Problem Solving - III - Hands On

1. Seating Arrangement

Problem statement :

You are required to create a program that determines the various arrangements in which a group of 'n' individuals can be seated in 'r' seats within a classroom or theater setting. The program should calculate an
arrangements.

Sample Test case:

```
Input 1 : n=4, r=2
Output 1: Possible arrangements : 6
Input 2: n=5 , r=3
Output 2: Possible arrangements: 10
```

Constraints:

```
1 ≤ n, r ≤ 10
```

Analysis

The problem requires the calculation of the number of possible arrangements of 'n' individuals in 'r' seats.

Code:

```
package com.hands_on;

import java.util.Scanner;

public class SeatingArrangement {
    public static int seatingArrangement(int n, int r){
        return (factorial(n)/(factorial(r)*factorial(n-r)));
    }

    public static int factorial(int num){
        if (num == 1){
            return 1;
        } else {
            return num * factorial(num -1);
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("n = ");
        int n = sc.nextInt();
        System.out.print("r = ");
        int r = sc.nextInt();
        System.out.print("The possible arrangement: " + seatingArrangement(n,r));
    }
}
```

Output:

```
n = 4
r = 2
The possible arrangement: 6
```

---

2. Reaching Points

Problem statement :

Given four integers sx, sy, tx, and ty, return true if it is possible to convert the point (sx, sy) to the point (tx, ty) through some operations, or false otherwise. The allowed operation on some point (x, y) is to convert
y).

Example 1:

```
Input: sx = 1, sy = 1, tx = 3, ty = 5
Output: true
```

> Explanation:
>
> One series of moves that transforms the starting point to the target is: (1, 1) -> (1, 2) (1, 2) -> (3, 2) (3, 2) -> (3, 5)

Example 2:

```
Input: sx = 1, sy = 1, tx = 2, ty = 2
Output: false
```

Example 3:

```
Input: sx = 1, sy = 1, tx = 1, ty = 1
Output: true
```

Constraints:

```
1 <= sx, sy, tx, ty <= 10^9
```

Analysis

- Get the input values of sx, sy, tx, ty.
- Check if the target point is reachable from the source point.
- If the target point is reachable, return true; otherwise, return false.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class ReachingPoints {
    public static boolean reachingPoint(int sx, int sy, int tx, int ty,boolean flag){
        if (sx>tx || sy > ty){
            return false;
        } else if (sx == tx && sy == ty){
            return true;
        } else if (sx < tx && flag){
            return reachingPoint(sx + sy , sy, tx, ty, false);
        }  else if (sy < ty && !flag){
            return reachingPoint(sx, sx + sy , tx, ty,true);
        }
        return false;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the value of sx: ");
        int sx = sc.nextInt();
        System.out.print("Enter the value of sx: ");
        int sy = sc.nextInt();
        System.out.print("Enter the value of sx: ");
        int tx = sc.nextInt();
        System.out.print("Enter the value of : ");
        int ty = sc.nextInt();
        System.out.println(reachingPoint(sx,sy,tx,ty,false));
    }
}
```

Output:

```
Enter the value of sx: 1
Enter the value of sy: 1
Enter the value of tx: 3
Enter the value of ty: 5
true
```

3. Ways To Express

Problem statement :

You are given the number ‘N’. The task is to find the number of ways to represent ‘N’ as a sum of two or more consecutive natural numbers.

Example:

```
N = 9
‘9’ can be represented as:
2 + 3 + 4 = 9
4 + 5 = 9
```

> The number of ways to represent ‘9’ as a sum of consecutive natural numbers is ‘2’. So, the answer is ‘2’.

> Note:

1. The numbers used to represent ‘N’ should be natural numbers (Integers greater than equal to 1).

Sample Input 1:

```
2
21
15
```

Sample Output 1:

```
3
3
```

> Explanation Of Sample Input 1:
>
> Test Case 1:
>
> ‘21’ can be represented as:
>
> 1 + 2 + 3 + 4 + 5 + 6 = 21
>
> 6 + 7 + 8 = 21
>
> 10 + 11 = 21

> The number of ways to represent ‘21’ as a sum of consecutive natural numbers is ‘3’. So, the answer is ‘3’.
>
> Test Case 2:
>
> ‘15’ can be represented as:
>
> 1 + 2 + 3 + 4 + 5 = 15
>
> 4 + 5 + 6 = 15
>
> 7 + 8 = 15
>
> The number of ways to represent ‘15’ as a sum of consecutive natural numbers is ‘3’. So, the answer is ‘3’.

Sample Input 2:

```
2
18
16
```

Sample Output 2:

```
2
0
```

Constraints:

```
1 <= T <= 1000
3 <= N <= 10 ^ 5
Where ‘T’ is the number of test cases, and ‘N’ is the given number.
```

Analysis

- Get the input values of T and N.
- Find the number of ways to represent N as a sum of two or more consecutive natural numbers.
- Return the number of ways.

Code:

```
package com.hands_on;

import java.util.Scanner;

public class WaysToExpress {
    public static int wayToExpress(int num){
        int  count = 0;
        for(int i=0;i<num;i++){
            int sum = 0;
            for(int j=i+1;j<num;j++){
                if (sum == num){
                    count += 1;
                }
                sum+= j;

            }
        }
        return count;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter any number: ");
        int num = sc.nextInt();
        System.out.print("The count of "+ num +" is "+ wayToExpress(num));
    }
}
```

Output:

```
Enter any number: 21
The count of 21 is 3
```

---

4. Butterfly star pattern

Problem statement:

Create a program to print a butterfly star pattern consisting of stars in a symmetrical butterfly shape. The pattern should be printed such that the stars form wings on both sides of the centerline. The number of st
decrease towards the centerline.

Input 1 :

```
rows = 5
```

Output 1:

```
*        *
**      **
***    ***
****  ****
```

```
**********
**********
****  ****
***    ***
**      **
*        *
```

Input 2:

```
rows =7
```

Output 2 :

```
*            *
**          **
***        ***
****      ****
*****    *****
******  ******
**************
**************
******  ******
*****    *****
****      ****
***        ***
**          **
*            *
```

Analysis

- Get the input value of rows.
- Print the butterfly star pattern with the given number of rows.

Code:

```
package com.hands_on;

import java.util.Scanner;

public class ButterFlyPattern  {
    public static void butterflyPattern(int num){
        for(int i=0;i<num;i++){
            for(int j=0;j<i+1;j++){
                System.out.print("*");
            }
            for(int k=num-1; k>i; k-- ){
                System.out.print("  ");
            }
            for(int j=0;j<i+1;j++){
                System.out.print("*");
            }
            System.out.println();
        }
        for(int i=num -1;i>=0;i--){
            for(int j=0;j<i+1;j++){
                System.out.print("*");
            }
            for(int k=num-1; k>i; k-- ){
                System.out.print("  ");
            }
            for(int j=0;j<i+1;j++){
                System.out.print("*");
            }
            System.out.println();
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter any number: ");
        int num = sc.nextInt();
        butterflyPattern(num);
    }
}
```

Output:

```
Enter any number: 5
*        *
**      **
***    ***
****  ****
**********
**********
****  ****
***    ***
**      **
*        *
```

5. Sum of two prime numbers

Problem statement:

Rita has to design a code that takes a positive integer or number which is required to be given by the user. Then the code should further identify whether that digit can be expressed as the sum of two prime numb can be expressed as sum of two prime numbers then, print the integer can be expressed as sum of two prime numbers as a result.

Sample Input 1:

```
34
```

Sample Output 1:

```
3,31
```

Sample Input1:

```
20
```

Sample Output2:

```
7,13
```

Constraints:

```
N>=1
```

Analysis

- Get the input value of N.
- Find the two prime numbers whose sum is equal to N.
- Print the two prime numbers.

Code:

```
package com.hands_on;

import java.util.Scanner;

public class SumOfTwoPrimeNumbers {
    public static int[] sumOfTwoPrimeNumber(int num){
        int[] arr = new int[2];
        outer:
        for(int i = 2; i<=num; i++){
            if (isPrime(i)) {
                for (int j = 2;j<=num ; j++ ) {
                    if (isPrime(j)){
                        if (i+j == num){
                            arr[0] = i;
                            arr[1] = j;
                            break outer;
                        }
                    }
                }
            }
        }

        return arr;
    }

    public static boolean isPrime(int n){
        if (n<=1){
            return false;
        }
        for(int i=2;i<n;i++){
            if (n%i == 0){
                return false;
            }
        }
        return true;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter any number: ");
        int num = sc.nextInt();
        int[] res = sumOfTwoPrimeNumber(num);
        System.out.println(res[0]+" "+res[1]);
    }
}
```

Output:

```
Enter any number: 34
3 31
```

6. Jumping Numbers

Problem statement

You are given a positive integer 'N'. Your task is to print all the jumping numbers smaller than or equal to 'N'. A number is called a jumping number if all adjacent digits in it differ by 1. All the single-digit numbers a
numbers.

> Note:
>
> The difference between ‘9’ and ‘0’ is not considered as 1. Example:
>
> Let’s say 'N' = 25. The jumping numbers less than or equal to 25 are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 21, 23. In all these numbers the adjacent digits differ by 1.

Sample Input 1:

```
2
25
45
```

Sample Output 1:

```
0 1 2 3 4 5 6 7 8 9 10 12 21 23
0 1 2 3 4 5 6 7 8 9 10 12 21 23 32 34 43 45
```

> Explanation 1: For the first test case, refer to the example explained before. For the second test case, N = 45. The jumping numbers less than 45 are: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 21, 23, 32, 34, 43, 45.

Sample Input 2:

