Problem Solving - I

1. Add Digits

Problem Statements:

Given an integer num, repeatedly add all its digits until the result has only one digit and return it.

Example 1:

```
Input: num = 38
Output: 2
```

Example 2:

```
Input: num = 0
Output: 0
```

Constraints:

```
0 <= num <= 2^31 - 1
```

Analysis:

- Get the number input from the user
- Check if the number is less than 10, then return the number
- If the number is greater than 10, then find the sum of the digits
- Repeat the process until the sum is less than 10
- Return the sum

Code:

```java
package com.hands_on;

import java.util.Scanner;
```

```java
public class AddDigits {
    public static int addDigits(int num) {
        int sum;
        if (num < 10) {
            return num;
        } else {
            sum = 0;
            while (num != 0) {
                int digit = num % 10;
                sum += digit;
                num /= 10;
            }
            return addDigits(sum);
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int userInput = sc.nextInt();
        System.out.println(addDigits(userInput));

    }
}
```

Output:

```
Enter any number: 38
2
```

---

2. Alternating Digit Sum

Problem statement:

You are given a positive integer n. Each digit of n has a sign according to the following rules: ###

- The most significant digit is assigned a positive sign.
- Each digit has an opposite sign to its adjacent digits.
- Return the sum of all digits with their corresponding sign.

Example 1:

```
Input: n = 521
Output: 4
Explanation: (+5) + (-2) + (+1) = 4.
```

Example 2:

```
Input: n = 111
Output: 1
Explanation: (+1) + (-1) + (+1) = 1.
```

Constraints:

```
1 <= n <= 10^9
```

Analysis:

- Get the number input from the user
- Convert the number to a string
- Initialize the sum with the first digit
- Iterate through the string
- If the digit is even, add it to the sum
- If the digit is odd, subtract it from the sum
- Return the sum

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class AlternatingDigitSum {
    public static int alternatingDigitSum(int n) {
        String num = String.valueOf(n);
        int sum = num.charAt(0) - '0';
        for (int i = 1; i < num.length(); i++) {
            if (i % 2 == 0) {
                sum += num.charAt(i) - '0';
```

```
            } else {
                sum -= num.charAt(i) - '0';
            }
        }
        return sum;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int userInput = sc.nextInt();
        System.out.println(alternatingDigitSum(userInput));
    }
}
```

Output:

```
Enter any number: 137
5
```

---

3. Divisor Game

Problem statement:

Alice and Bob take turns playing a game, with Alice starting first. Initially, there is a number n on the chalkboard. On each player's turn, that player makes a move consisting of: ###

- Choosing any x with 0 < x < n and n % x == 0.
- Replacing the number n on the chalkboard with n - x.
- Also, if a player cannot make a move, they lose the game.
- Return true if and only if Alice wins the game, assuming both players play optimally.

Example 1:

```
Input: n = 2
Output: true
Explanation: Alice chooses 1, and Bob has no more moves.
```

Example 2:

```
Input: n = 3
Output: false
Explanation: Alice chooses 1, Bob chooses 1, and Alice has no more moves.
```

Constraints:

```
1 <= n <= 1000
```

Analysis:

- Get the number input from the user
- If the number is even, Alice wins
- If the number is odd, Bob wins
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class DivisorGame {
    public static boolean divisorGame(int n) {
        return n % 2 == 0;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int userInput = sc.nextInt();
        System.out.println(divisorGame(userInput));
    }
}
```

Output:

```
Enter any number: 3
false
```

4. Happy Number

Problem statement:

Write an algorithm to determine if a number n is happy. ### A happy number is a number defined by the following process:

- Starting with any positive integer, replace the number by the sum of the squares of its digits.
- Repeat the process until the number equals 1 (where it will stay), or it loops endlessly in a cycle which does not include 1.
- Those numbers for which this process ends in 1 are happy.
- Return true if n is a happy number, and false if not.

Example 1:

```
Input: n = 19
Output: true
Explanation:
    1^2 + 9^2 = 82
    8^2 + 2^2 = 68
    6^2 + 8^2 = 100
    1^2 + 0^2 + 0^2 = 1
```

Example 2:

```
Input: n = 2
Output: false
```

Constraints:

```
1 <= n <= 2^31 - 1
```

Analysis:

- Get the number input from the user
- If the number is less than 10, check if it is equal to 1
- If the number is greater than 10, find the sum of the squares of the digits

- Repeat the process until the sum is less than 10
- Return the result

Code:

```
package com.hands_on;

import java.util.Scanner;

public class HappyNumber {
    public static int happyNumber(int num) {
        int sum;
        if (num < 10) {
            if (num == 1){
                return 1;
            }else{
                return 0;
            }
        } else {
            sum = 0;
            while (num != 0) {
                int digit = num % 10;
                sum += digit * digit;
                num /= 10;
            }
            return happyNumber(sum);
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int userInput = sc.nextInt();
        System.out.println((happyNumber(userInput) == 1));

    }
}
```

Output:

```
Enter any number: 19
true
```

5. Ugly Number

Problem statement:

An ugly number is a positive integer whose prime factors are limited to 2, 3, and 5. Given an integer n, return true if n is an ugly number.

