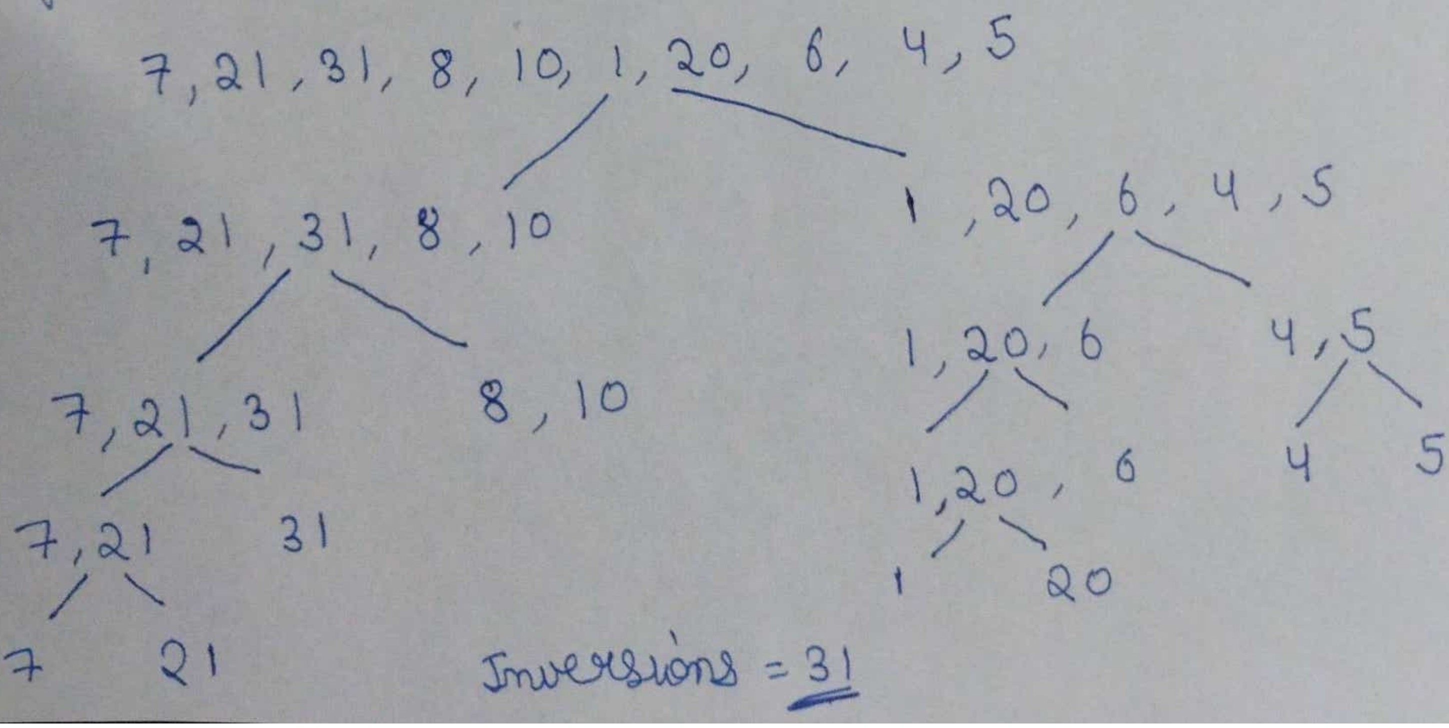
```
map <int, int> m;
for (int i=0; i<arr.size(); i++)
{ if (m.find(target - arr[i]) == m.end())
      m[arr[i]] = i;
  else
      cout << i << " " << m[arr[i]] << j << target-arr[i];
}}
```

Q8. which sorting is best for practical use?

Quicksort is fastest general purpose sort. In most practical situation quicksort is method of choice. If stability is important & space is available, merge sort will be best.

Q9. what do you mean by number of inversion in an array? Count the no. of inversion in arr[] = {7, 21, 31, 8, 10, 1, 20, 6, 4, 5} using merge sort.

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



Inversions = 31

Q10: In which cases Quick sort will give the best & worst case time complexity?

Worst case - when the pivot is always an extreme (smallest or largest). This happens when input array is sorted in reverse order.

Best case - when pivot element is as near to middle element. $O(n \log n)$.

Q11. Write recurrence relation of merge & quick sort in best & worst case? What are similarities & differences between complexities.

→ Merge sort → $T(n) = 2T(\frac{n}{2}) + O(n)$

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

Quick sort	Merge sort
• splitting is done in any ratio.	• Array is just parted into 2 halves.
• smaller array is suitable.	• suitable for any size of array.
• Inefficient for large array.	• More efficient.
• Internal sorting	• external sorting.
• Not stable	• stable