ASSIGNMENT-1

19BQ1A05L1
Sk.Vaseem Naazleen

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

Ans: selection sort: which is a in-place sorting algorithm. The algorithm divides the input list two parts. a sorted sublist of items which is built up from left to right at the front of the list and a sublist of the remaining unsorted items that occupy the rest of the list. Initially, The sorted sublist is empty and the unsorted sublists the entire Input list.
~~Initially, the second sublist is~~
The algorithm proceeds by finding the smallest element in the unsorted sublists, exchanging it with the leftmost unsorted array element, and moving the sublist boundaries one element to the right.

So, for given list {22, 22, 22, 22, 22, 22, 22}, every element in the list is same, hence when we search for the smallest element, we get the number itself is the smallest element and no need to exchanging. It happens to hole array.

We can each element with minimum element in the unsorted elements. Hence, same no element is exchanged. Since, The time complexity of selection sort on array {22, 22, 22, 22, 22, 22, 22} is $O(n^2)$, $O(1)$ swaps.

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

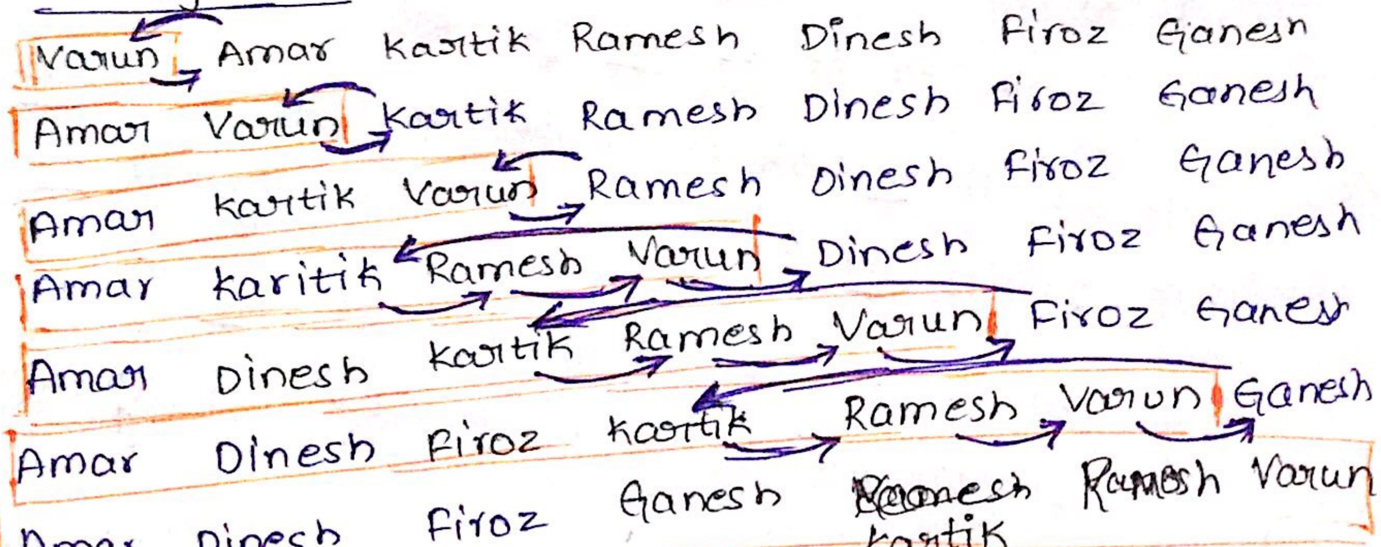
Q2. sort the following list of names using Insertion sort.
: Varun, Amar, karthik, Ramesh, Bhuvan, Dinesh, Firoz, and Ganesh.

Ans: About Insertion sort:

- It is a in-place comparison based sorting algorithm.
- We begin by assuming that a list with one item(poistion 0) is already sorted. On each pass, one for each item 1 through n-1, the current item is checked aganist those in the already sorted sublist. As we look back into the already sorted sublist, we shift those items that are greater to the right. When we reach a smaller item or the end of the sublist, the current item can be inserted.

Best case Time complexity: O(n)
Worst case Time complexity: O(n 2)

Sorting Names:-



Varun Amar Kartik Ramesh Dinesh Firoz Ganesh
Amar Varun Kartik Ramesh Dinesh Firoz Ganesh
Amar Kartik Varun Ramesh Dinesh Firoz Ganesh
Amar karitik Ramesh Varun Dinesh Firoz Ganesh
Amar Dinesh Kartik Ramesh Varun Firoz Ganesh
Amar Dinesh Firoz kartik Ramesh Varun Ganesh
Amar Dinesh Firoz Ganesh ~~Ramesh~~ Kartik Ramesh Varun

▭ - sorted sublist.
→ - swaps.