Example 1:

```
Input: n = 6
Output: true
Explanation: 6 = 2 × 3
```

Example 2:

```
Input: n = 1
Output: true
Explanation: 1 has no prime factors, therefore all of its prime factors are
limited to 2, 3, and 5
```

Constraints:

```
-2^31 <= n <= 2^31 - 1
```

Analysis:

- Get the number input from the user
- If the number is less than or equal to 0, return false
- Iterate through the array
- If the number is divisible by 2, 3, or 5, divide the number by 2, 3, or 5
- If the number is not divisible by 2, 3, or 5, return false
- Return true

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class UglyNumber {
    public static boolean uglyNumber(int n) {
        if (n <= 0) {
            return false;
        }
        while (n % 2 == 0) {
            n /= 2;
        }
        while (n % 3 == 0) {
            n /= 3;
        }
        while (n % 5 == 0) {
            n /= 5;
        }
        return n == 1;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int userInput = sc.nextInt();
        System.out.println(uglyNumber(userInput));
    }
}
```

Output:

```
Enter any number: 6
true
```

---

6. Find the Pivot Integer

Problem statement:

Given an integer array nums, find the pivot integer. An integer x is defined as the pivot integer if:

- The sum of all elements to the left of x is equal to the sum of all elements to the right of x.
- If the array has no pivot integer, return -1. If there are multiple pivot integers, return the left-most pivot integer.

Example 1:

```
Input: n = 8
Output: 6
Explanation: 6 is the pivot integer since:
    1 + 2 + 3 + 4 + 5 + 6 = 6 + 7 + 8 = 21.
```

Example 2:

```
Input: n = 1
Output: 1
Explanation: 1 is the pivot integer since: 1 = 1.
```

Constraints:

```
1 <= n <= 1000
```

Analysis:

- Get the number input from the user
- Initialize the left sum and right sum as 0
- Iterate through the array
- Calculate the right sum
- If the left sum is equal to the right sum, return the pivot integer
- If the pivot integer is not found, return -1

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class PivotInteger {
    public static int pivotInteger(int[] nums) {
        int leftSum = 0;
```

```
        int rightSum = 0;
        for (int j : nums) {
            rightSum += j;
        }
        for (int num : nums) {
            rightSum -= num;
            if (leftSum == rightSum) {
                return num;
            }
            leftSum += num;
        }
        return -1;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int size = sc.nextInt();
        int[] generatedArray = new int[size];
        for (int i = 0; i < size; i++) {
            generatedArray[i] = i+1;
        }
        System.out.println(pivotInteger(generatedArray));
    }
}
```

Output:

```
Enter any number: 8
6
```

---

7. Count of Matches in Tournament

Problem statement:

You are given an integer n, the number of teams in a tournament that has strange rules:

- If the current number of teams is even, each team gets paired with another team. A total of n / 2 matches are played, and n / 2 teams advance to the next round.

- If the current number of teams is odd, one team randomly advances in the tournament, and the rest gets paired. A total of (n - 1) / 2 matches are played, and (n - 1) / 2 + 1 teams advance to the next round.
- Return the number of matches played in the tournament until a winner is decided.

Example 1:

```
Input: n = 7
Output: 6
Explanation: 7 teams
    - 4 teams advance, 3 matches are played
    - 2 teams advance, 2 matches are played
    - 1 team wins
    Total matches = 3 + 2 + 1 = 6
```

Example 2:

```
Input: n = 14
Output: 13
Explanation: 14 teams
    - 7 teams advance, 7 matches are played
    - 4 teams advance, 3 matches are played
    - 2 teams advance, 2 matches are played
    - 1 team wins
    Total matches = 7 + 3 + 2 + 1 = 13
```

Constraints:

```
1 <= n <= 200
```

Analysis:

- Get the number input from the user
- Initialize the count as 0
- If the number is even, divide the number by 2
- If the number is odd, subtract 1 from the number and divide by 2
- Add the result to the count
- Repeat the process until the number is 1
- Return the count

Code:

```
package com.hands_on;

import java.util.Scanner;

public class CountOfMatchesInTournament {
    public static int countOfMatechesInTournament(int matches){
        if(matches == 1){
            return 0;
        }else{
            if(matches%2==0) {
                return countOfMatechesInTournament(matches / 2) + (matches/2)
            } else {
                return countOfMatechesInTournament((matches)/2)+((matches+1)/
            }
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int userInput = sc.nextInt();
        System.out.println("Total number of match played in tournament: "+cou
    }
}
```

Output:

```
Enter any number: 7
Total number of match played in tournament: 6
```

---

8. Find the Maximum Achievable Number

Problem statement:

You are given two integers, num and t. An integer x is called achievable if it can become equal to num after applying the following operation no more than t times:

- Increase or decrease x by 1, and simultaneously increase or decrease num by 1.
- Return the maximum possible achievable number.
- It can be proven that there exists at least one achievable number.

Example 1:

```
Input: num = 4, t = 1
Output: 6
Explanation: 4 + 1 = 5, 5 + 1 = 6.
```

Example 2:

```
Input: num = 3, t = 2
Output: 6
Explanation: 3 + 1 = 4, 4 + 2 = 6.
```

Constraints:

```
1 <= num, t <= 50
```

Analysis:

- Get the number input from the user
- Get the number of operations input from the user
- If the number of operations is less than or equal to 0, return the number
- If the number of operations is even, return the number + operations
- If the number of operations is odd, return the number + operations + 1

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class MaximumAchievableNumber {
    public static int maximumAchievableNumber(int num, int t) {
        if (t <= 0) {
            return num;
        } else {
            if (t % 2 == 0) {
```

```
                return num + t;
            } else {
                return num + t + 1;
            }
        }
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int userInput = sc.nextInt();
        System.out.print("Enter number of operations: ");
        int operations = sc.nextInt();
        System.out.println(
            "Maximum Achievable Number: "
            +maximumAchievableNumber(userInput, operations)
        );
    }
}
```

Output:

```
Enter any number: 4
Enter number of operations: 1
Maximum Achievable Number: 6
```

---

9. Number of handshakes

Problem statement:

The user is asked to take a number as integer n and find out the possible number of handshakes. For example, if there are n number of people in a meeting and find the possible number of handshakes made by the person who entered the room after all were settled.

- For the first person, there will be N-1 people to shake hands with
- For second person, there will be N -1 people available but as he has already shaken hands with the first person, there will be N-1-1 = N-2 shake-hands

- For third person, there will be N-1-1-1 = N-3, and So On… Therefore, the total number of handshake = (N – 1 + N – 2 + ...+ 1 + 0) = ((N-1) *N) / 2.

Input :

