Roll no: 19BQ1A05H8
Sec: CSE-C

Assignment -1

1) Assume that there is a list {22,22,22,22,22,22,22}. What happens when selection sort is applied on the list? Explain.

A) **Sorting:-** Sorting is an arrangement of set of elements either in ascending or descending order.

Selection sort:-

1. Let us consider an unsorted array
2. Find the smallest element of the unsorted part and swap it with the first element
3. Then, consider the smallest element chosen in previous step as a part of the sorted list. The size of the sorted list increases in this way.
4. Repeat the steps one to three until the whole list is sorted.

$$\text{Given list} = \{\underset{0}{22}, \underset{1}{22}, \underset{2}{22}, \underset{3}{22}, \underset{4}{22}, \underset{5}{22}, \underset{6.}{22}\}$$

$$\text{After sorting list} = \{\underset{0}{22}, \underset{1}{22}, \underset{2}{22}, \underset{3}{22}, \underset{4}{22}, \underset{5}{22}, \underset{6.}{22}\}$$

When selection sort is applied on the given list, we get the same list as output because all the elements are equal. In selection sort we find the smallest element of unsorted part and swap it with first element. In the given list there is no smallest element as all the elements are equal. So there are no swappings and the given list remains same.

2. Sort the following list of names using Insertion sort: Varun, Amar, Karthik, Ramesh, Bhuvan, Dinesh, Firoz and Ganesh

A. Insertion sort:- Insertion sort is the sorting mechanism where the sorted list is built having one time at a time. the list elements are compared with each other sequentially and then arranged simultaneously in some particular order. This sort works on the principle of inserting an element at a particular position, hence the name insertion sort. Insertion sort works as follows:

1. The first step involves the comparison of the element in question with its adjacent elements.
2. And if at every comparison reveals that the element in question can be inserted at a particular position, then space is created for it by shifting the other elements one position to the right and inserting the element at the suitable position.
3. The above procedure is repeated untill all the elements in the list is at their apt position.

Varun	Amar	Karthik	Ramesh	Bhuvan	Dinesh	Firoz	Ganesh
Amar	Varun	Karthik	Ramesh	Bhuvan	Dinesh	Firoz	Ganesh
Amar	Karthik	Varun	Ramesh	Bhuvan	Dinesh	Firoz	Ganesh
Amar	Karthik	Ramesh	Varun	Bhuvan	Dinesh	Firoz	Ganesh
Amar	Bhuvan	Karthik	Ramesh	Varun	Dinesh	Firoz	Ganesh
Amar	Bhuvan	Dinesh	Karthik	Ramesh	Varun	Firoz	Ganesh
Amar	Bhuvan	Dinesh	Firoz	Karthik	Ramesh	Varun	Ganesh
Amar	Bhuvan	Dinesh	Firoz ·	Ganesh	Karthik	Ramesh	Varun

Output:-
```
Amar
Bhuvan
Dinesh
Firoz
Ganesh
Karthik
Ramesh
Varun.
```

3. Sort the following numbers using Quick Sort :- 67, 54, 9, 21, 12, 65, 56, 43, 34, 79, 70 and 45

A. **Quick Sort:-** Quick Sort is a divide and conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot..

Following are the steps involved in quick sort algorithm:

1. After selecting an element as pivot, which is the last index of the array in our case, we divide the array for the first time.
2. In quick sort, we call this partitioning. It is not simple breaking down of array into 2 subarrays, but in case of partitioning, the array elements are so positioned that all the elements smaller than the pivot will be on the left side of the pivot and all the elements greater than the pivot will be on right side of it.
3. And the pivot element will be at its final sorted position.
4. The elements to the left and right, may not be sorted.
5. Then we pick subarrays, elements on the left of pivot and elements on the right of pivot, and we perform partitioning on them by choosing a pivot in the subarrays.

