20/09/2020 19BCIAC5L4 CSE-D

Data structures and Algorithm in JAVA

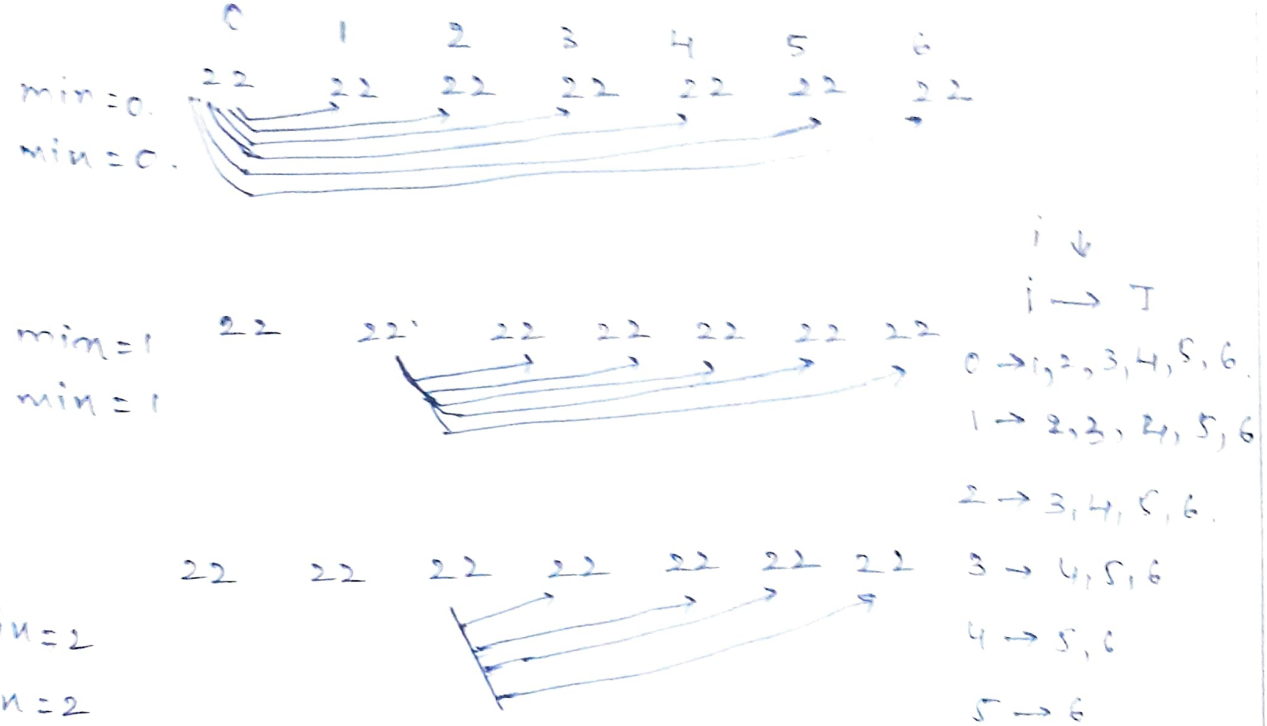
ASSIGNMENT - I

SIRILALITHA.A

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. se

Selection sort:

It follows human approach.



In the above list all the elements are in their exact position. so there will be no swaps and the array remains same.

Algorithm:-

Step 1: Set MIN to location 0.

Step 2: Search the minimum element in the list.

step 3: Swap with value at location MIN.

Step 4:- Increment MIN to point to next element.

step 5:- Repeat until list is sorted.

Code:-

```
public class SelectionSortADT
{
    public static void main(String args[])
    {
        int n, a[];
        Scanner sc = new scanner(System.in);
        System.out.println("enter size");
        n = sc.nextInt();
        System.out.println("enten elements");
        For (int i=0; i<n; i++)
                a[i] = sc.nextInt();
        System.out.println("Before sorting");
        display(a,n);
        SelectionSort(a,n);
        System.out.println("after sorting");
        display(a,n);
    }
```

```
public static void display (int a[], int n)
{
    For(int i=0; i<n; i++)
            System.out.println(a[i]+"  ");
}
public static void selectionsort(int a[], int n)
{
    int i,j,min,c;
    for(i=0; i<n; i++)
    {
        min=i
        For(j=i+1, j<n; j++)
        {
            if(a[min] >a[j])
                min = j;
        }
        If(min!=i)
        {
          c =a[min]; a[min] =a[i]; a[i]=c;
        }
    }
}
```

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

Insertion sort:

Every element is compared with previous elements in Insertionsort there is no swaps.

