Tutorial-3

Name :- Vikash Agarwal
Section :- B.Tech (IT)
Class Roll no :- 32
University Roll :- 2015563

1. Linear Search (array, target)

```
{
  Initializa index = 0;
  While (index < number of element in array)
  {
    If (array[index] == target
      Return index;
    Increment Index by 1
  }
  Return -1;
}
```

2. Insertion Sort Iterative Sol^n.

```
Void InsertionSort (array, n)
{
  int i, temp, j;
  for (i<- 1 to n)
  {
    temp = array[i]
    j = i-1
  while (j>=0 and arr[j] > temp)
  {
    arr[j+1] = arr[j];
    j = j-1;
  }
```

```
        arr[j+1] = temp;
    }
}
```

Recursive soln:-

```
Void InsertionSort (array, n)
{
    if (n<=1)
        return;
    InsertionSort (array, n-1);
    int last = array[n-1];
    int j=n-2;
    while (j>=0 and array[j] > last)
    {
        array[j+1] = array[j];
        Decrement j;
    }
    array [j+1] = last
}
```

An online algorithm is one that can process its input piece by piece in a serial fashion i.e. in the order that the input is feed to the algorithm without having the entire input at the begining.

So, only Insertion sort is online else are offline.

Ques 3

	Best case	Average case	Worst case	Space
Bubble	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
Insertion	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
merge	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Quick	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(n)$
heap	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$

Ques 4.

	Stable	Inplace	online
Bubble	✓	✓	X
Selection	X	✓	X
insertion	✓	✓	✓
Merge	✓	X	X
Quick	X	X	X
heap	X	✓	X

Ques 5. Iterative sol^n :-

```
Int binarySearch (array, left, right, target)
{
  while (left <= right)
  {
    int m = (left + right) / 2 ;
    if (array[m] = target)
      return m;
    if (array[m] < target)
      left = m+1 ;
    else right = m - 1 ;
```

```
    }
    return -1;
  }
```

Recursive Soln :-

```
int BinarySearch (array, lebl, right, target)
  {
    if (right >= lebl)
      int mid = (left + right)/2;
    else if (array[mid] > target)
      return BinarySearch (array, lebl, mid-1, target);
    else
      return (BinarySearch (array, mid+1, right, target);
    }
  return -1;
  }
```

T.C of Binary = O(logn) T.C of linear = O(n)

S.C of Binary = O(1) [for iterative]

= O(n) [for recursive]

S.C of linear = O(1) [for iterative]

= O(n) [for recursive]