```
2
10
1
```

Sample Output 2:

```
0 1 2 3 4 5 6 7 8 9 10
0 1
```

Constraints:

```
1 <= T <= 100
1 <= N <= 10^8
```

Analysis

- Get the input value of T and N.
- Find all the jumping numbers less than or equal to N.
- Print the jumping numbers.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class JumpingNumbers {
    public static void jumpingNumber(int num) {
        for(int i=1; i<=num;i++){
            if (i<=9){
                System.out.print(i+" ");
                continue;
            }
            int temp = i;
            int val = 0;
            while(temp>0){
                int digit = temp%10;
                val = abs(digit - val);
                temp/=10;
            }
            if (val == 1){
                System.out.print(i+" ");
            }
        }
    }

    public static int abs(int a){
        if (a<0)
            return -a;
        else
            return a;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter any number: ");
        int num = sc.nextInt();
        jumpingNumber(num);
    }
}
```

Output:

```
Enter any number: 25
0 1 2 3 4 5 6 7 8 9 10 12 21 23
```

7. Minimum One Bit Operations to Make Integers Zero

Problem Statement:

Given an integer n, you must transform it into 0 using the following operations any number of times: Change the rightmost (0th) bit in the binary representation of n. Change the ith bit in the binary representatio
to 1 and the (i-2)th through 0th bits are set to 0. Return the minimum number of operations to transform n into 0.

Example 1:

```
Input: n = 3
Output: 2
```

> Explanation: The binary representation of 3 is "11".
>
> "11" -> "01" with the 2nd operation since the 0th bit is 1.
>
> "01" -> "00" with the 1st operation.

Example 2:

```
Input: n = 6
Output: 4
```

> Explanation: The binary representation of 6 is "110".
>
> "110" -> "010" with the 2nd operation since the 1st bit is 1 and 0th through 0th bits are 0.
>
> "010" -> "011" with the 1st operation.
>
> "011" -> "001" with the 2nd operation since the 0th bit is 1.
>
> "001" -> "000" with the 1st operation.

Constraints:

```
0 <= n <= 10^9
```

Analysis

- Get the input value of n.
- Find the minimum number of operations to transform n into 0.
- Return the minimum number of operations.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class MinimumOneBitOperationstoMakeIntegersZero {
    public static int minimumOneBitOperationMakeIntegerZero(int n) {
        int ans = 0;
        int k = 0;
        int mask = 1;
        while (mask <= n) {
            if ((n & mask) != 0) {
                ans = (1 << (k + 1)) - 1 - ans;
            }
            mask <<= 1;
            k++;
        }
        return ans;
    }

    public static int minimum1OneBitOperationMakeIntegerZero(int n){
        int ans = 0;
        while (n > 0){
            ans^=n;
            n = n >> 1;
        }
        return ans;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number: ");
        int num = sc.nextInt();
        System.out.println(minimum1OneBitOperationMakeIntegerZero(num));
    }
}
```

Output:

```
Enter the number: 3
2
```

---

8. Median of 2 Sorted Arrays of Different Sizes

Problem Statement :

Given two sorted arrays array1 and array2 of size m and n respectively. Find the median of the two sorted arrays.

Example 1:

```
Input:
m = 3, n = 4
array1[] = {1,5,9}
array2[] = {2,3,6,7}
Output: 5
```

> Explanation: The middle element for {1,2,3,5,6,7,9} is 5

Example 2:

```
Input:
m = 2, n = 4
array1[] = {4,6}
array2[] = {1,2,3,5}
Output: 3.5
```

Constraints:

```
0 ≤ m,n ≤ 10^6
1 ≤ array1[i], array2[i] ≤ 10^9
```

Analysis

- Get the input values of m, n, array1, and array2.
- Merge the two arrays and sort them.
- Find the median of the merged array.
- Return the median.

Code:

```
package com.hands_on;

import java.util.Arrays;
import java.util.Scanner;

public class MedianOf2SortedArraysOfDifferentSizes {
    public static void medianOf2SortedArraysOfDifferentSizes(int[] arr1, int[] arr2){
        int[] arrayFinal = new int[arr1.length+arr2.length];
        int l=0;
        for (int j : arr1) {
            arrayFinal[l++] = j;
        }
        for (int j : arr2) {
            arrayFinal[l++] = j;
        }
        Arrays.sort(arrayFinal);
        if (arrayFinal.length%2 != 0){
            System.out.println("The Median of 2 Sorted Array is "+arrayFinal[arrayFinal.length/2]);
        }else{
            double a = (double)arrayFinal.length/2;
            double b = a+1;
            double median = (a+b)/2;
            System.out.println("The Median of 2 Sorted Array is "+median);


        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array-1: ");
        int sizeOfArray1 = sc.nextInt();
        System.out.print("Enter the elements of array-1: ");
        int[] array1 = new int[sizeOfArray1];
        for(int i=0;i<sizeOfArray1;i++){
            array1[i] = sc.nextInt();
        }
        System.out.print("Enter the size of array-2: ");
        int sizeOfArray2 = sc.nextInt();
        System.out.print("Enter the elements of array-2: ");
        int[] array2 = new int[sizeOfArray2];
        for(int i=0;i<sizeOfArray2;i++){
            array2[i] = sc.nextInt();
        }
        medianOf2SortedArraysOfDifferentSizes(array1,array2);
    }
}
```

Output:

```
Enter the size of array-1: 3
Enter the elements of array-1: 1 5 9
Enter the size of array-2: 4
Enter the elements of array-2: 2 3 6 7
The Median of 2 Sorted Array is 5
```

9. Subarrays With Equal 0’s, 1’s and 2’s

Problem statement :

You are given an array containing ‘N’ integers. In the array, the elements are 0, 1 and 2. You have a simple task, find the count of non-empty subarrays containing an equal number of 0’s, 1’s and 2’s. The subarray c
of the array ARR, i. e. the array ARRi, ARRi+1, ... , ARRj for some 0 ≤ i ≤ j ≤ N - 1.

For Example :

```
If ‘N’ = 5, ‘ARR’ = {1, 1, 0, 2, 1}
There are exactly two subarrays that contain an equal number of 0’s, 1’s and
2’s.
The first subarray is from ‘ARR[1]’ to ‘ARR[3]’, ie: {1, 0, 2}.
The second subarray is from ‘ARR[2]’ to ‘ARR[4]’, ie: {0, 2, 1}.
```

Sample Input 1 :

```
2
5
1 1 0 2 1
4
1 1 0 0
```

Sample Output 1 :

```
2
0
```

> Explanation For Sample Input 1 :
>
> For test case 1 :
>
> We will print 2 because:
>
> There are exactly two subarrays that contain an equal number of 0’s, 1’s and 2’s.
>
> The first subarray is from ARR[1] to ARR[3], ie: {1, 0, 2}, and the second subarray is from ARR[2] to ARR[4], ie: {0, 2, 1}
>
> For test case 2 :
>
> We will print 0 because:
>
> The input array doesn’t contain any element equal to 2, so it’s impossible to
>
> form a non-empty subarray with an equal number of 0’s, 1’s and 2’s.

Sample Input 2 :

```
2
6
1 0 2 1 0 2
6
1 1 0 0 2 2
```

Sample Output 2 :

```
5
1
```

Constraints :

```
1 ≤ T ≤ 10
1 ≤ N ≤ 1000
ARR[i] = {0, 1, 2}
```

Analysis

- Get the input value of T and N.
- Find the count of non-empty subarrays containing an equal number of 0’s, 1’s, and 2’s.
- Print the count of subarrays.

Code:

```
package com.hands_on;

import java.util.HashMap;
import java.util.Scanner;

public class SubarraysWithEqual0s1sand2s {

    // Referred method -> working
    public static int arraysWithEqual012(int[] arr, int n){
        if (n < 3)
            return 0;
        int ans = 0;
        int c0 = 0, c1 = 0, c2 = 0;
        String key = 0 + "#" + 0;
        HashMap<String, Integer> map = new HashMap<>();
        map.put(key, 1);
        for(int num : arr){
            if(num == 0)
                c0++;
```

```java
            else if(num == 1)
                c1++;
            else
                c2++;
            key = (c2-c1) + "#" + (c1-c0);
            if(map.containsKey(key)){
                ans += map.get(key);
                map.put(key, map.get(key)+1);
            }else{
                map.put(key, 1);
            }
        }
        return ans;
    }

    // my method -> not working
    public static int subarraysWithEqual012(int[] arr, int n){
        int cnt = 0;
        int flag = 0;
        for(int i=0;i<n-2;i++){
            int c = 0;

            for(int j=i; j < i+3; j++){
                if (arr[j] == 1) {
                    flag = 1;
                    c++;
                }
                else if (arr[j] == 2) {
                    flag = 1;
                    c+=2;
                }
                else if (arr[j] == 0) {
                    flag = 1;
                    c+=3;
                }

            }
            if (flag == 1 && c == 6 ){
                cnt++;
            }
        }
        return cnt;

    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array: ");
        int n = sc.nextInt();
        System.out.print("Enter the elements of the array: ");
        int[] arr = new int[n];
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println(arraysWithEqual012(arr, n));
    }
}
```

Output:

```
Enter the size of array: 5
Enter the elements of the array: 1 1 0 2 1
2
```

---

10. Chocolate & Sweetness

Problem statement :

Nim wants to try some sweets in a sweet shop. There are ‘N’ types of sweets available at that shop. Each sweet has a sweetness level and expiry date. Nim is fond of sweets, so he sets a specific sweetness level ‘X’ sweets having a sweetness level greater than equal to ‘X’. But Nim also wants fresher sweets, so he only buys having expiry date strictly greater than ‘Y’. Can you help Nim to find the number of sweets suitable for two arrays, ‘SWEET’ and ‘EXPIRY’, both having ‘N’ values corresponding to the sweetness and expiry date of ith sweet. Nim will ask ‘Q’ queries with ‘X’ as the minimum sweetness level and ‘Y’ as the minimum exp answer all ‘Q’ queries and tell the number of sweets satisfying the given condition for each query.

For Example

```
For the given if N = ‘5’, ‘SWEET’ = [1,3,6,7,2] and ‘EXPIRY = [10,7,2,6,4].
And the query is ‘X’=2 and ‘Y’ =6 ,then the number of sweets satisfying the
condition is only 1 (having sweetness 3 and expiry date 7).
```

Sample Input 1:

```
2
5 2
1 3 6 7 2
10 7 2 6 4
2 6
3 9
3 1
1 5 7
11 7 10
3 6
```

Sample Output 1:

```
1 0
2
```

> Explanation of sample input 1: For the first test case, For the first query, the number of sweets having sweetness greater than 2 and expiry greater than 6 is only 1(Sweet number 4). For the second query, there
> the condition. Hence the answer is 0. Hence the answer is [1,0]. For the second test case: The number of sweets satisfying both the conditions is 2. (2nd and the 3rd sweet.) Hence the answer is [2].

Sample Input 2:

```
2
5 2
2 9 4 8 7
3 8 3 2 8
7 9
3 3
4 3
9 2 6 4
1 10 10 2
1 2
7 2
2 2
```

Sample Output 2:

```
0 2
2 0 2
```

Constraints:

```
1 <= T <= 10
1 <= N <= 100000.
1 <= Q <= 100000.
1 <= SWEET[i] <=100000
1 <= EXPIRY[i] <= 100000
```

Analysis

- Get the input value of T, N, Q, SWEET, and EXPIRY.
- Find the number of sweets satisfying the given condition for each query.
- Print the number of sweets for each query.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class ChocolateAndSweetness {
    public static void chocolateAndSweetness(int size, int query,
                                        int[] sweetness,
                                        int[] expiry){
        Scanner sc = new Scanner(System.in);
        for(int i=0; i<query; i++){
            System.out.println("Enter the value of x: ");
            int x = sc.nextInt();
            System.out.println("Enter the value of y: ");
            int y = sc.nextInt();
            int ans = verify(size, x, y, sweetness, expiry);
            System.out.println("For the query x="+x+" y="+y
                    +" is the available chocolate is "+ans);
        }
    }

    public static int verify(int size, int x,
                    int y, int[] sweetness, int[] expiry){
        int cnt=0;
        for( int i=0; i<size; i++){
            if ((sweetness[i] > x) && (expiry[i] > y)){
                cnt++;
            }
        }
        return cnt;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter any number: ");
        int t = sc.nextInt();
        while(t> 0){
            System.out.println("Enter the size of array: ");
            int size = sc.nextInt();
            System.out.println("Enter the number of query: ");
            int query = sc.nextInt();
            int[] sweetness = new int[size];
            int[] expiry = new int[size];
            System.out.println("Enter the sweetness of the chocolate: ");
            for(int i = 0; i < size; i++){
                sweetness[i] = sc.nextInt();
            }
            System.out.println("Enter the expiry of the chocolate: ");
            for(int i = 0; i < size; i++){
                expiry[i] = sc.nextInt();
            }
            chocolateAndSweetness(size, query, sweetness, expiry);
            t--;
        }
    }
}
```

Output:

```
Enter any number: 2
Enter the size of array: 5
Enter the number of query: 2
Enter the sweetness of the chocolate: 1 3 6 7 2
Enter the expiry of the chocolate: 10 7 2 6 4
Enter the value of x: 2
Enter the value of y: 6
For the query x=2 y=6 is the available chocolate is 1
Enter the value of x: 3
Enter the value of y: 9
For the query x=3 y=9 is the available chocolate is 0
Enter the size of array: 3
Enter the number of query: 1
Enter the sweetness of the chocolate: 1 5 7
Enter the expiry of the chocolate: 11 7 10
Enter the value of x: 3
Enter the value of y: 6
For the query x=3 y=6 is the available chocolate is 2
```

---

11. Smallest Positive Integer that cannot be represented as Sum

Problem Statement :

Given an array of size N, find the smallest positive integer value that is either not presented in the array or cannot be represented as a sum(coz sum means you are adding two or more elements) of some element

Example 1:

```
Input:
N = 6
array[] = {1, 10, 3, 11, 6, 15}
Output:
2
```

> Explanation: 2 is the smallest integer value that cannot be represented as sum of elements from the array.

Example 2:

```
Input:
N = 3
array[] = {1, 1, 1}
Output:
4
```

> Explanation: 1 is present in the array 2 can be created by combining two 1s. 3 can be created by combining three 1s. 4 is the smallest integer value that cannot be represented as sum of elements from the array

Constraints:

```
1 ≤ N ≤ 10^6
1 ≤ array[i] ≤ 10^9
The array may contain duplicates
```

Analysis

- Get the input value of N and array.
- Find the smallest positive integer value that is either not presented in the array or cannot be represented as a sum of some elements from the array.
- Return the smallest positive integer.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class SmallestPositiveIntegerThatCannotBeRepresentedAsSum {
    public static int smallestPositiveInteger(int[] arr, int n){
        int res = 1;
        for (int i = 0; i < n && arr[i] <= res; i++)
            res = res + arr[i];
        return res;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println(smallestPositiveInteger(arr, n));
    }
}
```

Output:

```
Enter the size of array: 3
Enter the elements of the array: 1 1 1
4
```

---

12. Candy

Problem Statement :

There are N children standing in a line. Each child is assigned a rating value given in the integer array ratings. You are giving candies to these children subjected to the following requirements:

- Each child must have atleast one candy.
- Children with a higher rating than its neighbors get more candies than their neighbors. Return the minimum number of candies you need to have to distribute.

Example 1:

```
Input:
N = 3
ratings = [1, 0, 2]
Output:
5
```

> Explanation: You can allocate to the first, second and third child with 2, 1, 2 candies respectively.

Example 2:

```
Input:
N = 3
ratings = [1, 2, 2]
Output:
4
```

> Explanation: You can allocate to the first, second and third child with 1, 2, 1 candies respectively. The third child gets 1 candy because it satisfies the above two conditions.

Constraints:

```
1 ≤ N ≤ 10^6
0 ≤ ratingsi ≤ 10^
```

Analysis

- Get the input value of N and ratings.
- Distribute the candies to the children subjected to the given requirements.
- Return the minimum number of candies.

Code:

```java
package com.hands_on;

import java.util.Arrays;
import java.util.Scanner;

public class Candy {
    public static int candy(int[] ratings, int n){
        int[] candies = new int[n];
        Arrays.fill(candies, 1);
        for (int i = 1; i < n; i++) {
            if (ratings[i] > ratings[i - 1]) {
                candies[i] = candies[i - 1] + 1;
            }
        }
        for (int i = n - 2; i >= 0; i--) {
            if (ratings[i] > ratings[i + 1]) {
                candies[i] = Math.max(candies[i], candies[i + 1] + 1);
            }
        }
        int sum = 0;
        for (int i = 0; i < n; i++) {
            sum += candies[i];
        }
        return sum;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array: ");
        int n = sc.nextInt();
        int[] ratings = new int[n];
        System.out.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            ratings[i] = sc.nextInt();
        }
        System.out.println(candy(ratings, n));
    }
}
```

Output:

```
Enter the size of array: 3
Enter the elements of the array: 1 2 2
4
```

---

13. Maximum Sum Rectangle

Problem statement :

You are given an M X N matrix of integers ARR. Your task is to find the maximum sum rectangle. Maximum sum rectangle is a rectangle with the maximum value for the sum of integers present within its boundary, rectangles that can be formed from the elements of that matrix. A rectangle is a 2-D polygon with opposite sides parallel and equal to each other.

For example:

Consider following matrix: 1 2 -1 -4 -20 -8 -3 4 2 1 3 8 10 1 3 -4 -1 1 7 -6 The rectangle (1,1) to (3,3) is the rectangle with the maximum sum, i.e. 29. 1 2 -1 -4 -20 -8 |-3 4 2 | 1 3 | 8 10 1 | 3 -4 |-1 1 7 | -6

Sample Input 1:

```
3
1 1
1
1 2
-1 1
2 2
1 -1
2 2
```

Sample Output 1:

```
1
1
4
```

> Explanation of Input 1:
>
> (i) Maximum sum rectangle is (0,0)-(0,0)
>
> (ii) Maximum sum rectangle is (0,1)-(0,1)
>
> (iii) Maximum sum rectangle is (1,0)-(1,1)

Sample Input 2:

```
3
1 2
0 0
2 2
1 0
0 1
4 5
1 2 -1 -4 -20
-8 -3 4 2 1
3 8 10 1 3
-4 -1 1 7 -6
```

Sample Output 2:

```
0
2
29
```

> Explanation of Input 2:
>
> (i) Maximum sum rectangle is (0,0)-(0,1)
>
> (ii) Maximum sum rectangle is (0,0)-(1,1)
>
> (iii) Maximum sum rectangle is (1,1)-(3,3)

Constraints:

```
1 <= T <= 10
1 <= M, N <= 100
-10^5 <= ARR[i] <=10^5
```

Analysis

- Get the input value of T, M, N, and ARR.
- Find the maximum sum rectangle.
- Return the maximum sum.

Code:

```
package com.hands_on;

import java.util.Scanner;

public class MaximumSumRectangle {
```

```java
    public static int maximumSumRectangle(int[][] arr, int m, int n){
        int maxSum = Integer.MIN_VALUE;
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = i; k < m; k++) {
                    for (int l = j; l < n; l++) {
                        int sum = 0;
                        for (int p = i; p <= k; p++) {
                            for (int q = j; q <= l; q++) {
                                sum += arr[p][q];
                            }
                        }
                        maxSum = Math.max(maxSum, sum);
                    }
                }
            }
        }
        return maxSum;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array: ");
        int t = sc.nextInt();
        while(t>0){
            System.out.print("Enter the size of row: ");
            int m = sc.nextInt();
            System.out.print("Enter the size of column: ");
            int n = sc.nextInt();
            int[][] arr = new int[m][n];
            System.out.print("Enter the elements of the array: ");
            for (int i = 0; i < m; i++) {
                for (int j = 0; j < n; j++) {
                    arr[i][j] = sc.nextInt();
                }
            }
            System.out.println(maximumSumRectangle(arr, m, n));
            t--;
        }
    }
}
```

Output:

```
Enter the size of array: 1
Enter the size of row: 3
Enter the size of column: 3
Enter the elements of the array:
1 2 -1
1 2 -1
1 2 -1
4
```

---

14. Sum of Floored Pairs

Problem Statement :

Given an integer array nums, return the sum of floor(nums[i] / nums[j]) for all pairs of indices 0 <= i, j < nums.length in the array. Since the answer may be too large, return it modulo 10^9 + 7. The floor() function r
the division.

Example 1:

```
Input: nums = [2,5,9]
Output: 10
```

> Explanation:
>
> floor(2 / 5) = floor(2 / 9) = floor(5 / 9) = 0
>
> floor(2 / 2) = floor(5 / 5) = floor(9 / 9) = 1
>
> floor(5 / 2) = 2
>
> floor(9 / 2) = 4
>
> floor(9 / 5) = 1
>
> We calculate the floor of the division for every pair of indices in the array then sum them up.

Example 2:

```
Input: nums = [7,7,7,7,7,7,7]
Output: 49
```

Constraints:

```
1 <= nums.length <= 10^5
1 <= nums[i] <= 10^5
```

Analysis

- Get the input value of nums.

- Find the sum of floor(nums[i] / nums[j]) for all pairs of indices 0 <= i, j < nums.length in the array.
- Return the sum modulo 10^9 + 7.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class SumOfFlooredPairs {
    public static int sumOfFlooredPairs(int[] nums){
        int mod = 1000000007;
        int max = 0;
        for (int num : nums) {
            max = Math.max(max, num);
        }
        int[] count = new int[max + 1];
        for (int num : nums) {
            count[num]++;
        }
        int[] prefix = new int[max + 1];
        for (int i = 1; i <= max; i++) {
            prefix[i] = prefix[i - 1] + count[i];
        }
        long res = 0;
        for (int i = 1; i <= max; i++) {
            if (count[i] == 0) {
                continue;
            }
            for (int j = i; j <= max; j += i) {
                res += (long) count[i] * (prefix[Math.min(max, j + i - 1)] - prefix[j - 1]) * (j / i);
            }
        }
        return (int) (res % mod);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array: ");
        int n = sc.nextInt();
        int[] nums = new int[n];
        System.out.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            nums[i] = sc.nextInt();
        }
        System.out.println(sumOfFlooredPairs(nums));
    }
}
```

Output:

```
Enter the size of array: 3
Enter the elements of the array: 2 5 9
10
```

---

15. Maximum subset XOR

Problem Statement :

Given an array arr[] of N positive integers. Find an integer denoting the maximum XOR subset value in the given array arr[].

Example 1:

```
Input :
N = 3
arr[] = {2, 4, 5}
Output : 7
```

> Explanation : The subset {2, 5} has maximum subset XOR value.

Example 2 :

```
Input :
N= 3
arr[] = {9, 8, 5}
Output : 13
```

> Explanation : The subset {8, 5} has maximum subset XOR value.

Contraints :

```
1 <= N <= 10^5
1 <= arr[i] <= 10^6
```

Analysis

- Get the input value of N and arr.
- Find an integer denoting the maximum XOR subset value in the given array arr[].

- Return the maximum XOR subset value.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class MaximumSubsetXOR {
    public static int maximumSubsetXOR(int[] arr, int n){
        int index = 0;
        for (int i = 31; i >= 0; i--) {
            int maxElement = Integer.MIN_VALUE;
            int maxIndex = index;
            for (int j = index; j < n; j++) {
                if ((arr[j] & (1 << i)) != 0 && arr[j] > maxElement) {
                    maxElement = arr[j];
                    maxIndex = j;
                }
            }
            if (maxElement == Integer.MIN_VALUE) {
                continue;
            }
            int temp = arr[index];
            arr[index] = arr[maxIndex];
            arr[maxIndex] = temp;
            maxIndex = index;
            for (int j = 0; j < n; j++) {
                if (j != maxIndex && (arr[j] & (1 << i)) != 0) {
                    arr[j] = arr[j] ^ arr[maxIndex];
                }
            }
            index++;
        }
        int res = 0;
        for (int i = 0; i < n; i++) {
            res ^= arr[i];
        }
        return res;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println(maximumSubsetXOR(arr, n));
    }
}
```

Output:

```
Enter the size of array: 3
Enter the elements of the array: 9 8 5
13
```

---

16. Poor Pigs

Problem Statement:

There are buckets buckets of liquid, where exactly one of the buckets is poisonous. To figure out which one is poisonous, you feed some number of (poor) pigs the liquid to see whether they will die or not. Unfort
minutesToTest minutes to determine which bucket is poisonous. You can feed the pigs according to these steps: Choose some live pigs to feed. For each pig, choose which buckets to feed it. The pig will consume
simultaneously and will take no time. Each pig can feed from any number of buckets, and each bucket can be fed from by any number of pigs. Wait for minutesToDie minutes. You may not feed any other pigs duri
minutesToDie minutes have passed, any pigs that have been fed the poisonous bucket will die, and all others will survive. Repeat this process until you run out of time Given buckets, minutesToDie, and minutesT
number of pigs needed to figure out which bucket is poisonous within the allotted time.

Example 1:

```
Input: buckets = 4, minutesToDie = 15, minutesToTest = 15
Output: 2
```

> Explanation: We can determine the poisonous bucket as follows: At time 0, feed the first pig buckets 1 and 2, and feed the second pig buckets 2 and 3. At time 15, there are 4 possible outcomes:

- If only the first pig dies, then bucket 1 must be poisonous.
- If only the second pig dies, then bucket 3 must be poisonous.
- If both pigs die, then bucket 2 must be poisonous.
- If n either pig dies, then bucket 4 must be poisonous.

Example 2:

```
Input: buckets = 4, minutesToDie = 15, minutesToTest = 30
Output: 2
```

> Explanation: We can determine the poisonous bucket as follows: At time 0, feed the first pig bucket 1, and feed the second pig bucket 2. At time 15, there are 2 possible outcomes:

- If either pig dies, then the poisonous bucket is the one it was fed.

- If neither pig dies, then feed the first pig bucket 3, and feed the second pig bucket 4. At time 30, one of the two pigs must die, and the poisonous bucket is the one it was fed.

Constraints:

```
1 <= buckets <= 1000
1 <= minutesToDie <= minutesToTest <= 100
```

Analysis

- Get the input value of buckets, minutesToDie, and minutesToTest.
- Return the minimum number of pigs needed to figure out which bucket is poisonous within the allotted time.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class PoorPigs {
    public static int poorPigs(int buckets, int minutesToDie, int minutesToTest){
        int pigs = 0;
        while (Math.pow((minutesToTest / minutesToDie + 1), pigs) < buckets) {
            pigs++;
        }
        return pigs;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of buckets: ");
        int buckets = sc.nextInt();
        System.out.print("Enter the minutes to die: ");
        int minutesToDie = sc.nextInt();
        System.out.print("Enter the minutes to test: ");
        int minutesToTest = sc.nextInt();
        System.out.println(poorPigs(buckets, minutesToDie, minutesToTest));
    }
}
```

Output:

```
Enter the number of buckets: 4
Enter the minutes to die: 15
Enter the minutes to test: 15
2
```

---

17. Count Array Pairs Divisible by K

Problem Statement :

Given a 0-indexed integer array nums of length n and an integer k, return the number of pairs (i, j) such that:

- 0 <= i < j <= n - 1 and
- nums[i] * nums[j] is divisible by k.

Example 1:

```
Input: nums = [1,2,3,4,5], k = 2
Output: 7
```

> Explanation: The 7 pairs of indices whose corresponding products are divisible by 2 are (0, 1), (0, 3), (1, 2), (1, 3), (1, 4), (2, 3), and (3, 4). Their products are 2, 4, 6, 8, 10, 12, and 20 respectively. Other pairs such a
> products 3 and 15 respectively, which are not divisible by 2.

Example 2:

```
Input: nums = [1,2,3,4], k = 5
Output: 0
```

> Explanation: There does not exist any pair of indices whose corresponding product is divisible by 5.

Constraints:

```
1 <= nums.length <= 10^5
1 <= nums[i], k <= 10^5
```

Analysis

- Get the input value of nums and k.
- Return the number of pairs (i, j) such that nums[i] * nums[j] is divisible by k.

Code:

```
package com.hands_on;

import java.util.Scanner;
```

```
public class CountArrayPairsDivisibleByK {
    public static int countArrayPairsDivisibleByK(int[] nums, int n, int k){
        int count = 0;
        for (int i = 0; i < nums.length; i++) {
            for (int j = i + 1; j < nums.length; j++) {
                if ((nums[i] * nums[j]) % k == 0) {
                    count++;
                }
            }
        }
        return count;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array: ");
        int n = sc.nextInt();
        int[] nums = new int[n];
        System.out.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            nums[i] = sc.nextInt();
        }
        System.out.print("Enter the value of k: ");
        int k = sc.nextInt();
        System.out.println(countArrayPairsDivisibleByK(nums, n, k));
    }
}
```

Output:

```
Enter the size of array: 5
Enter the elements of the array: 1 2 3 4 5
Enter the value of k: 2
7
```

18. Max min Height

Problem Statement :

You have a garden with n flowers lined up in a row. The height of ith flower is ai units. You will water them for k days. In one day you can water w continuous flowers (you can do this only once in a single day). Whe
height increases by 1 unit. You need to maximize the height of the smallest flower all the time.

Example 1:

```
Input:
n=6
k=2
w=3
a[]={2,2,2,2,1,1}
Output:
2
```

> Explanation: Water last three flowers for first day & first three flowers for second day.The new heights will be {3,3,3,3,2,2}

Example 2:

```
Input:
n=2
k=5
w=1
a[]={5,8}
Output:
9
```

> Explanation: For the first four days water the first flower then water the last flower once.

Constraints:

```
1 <= n<= 10^5
1<=w<=n
1<=k<=10^5
1 <= a[i] <= 10^9
```

Analysis

- Get the input value of n, k, w, and a.
- Maximize the height of the smallest flower all the time.
- Return the maximum height of the smallest flower.

Code:

```
package com.hands_on;

import java.util.Scanner;

public class MaxMinHeight {
    private static long maxMinHeight(int n, int k, int w, int[] a) {
        int high = Integer.MAX_VALUE / 4;
        int low = 0;
        long ans = -1;
        while(low <= high){
            int mid = low + (high - low) / 2;
            if(isPossible(a , w , mid , k)){
                low = mid + 1;
                ans = (long)Math.max((long)mid , ans);
            }else{
                high = mid - 1;
            }
        }
        return ans;
    }

    private static boolean isPossible(int[] arr , int l , int maxHeight , int days){
        int[] water = new int[arr.length];
        for(int i = 0 ; i < arr.length ; i++){
            if(i > 0){
                water[i] = water[i-1];
            }
            int curHei = water[i] + arr[i];
            if(i >= l){
                curHei -= (water[i-l]);
            }
            if(curHei < maxHeight){
                water[i] += (maxHeight - curHei);
                days -= (maxHeight - curHei);
            }
            if(days < 0) return false;
        }
        return true;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array: ");
        int n = sc.nextInt();
        System.out.print("Enter the value of k: ");
        int k = sc.nextInt();
        System.out.print("Enter the value of w: ");
        int w = sc.nextInt();
        int[] a = new int[n];
        System.out.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            a[i] = sc.nextInt();
        }
        System.out.println(maxMinHeight(n, k, w, a));
    }
}
```

Output:

```
Enter the size of array: 6
Enter the value of k: 2
Enter the value of w: 3
Enter the elements of the array: 2 2 2 2 1 1
2
```

---

19. Split Array Largest Sum

Problem Statement :

Given an array arr[] of N elements and a number K., split the given array into K subarrays such that the maximum subarray sum achievable out of K subarrays formed is minimum possible. Find that possible suba

Example 1:

```
Input:
N = 4, K = 3
arr[] = {1, 2, 3, 4}
Output: 4
```

> Explanation: Optimal Split is {1, 2}, {3}, {4}. Maximum sum of all subarrays is 4, which is minimum possible for 3 splits.

Example 2:

```
Input:
N = 3, K = 2
A[] = {1, 1, 2}
Output:
2
```

> Explanation: Splitting the array as {1,1} and {2} is optimal. This results in a maximum sum subarray of 2.

Constraints:

```
1 ≤ N ≤ 10^5
1 ≤ K ≤ N
1 ≤ arr[i] ≤ 10^4
```

Analysis

- Get the input value of N, K, and arr.
- Split the given array into K subarrays such that the maximum subarray sum achievable out of K subarrays formed is minimum possible.
- Find that possible subarray sum.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class SplitArrayLargestSum {
    public static int splitArrayLargestSum(int[] arr, int n, int k){
        int low = 0;
        int high = 0;
        for (int i = 0; i < n; i++) {
            high += arr[i];
            low = Math.max(low, arr[i]);
        }
        while (low < high) {
            int mid = low + (high - low) / 2;
            int sum = 0;
            int count = 1;
            for (int i = 0; i < n; i++) {
                sum += arr[i];
                if (sum > mid) {
                    sum = arr[i];
                    count++;
                }
            }
            if (count <= k) {
                high = mid;
            } else {
                low = mid + 1;
            }
        }
        return low;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array: ");
        int n = sc.nextInt();
        System.out.print("Enter the value of k: ");
        int k = sc.nextInt();
        int[] arr = new int[n];
        System.out.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println(splitArrayLargestSum(arr, n, k));
    }
}
```

Output:

```
Enter the size of array: 4
Enter the value of k: 3
Enter the elements of the array: 1 2 3 4
4
```

---

20. Max Circular Subarray Sum

Problem Statement :

Given an array arr[] of N integers arranged in a circular fashion. Your task is to find the maximum contiguous subarray sum

Example 1:

```
Input:
N = 7
arr[] = {8,-8,9,-9,10,-11,12}
Output:
22
```

> Explanation: Starting from the last element of the array, i.e, 12, and moving in a circular fashion, we have max subarray as 12, 8, -8, 9, -9, 10, which gives maximum sum as 22.

Example 2:

```
Input:
N = 8
arr[] = {10,-3,-4,7,6,5,-4,-1}
Output:
23
```

> Explanation: Sum of the circular subarray with maximum sum is 23

Constraints:

```
1 <= N <= 10^6
-10^6 <= Arr[i] <= 10^6
```

Sample Code:

```
static int normalMaxSum(int a[],int n){
    int res=a[0],maxEnd=a[0];
    for(int i=1;i<n;i++){
        maxEnd=Math.max(maxEnd+a[i],a[i]);
        res=Math.max(res,maxEnd);
    }
    return res;
}
static int circularSubarraySum(int a[], int n) {
    int max_normal=normalMaxSum(a,n);
    if(max_normal<0){
        return max_normal;
    }
    int arr_sum=0;
    for(int i=0;i<n;i++){
        arr_sum+=a[i];
        a[i]=-a[i];
    }
    int maxCircular=arr_sum+normalMaxSum(a,n);
    return Math.max(maxCircular,max_normal);
}
```

Analysis

- Get the input value of N and arr.
- Find the maximum contiguous subarray sum.
- Return the maximum contiguous subarray sum.

Code:

```
package com.hands_on;

import java.util.Scanner;

public class MaxCircularSubarraySum {
    public static int normalMaxSum(int[] a, int n){
        int res = a[0];
        int maxEnd = a[0];
        for (int i = 1; i < n; i++) {
            maxEnd = Math.max(maxEnd + a[i], a[i]);
            res = Math.max(res, maxEnd);
        }
        return res;
    }
    public static int circularSubarraySum(int[] a, int n) {
        int max_normal = normalMaxSum(a, n);
        if (max_normal < 0) {
            return max_normal;
        }
        int arr_sum = 0;
        for (int i = 0; i < n; i++) {
            arr_sum += a[i];
            a[i] = -a[i];
        }
        int maxCircular = arr_sum + normalMaxSum(a, n);
        return Math.max(maxCircular, max_normal);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.print("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println(circularSubarraySum(arr, n));
    }
}
```

Output:

```
Enter the size of array: 7
Enter the elements of the array: 8 -8 9 -9 10 -11 12
22
```

21.Mining Diamonds

Problem statement :

There are ‘N’ diamonds in a mine. The size of each diamond is given in the form of integer array ‘A’. If the miner mines a diamond, then he gets 'size of previous unmined diamond * size of currently mined diamon
diamond' number of coins. If there isn’t any next or previous unmined diamond then their size is replaced by 1 while calculating the number of coins.Vladimir, a dumb miner was assigned the task to mine all diar
asks for your help to determine the maximum number of coins that he can earn by mining the diamonds in an optimal order.

For example:

```
Suppose ‘N’ = 3, and ‘A’ = [7, 1, 8]
The optimal order for mining diamonds will be [2, 1, 3].
State of mine - [7, 1, 8] [7, 8] [8]
Coins earned - (7*1*8) + (1*7*8) + (1*8*1) = 56 + 56 + 8 = 120
Hence output will be 120.
```

Sample Input 1 :

```
2
3
7 1 8
2
9 1
```

Sample Output 1 :

```
120
18
```

> Explanation For Sample Input 1 :
>
> For First Case - Same as explained in above example.
>
> For the second case - ‘N’ = 2, and ‘A’ = [9, 1]
>
> The optimal order for mining diamonds will be [2, 1].
>
> State of mine - [9, 1] [9]
>
> Coins earned - (1*9*1) + (1*9*1) = 9 + 9 = 18
>
> Hence output will be 18..

Sample Input 2 :

```
2
5
1 2 3 4 5
4
1 5 2 8
```

Sample Output 2 :