```
3
```

Output :

```
Possible number of handshakes: 3
```

Input :

```
5
```

Output :

```
Possible number of handshakes: 10
```

Analysis:

- Get the number input from the user
- Calculate the number of handshakes using the formula (n*(n-1))/2
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class NumberOfHandshakes {
    public static int numberOfHandshakes(int n) {
        return (n * (n - 1)) / 2;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int userInput = sc.nextInt();
        System.out.println(
```

```
            "Possible number of handshakes: "
            +numberOfHandshakes(userInput)
        );
    }
}
```

Output:

```
Enter any number: 5
Possible number of handshakes: 10
```

---

10. Climbing Stairs

Problem statement:

You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

```
Input: n = 2
Output: 2
Explanation: There are two ways to climb to the top.
    1. 1 step + 1 step
    2. 2 steps
```

Example 2:

```
Input: n = 3
Output: 3
Explanation: There are three ways to climb to the top.
    1. 1 step + 1 step + 1 step
    2. 1 step + 2 steps
    3. 2 steps + 1 step
```

Constraints:

```
1 <= n <= 45
```

Analysis:

- Get the number input from the user
- Initialize the first and second steps as 1
- Iterate through the array
- Calculate the number of ways to reach the top
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class ClimbingStairs {
    public static int climbingStairs(int n) {
        int first = 1;
        int second = 1;
        for (int i = 2; i <= n; i++) {
            int temp = first + second;
            first = second;
            second = temp;
        }
        return second;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int userInput = sc.nextInt();
        System.out.println(climbingStairs(userInput));
    }
}
```

Output:

```
Enter any number: 3
3
```

11. Encrypt the given number

Problem statement:

Micheal has created his own encryption technique to encrypt a number. He makes use of the logic behind factorial. In a factorial, we multiply the number by its previous number and so on but if we want to encrypt a number we don't multiply in every step like in thecase of factorial but multiply, divide, add and subtract and repeat in the same order.
So your task is to find the encrypted form of a number using the Micheal's encryption technique and you were being provided with the number

Sample Input 1:

```
2
5
8
```

Sample Output 1:

```
7
9
```

Explanation For Sample Input 1:

> **Test Case 1:**
>
> > For the first test case, given number is `5` so using the encryption technique we follow the steps: (5 * 4 / 3 + 2 - 1) = 7
>
> **Test Case 2:**
>
> > For the first test case, the given number is `8` so using the encryption technique we follow the steps: (8 * 7 / 6 + 5 - 4 * 3 / 2 - 1) = 9

Sample Input 2:

```
1
12
```

Sample Output 2:

```
13
```

Analysis:

- Get the number input from the user
- Initialize the result as the number
- Iterate through the array
- Calculate the encrypted number
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class EncryptNumber {
    public static int encryptNumber(int n) {
        int result = n;
        for (int i = n - 1; i >= 1; i--) {

            result = result * i;
            i--;
            if (i == 1) {
                break;
            }
            result = result / i;
            if (i == 1) {
                break;
            }
            i--;
            result = result + i;
            if (i == 1) {
                break;
            }
```

```
            i--;
            result = result - i;
        }
        return result;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int userInput = sc.nextInt();
        System.out.println(encryptNumber(userInput));
    }
}
```

Output:

```
Enter any number: 8
9
```

---

12. Cube Sum Pairs

Problem statement:

You are given a positive integer N, and you have to find the number of ways to represent N as a sum of cubes of two integers(let's say A and B), such that: N = A^3 + B^3.

> **Note**:
>
> 1. A should be greater than or equal to one (A>=1).
> 2. B should be greater than or equal to zero (B>=0).
> 3. (A, B) and (B, A) should be considered different solutions, if A is not equal to B, i.e (A, B) and (B, A) will not be distinct if A=B.

Sample Input 1:

```
9
```

Sample Output 1:

```
2
```

Explanation For Sample Input 1:

```
There are 2 ways to represent N in the (A^3 + B^3).
Eg. 1^3 + 2^3 = 9 and 2^3 + 1^3 = 9.
```

Sample Input 2:

```
27
```

Sample Output 2:

```
1
```

Explanation For Sample Input 2:

```
There is only 1 way to represent N in the (A^3 + B^3).
Eg. 3^3 + 0^3 = 27.
```

Constraints:

```
1 <= T <= 10^3
1 <= N <= 10^8
```

Analysis:

- Get the number input from the user
- Initialize the count as 0
- Iterate through the array
- Calculate the cube sum pairs
- Return the result

Code:

```java
package com.hands_on;
```

```java
import java.util.Scanner;

public class CubeSumPairs {
    public static int cubeSumPairs(int n) {
        int count = 0;
        for (int i = 1; i * i * i < n; i++) {
            int b = n - i * i * i;
            int c = (int) Math.cbrt(b);
            if (c * c * c == b) {
                count++;
            }
        }
        return count;
    }

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

Output:

```
Enter any number: 9
2
```

---

13. Star pattern

Problem statement:

Print the following pattern

Sample Input 1:

```
5
```

Sample Output 1:

```
The pattern is:
     *
    ***
   *****
  *******
 *********
```

Sample Input 2:

```
3
```

Sample Output 2:

```
The pattern is:
  *
 ***
*****
```

Analysis:

- Get the number input from the user
- Print the pattern

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class StarPattern {
    public static void starPattern(int n) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n - i; j++) {
                System.out.print(" ");
            }
            for (int k = 0; k <= i; k++) {
                System.out.print("*");
            }
            for (int l = 1; l <= i; l++) {
                System.out.print("*");
```

```
            }
            System.out.println();
        }
    }

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

Output:

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

---

14. Square pattern

Problem statement:

Print the following pattern
Sample Input 1:

```
7
```

Sample Output 1:

```
7777777
7777777
7777777
7777777
7777777
```

```
7777777
7777777
```

Sample Input 2:

```
6
```

Sample Output 2:

```
666666
666666
666666
666666
666666
666666
```

Constraints

```
0 <= N <= 50
```

Analysis:

- Get the number input from the user
- Print the pattern

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class SquarePattern {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter any number: ");
        int num = sc.nextInt();
        squarePattern(num);
    }
        public static void squarePattern(int num){
                for(int i = 0;i < num; i++) {
```

```
                        for(int j = 0; j < num; j++) {
                                System.out.print(num+" ");
                        }
                System.out.println();
                }

        }

}
```

Output:

```
Enter any number: 7
7 7 7 7 7 7 7
7 7 7 7 7 7 7
7 7 7 7 7 7 7
7 7 7 7 7 7 7
7 7 7 7 7 7 7
7 7 7 7 7 7 7
7 7 7 7 7 7 7
```

Number of 1 Bits

Problem statement:

Write a function that takes an integer and returns the number of 1 bit it has.