67 54 9 21 12 65 56 43 34 79 70 45

key

L to R →

key

45 54 9 21 12 65 56 43 34 79 70 67

← R to L

key

45 54 9 21 12 65 56 43 34 67 70 79

L to R →

key

45 34 9 21 12 65 56 43 54 67 70 79

key ← R to L

45 34 9 21 12 65 56 43 | 54 67 | 70 79

L to R →

43 34 9 21 12 65 56 45 54 67 70 79

← R to L key

key

43 34 9 21 12 |45| |56 65| 54 67 | 70 79

L to R →

key ← R to L

43 12 9 21 34 |45| |56 . 65 54 67 | 70 79

← R to L

43 12 9 21 34 |45| 54 65 56 67 | 70 79

L to R → key

43 12 9 21 34 |45| |54| 65 56 67 70 79

key ← R to L key

34 12 9 21 43 45 |54| 56 65 67 70 79

key

L to R →

34 12 9 21 |43| 45 54 56 65 67 70 79

key ← R to L

21 12 9 34 43 45 54 56 65 67 70 79

key

L to R →

21 12 9 |34| 43 45 54 56 65 67 70 79

key ← R to L

9 12 21 |34| 43 45 54 56 65 67 70 79

Given list is sorted using quicksort.

Output:- The sorted array is

```
9
12
21
34
43
45
54
56
65
67
70
79
```

4. Implement Linear Search and Binary Search using Recursion.

A. Linear search in Java using recursion:-

```
Class LinearSearch
{
    Static int arr[] = {12, 34, 54, 2, 3};
    static int recSearch (int arr [], int l, int r, int x)
    {
        if (r<l)
            return -1;
        if (arr[l]==x)
            return l;
        if (arr[r]==x)
            return r;
        return recSearch (arr, l+1, r-1, x);
    }
```

Scanned with CamScanner

```
                if(flag!=1)
                {
                    while((a[loc]>=a[left]) && (loc!=left))
                        left++;
                        if(loc==left)
                            flag=1;
                        else if (a[loc]<a[left])
                            temp=a[loc]
                            a[loc]=a[left]
                            a[left]=temp
                            loc=left;

                    }

            }
            return loc;
}
static void quickSort (int a[], int beg, int end)
{
    int loc;
    if (beg< end)
    {
        loc=partition (a,beg,end);
        quickSort (a, beg, loc-1);
        quickSort (a, loc+1, end);
    }
 }
}
```

```
public static void main (String args[])
{
    int x = 3
    int index = recSearch (arr, 0, arr.length-1, x);
    if (index! = -1)
        System.out.println ("Element"+ x +" is present at
                            index" + index);
    else
        System.out.println ("Element"+x+" is not present");
  }
}
```

Output:- Element 3 is present at index 4

Java program to implement Binary Search using recursion:-

```
Class BinarySearch {
    int binarySearch (int arr [], int l, int r, int x)
    {
      if (r>=1) {
          int mid= l+(r-1)/2;
          if(arr[mid = = x)
              return mid;
          if (arr[mid] > x)
              return binarySearch (arr, l, mid-1, x);
          return binarySearch (arr, mid+1, r, x);
      }
      return -1;
    }
    public static void main(String args[])
    {
        BinarySearch ob = new BinarySearch();
        int arr[] = { 2, 3, 4, 10, 40 };
        int n = arr.length;
        int x = 10;
        int result = ob.binarySearch(arr, 0, n-1, x);
        if(result == -1)
            System.out.println("Element not present");
        else
            System.out.println("Element found at index"+result);
    }
}
```

Output:- Element found at index 3

5. Explain in brief the various factors that determine the selection of an algorithm to solve a computational problem.

A. * An algorithm is a sequence of steps to solve a particular problem.

* One problem can have 'n' number of solutions (i.e., multiple solutions).
* To find out best solution among 'n' solutions, we apply some analysis.
* Analysis of algorithms is the determination of the amount of time and space resources required to execute it.

* The performance of an algorithm can be measured by two properties: Time and space.
* "Time Complexity" of an algorithm quantifies the amount of time taken by an algorithm to run as a function of the length of the input.
* Similarly "Space Complexity of an algorithm quantifies the amount of space or memory taken by an algorithm to run as a function of the length of the input.

Space Complexity:- An algorithm represents the amount of memory space needed by the algorithm in its life cycle.

* Space needed by an algorithm is equal to the sum of the following two components

Space S(p) = Fixed part (A)+ Variable part Sp (I)

* A fixed part that is a space required to store certain data and variables (i.e., simple variables and constants, program size etc.), that are not dependent of the size of the problem.
* A variable part is a space required by variables, whose size is totally dependent on the size of the problem.
* For example, recursion stack space, dynamic memory allocation

In Case of Time Complexity, the running time of an algorithm depends on the following.

* Whether it is running on Single processor machine or Multi processor machine
* Whethere it is a 32-bit machine.
* Read and Write speed of the machine

* The amount of time required by an algorithm to perform Arithmetic operations, logical operations, return value and assignment operations etc.,
* Input data.