0 1 2 3 4 5 6

Temp → Amar

Varun, Amar, Karthik, Ramesh, Bhuvan, Dinesh, Firoz, Ganesh.

Amar, Varun, Karthik, Ramesh, Bhuvan, Dinesh, Firoz, Ganesh.

Temp → Karthik.

Amar, Karthik, Varun, Ramesh, Bhuvan, Dinesh, Firoz, Ganesh.

Temp → Ramesh

Amar, Karthik, Varun, Ramesh, Bhuvan, Dinesh, Firoz, Ganesh.

Temp → Bhuvan

Amar, Bhuvan, Karthik, Ramesh, Varun, Dinesh, Firoz, Ganesh.

Temp → Dinesh

Amar, Bhuvan, Dinesh, Karthik, Ramesh, Varun, Firoz, Ganesh.

Temp → Firoz

Amar, Bhuvan, Dinesh, Firoz, Karthik, Ramesh, Varun, Ganesh.

Temp → Ganesh

Amar, Bhuvan, Dinesh, Firoz, Ganesh, Karthik, Ramesh, Varun.

In the above list, every element First index is compared With the previous element First Index.

0 - no.
1 - 0.
2 - 1,0.
3 - 2,1,0.
4 - 3,2,1,0.
5 - 4,3,2,1,0.
6 - 5,4,3,2,1,0.
7 - 6,5,4,3,2,1,0.

Time complexity

no. of comp = $\frac{n \times n-1}{2}$

$= O(n^2)$ (worst case).

no. of swaps = 0.

Code:

```
import java.util.*;
public class InsertionSortADT
{
    public static void main(string args[])
    {
        int i;
        Scanner sc = new Scanner(System.in);
        System.out.println("enter size");
        int n = sc.nextInt();
        String a[] = new string[n+1];
        System.out.println("enter elements");
        For (i=0; i<=n; i++)
                a[i] = sc.next();
        System.out.println("after sorting");
        display(a,n);
    }
```

```
public static void display(String a[], int n)
{
    for(int i=1; i<=n; i++)
        System.out.println(a[i]+ "  ");
}
public static void insertsort (String a[], int n)
{
    int i, j;
    String t;
    for(i=1; i<=n; i++)
    {
      t = a[i];
      j = i-1;
      while ((j>0)&&(a[j].charAt(0)>t.charAt(0)))
      {
          a[j+1] = a[j];
          j--;
      }
    }
}
}
```

3. Sort the following numbers using QuickSort:-

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

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

Key

Right to left

Smaller element then key.

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

Left to right

Greater element then key

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

L TO R

key

left

right

R TO L

(45) 54 9 21 12 65 56 43 34

Key

R TO L

(70) 79

key

34 54 9 21 12 65 56 43 54 70 79

L TO R

R TO L

34 43 9 21 12 65 56 (45) 54

L TO R

34 43 9 21 12 | (45) | 56 65 54

(34) 43 9 21 12 (56) 65 34

Key

key

12 43 9 21 (34) 54 65 (56)

12 43 9 21 (34)

→

12 (34) 9 21 43

←

12 21 9 |34| 43

(12) 21 9

Key ←

9 21 (12)

→

9 12 21.

(56) 65 54

Key ←

54 65 (56)

→

54 56 65.

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

CODE:-

```
public class QuickSortADT
{
    public static void main(String args[])
    {
        int n;
        Scanner sc = new Scanner(System.in);
        System.out.println("enter size:");
        n = sc.nextInt();
        int a[] = new int[n];
        System.out.println("enter elements:");
        For (int i=0; i<n; i++)
                a[i] = sc.nextInt();
```

```
System.out.println("before sorting");
 display(a,n);
 Quick Sort(a,0,n-1);
 System.out.println("after sorting");
 display(0,n);
}
public static void display(int a[], int n)
{
    For(i=0; i<n; i++)
            System.out.println(a[i]+"  ");
 }
 public static void quicksort(int a[], int left, int right)
 {
    int pivot, l, u, temp;
    l = left;
    u = right;
    pivot = left;
    while(left != right)
    {
       while((a[right] >= a[pivot]) && (left != right))
            right --
       If(left != right)
          {
            temp = a[pivot];
```

```
        a[pivot] = a[right];
        a[right] = temp;
        pivot = right;
    }
    while ((a[left] <= a[pivot]) && (left != right))
        left++
    If (left != right)
    {
        temp = a[pivot];
        a[pivot] = a[left];
        a[left] = temp;
        pivot = left;
    }
}
If (l < pivot)
        quicksort (a, l, pivot-1);
If (pivot < u)
        quick sort (a, pivot+1, u);
}
}
```

4. Implement Linear search and Binary search using Recursion.

Linear Search :-

To find the Index of element in sequential order in an array. It is also called sequential (or) brute force search.