Example Input

```
Input1: 11
Output1: 3
```

Explanation:

```
11 is represented as 1101 in binary
```

Constraints:

```
1<= A <= 4294967295
```

Analysis:

- Get the number input from the user
- Convert the number to binary
- Count the number of 1 bits
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class NumberOf1Bits {
    public static int numberOf1Bits(int n) {
        int count = 0;
        while (n != 0) {
            n = n & (n - 1);
            count++;
        }
        return count;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int userInput = sc.nextInt();
        System.out.println(numberOf1Bits(userInput));
    }
}
```

Output:

```
Enter any number: 11
3
```

---

16. Decode XORed Array

Problem Statement:

There is a hidden integer array arr that consists of n non-negative integers. It was encoded into another integer array encoded of length n - 1, such that encoded[i] = arr[i] XOR arr[i + 1]. For example, if arr = [1,0,2,1], then encoded = [1,2,3]. You are given the encoded array. You are also given an integer first, that is the first element of arr, i.e. arr[0].Return the original array arr. It can be proved that the answer exists and is unique.

Example 1:

```
Input: encoded = [1,2,3], first = 1
Output: [1,0,2,1]
```

Explanation:

```
If arr = [1,0,2,1], then first = 1 and encoded = [1 XOR 0, 0 XOR 2, 2 XOR
1] = [1,2,3]
```

Example 2:

```
Input: encoded = [6,2,7,3], first = 4
Output: [4,2,0,7,4]
```

Constraints:

```
1<= n <= 10^4
```

Analysis:

- Get the number input from the user
- Get the first element of the array
- Initialize the result array with the first element
- Iterate through the array
- Calculate the original array
- Return the result

Code:

```
package com.hands_on;

import java.util.Scanner;

public class DecodeXORedArray {
            public static int[] decodeXORedArray(int[] encoded, int first) {
                int n = encoded.length;
                int[] arr = new int[n + 1];
                arr[0] = first;
                for (int i = 0; i < n; i++) {
                    arr[i + 1] = arr[i] ^ encoded[i];
                }
                return arr;
        }
        public static void main(String[] args) {
                Scanner sc = new Scanner(System.in);
                System.out.print("Enter the number of elements: ");
                int n = sc.nextInt();
                int[] encoded = new int[n];
                System.out.println("Enter the elements of the array: ");
                for (int i = 0; i < n; i++) {
                        encoded[i] = sc.nextInt();
                }
                System.out.print("Enter the first element: ");
                int first = sc.nextInt();
                int[] result = decodeXORedArray(encoded, first);
                System.out.println("The original array is: ");
                for (int i : result) {
                        System.out.print(i + " ");
                }
        }
}
```

Output:

```
Enter the number of elements: 3
Enter the elements of the array:
1 2 3
Enter the first element: 1
```

```
The original array is:
1 0 2 1
```

---

17. Number of Even and Odd Bits

Problem Statement:

You are given a positive integer n.Let even denote the number of even indices in the binary representation of n (0-indexed) with value 1.Let odd denote the number of odd indices in the binary representation of n (0-indexed) with value 1.Return an integer array answer where answer = [even, odd].

Example 1:

```
Input: n = 17
Output: [2,0]
```

> **Explanation:** The binary representation of 17 is 10001. It contains 1 on the 0th and 4th indices. There are 2 even and 0 odd indices.

Example 2:

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

> **Explanation:** The binary representation of 2 is 10. It contains 1 on the 1st index. There are 0 even and 1 odd indices.

Constraints:

```
1 <= n <= 1000
```

Analysis:

- Get the number input from the user
- Convert the number to binary
- Count the number of even and odd bits

- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class NumberOfEvenAndOddBits {
    public static int[] numberOfEvenAndOddBits(int n) {
        int even = 0, odd = 0;
        String binary = Integer.toBinaryString(n);
        for (int i = 0; i < binary.length(); i++) {
            if (i % 2 == 0 && binary.charAt(i) == '1') {
                even++;
            } else if (i % 2 == 1 && binary.charAt(i) == '1') {
                odd++;
            }
        }
        return new int[]{even, odd};
    }
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter any number: ");
        int n = sc.nextInt();
        int[] arr = numberOfEvenAndOddBits(n);
        System.out.println("["+arr[0]+","+arr[1]+"]");
    }
}
```

Output

```
Enter any number: 17
[2,0]
```

---

18. Hamming Distance

Problem Statement:

The Hamming distance between two integers is the number of positions at which the corresponding bits are different. Given two integers x and y, return the Hamming distance between them

Example 1:

```
Input: x = 1, y = 4
Output: 2
```

Explanation:

```
1 (0 0 0 1)
4 (0 1 0 0)
     ↑   ↑
The above arrows point to positions where the corresponding bits are
different.
```

Example 2:

```
Input: x = 3, y = 1
Output: 1
```

Constraints:

```
0 <= x, y <= 2^31 - 1
```

Analysis:

- Get the number input from the user
- Calculate the Hamming distance
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class HammingDistance {
```