Sorted Array:

Amar Dinesh Firoz Ganesh Kartik Ramesh Varun

Q: sort the following numbers using Quicksort :
67, 54, 9, 21, 12, 65, 56, 43, 34, 79, 70 and 45.

Take last element as pivot.

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

Arrange the numbers according to pivot.

67 54 9 21 12 65 56 43 34 79 70 45
↑

67 54 9 21 12 65 56 43 34 79 70 45
↑

[illegible]

9 54 67 21 12 65 56 43 34 79 70 45
↑

9 21 12 54 67 65 56 43 34 79 70 45
↑

9 21 12 54 67 65 56 43 34 79 70 45
↑

9 21 12 54 67 65 56 43 34 79 70 45
↑

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

9 21 12 34 43 54 67 65 56 79 70 45.
↑

9 21 12 34 43 54 67 65 56 79 70 45.
↑

i = 4+1 = $\boxed{5}$

9 21 12 34 43 54 67 65 56 79 70 45.
↑
0 1 2 3 4 5 6 7 8 9 10 11

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

↑
Pivot is in correct position

Divide the array into 2 halfs

9 21 12 43 34
↑

9 21 12 43 34

9 21 12 43 34

9 21 12 43 34

9 21 12 34 43
↑
Pivot is in correct place.

9 21 12 43
↑

9 21 12

9 12 21
↑
Pivot is in correct place.

9 12 21 34 43

56 54 67 79 70 65
↑

56 54 67 79 70 65

56 54 67 79 70 65

56 54 67 79 70 65

56 54 67 79 70 65

56 54 65 79 70 67
↑
pivot is in correct place

56 54
↑

54 56
↑
correct place.

79 70 67
↑

79 70 67

67 70 79
↑
correct place

70 79
↑ correct plae

70 79

54 56 65 67 70 79.

Sorted Array:

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

Q Implement Linear Search & Binary search using Recursion.

Linear Search:

```
import java.io.*;
import java.util.scanner;

public class LinearSearchRecursion {
        public static void main(string[] args) {
              scanner in = new Scanner(system.in);

              System.out.println("Enter the size of the array: ");
              int n= in.nextInt();

              Int[] arr = new int[n];
              system.out.println("Enter array elements :  ");
              for(int i=0; i<n; i++) {
                   arr[i] = in.nextInt();
              }
              system.out.println("Enter the element to find: ");
              int key = in.nextInt();
              int index = linearSearch (arr, 0, key);
              If (index == -1) {
                  System.out.println("Key not found :C ");
              } else {
                  system.out.println (" Key found at " + (index+1) +
                                                  " position ");
              }
        }
     public static int linearSearch(int[] arr, int ind, int key) {
        if (arr.length == ind) {
              return -1;
        }

        if (arr[ind] == key) {
              return ind;
        }
        return linear search(arr, ind+1, key);
   }
}
```

Binary Search:

```
Import java.util.Scanner;

class BinarySearchRecursion {
    public static int search(int[] nums, int lb, int ub, int key){
        if( lb > ub) {
            return -1;
        }
        int mid = lb + (ub - lb)/2;
        if (nums[mid] ==Key) {
            return mid;
        } else if (num[mid] < Key) {
            return search (nums, mid+1, ub, key);
        } else {
            return search (nums, lb, mid+1, key);
        }
    }
    public static int binarySearch(int[] nums, int key) {
        return search (nums, 0, nums.length-1, key);
    }
    public static void main (String[] args) {
        int a[];
        int n;

        Scanner sc = new Scanner(system.in);
        System.out.println("Enter the size of the array: ");
        n = sc.nextInt();
        a = new int[n];
        System.out.println("Enter values into array : ");
        for (int i=0; i<n; i++) {
            System.out.printf("Enter %d index elem: ", i);
            a[i] = sc.nextInt();
        }

        int position = binarySearch(a, key);
        if ( position == -1) {
            system.out.println("Key not found : ( ");
        } else {
            system.out.println(" Key foundat " +(position +1)+
                                            " position");
        }
    }
}
```

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

Ans: In computer science, a computational problem is a problem that a computer might be able to solve, or a question that a computer may be able to answer.

A computational problem can be viewed as an infinite collection of instances together with a, possible empty, set of solutions for every instance.

i, To solve a computational problem, first we have to choose a data structure to solve the problem.

ii, write down the steps to solve the problem (algorithm)

iii, Then try to analyse the time complexity and space complexity taken by the algorithm.

iv, Then try to optimize the algorithm. By thinking, can we do better?

v, If possible, then we should try to optimize the algorithm.

vi, Hence, we can colve any computational problem by following above steps.