**Batch- T8**

**Practical No. 2.**

**Searching Algorithm**

**Name: Siya Pondkule**

**PRN: 23520008**

1. You are an IT company's manager. Based on their performance over the last N working days, you must rate your employee. You are given an array of N integers called workload, where workload[i] represents the number of hours an employee worked on an ith day. The employee must be evaluated using the following criteria:

* Rating = the maximum number of consecutive working days when the employee has worked more than 6 hours.

You are given an integer *N*where *N* represents the number of working days. You are given an integer array *workload*where *workload[i]* represents the number of hours an employee worked on an ith day.

**Task**

Determine the employee rating.

import java.util.\*;

public class EmpRating {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the number of employees");

        int n=sc.nextInt();

        System.out.println("Enter workload of employees:");

        int[] workload=new int[n];

        for(int i=0;i<n;i++){

            workload[i]=sc.nextInt();

        }

        sc.close();

        System.out.println("Rating:"+employeeRating(workload));

    }

    public static int employeeRating(int[] workload){

        int currDayRating=0;

        int maxRating=0;

        for(int h:workload){

            if(h>6){

                currDayRating++;

            }else{

                if(currDayRating>maxRating){

                    maxRating=currDayRating;

                }

                currDayRating=0;

            }

        }

        if(currDayRating>maxRating){

            maxRating=currDayRating;

        }

        return maxRating;

    }

}

**Pseudo code:**

Procedure main:

Prompt user to enter the number of employees

Read integer n

Initialize an array workload of size n

Prompt user to enter the workload of employees

For i from 0 to n-1:

Read integer workload[i]

Call employeeRating(workload) and store the result in rating

Print "Rating: " followed by rating

End Procedure

Function employeeRating(workload):

Initialize currDayRating to 0

Initialize maxRating to 0

For each h in workload:

If h > 6

Increment currDayRating by 1

Else:

If currDayRating > maxRating:

Update maxRating to currDayRating

Reset currDayRating to 0

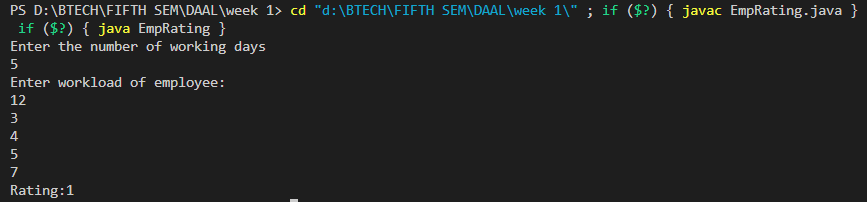
If currDayRating > maxRating:

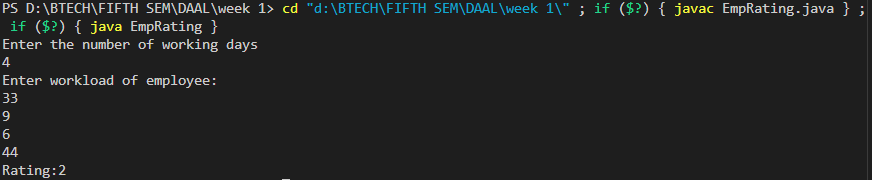
Update maxRating to currDayRating

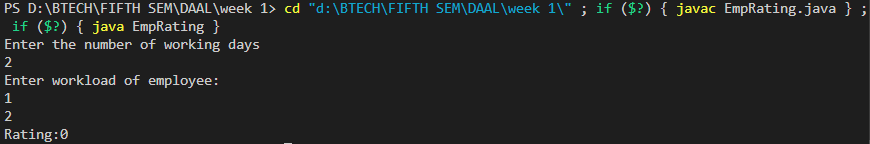
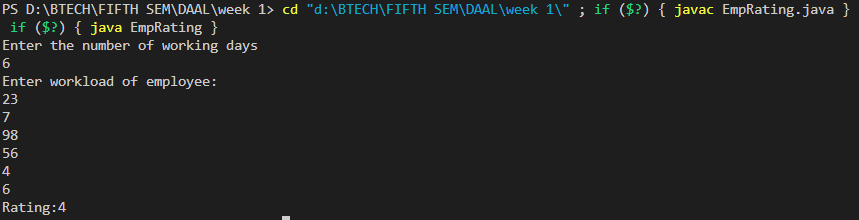
Return maxRating