```
        public static int hammingDistance(int x, int y) {
                int xor = x ^ y;
                int count = 0;
                while (xor != 0) {
                        count += xor & 1;
                        xor >>= 1;
                }
                return count;
        }
        public static void main(String[] args){
                Scanner sc = new Scanner(System.in);
                System.out.print("Enter the 1st number: ");
                int firstNum = sc.nextInt();
                System.out.print("Enter the 2nd number: ");
                int secondNum = sc.nextInt();
                System.out.println(
                        "The Hamming Distance between "+ firstNum
                        +" and "+ secondNum +" is "
                        + hammingDistance(firstNum, secondNum)
                );
        }
}
```

Output:

```
Enter the 1st number: 1
Enter the 2nd number: 4
The Hamming Distance between 1 and 4 is 2
```

---

19. Check if Bitwise OR Has Trailing Zeros

Problem Statement:

You are given an array of positive integers nums. You have to check if it is possible to select two or more elements in the array such that the bitwise OR of the selected elements has at least one trailing zero in its binary representation. Return true if it is possible to select two or more elements whose bitwise OR has trailing zeros, return false otherwise.

Example 1:

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

> Explanation: If we select the elements 2 and 4, their bitwise OR is 6, which has the binary representation "110" with one trailing zero.

Example 2:

```
Input: nums = [2,4,8,16]
Output: true
```

> Explanation: If we select the elements 2 and 4, their bitwise OR is 6, which has the binary representation "110" with one trailing zero. Other possible ways to select elements to have trailing zeroes in the binary representation of their bitwise OR are: (2, 8), (2, 16), (4, 8), (4, 16), (8, 16), (2, 4, 8), (2, 4, 16), (2, 8, 16), (4, 8, 16), and (2, 4, 8, 16).

Constraints:

- 2 <= nums.length <= 100
- 1 <= nums[i] <= 100

Analysis:

- Get the number input from the user
- Check if the bitwise OR has trailing zeros
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class BitwiseORTrailingZeros {
        public static boolean bitwiseORTrailingZeros(int[] nums) {
        int n = nums.length;
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if ((nums[i] | nums[j]) % 2 == 0) {
                        return true;
                }
```

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

Output:

```
Enter the size: 5
Enter the elements of array:
1 2 3 4 5
true
```

---

20. Longest Consecutive 1's

Problem Statement:

Given a number N. Find the length of the longest consecutive 1s in its binary representation

Example 1:

```
Input: N = 14
Output: 3
```

> Explanation: Binary representation of 14 is 1110, in which 111 is the longest consecutive set bits of length is 3.

Example 2:

```
Input: N = 222
Output: 4
```

> Explanation: Binary representation of 222 is 11011110, in which 1111 is the longest consecutive set bits of length 4.

Constraints:

```
1 <= N <= 10^6
```

Analysis:

- Get the number input from the user
- Convert the number to binary
- Calculate the longest consecutive 1's
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class LongestConsecutiveOnes {
    public static int longestConsecutiveOnes(int n) {
        int count = 0;
        int maxCount = 0;
        while (n != 0) {
            if ((n & 1) == 1) {
                count++;
            } else {
                maxCount = Math.max(maxCount, count);
                count = 0;
            }
            n >>= 1;
        }
        return Math.max(maxCount, count);
    }
    public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int userInput = sc.nextInt();
        System.out.println(longestConsecutiveOnes(userInput));
    }
}
```

Output:

```
Enter any number: 14
3
```

---

21. Gray to Binary equivalent

Problem Statement:

Given a integer number n in Gray Code, find the binary equivalent of the number n. Return the decimal representation of the binary equivalent.

Example 1:

```
Input:
    n = 4
Output:
    7
```

> **Explanation**:
>
> - Given 4, its gray code = 110.
> - Binary equivalent of the gray code 110 is 100.
> - Return 7 representing gray code 100.

Example 2:

```
Input:
    n = 15
```

```
Output:
    10
```

> **Explanation**:
>
> - Given 15 representing gray code 1000.
> - Binary equivalent of gray code 1000 is 1111.
> - Return 10 representing gray code 1111
> - ie binary 1010.

Constraints:

```
0 <= n <= 10^8
```

Analysis:

- Get the number input from the user
- Convert the Gray code to binary
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class GrayToBinaryEquivalent {
    public static int grayToBinaryEquivalent(int n) {
        int result = n;
        while (n > 0) {
            n >>= 1;
            result ^= n;
        }
        return result;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int userInput = sc.nextInt();
        System.out.println(grayToBinaryEquivalent(userInput));
```

```
    }
}
```

Output:

```
Enter any number: 4
7
```

---

22. Minimum Bit Flips to Convert Number

Problem Statement:

A bit flip of a number x is choosing a bit in the binary representation of x and flipping it from either 0 to 1 or 1 to 0. Given two integers start and goal, return the minimum number of bit flips to convert start to goal.

Example 1:

```
Input: start = 10, goal = 7
Output: 3
```

> Explanation: The binary representation of 10 and 7 are 1010 and 0111 respectively. We can convert 10 to 7 in 3 steps:
>
> - Flip the first bit from the right: 1010 -> 1011.
> - Flip the third bit from the right: 1011 -> 1111.
> - Flip the fourth bit from the right: 1111 -> 0111.
>
> It can be shown we cannot convert 10 to 7 in less than 3 steps. Hence, we return 3.

Example 2:

```
Input: start = 3, goal = 4
Output: 3
```

> Explanation: The binary representation of 3 and 4 are 011 and 100 respectively. We can convert 3 to 4 in 3 steps:

- Flip the first bit from the right: 011 -> 010.
- Flip the second bit from the right: 010 -> 000.
- Flip the third bit from the right: 000 -> 100.

It can be shown we cannot convert 3 to 4 in less than 3 steps. Hence, we return 3.

