Regd. NO : 19BQIA05L7

Data Structures

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

The algorithm steps :

1. Initially, the whole array is unsorted.
2. Find the minimum of the unsorted part and swap it with the first element.
3. Then, consider the minimum number chosen in previous step as part of the sorted array. The size of the sorted part grows this way.
4. continue steps one through three until there are no elements left in the unsorted part.

Given list {22, 22, 22, 22, 22, 22, 22}

0	1	2	3	4	5	6
22	22	22	22	22	22	22

Min = 0

no swap

22 22 22 22 22 22 22

If we apply the selection sort on the given list there is no change in the list. Because every element in the given list is equal. First, we assume minimum is '0' index if we compare with the (minim) right side elements no element should not be less than minimum number. So, the minimum position does not change until the sorting. last position. Because, every element in the list is equal. If we apply Sorting (the) to the unequal elements in the list. In this list we apply selection sort there is no change before sorting and after sorting.

Sort the following list of names using insertion sort:
Varun, Amar, Karthik, Ramesh, Bhuvan, Dinesh, Firoz & Ganesh.

Algorithm Steps:

To sort an array of size n in ascending order:

1. Iterate from a[1] to a[n] over the array.
2. Compare the current element (key) to its predecessor.
3. If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.
4. For the strings take the ASCII values.

Given list {Varun, Amar, Karthik, Ramesh, Bhuvan, Dinesh, Firoz, Ganesh}

0	1	2	3	4	5	6	7
Varun	Amar	Karthik	Ramesh	Bhuvan	Dinesh	Firoz	Ganesh
86	65	75	82	66	68	70	71

Temp: 65

Amar	Varun	Karthik	Ramesh	Bhuvan	Dinesh	Firoz	Ganesh
65	86	75	82	66	68	70	71

Temp: 75

Amar	Karthik	Varun	Ramesh	Bhuvan	Dinesh	Firoz	Ganesh
65	75	86	82	66	68	70	71

Temp: 82

Amar	Karthik	Ramesh	Varun	Bhuvan	Dinesh	Firoz	Ganesh
65	75	82	86	66	68	70	71

Temp: 66

Amar	Bhuvan	Karthik	Ramesh	Varun	Dinesh	Firoz	Ganesh
65	66	75	82	86	68	70	71

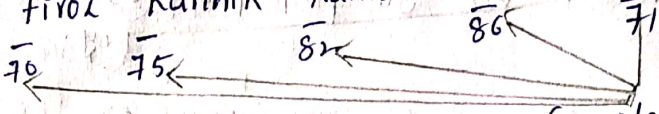
Temp: 68

Amar	Bhuvan	Dinesh	Karthik	Ramesh	Varun	Firoz	Ganesh
65	66	68	75	82	86	70	71

Temp: 70

Amar Bhuvan Dinesh Firoz Karthik Ramesh Varun Ganesh
65 66 68 70 75 82 86 71

Temp=71



Amar Bhuvan Dinesh Firoz Ganesh Karthik Ramesh Varun
65 66 68 70 71 75 82 86

Then the list is sorted by using insertion sort.

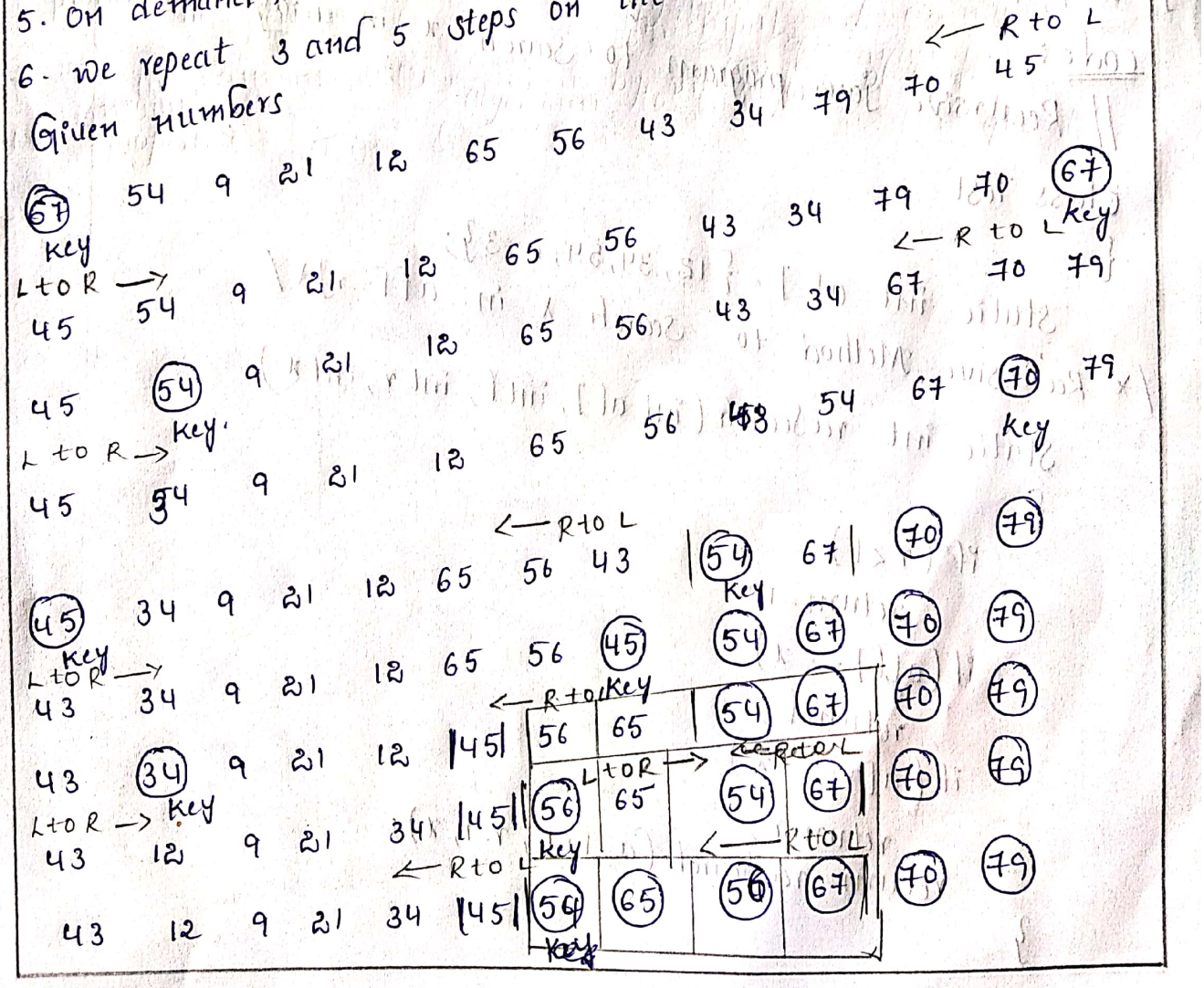
3) Sort the following numbers using Quicksort : 67, 54, 9, 21, 12, 65, 56, 43, 34, 79, 70, 45.

