Tutorial Sheet - 3

Ques 1 Linear Search (array, target)

```
{
    Initialize index = 0;
    while ( index < number of Element in array)
    {
        If (array [index] == target)
            Return index;

        Increment Index by 1

    }
    Return -1;
}
```

Insertion Sort Iterative soln.

Ques 2 Insertion Sort

```
Insertion Sort ( array, n)
void
{   int i, temp, j;
    for( i ← 1 to n)
    { temp = array [i]
        j = i - 1
        while( j >= 0 and arr [j] > temb)
        { 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 i = 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 fed to the algorithm without having the Entire Input at the beginning.

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(nlogn)	O(nlogn)	O(n²)	O(n)
Heap	O(nlogn)	O(nlogn)	O(nlogn)	O(1)

Ques 4

	~~Bread~~ Stable	Inplace	Online
Bubble	✓	✓	✗
Selection	✗	✓	✗
Insertion	✓	✓	✓
merge	✓	✗	✗
Quick	✗	✗	✗
Heap	✗	✓	✗

Ques 5 Iterative Soln

```
Int binary Search (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 Binary Search ( array, left, right, target)
{   if ( right >= left)
        int mid = (left + right) / 2 ;

    else if ( array [mid] > target)
        return Binary Search ( array, left, mid-1,
                                        target);

    else
      return (Binary Search ( array, mid +1, right, target);

    }
    return -1;
}
```

TC of Binary = O (logn)   TC of linear = O(n)

SC of Binary = O(1) [for iterative]

= O(n) [for recursive]

SC of linear = O(1) [for iterative]

= O(n) [for recursive]

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

Ques 7

```
void Index ( array, target) {
    unordered_set <int> st;
    for ( i = 0; i < array size; ++i)
    {
```

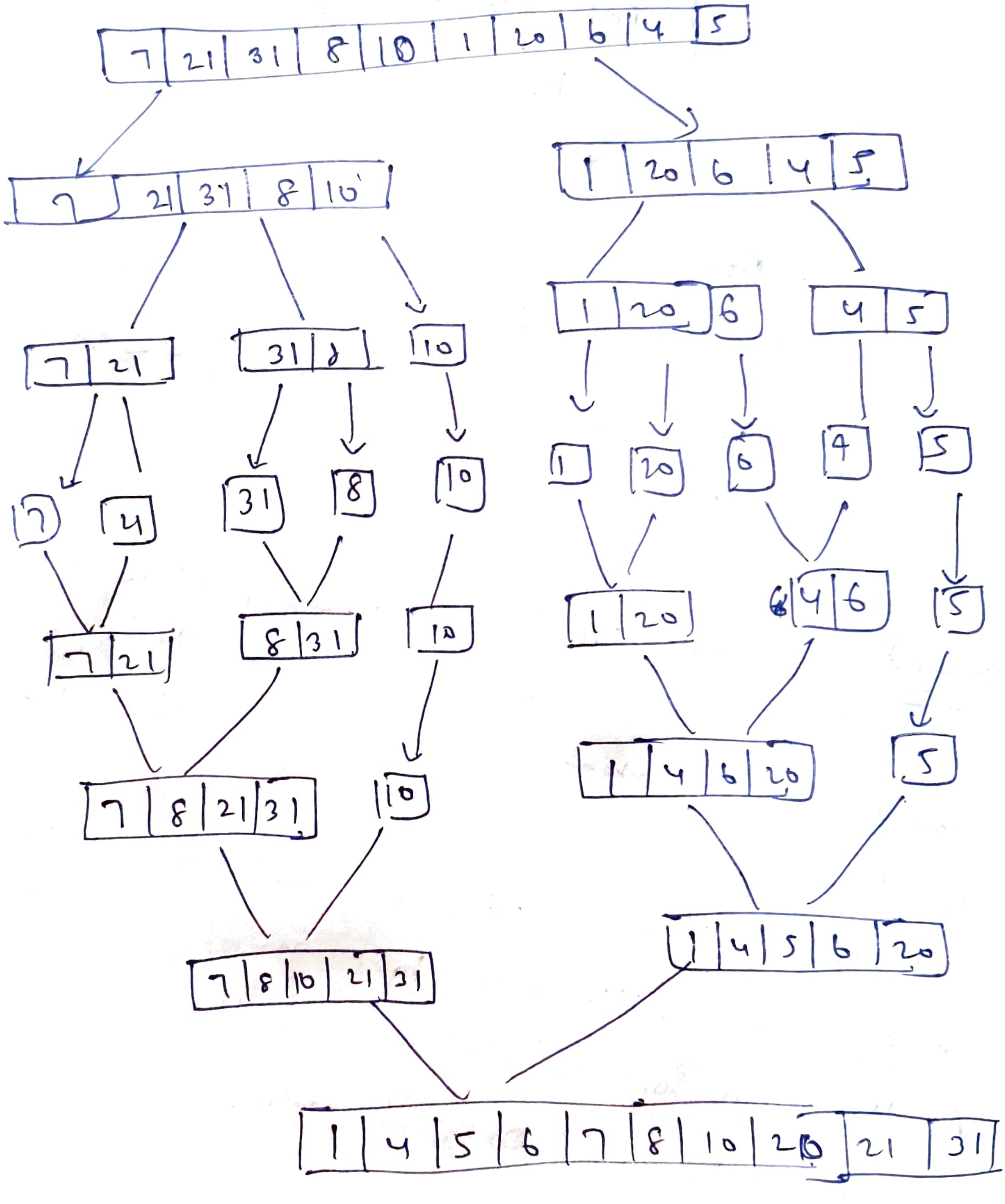
```
int diff = target - array[i]
if ( st.find(diff) == end)
   { st.insert(array[i]); }

else
   { int find = diff;
     break;
   }

}
int j = BinarySearch(array, find);
cout << i << " " << j << endl;
}
```

Ques 8 Quicksort is fastest general purpose sort. In most practical situations, quicksort is the method of choice. If stability is important and space is available, mergesort might be best
for larger data sets the Quick sort proves to be inefficient in that case so algorithms like merge sort are preferred and the element compared As the Merge sort is stable
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 Inversions are = ~~20~~ ~~22~~ 32



Ques10 The Best case for Quicksort will be when the portition 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 or array is sorted in decreasing order

Q11 Quick Sort $\Rightarrow$ $T(n) = 2T(n/2) + \Theta(n)$

Merge sort $\Rightarrow$ $T(n) = 2T(n/2) + n$

Similarity

1. Both the method follow divide & Conquer Algorithm
2. Both divide the Array in two parts.
3. Both have best TC of $O(n \log n)$

difference

1. The Merge Sort is stable as compared to Quick sort
2. The worst & best TC of merge is same whereas for Quick both are different i.e $\Theta(n^2) \rightarrow$ worst
   $\Theta(n \log n) \rightarrow$ Best.
3. The Quick Sort is not viable in large data set as its complexity goes on to $O(n^2)$ ~~bad~~ 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 techniques. 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 cannot access element whenever we want as its not available in memory.