Tutorial - 3

① int linearSearch (int *arr, int n, int key)

```
for i=0 to n-1
    if (arr[i] == key)
        return i
return -1
```

② Iterative insertion sort

```
void insertionSort (int arr[], int n)
{
  int i, j, temp;
  for i =1 to n
  {
    temp = arr[i];
    j = i-1;
   while (j ≥0 AND arr[j] >=temp)
    {
      arr[j+1] = arr[j];
      j = j-1;
      }
      arr[j+1] = temp;
  }
}
```

Recursive insertion sort

```
Void reInsertionSort (int arr[], int n)
 if (n ≤ 1)
    return
 reInsertionSort (arr, n-1)
 int temp = arr[n-1]
 int i = n-2
 while (i>=0 && arr[i] > temp)
  {
    arr[i+1] = arr[i]
       i = i-1
  }
```

arr [i+1] = temp;
}

sertion sort considers one input element per eration and produces partial solution without nsidering future. Thus insertion sort is called line sorting.

	Time Complexity (Best case)
Bubble sort	$O(n^2)$
Selection Sort	$O(n^2)$
Insertion Sort	$O(n)$
Merge Sort	$O(n \log n)$
Quick Sort	$O(n \log n)$
Heap Sort	$O(n \log n)$

	Inplace	Stable	Online
Bubble Sort	✓	✓	
Selection Sort	X	X	X
Insertion sort	✓	✓	✓
Merge sort	✓	✓	
Quick sort	X	X	
Heap sort	X	✓	

⑤ Iterative binary search

```
Int BinarySearch (int arr[], int l, int r, int key)
{
  while (l ≤ r)
    {
      int mid = (l+r)/2
      if (arr[mid] == key)
        return mid;
      if (arr[mid] < key)
        l = mid+1;
      else
        r = mid-1;
    }
  return -1;
}
```

Time Complexity

Best = O(1)

Average = O($\log_2 n$)

Worst = O($\log_2$)

Space complexity

O(1)

Recursive Binary Search

```
Int BinarySearch(int arr[], int l, int r, int key)
{
  if (l ≤ r) {
    int mid = (l+r)/2
    if (arr[mid] == key)
      return mid;
    else if (arr[mid] < key)
      return BinarySearch(arr, mid+1, r, key)
    else
      return BinarySearch(arr, l, mid-1, key)
  }
  return -1;
}
```

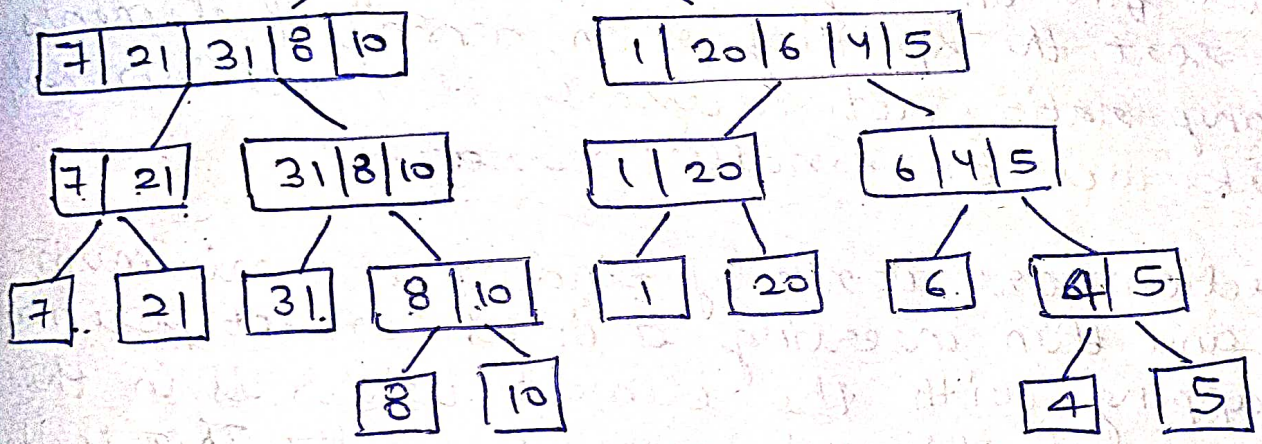
⑥ Re... ...ary recursive search

$T(n) = T(n/2) + 1$

⑧ Quicksort is the fastest general-purpose sort. In most practical situations, quick sort is the method of choice. If stability is important and space available merge sort might be best.

⑨ Inversion count for an array indicates how far or close the array is from being sorted. If the array is already sorted then inversion count 0 and if array is reverse sorted the inversion count is maximum. $a[i] > a[j]$ and $i < j$

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



Number of inversions = 31

⑩ Best case occurs when the array is in completely random manner, Time complexity $O(n \log n)$

Worst case occurs when array is already sorted either in ascending or descending order.

Time complexity $O(n^2)$

(11) Recurrence relation Merge sort in Best and worst case

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

Recurrence relation of Quick sort in

Best case $= 2T(\frac{n}{2}) + n$

Worst case $= T(n-1) + n$

Similarity -

Best Case time complexity $= n \log n$

Difference -

Worst Case time complexity of merge sort $O(n \log n)$ and of quick sort $O(n^2)$

Merge sort array is parted into 2 halves and it operates fine on any size of array whereas in quick sort the splitting of an array of elements is in any ratio, not necessarily divided into half it works well on smaller array.

(12) Selection sort works by finding the minimum element and then inserting it in its correct position by swapping with the element which is in the position of of this minimum element. This is what makes it unstable.

Selection sort can be made stable if instead of swapping, the minimum element is placed without swapping i.e; by placing the number in its position by pushing every element one step forward. (Insertion sort)

(13) If our computer has a RAM of 2 GB and we are given an array of 4 GB for sorting then we divide our source file into temporary files of size equal to the size of RAM and first sort these files by merge sort algorithm or quick sort algorithm.

Internal sorting- If the input data is such that it can be adjusted in the main memory at once.

External sorting- If the input data is such that it cannot be adjusted in the memory entirely at once, it needs to be stored in hard disk, floppy disk or any other storage device.