Q.6. $T(n) = T(\frac{n}{2}) + 1$

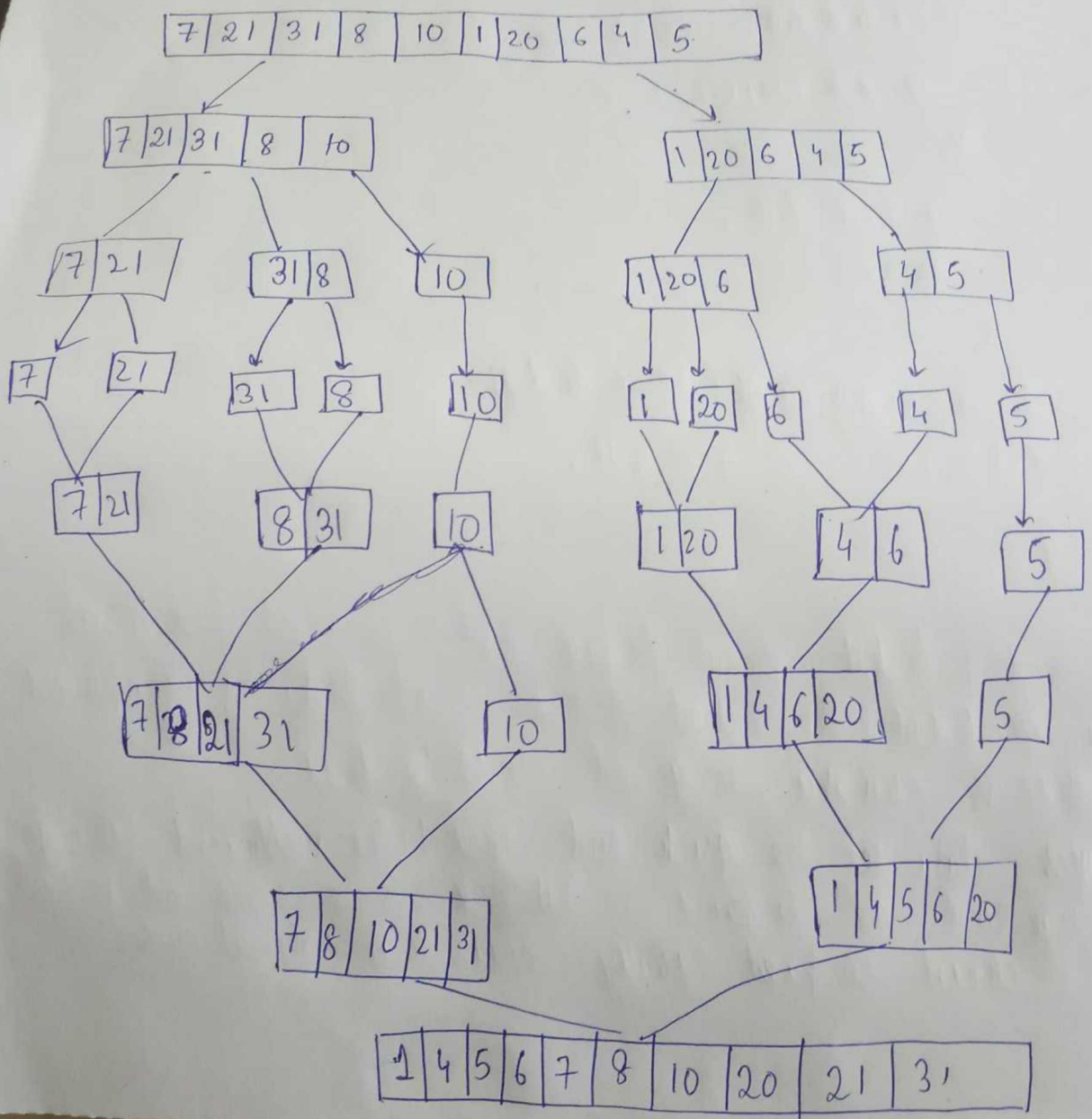
Q.7.
```
Void Index (array, target) {
  Unordered_set<int> st;
  for (i=0; i<array size ; ++i)
  {
    int diff = target - array[i]
    if (st.find (diff) == end)
      {
        st.insert (array [i]); }
    else
      {
        int find= diff ;
        break ;
      }
  }
  int  j= BinarySearch (array, find);
  cout << i << " " << j << endl;
}
```

Q.8. Quick sort is fastest general purpose sort. In most practical situation, quick sort is the method of choice. if stability is important and space is available, merge sort might be best.
for layers data set the quick sort proves to be inefficient so algorithm like merge sort are preferred in that case as the merge sort is stable and the element compared equally retain their original order.

Ques 9 for an array, inversion count indicates how far or close the array is from being sorted. If the array is already sorted then inversion count is 0. If an array is sorted in reverse order then inversion count is maximum.

{ 7, 21, 31, 8, 10, 1, 20, 6, 4, 5 }

for given array total no. of inversion are 32



Ques 10. The Best case for Quicksort will be when the partition process picks up the middle element as pivot. The worst case for Quicksort will be when the partition picks up first element of the array as array is sorted in decreasing order.

Q-11. Quick sort $\Rightarrow T(n) = 2T(n/2) + n$

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

Similarity

① Both the method follows divide and conquer algorithm.
② both divide the array in two parts.
③ both have best T.C of $O(n \log n)$

difference

① The merge sort is stable as compared to Quick sort.
② The worst and best T.C of merge is same whereas for Quick both are different i.e. $O(n^2) \rightarrow$ worst
$O(n \log n) \rightarrow$ best.
③ The quick sort is not viable in large dataset as its complexity goes on to $O(n^2)$ but for merge it is same.

Ques 12.

```
Void SelectionSort (int arr[], int n)
{
  int i, j, min_idx ;
  for (i=0 ; i<n ; ++i) {
     min_idx = i ;
     for (j=i+1 ; j<n ; ++j){
        if (arr[i] > arr[j])
           min_idx=j ;
     }
```

```
        int temp = arr[min_idx];
        for (j = min_idx ; j > i; --j )
          arr[j] = arr[j-1];
        }
      arr[i] = temp;
      }
}
```

Ques 13. To achieve this we will use External sorting technique. In internal sorting all the data to sort is stored in memory at all time while sorting is in progress. In external sorting data is stored outside on the disk and only loaded in memory in small chunks. External sorting is usually applied in cases when data can't fit into memory entirely. There is drawback of external sorting as we can access element whenever we want as its not available in memory.