Constraints:

```
0 <= start, goal <= 10^9
```

Analysis:

- Get the number input from the user
- Calculate the minimum bit flips
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class MinimumBitFlipsToConvertNumber {
    public static int minimumBitFlipsToConvertNumber(int start, int goal) {
        int count = 0;
        while (start != 0 || goal != 0) {
            if ((start & 1) != (goal & 1)) {
                count++;
            }
            start >>= 1;
            goal >>= 1;
        }
        return count;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the start number: ");
        int start = sc.nextInt();
        System.out.print("Enter the goal number: ");
        int goal = sc.nextInt();
        System.out.println(minimumBitFlipsToConvertNumber(start, goal));
```

```
    }
}
```

Output:

```
Enter the start number: 10
Enter the goal number: 7
3
```

---

23. Toeplitz Matrix

Problem Statement:

Given an m x n matrix, return true if the matrix is Toeplitz. Otherwise, return false. A matrix is Toeplitz if every diagonal from top-left to bottom-right has the same elements.

Example 1:

```
Input: matrix = [[1,2,3,4],[5,1,2,3],[9,5,1,2]]
Output: true
```

> **Explanation:**
>
> In the above grid, the diagonals are: "[9]", "[5, 5]", "[1, 1, 1]", "[2, 2, 2]", "[3, 3]", "[4]".
>
> In each diagonal all elements are the same, so the answer is True.

Example 2:

```
Input: matrix = [[1,2],[2,2]]
Output: false
```

> **Explanation:**
>
> The diagonal "[1, 2]" has different elements.

Constraints:

```
m == matrix.length
n == matrix[i].length
1 <= m, n <= 20
0 <= matrix[i][j] <= 99
```

Analysis:

- Get the matrix input from the user
- Check if the matrix is Toeplitz
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class ToeplitzMatrix {
    public static boolean isToeplitzMatrix(int[][] matrix) {
        for (int i = 0; i < matrix.length - 1; i++) {
            for (int j = 0; j < matrix[0].length - 1; j++) {
                if (matrix[i][j] != matrix[i + 1][j + 1]) {
                    return false;
                }
            }
        }
        return true;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of rows: ");
        int rows = sc.nextInt();
        System.out.print("Enter the number of columns: ");
        int cols = sc.nextInt();
        int[][] matrix = new int[rows][cols];
        System.out.println("Enter the elements of the matrix: ");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }
        System.out.println(isToeplitzMatrix(matrix));
```

```
    }
}
```

Output:

```
Enter the number of rows: 3
Enter the number of columns: 4
Enter the elements of the matrix:
1 2 3 4
5 1 2 3
9 5 1 2
true
```

---

24. Build Array from Permutation

Problem Statement:

Given a zero-based permutation nums (0-indexed), build an array ans of the same length where ans[i] = nums[nums[i]] for each 0 <= i < nums.length and return it. A zero-based permutation nums is an array of distinct integers from 0 to nums.length - 1 (inclusive). ###### Example 1: Input: nums = [0,2,1,5,3,4] Output: [0,1,2,4,5,3]

> **Explanation:** The array ans is built as follows:
>
> ans = [nums[nums[0]], nums[nums[1]], nums[nums[2]], nums[nums[3]],
>
> nums[nums[4]], nums[nums[5]]]
>
> = [nums[0], nums[2], nums[1], nums[5], nums[3], nums[4]]
>
> = [0,1,2,4,5,3]

Example 2:

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

> **Explanation:** The array ans is built as follows:

> ans = [nums[nums[0]], nums[nums[1]], nums[nums[2]], nums[nums[3]],
>
> nums[nums[4]], nums[nums[5]]]
>
> = [nums[5], nums[0], nums[1], nums[2], nums[3], nums[4]]
>
> = [4,5,0,1,2,3]

Constraints:

- 1 <= nums.length <= 1000
- 0 <= nums[i] < nums.length
- The elements in nums are distinct

Analysis:

- Get the number input from the user
- Build the array from permutation
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

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

```
        }
        int[] result = buildArrayFromPermutation(arr);
        System.out.println("The array is: ");
        for (int i : result) {
            System.out.print(i + " ");
        }
    }
}
```

Output:

```
Enter the size: 6
Enter the elements of the array:
5 0 1 2 3 4
The array is:
4 5 0 1 2 3
```

---

25. Concatenation of Array

Problem Statement: Given an integer array nums of length n, you want to create an array ans of length 2n where ans[i] == nums[i] and ans[i + n] == nums[i] for 0 <= i < n (0-indexed). Specifically, ans is the concatenation of two nums arrays. Return the array ans.

Example 1:

```
Input: nums = [1,2,1]
Output: [1,2,1,1,2,1]
```

> **Explanation:** The array ans is formed as follows:
>
> - ans = [nums[0],nums[1],nums[2],nums[0],nums[1],nums[2]]
> - ans = [1,2,1,1,2,1]

Example 2:

```
Input: nums = [1,3,2,1]
Output: [1,3,2,1,1,3,2,1]
```

> Explanation: The array ans is formed as follows:
>
> - ans = [nums[0],nums[1],nums[2],nums[3],nums[0],nums[1],nums[2],nums[3]]
> - ans = [1,3,2,1,1,3,2,1]

Constraints:

```
n == nums.length
1 <= n <= 1000
1 <= nums[i] <= 1000
```

Analysis:

- Get the number input from the user
- Concatenate the array
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class ConcatenationOfArray {
    public static int[] concatenationOfArray(int[] nums) {
        int n = nums.length;
        int[] ans = new int[2 * n];
        for (int i = 0; i < n; i++) {
            ans[i] = nums[i];
            ans[i + n] = nums[i];
        }
        return ans;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size: ");
        int size = sc.nextInt();
        int[] arr = new int[size];
        System.out.println("Enter the elements of the array: ");
        for (int i = 0; i < size; i++) {
            arr[i] = sc.nextInt();
        }
```

```
        int[] result = concatenationOfArray(arr);
        System.out.println("The array is: ");
        for (int i : result) {
            System.out.print(i + " ");
        }
    }
}
```

Output:

```
Enter the size: 3
Enter the elements of the array:
1 2 1
The array is:
1 2 1 1 2 1
```

---

26. Find N Unique Integers Sum up to Zero

Problem statement :

Given an integer n, return any array containing n unique integers such that they add up to 0. The integers for sum can be either positive or negative.

Example 1:

```
Input: n = 5
Output: [-7,-1,1,3,4]
```

> **Explanation:** These arrays also are accepted [-5,-1,1,2,3] , [-3,-1,2,-2,4].

Example 2:

```
Input: n = 3
Output: [-1,0,1]
```

Constraints:

```
1 <= n <= 1000
```

Analysis:

- Get the number input from the user
- Find the unique integers that sum up to zero
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class FindNUniqueIntegersSumUpToZero {
    public static int[] findNUniqueIntegersSumUpToZero(int n) {
        int[] result = new int[n];
        for (int i = 0; i < n - 1; i++) {
            result[i] = i + 1;
        }
        result[n - 1] = -(n * (n - 1)) / 2;
        return result;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int userInput = sc.nextInt();
        int[] result = findNUniqueIntegersSumUpToZero(userInput);
        System.out.println("The array is: ");
        for (int i : result) {
            System.out.print(i + " ");
        }
    }
}
```

Output:

```
Enter any number: 5
The array is:
```

```
1 2 3 4 -10
```

---

27. Matrix Diagonal Sum

Problem statement: Given a square matrix mat, return the sum of the matrix diagonals. Only include the sum of all the elements on the primary diagonal and all the elements on the secondary diagonal that are not part of the primary diagonal.

Example 1:

```
Input: mat =
    [[1,2,3],
    [4,5,6],
    [7,8,9]]
Output: 25
```

> Explanation: Diagonals sum: 1 + 5 + 9 + 3 + 7 = 25
>
> Notice that element mat[1][1] = 5 is counted only once.

Example 2:

```
Input: mat =
    [[1,1,1,1],
    [1,1,1,1],
    [1,1,1,1],
    [1,1,1,1]]
Output: 8
```

Constraints:

```
n == mat.length == mat[i].length
1 <= n <= 100
1 <= mat[i][j] <= 100
```

Analysis:

- Get the matrix input from the user

- Calculate the matrix diagonal sum
- Return the result

Code:

```
package com.hands_on;

import java.util.Scanner;

public class MatrixDiagonalSum {
    public static int matrixDiagonalSum(int[][] mat) {
        int n = mat.length;
        int sum = 0;
        for (int i = 0; i < n; i++) {
            sum += mat[i][i];
            sum += mat[i][n - i - 1];
        }
        if (n % 2 == 1) {
            sum -= mat[n / 2][n / 2];
        }
        return sum;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of rows: ");
        int rows = sc.nextInt();
        System.out.print("Enter the number of columns: ");
        int cols = sc.nextInt();
        int[][] matrix = new int[rows][cols];
        System.out.println("Enter the elements of the matrix: ");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }
        System.out.println("The Diagonal Sum of Matrix: "+matrixDiagonalSum(m
    }
}
```

Output:

```
Enter the number of rows: 3
Enter the number of columns: 3
Enter the elements of the matrix:
1 2 3
4 5 6
7 8 9
The Diagonal Sum of Matrix: 25
```

---

28. Modify the Matrix

Problem statement :

Given a 0-indexed m x n integer matrix matrix, create a new 0-indexed matrix called answer. Make answer equal to matrix, then replace each element with the value -1 with the maximum element in its respective column. Return the matrix answer.

Example 1:

```
Input: matrix = [[1,2,-1],[4,-1,6],[7,8,9]]
Output: [[1,2,9],[4,8,6],[7,8,9]]
```

> **Explanation:** The diagram above shows the elements that are changed (in blue).
>
> - We replace the value in the cell [1][1] with the maximum value in the column 1, that is 8.
> - We replace the value in the cell [0][2] with the maximum value in the column 2, that is 9.

Example 2:

```
Input: matrix = [[3,-1],[5,2]]
Output: [[3,2],[5,2]]
```

> Explanation: The diagram above shows the elements that are changed (in blue).

Constraints:

```
- m == matrix.length
- n == matrix[i].length
- 2 <= m, n <= 50
```

```
- -1 <= matrix[i][j] <= 100
- The input is generated such that each column contains at least one non-nega
```

Analysis:

- Get the matrix input from the user
- Modify the matrix
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class ModifyTheMatrix {
    public static int[][] modifyTheMatrix(int[][] matrix) {
        int rows = matrix.length;
        int cols = matrix[0].length;
        int[][] result = new int[rows][cols];
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                if (matrix[i][j] == -1) {
                    int max = Integer.MIN_VALUE;
                    for (int k = 0; k < rows; k++) {
                        max = Math.max(max, matrix[k][j]);
                    }
                    result[i][j] = max;
                } else {
                    result[i][j] = matrix[i][j];
                }
            }
        }
        return result;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of rows: ");
        int rows = sc.nextInt();
        System.out.print("Enter the number of columns: ");
        int cols = sc.nextInt();
        int[][] matrix = new int[rows][cols];
```

```
        System.out.println("Enter the elements of the matrix: ");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }
        int[][] result = modifyTheMatrix(matrix);
        System.out.println("The modified matrix is: ");
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                System.out.print(result[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Output:

```
Enter the number of rows: 3
Enter the number of columns: 3
Enter the elements of the matrix:
1 2 -1
4 -1 6
7 8 9
The modified matrix is:
1 2 9
4 8 6
7 8 9
```

---

29. Number of Students Doing Homework at a Given Time

Problem statement:

Given two integer arrays startTime and endTime and given an integer queryTime.The ith student started doing their homework at the time startTime[i] and finished it at time endTime[i].Return the number of students doing their homework at time queryTime. More formally, return the number of students where queryTime lays in the interval [startTime[i], endTime[i]] inclusive.