```
110
136
```

Analysis

- Get the input value of T, N, and A.
- Determine the maximum number of coins that he can earn by mining the diamonds in an optimal order.
- Return the maximum number of coins.

Code:

```
package com.hands_on;

import java.util.Scanner;

public class MiningDiamonds {
    public static int miningDiamonds(int[] a, int n){
        int[] b = new int[n+2];
        b[0] = b[n+1] = 1;
        for (int i = 1; i <= n; i++) {
            b[i] = a[i-1];
        }
        int[][] dp = new int[n+2][n+2];
        for (int i = n; i >= 1; i--) {
            for (int j = i; j <= n; j++) {
                for (int k = i; k <= j; k++) {
                    dp[i][j] = Math.max(dp[i][j], dp[i][k-1] + dp[k+1][j] + b[i-1] * b[k] * b[j+1]);
                }
            }
        }
        return dp[1][n];
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the value of T: ");
        int t = sc.nextInt();
        while(t>0){
            System.out.print("Enter the size of array: ");
            int n = sc.nextInt();
            int[] a = new int[n];
            System.out.print("Enter the elements of the array: ");
            for (int i = 0; i < n; i++) {
                a[i] = sc.nextInt();
            }
            System.out.println(miningDiamonds(a, n));
            t--;
        }
    }
}
```

Output:

```
Enter the value of T: 2
Enter the size of array: 3
Enter the elements of the array: 7 1 8
120
Enter the size of array: 2
Enter the elements of the array: 9 1
18
```

---

22. Matrix Exponentiation

Problem statement :

Sai is playing a game, he is standing on index 1 in a linear array of size ‘N’, then he rolls a dice and moves forward according to the number on the top of the dice. He repeats this operation until he reached index ‘N
expected number of moves required to reach position ‘N’ starting from ‘1’.

> Note:
>
> 1. The expected number of moves can be fractional.
> 2. Sai cannot go outside the array i.e if he is at (n - 1)-th position he can only move if he gets 1 as an outcome in dice.

Sample Input 1:

```
2
2
8
```

Sample Output 1:

```
1
1
```

> Explanation of Sample Input 1:
>
> Test case 1: Here the answer is equal to the expected number of dice rolls to get the first 1 as an outcome because we cannot go outside the array. And the expected number of dice rolls to get the first '1' is 6. i.
> roll a dice 6 times to get at position 2.
>
> Test case 2. If we are 7 steps away, then we can get from 1 by getting '6' with probability (1 / 6) and expected dice roll equal to 6, similarly, from 2 we can get 5, from 3 by 4, and so on. with all same probability i.
> dice roll 6. Now the average value will be 1 + ((6 * 6) / 6 = 7. Hence, for 7 steps away, our answer is 7.

Sample Input 2:

```
1
3
```

Sample Output 2:

```
1
```

> Explanation of Sample Input 2:
>
> Test case 1: As the Input is less than 7, the expected number of operations will be 6.

Constraints:

```
1 <= T <= 5
1 <= N <= 10^9
```

Analysis

- Get the input value of T and N.
- Find the expected number of moves required to reach position ‘N’ starting from ‘1’.
- Return the expected number of moves.

Code:

```
package com.hands_on;

import java.util.*;

public class MatrixExponentiation {
    public static double [][] matrixProduct(double [][] a, double [][] b)
    {
        double [][] temp = new double[7][7];
        for (int i = 0; i < 7; i++) {
            for (int j = 0; j < 7; j++) {
                for (int k = 0; k < 7; k++) {
                    temp[i][j] += a[i][k] * b[k][j];
                }
            }
        }
        return temp;
    }

    public static double [][] matExpo(double [][] mul, int n) {
```

```java
        double [][] s = {{ 1, 0, 0, 0, 0, 0, 0 },
                { 0, 1, 0, 0, 0, 0, 0 },
                { 0, 0, 1, 0, 0, 0, 0 },
                { 0, 0, 0, 1, 0, 0, 0 },
                { 0, 0, 0, 0, 1, 0, 0 },
                { 0, 0, 0, 0, 0, 1, 0 },
                { 0, 0, 0, 0, 0, 0, 1 }};
        while (n != 1) {
            if (n % 2 == 1) {
                s = matrixProduct(s, mul);
            }
            mul = matrixProduct(mul, mul);
            n /= 2;
        }
        return matrixProduct(mul, s);
    }

    public static double matrixExponentiation(int n) {
        n = n - 1;
        if(n <= 6) {
            return 6.00;
        }
        double [][]mul = {{(double)7 / 6, 1, 0, 0, 0, 0, 0 },
                { 0, 0, 1, 0, 0, 0, 0 },
                { 0, 0, 0, 1, 0, 0, 0 },
                { 0, 0, 0, 0, 1, 0, 0 },
                { 0, 0, 0, 0, 0, 1, 0 },
                { 0, 0, 0, 0, 0, 0, 1 },
                {(double) - 1 / 6, 0, 0, 0, 0, 0, 0 }};
        mul = matExpo(mul, n - 6);
        double ans = (mul[0][0] + mul[1][0] + mul[2][0] + mul[3][0] + mul[4][0] + mul[5][0]) * 6;
        return ans;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        int t = scanner.nextInt();
        while (t-- > 0) {
            int n = scanner.nextInt();
            if (n <= 6) {
                System.out.println("1");
            } else {
                double expectedMoves = matrixExponentiation(n);
                System.out.println((int)expectedMoves%6);
            }
        }
        scanner.close();
    }
}
```

Output:

```
2
2
8
1
1
```

---

23. Maximum Building Height

Problem Statement :

You want to build n new buildings in a city. The new buildings will be built in a line and are labeled from 1 to n. However, there are city restrictions on the heights of the new buildings:

- The height of each building must be a non-negative integer.
- The height of the first building must be 0.
- The height difference between any two adjacent buildings cannot exceed 1.

Additionally, there are city restrictions on the maximum height of specific buildings. These restrictions are given as a 2D integer array restrictions where restrictions[i] = [idi, maxHeighti] indicates that building idi than or equal to maxHeighti. It is guaranteed that each building will appear at most once in restrictions, and building 1 will not be in restrictions. Return the maximum possible height of the tallest building.

Example 1:

```
Input: n = 5, restrictions = [[2,1],[4,1]]
Output: 2
```

> Explanation: The green area in the image indicates the maximum allowed height for each building. We can build the buildings with heights [0,1,2,1,2], and the tallest building has a height of 2.

Example 2:

```
Input: n = 6, restrictions = []
Output: 5
```

> Explanation: The green area in the image indicates the maximum allowed height for each building. We can build the buildings with heights [0,1,2,3,4,5], and the tallest building has a height of 5.

Constraints:

```
2 <= n <= 10^9
0 <= restrictions.length <= min(n - 1, 10^5)
2 <= idi <= n
idi is unique.
0 <= maxHeighti <= 10^9
```

Analysis

- Get the input value of n and restrictions.
- Return the maximum possible height of the tallest building.

Code:

```java
package com.hands_on;

import java.util.*;

public class MaximumBuildingHeight {
    public static int maxBuilding(int n, int[][] restrictions) {
        Arrays.sort(restrictions, (a, b) -> Integer.compare(a[0], b[0]));
        int prevInd = 1, prevH = 0;
        for (int i=0; i<restrictions.length; i++) {
            restrictions[i][1] = Math.min(restrictions[i][1], prevH + (restrictions[i][0] - prevInd));
            prevInd = restrictions[i][0];
            prevH = restrictions[i][1];
        }
        for (int i=restrictions.length-2; i>=0; i--) {
            restrictions[i][1] = Math.min(restrictions[i][1], restrictions[i+1][1] + (restrictions[i+1][0] - restrictions[i][0]));
        }

        int ph = 0, pInd = 1, highest = 0;
        for (int i=0; i<restrictions.length; i++) {
            int ind = restrictions[i][0];
            int h = restrictions[i][1];
            if (ph < h) {
                h = Math.min(h, ph + (ind - pInd));
                int remains = Math.max(0, (ind-pInd) - (h - ph));
                highest = Math.max(highest, h + remains/2);
            } else {
                int remains = (ind-pInd) - (ph-h);
                highest = Math.max(highest, ph + remains/2);
            }
            ph = h;
            pInd = ind;;
        }
        highest = Math.max(highest, ph + (n-pInd));
        return highest;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the value of n: ");
        int n = sc.nextInt();
        System.out.print("Enter the number of restrictions: ");
        int m = sc.nextInt();
        int[][] restrictions = new int[m][2];
        for (int i = 0; i < m; i++) {
            System.out.print("Enter the building id and max height: ");
            restrictions[i][0] = sc.nextInt();
            restrictions[i][1] = sc.nextInt();
        }
        System.out.println(maxBuilding(n, restrictions));
    }
}
```

Output:

```
Enter the value of n: 5
Enter the number of restrictions: 2
Enter the building id and max height: 2 1
Enter the building id and max height: 4 1
2
```

---

24. Perfect Rectangle

Problem Statement :

Given an array rectangles where rectangles[i] = [xi, yi, ai, bi] represents an axis-aligned rectangle. The bottom-left point of the rectangle is (xi, yi) and the top-right point of it is (ai, bi).Return true if all the rectangle cover of a rectangular region.

Example 1:

```
Input: rectangles = [[1,1,3,3],[3,1,4,2],[3,2,4,4],[1,3,2,4],[2,3,3,4]]
Output: true
```

> Explanation: All 5 rectangles together form an exact cover of a rectangular region.

Example 2:

```
Input: rectangles = [[1,1,2,3],[1,3,2,4],[3,1,4,2],[3,2,4,4]]
Output: false
```

> Explanation: Because there is a gap between the two rectangular regions.

Constraints:

```
1 <= rectangles.length <= 2 * 10^4
rectangles[i].length == 4
-10^5 <= xi, yi, ai, bi <= 10^5
```

Analysis

- Get the input value of rectangles.
- Return true if all the rectangles together form an exact cover of a rectangular region.

Code:

```
package com.hands_on;

import java.util.Scanner;

public class PerfectRectangle {
    public static boolean isRectangleCover(int[][] rectangles) {
        int x1 = Integer.MAX_VALUE, x2 = Integer.MIN_VALUE, y1 = Integer.MAX_VALUE, y2 = Integer.MIN_VALUE;
        int area = 0;
        for (int[] rect : rectangles) {
            x1 = Math.min(x1, rect[0]);
            y1 = Math.min(y1, rect[1]);
            x2 = Math.max(x2, rect[2]);
            y2 = Math.max(y2, rect[3]);
            area += (rect[2] - rect[0]) * (rect[3] - rect[1]);
        }
        int expectedArea = (x2 - x1) * (y2 - y1);
        if (area != expectedArea) {
            return false;
        }
        boolean[] corners = new boolean[4];
        for (int[] rect : rectangles) {
            if (isValidCorner(rect[0], rect[1], x1, y1, corners, 0)) {
                return false;
            }
            if (isValidCorner(rect[0], rect[3], x1, y2, corners, 1)) {
                return false;
            }
            if (isValidCorner(rect[2], rect[1], x2, y1, corners, 2)) {
                return false;
            }
            if (isValidCorner(rect[2], rect[3], x2, y2, corners, 3)) {
                return false;
            }
        }
        return true;
    }

    private static boolean isValidCorner(int x, int y, int x1, int y1, boolean[] corners, int index) {
        if (x == x1 && y == y1) {
            if (corners[index]) {
                return true;
            }
            corners[index] = true;
        }
        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of rectangles: ");
        int n = sc.nextInt();
        int[][] rectangles = new int[n][4];
        for (int i = 0; i < n; i++) {
            System.out.print("Enter the coordinates of rectangle " + (i+1) + ": ");
            for (int j = 0; j < 4; j++) {
                rectangles[i][j] = sc.nextInt();
            }
        }
        System.out.println(isRectangleCover(rectangles));
    }
}
```

Output:

```
Enter the number of rectangles: 5
Enter the coordinates of rectangle 1: 1 1 3 3
Enter the coordinates of rectangle 2: 3 1 4 2
Enter the coordinates of rectangle 3: 3 2 4 4
Enter the coordinates of rectangle 4: 1 3 2 4
Enter the coordinates of rectangle 5: 2 3 3 4
true
```

---

25. Print Spiral

Problem statement :

For a given two-dimensional integer array/list of size (N x M), print it in a spiral form. That is, you need to print in the order followed for every iteration: a. First row(left to right) b. Last column(top to bottom) c. Las
column(bottom to top) Mind that every element will be printed only once. Refer to the Image: ####### Sample Input 1: 1 4 4 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Sample Output 1:

```
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
```

Sample Input 2:

```
2
3 3
1 2 3
4 5 6
7 8 9
3 1
10
20
30
```

Sample Output 2:

```
1 2 3 6 9 8 7 4 5
10 20 30
```

Constraints :

```
1 <= t <= 10^2
0 <= N <= 10^3
0 <= M <= 10^3
```

Analysis

- Get the input value of t, N, and M.
- Print the two-dimensional integer array/list in a spiral form.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class PrintSpiral {
    public static void printSpiral(int[][] arr, int n, int m) {
        int top = 0, bottom = n - 1, left = 0, right = m - 1;
        while (top <= bottom && left <= right) {
            for (int i = left; i <= right; i++) {
                System.out.print(arr[top][i] + " ");
            }
            top++;
            for (int i = top; i <= bottom; i++) {
                System.out.print(arr[i][right] + " ");
            }
            right--;
            if (top <= bottom) {
                for (int i = right; i >= left; i--) {
                    System.out.print(arr[bottom][i] + " ");
                }
                bottom--;
            }
            if (left <= right) {
                for (int i = bottom; i >= top; i--) {
                    System.out.print(arr[i][left] + " ");
                }
                left++;
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the value of t: ");
        int t = sc.nextInt();
        while (t-- > 0) {
            System.out.print("Enter the value of N and M: ");
            int n = sc.nextInt();
            int m = sc.nextInt();
            int[][] arr = new int[n][m];
            System.out.println("Enter the elements of the array: ");
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < m; j++) {
                    arr[i][j] = sc.nextInt();
                }
            }
            printSpiral(arr, n, m);
            System.out.println();
        }
    }
}
```

Output:

```
Enter the value of t: 1
Enter the value of N and M: 4 4
Enter the elements of the array:
1 2 3 4
5 6 7 8
```

```
9 10 11 12
13 14 15 16
1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
```

---

26. Maximum Number of Darts Inside of a Circular Dartboard

Problem Statement :

Maximum Number of Darts Inside of a Circular Dartboard Problem Statement : Alice is throwing n darts on a very large wall. You are given an array darts where darts[i] = [xi, yi] is the position of the ith dart that Ali knows the positions of the n darts on the wall. He wants to place a dartboard of radius r on the wall so that the maximum number of darts that Alice throws lie on the dartboard. Given the integer r, return the max can lie on the dartboard

Analysis :

- Get the input value of r.
- Return the maximum number of darts that can lie on the dartboard.

Code :

```
package com.hands_on;

import java.util.Scanner;

public class MaximumDart {
    private static int maximumDart(int r, int rows, int cols, int[][] arr){
        int count = 0;
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if(Math.abs(arr[i][j])==r){
                    count++;
                    break;
                }
            }
        }
        return count;
    }
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        int r = input.nextInt();
        int rows = input.nextInt();
        int cols = input.nextInt();
        int[][] arr = new int[rows][cols];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                arr[i][j] = input.nextInt();
            }
        }
        System.out.println(maximumDart(r,rows, cols, arr));
    }
}
```

Ouput :

```
2
4 2
-2 0
2 0
0 2
0 -2
4
```

---

27. Paint House

Problem statement :

Alex has started a painting business recently. He got a contract to paint 'N' houses in a city. Alex has 'K' colors to choose from. But the client has a condition that no two adjacent houses have the same color. The
with a certain color is represented by an N x K cost matrix. For example, costs[0][0] is the cost of painting house 0 with color 0; costs[1][2] is the cost of painting house 1 with color 2, and so on. Your task is to help
cost required to paint houses according to this condition. For Example : Let say N = 2 and K = 3 and costs = [[1,5,3] , [2,9,4]] In this case, Alex can paint house 0 into color 0, paint house 1 into color 2. Minimum cos
into color 2, paint house 1 into color 0. Minimum cost: 3 + 2 = 5 . Note : Assume 0 based indexing

Sample Input 1 :

```
2
2 3
1 5 3
2 9 4
3 3
1 4 5
2 3 5
6 7 8
```

Sample Output 1 :

```
5
10
```

> ### Explanation Of Sample Input 1 :
>
> #### Test Case 1 :
>
> In this case, Alex can paint house 0 into color 0, paint house 1 into color 2.
>
> Minimum cost: 1 + 4 = 5;
>
> Or paint house 0 into color 2, paint house 1 into color 0. Minimum cost: 3 + 2 = 5
>
> Hence the minimum cost will be 5.
>
> #### Test case 2 :
>
> In this case, Alex can paint house 0 into color 0, paint house 1 into color 1,
>
> paint house 2 in color 0. Minimum cost : 1 + 3 + 6 = 10.

Sample Input 2 :

```
2
2 3
1 2 3
10 11 12
1 2
4 2
```

Sample output 2 :

```
12
2
```

Analysis

- Get the input value of T, N, K, and costs.
- Find the minimum cost required to paint houses according to this condition.
- Return the minimum cost.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class PaintHouse {
    public static int minCost(int[][] costs, int n, int k) {
        int[][] dp = new int[n][k];
        for (int i = 0; i < k; i++) {
            dp[0][i] = costs[0][i];
        }
        for (int i = 1; i < n; i++) {
            for (int j = 0; j < k; j++) {
                dp[i][j] = Integer.MAX_VALUE;
                for (int l = 0; l < k; l++) {
                    if (l != j) {
                        dp[i][j] = Math.min(dp[i][j], dp[i - 1][l] + costs[i][j]);
                    }
                }
            }
        }
        int minCost = Integer.MAX_VALUE;
        for (int i = 0; i < k; i++) {
            minCost = Math.min(minCost, dp[n - 1][i]);
        }
        return minCost;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the value of T: ");
        int t = sc.nextInt();
        while (t-- > 0) {
            System.out.print("Enter the value of N and K: ");
            int n = sc.nextInt();
            int k = sc.nextInt();
            int[][] costs = new int[n][k];
            System.out.println("Enter the elements of the array: ");
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < k; j++) {
                    costs[i][j] = sc.nextInt();
                }
            }
            System.out.println(minCost(costs, n, k));
        }
    }
}
```

Output:

```
Enter the value of T: 2
Enter the value of N and K: 2 3
Enter the elements of the array:
1 5 3
```