Code :-

```
import java.util.*;
public class ArrayADT
{
   public static void main(String args[])
   {
    int  a[] = {1,2,3,4,5};
    int  i, pos;
    pos = linearSearch (a,0, a.length-1, 8);
   If (pos == -1)
          System.out.println ("Key not found");
   else
     System.out.println("Key found at " + " +pos);
   }
  public static int linearSearch(int b[], int l, int n, key)
  {
       if (n< 1)
            return -1;
       else if (b[1] == key)
              return 1;
```

```
else if (b[n] == key)
            return n;
 else
     return linearsearch(b, l+1, n-1, key);
 }
}
```

Binary search :-

It decreases the no. of comparisions drastically.

It is to overcome the draw back of linear search.

code :-

```
import java.util.*;
Public class ArrayADT;
{
    public static void main(String args[])
    {
        int a[] = {1,2,3,4,5};
        int i, pos;
        pos = bsearch(a, 0, a.length-1, key);
        If (pos == -1)
            System.out.println("key not found");
        else
            System.out.println("key found at" + " " +pos);
    }
```

```
public static int bsearch(int b[], int l, int u, int key)
{
    int lb, ub, mind;
    lb = l;
    ub = u;
    while(lb<=ub)
    {
        mid = (lb+ub)/2;
        If (b[mid] == key)
                return mid;
        else if (b[mid] > key)
                return bsearch(b, lb, mid-1, key);
        else
                return bsearch(b, mid+1, ub, key);
    }
}
```

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

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

The factors that determine the selection of an algorithm to solve a computational problem are performance analysis of algorithm.

The performance of an algorithm can be measured by

1. Time
2. Space.

1. Time :-

The time complexity of an algorithm quantifies amount of time taken by an algorithm to run as a function of length of input.

2. Space :-

The space complexity quantifies amount of space/me - mory taken by an algorithm to run as a function of length of input.

→ However, even time and space complexity depends on lot of things like hardware, os, processor etc.. we don't consider any of these factors while analysing algorithm we consider only execution ~~of~~time algorithm.

Efficiency can be measured at 2 stages.

Before and After implementations as.

1. priori analysis.
2. posterior analysis.

1. Priori Analysis :-

This is defined theoretical analysis.

Efficiency can be measured by assuming all factors and have no effect after implementation.

2. Posterior Analysis :-

This is defined as empirical analysis of algorithm. The chosen algorithm is implemented using programming language and executed on target computer machine. In analysis stats like running time and space needed are collected.

- Space Complexity :-

This shows amount of space needed by algorithm in its life cycle.

Space needed by algorithm $S(p)$ = Fixed part (A) + Variable part (B).

Fixed part is space required by variables that aren't dependent on size of program.

Variable part is space required by variables that are dependent on size of program.

In other words, space can be calculated based on amount of space required to store constant, variables, program instructions, function calls etc...

Eg:-

```
int Add(int A[], int n)
{
    int s=0, i;
    For (i=0; i<n; i++)
        s = s+A[i];
    return s;
}
```

In the above code it requires:-

→ n * 2 bytes of memory to store array A[]

→ 2 bytes for integer parameter 'n'.

→ 4 bytes for sum variables 's' and 'i' (2 bytes each).

→ 2 bytes for return variables.

Time Complexity:-

The running time of an algorithm

This depends on the following.

- Whether it is running on single/multi-processor machine.
- Whether it is 32/64 bit machine.
- Read and write speed of machine.
- Time required for algorithm to perform arithmetic, logical assignment operation etc...

- Input data

Eg:-

```
int Add(int a[], int n)                 cost (Time for line)   Repetition (No. of lines)   Total (Time for worst case)
{
    int s=0, i;                         1                      1                           1
    For(i=0; i<n; i++)                  1+1+1                  1+(n+1)+n                   2n+2
        s = s+a[i];                     2                      n                           2n
    return s;                           1                      1                           1
}                                                                                          ------
                                                                                           4n + 4
```

Totally, it takes $4n+4$ units of time to compile an algorithm.

---------- x ---------- x ----------