Example 1:

```
Input: startTime = [1,2,3], endTime = [3,2,7], queryTime = 4
Output: 1
```

> **Explanation:** We have 3 students where:
>
> - The first student started doing homework at time 1 and finished at time 3 and wasn't doing anything at time 4.
> - The second student started doing homework at time 2 and finished at time 2 and also wasn't doing anything at time 4.
> - The third student started doing homework at time 3 and finished at time 7 and was the only student doing homework at time 4.

Example 2:

```
Input: startTime = [4], endTime = [4], queryTime = 4
Output: 1
```

> **Explanation:** The only student was doing their homework at the queryTime.

Constraints:

```
startTime.length == endTime.length
1 <= startTime.length <= 100
1 <= startTime[i] <= endTime[i] <= 1000
1 <= queryTime <= 1000
```

Analysis:

- Get the number input from the user
- Calculate the number of students doing homework at a given time
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;
```

```java
public class NumberOfStudentsDoingHomeworkAtAGivenTime {
    public static int numberOfStudentsDoingHomeworkAtAGivenTime(
        int[] startTime,
        int[] endTime,
        int queryTime
    ) {
        int count = 0;
        for (int i = 0; i < startTime.length; i++) {
            if (startTime[i] <= queryTime && queryTime <= endTime[i]) {
                count++;
            }
        }
        return count;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of students: ");
        int n = sc.nextInt();
        int[] startTime = new int[n];
        int[] endTime = new int[n];
        System.out.println("Enter the start time of the students: ");
        for (int i = 0; i < n; i++) {
            startTime[i] = sc.nextInt();
        }
        System.out.println("Enter the end time of the students: ");
        for (int i = 0; i < n; i++) {
            endTime[i] = sc.nextInt();
        }
        System.out.print("Enter the query time: ");
        int queryTime = sc.nextInt();
        System.out.println(
            "The number of students doing their
            homework at the given time is: "
            + numberOfStudentsDoingHomeworkAtAGivenTime(
                startTime, endTime, queryTime)
        );
    }
}
```

Output:

```
Enter the number of students: 3
Enter the start time of the students:
1 2 3
Enter the end time of the students:
3 2 7
Enter the query time: 4
The number of students doing their homework at the given time is: 1
```

---

30. Count Hills and Valleys in an Array

Problem statement :

You are given a 0-indexed integer array nums. An index i is part of a hill in nums if the closest non-equal neighbors of i are smaller than nums[i]. Similarly, an index i is part of a valley in nums if the closest non-equal neighbors of i are larger than nums[i]. Adjacent indices i and j are part of the same hill or valley if nums[i] == nums[j]. Note that for an index to be part of a hill or valley, it must have a non-equal neighbor on both the left and right of the index.Return the number of hills and valleys in nums.

Example 1:

```
Input: nums = [2,4,1,1,6,5]
Output: 3
```

> **Explanation:**
>
> - At index 0: There is no non-equal neighbor of 2 on the left, so index 0 is neither a hill nor a valley.
> - At index 1: The closest non-equal neighbors of 4 are 2 and 1. Since 4 > 2 and 4 > 1, index 1 is a hill.
> - At index 2: The closest non-equal neighbors of 1 are 4 and 6. Since 1 < 4 and 1 < 6, index 2 is a valley.
> - At index 3: The closest non-equal neighbors of 1 are 4 and 6. Since 1 < 4 and 1 < 6, index 3 is a valley, but note that it is part of the same valley as index 2.
> - At index 4: The closest non-equal neighbors of 6 are 1 and 5. Since 6 > 1 and 6 > 5, index 4 is a hill.

- At index 5: There is no non-equal neighbor of 5 on the right, so index 5 is neither a hill nor a valley.
- There are 3 hills and valleys so we return 3.

Example 2:

```
Input: nums = [6,6,5,5,4,1]
Output: 0
```

Explanation:

- At index 0: There is no non-equal neighbor of 6 on the left, so index 0 is neither a hill nor a valley.
- At index 1: There is no non-equal neighbor of 6 on the left, so index 1 is neither a hill nor a valley.
- At index 2: The closest non-equal neighbors of 5 are 6 and 4. Since 5 < 6 and 5 > 4, index 2 is neither a hill nor a valley.
- At index 3: The closest non-equal neighbors of 5 are 6 and 4. Since 5 < 6 and 5 > 4, index 3 is neither a hill nor a valley.
- At index 4: The closest non-equal neighbors of 4 are 5 and 1. Since 4 < 5 and 4 > 1, index 4 is neither a hill nor a valley.
- At index 5: There is no non-equal neighbor of 1 on the right, so index 5 is neither a hill nor a valley.
- There are 0 hills and valleys so we return 0.

Constraints:

```
3 <= nums.length <= 100
1 <= nums[i] <= 100
```

Analysis:

- Get the number input from the user
- Count the hills and valleys in the array
- Return the result

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class CountHillsAndValleysInAnArray {
    public static int countHillsAndValleysInAnArray(int[] nums) {
        int count = 0;
        for (int i = 1; i < nums.length - 1; i++) {
            if (nums[i - 1] != nums[i] && nums[i] != nums[i + 1]) {
                if (nums[i - 1] < nums[i] && nums[i] > nums[i + 1]) {
                    count++;
                } else if (nums[i - 1] > nums[i] && nums[i] < nums[i + 1]) {
                    count++;
                }
            }
        }
        return count;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size: ");
        int size = sc.nextInt();
        int[] arr = new int[size];
        System.out.println("Enter the elements of the array: ");
        for (int i = 0; i < size; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println(
            "The number of hills and valleys in the array is: "
            + countHillsAndValleysInAnArray(arr)
        );
    }
}
```

Output:

```
Enter the size: 6
Enter the elements of the array:
2 4 1 1 6 5
The number of hills and valleys in the array is: 3
```