```
2 9 4
5
Enter the value of N and K: 3 3
Enter the elements of the array:
1 4 5
2 3 5
6 7 8
10
```

---

28. Bricks Falling when hit

Problem Statement:

You are given an m x n binary grid, where each 1 represents a brick and 0 represents an empty space. A brick is stable if:

- It is directly connected to the top of the grid, or
- At least one other brick in its four adjacent cells is stable.

You are also given an array hits, which is a sequence of erasures we want to apply. Each time we want to erase the brick at the location hits[i] = (rowi, coli). The brick on that location (if it exists) will disappear. Son
longer be stable because of that erasure and will fall. Once a brick falls, it is immediately erased from the grid (i.e., it does not land on other stable bricks). Return an array result, where each result[i] is the numbe
the ith erasure is applied. Note that an erasure may refer to a location with no brick, and if it does, no bricks drop.

Example 1:

```
Input: grid = [[1,0,0,0],[1,1,1,0]], hits = [[1,0]]
Output: [2]
```

> Explanation: Starting with the grid:
>
> [[1,0,0,0],
>
> [1,1,1,0]]
>
> We erase the underlined brick at (1,0), resulting in the grid:
>
> [[1,0,0,0],
>
> [0,1,1,0]]
>
> The two underlined bricks are no longer stable as they are no longer
>
> connected to the top nor adjacent to another stable brick, so they will fall.
>
> The resulting grid is:
>
> [[1,0,0,0],

> [0,0,0,0]]
>
> Hence the result is [2].

Example 2:

```
Input: grid = [[1,0,0,0],[1,1,0,0]], hits = [[1,1],[1,0]]
Output: [0,0]
```

> Explanation: Starting with the grid:
>
> [[1,0,0,0],
>
> [1,1,0,0]]
>
> We erase the underlined brick at (1,1), resulting in the grid:
>
> [[1,0,0,0],
>
> [1,0,0,0]]
>
> All remaining bricks are still stable, so no bricks fall. The grid remains the
>
> same:
>
> [[1,0,0,0],
>
> [1,0,0,0]]
>
> Next, we erase the underlined brick at (1,0), resulting in the grid:
>
> [[1,0,0,0],
>
> [0,0,0,0]]
>
> Once again, all remaining bricks are still stable, so no bricks fall.
>
> Hence the result is [0,0].

Analysis

- Get the input value of grid and hits.
- Return an array result, where each result[i] is the number of bricks that will fall after the ith erasure is applied.

Code:

```
package com.hands_on;

import java.util.Scanner;
```

```java
public class BricksFalling {
    public static int[] hitBricks(int[][] grid, int[][] hits) {
        int m = grid.length, n = grid[0].length;
        for (int[] hit : hits) {
            grid[hit[0]][hit[1]]--;
        }
        for (int i = 0; i < n; i++) {
            dfs(grid, 0, i);
        }
        int[] res = new int[hits.length];
        int[][] dirs = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
        for (int i = hits.length - 1; i >= 0; i--) {
            int r = hits[i][0], c = hits[i][1];
            if (grid[r][c] == 0) {
                continue;
            }
            grid[r][c]++;
            if (!isConnected(grid, r, c)) {
                continue;
            }
            res[i] = dfs(grid, r, c) - 1;
        }
        return res;
    }

    private static int dfs(int[][] grid, int r, int c) {
        if (r < 0 || r >= grid.length || c < 0 || c >= grid[0].length || grid[r][c] != 1) {
            return 0;
        }
        grid[r][c] = 2;
        return 1 + dfs(grid, r + 1, c) + dfs(grid, r - 1, c) + dfs(grid, r, c + 1) + dfs(grid, r, c - 1);
    }

    private static boolean isConnected(int[][] grid, int r, int c) {
        if (r == 0) {
            return true;
        }
        int[][] dirs = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
        for (int[] dir : dirs) {
            int nr = r + dir[0], nc = c + dir[1];
            if (nr >= 0 && nr < grid.length && nc >= 0 && nc < grid[0].length && grid[nr][nc] == 2) {
                return true;
            }
        }
        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of rows and columns: ");
        int m = sc.nextInt();
        int n = sc.nextInt();
        int[][] grid = new int[m][n];
        System.out.println("Enter the elements of the grid: ");
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                grid[i][j] = sc.nextInt();
            }
        }
        System.out.print("Enter the number of hits: ");
        int k = sc.nextInt();
        int[][] hits = new int[k][2];
        System.out.println("Enter the elements of the hits: ");
        for (int i = 0; i < k; i++) {
            for (int j = 0; j < 2; j++) {
                hits[i][j] = sc.nextInt();
            }
        }
        int[] result = hitBricks(grid, hits);
        System.out.print("Output: ");
        for (int i : result) {
            System.out.print(i + " ");
        }
    }
}
```

Output:

```
Enter the number of rows and columns: 2 2
Enter the elements of the grid:
1 0
1 1
Enter the number of hits: 1
Enter the elements of the hits:
1 0
Output: 2
```

---

29. Transform to Chessboard

Problem Statement:

You are given an n x n binary grid board. In each move, you can swap any two rows with each other, or any two columns with each other. Return the minimum number of moves to transform the board into a ches
impossible, return -1. A chessboard board is a board where no 0's and no 1's are 4-directionally adjacent.

Example 1:

```
Input: board = [[0,1,1,0],[0,1,1,0],
Output: 2
```

> Explanation: One potential sequence of moves is shown. The first move swaps the first and second column. The second move swaps the second and third row.

Example 2:

```
Input: board = [[0,1],[1,0]]
Output: 0
```

> Explanation: Also note that the board with 0 in the top left corner, is also a valid chessboard.

Example 3:

```
Input: board = [[1,0],[1,0]]
Output: -1
```

> Explanation: No matter what sequence of moves you make, you cannot end with a valid chessboard.

Constraints:

```
n == board.length
n == board[i].length
2 <= n <= 30
board[i][j] is either 0 or 1.
```

Analysis

- Get the input value of board.
- Return the minimum number of moves to transform the board into a chessboard board.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class TransformChessboard {
    public static int movesToChessboard(int[][] board) {
        int n = board.length;
        int rowSum = 0, colSum = 0, rowSwap = 0, colSwap = 0;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if ((board[0][0] ^ board[i][0] ^ board[0][j] ^ board[i][j]) == 1) {
                    return -1;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            rowSum += board[0][i];
            colSum += board[i][0];
            if (board[i][0] == i % 2) {
                rowSwap++;
            }
            if (board[0][i] == i % 2) {
                colSwap++;
            }
        }
        if (rowSum != n / 2 && rowSum != (n + 1) / 2) {
            return -1;
        }
        if (colSum != n / 2 && colSum != (n + 1) / 2) {
            return -1;
        }
        if (n % 2 == 1) {
            if (colSwap % 2 == 1) {
                colSwap = n - colSwap;
            }
            if (rowSwap % 2 == 1) {
                rowSwap = n - rowSwap;
            }
        } else {
            colSwap = Math.min(n - colSwap, colSwap);
            rowSwap = Math.min(n - rowSwap, rowSwap);
        }
        return (colSwap + rowSwap) / 2;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of rows and columns: ");
        int n = sc.nextInt();
        int[][] board = new int[n][n];
        System.out.println("Enter the elements of the board: ");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                board[i][j] = sc.nextInt();
            }
        }
        System.out.println(movesToChessboard(board));
    }
}
```

Output:

```
Enter the number of rows and columns: 2
Enter the elements of the board:
0 1
1 0
0
```

---

30. Kth Smallest Amount With Single Denomination Combination

Problem statement :

You are given an integer array coins representing coins of different denominations and an integer k. You have an infinite number of coins of each denomination. However, you are not allowed to combine coins of
Return the kth smallest amount that can be made using these coins.

Example 1:

```
Input: coins = [3,6,9], k = 3
Output: 9
```

> Explanation: The given coins can make the following amounts: Coin 3 produces multiples of 3: 3, 6, 9, 12, 15, etc. Coin 6 produces multiples of 6: 6, 12, 18, 24, etc. Coin 9 produces multiples of 9: 9, 18, 27, 36, et
> produce: 3, 6, 9, 12, 15, etc.

Example 2:

```
Input: coins = [5,2], k = 7
Output: 12
```

> Explanation: The given coins can make the following amounts: Coin 5 produces multiples of 5: 5, 10, 15, 20, etc. Coin 2 produces multiples of 2: 2, 4, 6, 8, 10, 12, etc. All of the coins combined produce: 2, 4, 5, 6,

Constraints:

```
1 <= coins.length <= 15
1 <= coins[i] <= 25
1 <= k <= 2 * 10^9
coins contains pairwise distinct integers
```

Analysis

- Get the input value of coins and k.
- Return the kth smallest amount that can be made using these coins.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class KthSmallestAmount {
    public static int kthSmallestAmount(int[] coins, int k) {
        int n = coins.length;
        int[] dp = new int[k + 1];
        dp[0] = 1;
        for (int i = 0; i < n; i++) {
            for (int j = coins[i]; j <= k; j++) {
                dp[j] += dp[j - coins[i]];
            }
        }
        for (int i = 1; i <= k; i++) {
            if (dp[i] >= k) {
                return i;
            }
        }
        return 0;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of coins: ");
        int n = sc.nextInt();
        int[] coins = new int[n];
        System.out.println("Enter the elements of the coins: ");
        for (int i = 0; i < n; i++) {
            coins[i] = sc.nextInt();
        }
        System.out.print("Enter the value of k: ");
        int k = sc.nextInt();
        System.out.println(kthSmallestAmount(coins, k));
    }
}
```

Output:

```
Enter the number of coins: 3
Enter the elements of the coins:
```

```
3 6 9
Enter the value of k: 3
9
```