A: Algorithm Steps:

1. We take a list of elements.
2. We identify the first element as the key/pivot element.
3. comparison starts from right to left.
4. Later on from left to right.
5. On demand, we divide the list into two halves.
6. We repeat 3 and 5 steps on the left and right sublist.

Given numbers

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



L to R →

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

← R to L

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.

4] Implement Linear Search & Binary Search using Recursion.

code:

```
// Recursive java program to search x in array
class Test
{
    static int a[] = {12, 34, 54, 2, 3};
/* Recursive Method to search x in a[l..r] */
    static int recSearch (int a[], int l, int r, int x)
    {
        if (r < l)
            return -1;
        if (a[l] == x)
            return l;
        if (a[r] == x)
            return r;
        return recSearch (a, l+1, r-1, x);
    }
```

```
    // Driver method
public static void main (String[] args)
{
    int x = 3;
    // Method call to find x
    int index = recSearch (a, 0, a.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");
}
```

o/p: Element 3 is present at index 4

```
}
// Java implementation of recursive Binary Search
class BinarySearch {
    // Returns index of x if it is present in a[l..r], else //return -1
static int binarySearch (int a[], int l, int r, int x)
{   {
    if (r >= l) {
        int mid = l + (r - l) / 2;
        // if the element is present at middle itself
        if (a[mid] == x)
            return mid;
        // if element is smaller than mid, then it can only
            be present in left subarray
        if (a[mid] > x)
            return binarySearch (a, l, mid-1, x);
// else the element can only be present in right subarray
        return binary Search (a, mid+1, r, x);
    }
```

```
// Driver method to test above
public static void main (String args[])
{
    BinarySearch ob = new BinarySearch();
    int a[] = {2,3,4,10,40}
    int n = a.length;
    int x = 10;
    int result = ob.binarySearch(a, 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' no. 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 & space resources required to execute it

The performance of an algorithm can be measured by two properties: Time & Space.

Time complexity of an algorithm to run as a fun) quantifies the amount of time taken by an algorithm to run as a function of the length of the input.

19BQ1A0527

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 the algorithm in its life cycle.

Space needed by an algorithm is equal to the sum of the following two component.

Space S(P) = Fixed part(A) + variable part SP(I)

A fixedpart that is a space required to store certain data & variables, 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.
eg: recursion stack space, dynamic memory allocation etc.
Space can be calculated based on amount of space required

- To store program instructions.
- To store constant values.
- To store variable values.
- And for few other things like function calls, jumping statement etc.

eg:

```
int Sum(int a[], int n)
{
    int sum=0, i;
    for(i=0; i<n; i++)
        sum = sum + a[i];
    return sum;
}
```

In the above piece of code it requires
- 'n * 4' bytes of memory to store array variable 'a[]'.
- 4 bytes of memory for integer parameter 'n'.
- 8 bytes of memory for local integer variables 'sum' & 'i' (4 bytes each)
- 4 bytes of memory for return value.

In case of Time complexity, the running Time of an algorithm depends upon the following...

- whether it is running on single processor machine or Multi processor machine.
- whether it is a 32 bit machine or 64 bit machine.
- Read and write speed of the machine.
- The amount of time required by an algorithm to perform Arithmatic operations, logical operations, return value and assignment operations etc.,
- Input data.