End Function

Output:





1. You have N boxes numbered 1 through N and K candies numbered 1 through K. You put the candies in the boxes in the following order:

* first candy in the first box,
* second candy in the second box,
* .......
* ........
* so up to N-th candy in the Nth box,
* the next candy in (N - 1)-th box,
* the next candy in (N - 2)-th box
* ........
* .......
* and so on up to the first box,
* then the next candy in the second box
* ......    and so on until there is no candy left.

So you put the candies in the boxes in the following order:

Find the index of the box where you put the K-th candy.

import java.util.\*;

public class candies\_in\_box {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter number of Boxes:");

        int n=sc.nextInt();

        System.out.println("Enter number of candies:");

        int k=sc.nextInt();

        sc.close();

        int indexOfBox=findBoxIndex(n,k);

        System.out.println("Kth Candies index:"+indexOfBox);

    }

    public static int findBoxIndex(int n,int k){

        int currBox=0;

        int move=1;

        for(int i=1;i<=k;i++){

           if(move==1){

            if(currBox<n){

                currBox++;

            }else{

                currBox--;

                move=-1;

            }

           }else{

            if(currBox>1){

                currBox--;

            }else{

                currBox++;

                move=1;

            }

           }

        }

        return currBox;

    }

}

**Pseudo code:**

Procedure main:

Prompt user to enter the number of boxes

Read integer n (number of boxes)

Prompt user to enter the number of candies

Read integer k (number of candies)

Call findBoxIndex(n, k) and store the result in indexOfBox

Print "Kth Candies index: " followed by indexOfBox

End Procedure

Function findBoxIndex(n, k):

Initialize currBox to 0

Initialize move to 1

For i from 1 to k:

If move == 1:

If currBox < n:

Increment currBox by 1

Else:

Decrement currBox by 1

Set move to -1 (reverse direction)

Else:

If currBox > 1:

Decrement currBox by 1

Else:

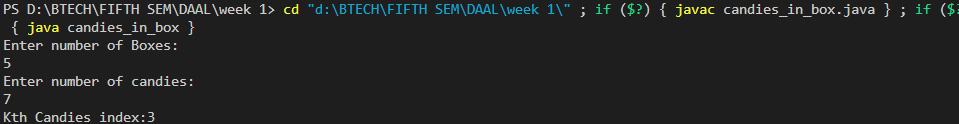
Increment currBox by 1

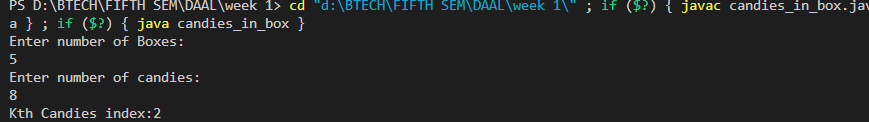
Set move to 1 (reverse direction)

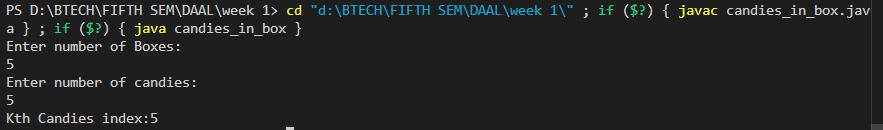
Return currBox

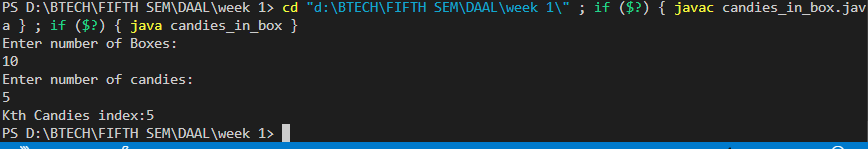
End Function

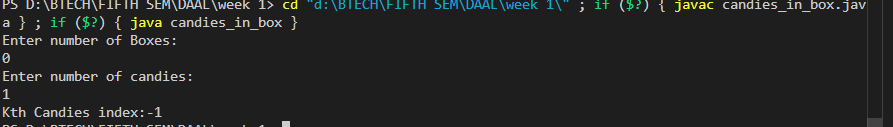
Output:











1. Implement and Explain Tower of Hanoi Algorithm.

Algorithm:

1. Start
2. If there is only one disk(n==1)

then move the disk directly from source to destination.

1. Otherwise move n-1 disk from source to auxiliary rod

Use destination rod as an intermediate

* 1. Move nth(largest) disk from source rod to destination rod
  2. Move nth disk from auxiliary rod to destination rod

1. End.

**Pseudo code:**

Procedure main:

Prompt user to enter the number of disks

Read integer n (number of disks)

Print "Tower of Hanoi Result:"

Call tower(n, 'A', 'C', 'B') to solve the Tower of Hanoi problem

Close the scanner

End Procedure

Function tower(n, source, dest, auxiliary):

If n == 1:

Print "Move disk 1 from source " + source + " to destination " + dest

Return

Call tower(n - 1, source, auxiliary, dest) to move n-1 disks from source to auxiliary

Print "Move disk " + n + " from source " + source + " to destination " + dest

Call tower(n - 1, auxiliary, dest, source) to move n-1 disks from auxiliary to destination

End Function

import java.util.\*;

public class tower\_of\_hanoi {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter number of Disks:");

        int n=sc.nextInt();

        System.out.println("Tower of Hanoi Result:");

        tower(n, 'A', 'C', 'B');

        sc.close();

    }

    public static void tower(int n,char source,char dest,char auxillary){

        if(n==1){

            System.out.println("Move disk 1 from source "+source+" to destination "+dest);

            return;

        }

        tower(n-1, source, auxillary, dest);

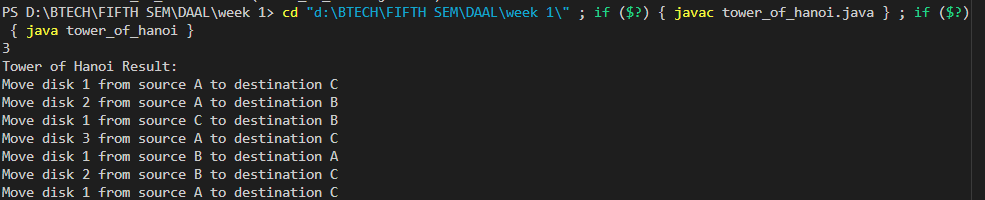
        System.out.println("Move disk "+n+" from source "+source+" to destination "+dest);

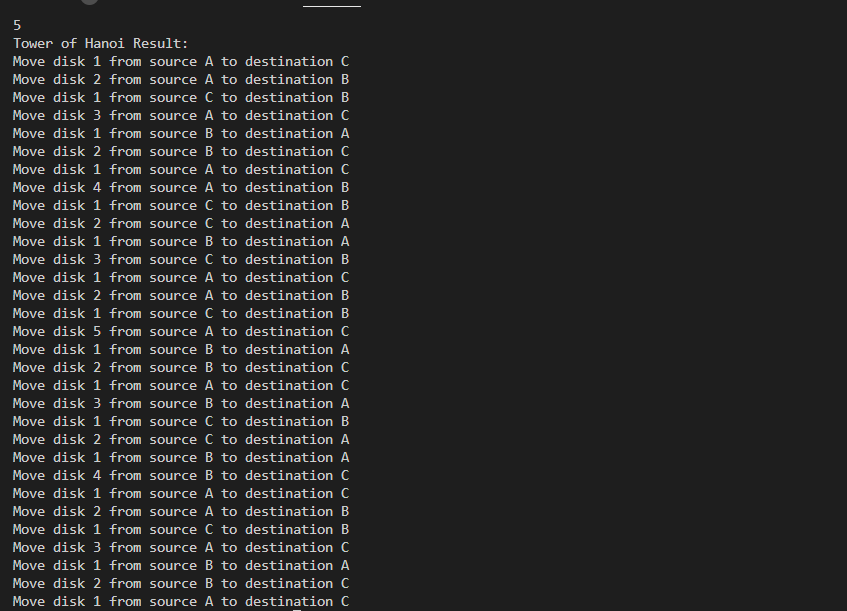
        tower(n-1, auxillary, dest, source);

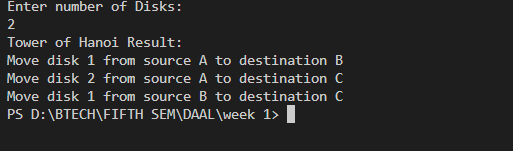
    }

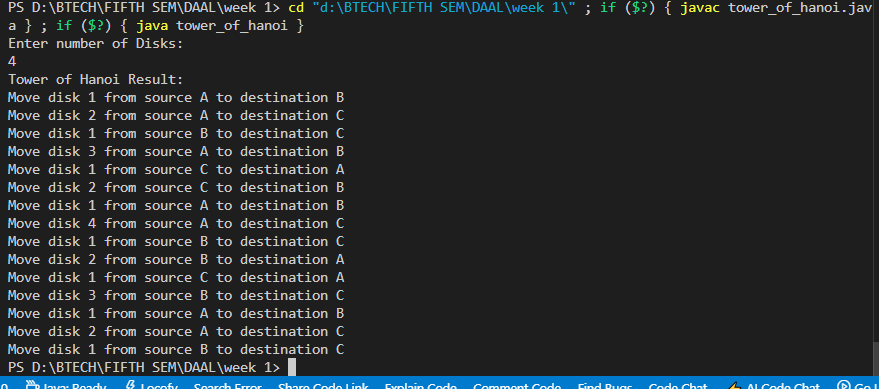
}

Output:









1. There is a frog initially placed at the origin of the coordinate plane. In exactly 1 second, the frog can either move up 1 unit, move right 1 unit, or stay still. In other words, from position (x, y), the frog can spend 1 second to move to:

* (x+1,y)
* (x,y+1)
* (x, y)

The T seconds , a villager who sees the frog reports that the frog lies on or inside a square of side-length s with coordinates (X,Y), (X+s, Y), (X,Y+s), (X+s, Y+s).

Calculate how many points with integer coordinates on or inside this square could be the frog’s position after exactly T seconds.

Input format:

The first and only line of input contains four space separated integers: X, Y, s and T.

Output format:

Print the number of points with integer coordinates that could be the frog’s position after T seconds.

import java.util.Scanner;

public class frog\_movement {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter value of X:");

        int X = scanner.nextInt();

        System.out.println("Enter value of Y:");

        int Y = scanner.nextInt();

        System.out.println("Enter value of S:");

        int s = scanner.nextInt();

        System.out.println("Enter value of T:");

        int T = scanner.nextInt();

        int count = 0;

        for (int x = X; x <= X + s; x++) {

            for (int y = Y; y <= Y + s; y++) {

                int steps = (x - X) + (y - Y);

                if (steps <= T && steps + (T - steps) <= T) {

                    count++;

                }

            }

        }

        System.out.println("Frog position:"+count);

        scanner.close();

    }

}

**Pseudo code:**

Procedure main:

Prompt user to enter value of X

Read integer X

Prompt user to enter value of Y

Read integer Y

Prompt user to enter value of S

Read integer S

Prompt user to enter value of T

Read integer T

Initialize count to 0

For each x from X to X + S:

For each y from Y to Y + S:

Calculate steps as (x - X) + (y - Y)

If steps <= T and (steps + (T - steps)) <= T:

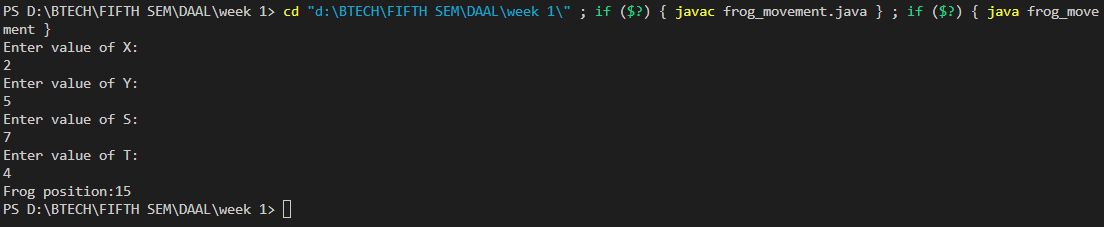
Increment count

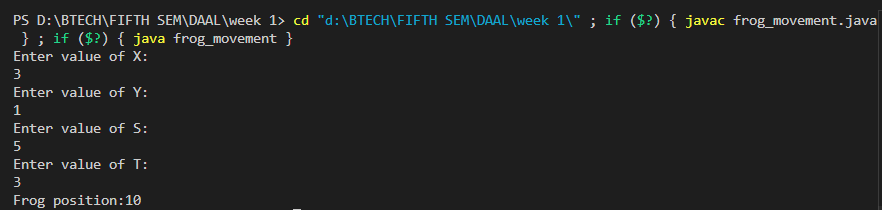
Print "Frog position:" + count

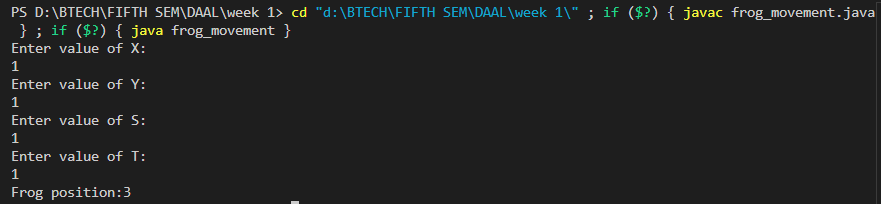
Close the scanner

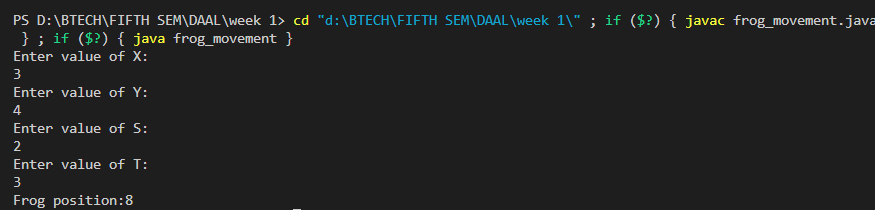
End Procedure

Output:









1. Implement linear search Algorithm.

import java.util.\*;

public class linear\_search {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter number of elements in array:");

        int n=sc.nextInt();

        System.out.println("Enter elements in array:");

        int[] arr=new int[n];

        for(int i=0;i<n;i++){

            arr[i]=sc.nextInt();

        }

        System.out.println("Enter target element:");

        int target=sc.nextInt();

        sc.close();

        int Result=linearSearch(arr, target);

        System.out.println("Index of target element:"+Result);

    }

    public static int linearSearch(int[] arr,int target){

    for(int i=0;i<arr.length;i++){

        if(arr[i]==target){

            return i;

        }

    }

    return -1;

  }

}

**Pseudo code:**

Procedure main:

Prompt user to enter number of elements in array

Read integer n

Prompt user to enter elements in array

Initialize array arr of size n

For each index i from 0 to n-1:

Read integer and assign it to arr[i]

Prompt user to enter target element

Read integer target

Close the scanner

Call linearSearch(arr, target) and store the result in Result

Print "Index of target element: " + Result

End Procedure

Procedure linearSearch(arr, target):

For each index i from 0 to length of arr - 1:

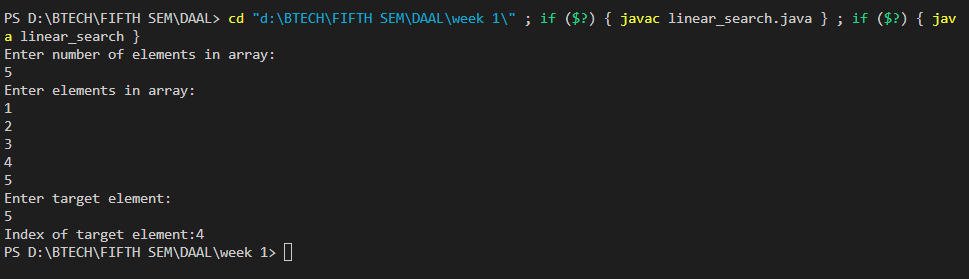
If arr[i] is equal to target:

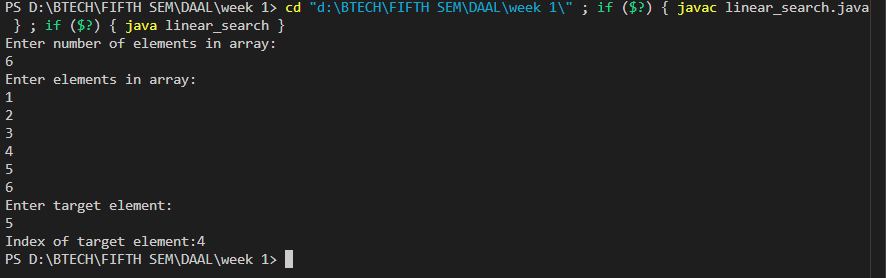
Return i

Return -1

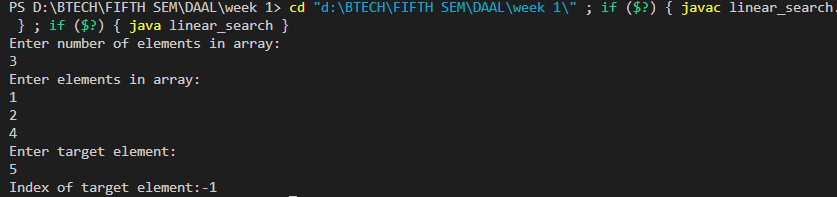
End Procedure

Output:









6. Implement Binary Search algorithm.

import java.util.Scanner;

public class binary\_search {

    public static void main(String[] args) {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter number of elements in array:");

        int n=sc.nextInt();

        System.out.println("Enter array elements:");

        int[] arr=new int[n];

        for(int i=0;i<n;i++){

            arr[i]=sc.nextInt();

        }

        System.out.println("Enter target element:");

        int target=sc.nextInt();

        sc.close();

        int Result=binarySearch(arr, target);

        System.out.println("Result:"+Result);

    }

    public static int binarySearch(int[] arr,int target){

        int left=0;

        int right=arr.length-1;

        while(left<=right){

            int mid=(left+right)/2;

            if(arr[mid]==target){

                return mid;

            }

            if(arr[mid]<target){

                left=mid+1;;

            }

            if(arr[mid]>target){

                right=mid-1;

            }

        }

        return -1;

    }

}

Pseudo code:

Procedure main:

Prompt user to enter number of elements in array

Read integer n

Prompt user to enter array elements

Initialize array arr of size n

For each index i from 0 to n-1:

Read integer and assign it to arr[i]

Prompt user to enter target element

Read integer target

Close the scanner

Call binarySearch(arr, target) and store the result in Result

Print "Result: " + Result

End Procedure

Procedure binarySearch(arr, target):

Initialize left to 0

Initialize right to length of arr - 1

While left is less than or equal to right:

Calculate mid as (left + right) / 2

If arr[mid] is equal to target:

Return mid

If arr[mid] is less than target:

Set left to mid + 1

If arr[mid] is greater than target:

Set right to mid - 1

Return -1

End Procedure

Output:

