Problem Solving - II

1. Fibonacci Sum

Problem statement :

Given a number positive number N, find value of f0 + f1 + f2 + . + fN where fi indicates ith Fibonacci number. Remember that f0 = 0, f1 = 1, f2 = 1, f3 = 2, f4 = 3, f5 = 5, Since the answer can be very large, answer modulo 1000000007 should be returned.

Example 1:

```
Input:
    N = 3
Output:
    4
```

> **Explanation:**
>
> 0 + 1 + 1 + 2 = 4

Example 2:

```
Input :
    N = 4
Output :
    7
```

> Explanation :
>
> 0 + 1 + 1 + 2 + 3 = 7

Constraints:

```
1 <= N <= 100000
```

Analysis:

- Get the input number from the user.
- Initialize sum = 1, f = 0, s = 1.
- Run a loop from 2 to n.
- Calculate the next Fibonacci number.
- Update the values of f and s.
- Add the next Fibonacci number to the sum.
- Return the sum.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class FibonacciSum {
    public static int fibonacciSum(int n) {
        int sum = 1,f = 0, s = 1;
        for(int i = 2; i <= n; i++) {
            int next = f + s;
```

```
            f = s;
            s = next;
            sum += next;
        }
        return sum % 1000000007;
    }

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

Output

```
Enter the number :
4
7
```

---

2. The kth Factor of n

Problem statement :

You are given two positive integers n and k. A factor of an integer n is defined as an integer i where n % i == 0. Consider a list of all factors of n sorted in ascending order, return the kth factor in this list or return -1 if n has less than k factors.
#####

Example 1:

```
Input: n = 12, k = 3
Output: 3
```

> Explanation: Factors list is [1, 2, 3, 4, 6, 12], the 3rd factor is 3.

Example 2:

```
Input: n = 7, k = 2
Output: 7
```

> Explanation: Factors list is [1, 7], the 2nd factor is 7.

Constraints:

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

Analysis:

- Get the input number and position from the user.
- Find all the factors of the given number.
- Return the kth factor from the list of factors.

Code:

```java
package com.hands_on;

import java.util.Scanner;
```

```java
public class KthFactorOfN {
    public static int kthFactorOfN(int n,int pos){
        int[] factors = findFactors(n);
        return factors[pos-1];
    }

    public static int[] findFactors(int n){
        int[] arr = new int[n];
        int k = 0;
        for(int i=1; i<=n; i++){
            if (n%i == 0){
                arr[k++] = i;
            }
        }
        return arr;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter any number: ");
        int n = sc.nextInt();
        System.out.println("Enter the position: ");
        int pos = sc.nextInt();
        System.out.println(kthFactorOfN(n, pos));

    }
}
```

Output

```
Enter any number:
12
Enter the position:
3
3
```

---

3. Find the Winner of the Circular Game

Problem Statement:

There are n friends that are playing a game. The friends are sitting in a circle and are numbered from 1 to n inclockwise order. More formally, moving clockwise from the ith friend brings you to the (i+1)th friend for 1 <= i < n, and moving clockwise from the nth friend brings you to the 1st friend.

The rules of the game are as follows:

1. Start at the 1st friend.
2. Count the next k friends in the clockwise direction including the friend you started at. The counting wraps around the circle and may count some friends more than once.
3. The last friend you counted leaves the circle and loses the game.
4. If there is still more than one friend in the circle, go back to step 2 starting from the friend immediately clockwise of the friend who just lost and repeat.
5. Else, the last friend in the circle wins the game.

Given the number of friends, n, and an integer k, return the winner of the game.

Input:

```
n = 5, k = 2
```

Output:

```
3
```

> Explanation: Here are the steps of the game:
>
> 1. Start at friend 1.
> 2. Count 2 friends clockwise, which are friends 1 and 2.
> 3. Friend 2 leaves the circle. Next start is friend 3.
> 4. Count 2 friends clockwise, which are friends 3 and 4.
> 5. Friend 4 leaves the circle. Next start is friend 5.
> 6. Count 2 friends clockwise, which are friends 5 and 1.
> 7. Friend 1 leaves the circle. Next start is friend 3.
> 8. Count 2 friends clockwise, which are friends 3 and 5.
> 9. Friend 5 leaves the circle. Only friend 3 is left, so they are the winner.

Input:

```
n = 6, k = 5
```

```
Output: 1
```

> Explanation: The friends leave in this order: 5, 4, 6, 2, 3. The winner is friend 1.

Constraints:

```
1 <= k <= n <= 500
```

Analysis:

- Get the number of friends and the value of k from the user.
- Create a list of friends from 1 to n.
- Initialize the start index to 0.
- Run a loop until the length of the friends list is greater than 1.
- Calculate the index of the friend to be removed.
- Remove the friend from the list.
- Update the start index.
- Return the remaining friend.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class CircularGame {
    public static int findTheWinner(int n, int k) {
        int[] friends = new int[n];
        for (int i = 0; i < n; i++) {
            friends[i] = i + 1;
        }
        int start = 0;
```

```
        while (friends.length > 1) {
            int removeIndex = (start + k - 1) % friends.length;
            int[] newFriends = new int[friends.length - 1];
            int l = 0;
            for (int i = 0; i < friends.length; i++) {
                if (i != removeIndex) {
                    newFriends[l++] = friends[i];
                }
            }
            friends = newFriends;
            start = removeIndex % friends.length;
        }
        return friends[0];
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of friends: ");
        int n = sc.nextInt();
        System.out.println("Enter the value of k: ");
        int k = sc.nextInt();
        System.out.println("The Winner Of the Game is "+ findTheWinner(n, k));
    }
}
```

Output

```
Enter the number of friends:
8
Enter the value of k:
```

```
2
The Winner Of the Game is 1
```

---

4. Egg Drop With 2 Eggs and N Floors

Problem Statement:

You are given two identical eggs and you have access to a building with n floors labeled from 1 to n. You know that there exists a floor f where 0 <= f <= n such that any egg dropped at a floor higher than f will break, and any egg dropped at or below floor f will not break. In each move, you may take an unbroken egg and drop it from any floor x (where 1 <= x <= n). If the egg breaks, you can no longer use it. However, if the egg does not break, you may reuse it in future moves. Return the minimum number of moves that you need to determine with certainty what the value of f is.

Example 1:

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

> Explanation: We can drop the first egg from floor 1 and the second egg from floor 2.
>
> If the first egg breaks, we know that f = 0.
>
> If the second egg breaks but the first egg didn't, we know that f = 1. Otherwise, if both eggs survive, we know that f = 2.

Example 2:

```
Input: n = 100
Output: 14
```

> Explanation: One optimal strategy is:
>
> - Drop the 1st egg at floor 9. If it breaks, we know f is between 0 and 8. Drop the 2nd egg starting from floor 1 and going up one at a time to find f within 8 more drops. Total drops is 1 + 8 = 9.
> - If the 1st egg does not break, drop the 1st egg again at floor 22. If it breaks, we know f is between 9 and 21. Drop the 2nd egg starting from floor 10 and going up one at a time to find f within 12 more drops. Total drops is 2 + 12 = 14.
> - If the 1st egg does not break again, follow a similar process dropping the 1st egg from floors 34, 45, 55, 64, 72, 79, 85, 90, 94, 97, 99, and 100.

Constraints:

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

Analysis:

- Get the number of floors from the user.
- Initialize the number of moves to 0.
- Run a loop until the number of floors is greater than 0.
- Increment the number of moves.
- Subtract the number of moves from the number of floors.
- Update the number of floors.
- Return the number of moves.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class EggDrop {
    public static int eggDrop(int n) {
        int moves = 0;
        while (n > 0) {
            moves++;
            n -= moves;
        }
        return moves;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of floors: ");
        int n = sc.nextInt();
        System.out.println(
            "Minimum number of moves required to determine the value of f is "
            + eggDrop(n)
        );
    }
}
```

Output

```
Enter the number of floors:
100
Minimum number of moves required to determine the value of f is 14
```

---

5. Airplane Seat Assignment Probability

Problem Statement :

n passengers board an airplane with exactly n seats. The first passenger has lost the ticket and picks a seat randomly. But after that, the rest of the passengers will: - Take their own seat if it is still available, and - Pick other seats randomly when they find their seat occupied - Return the probability that the nth person gets his own seat.

Example 1:

```
Input: n = 1
Output: 1.00000
```

> Explanation: The first person can only get the first seat.

Example 2:

```
Input: n = 2
Output: 0.50000
```

> Explanation: The second person has a probability of 0.5 to get the second seat (when first person gets the first seat).

Constraints:

```
1 <= n <= 10^5
```

Analysis:

- Get the number of passengers from the user.
- Find the probability of the nth person getting his own seat.
- Return the probability.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class AirplaneSeat {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter any number: ");
        int n = sc.nextInt();
        System.out.printf(
            "The Probability of getting airplane seat is %.5f"
            , (1/(double)n)
        );
    }
}
```

Output

```
Enter any number:
5
The Probability of getting airplane seat is 0.20000
```

---

6. Closest Divisor

Problem Statement:

Given an integer num, find the closest two integers in absolute difference whose product equals num + 1 or num + 2. Return the two integers in any order.

Example 1:

```
Input: num = 8
Output: [3,3]
```

> **Explanation:** For num + 1 = 9, the closest divisors are 3 & 3, for num + 2 = 10, the closest divisors are 2 & 5, hence 3 & 3 is chosen.

Example 2:

```
Input: num = 123
Output: [5,25]
```

Constraints:

```
1 <= num <= 10^9
```

Analysis:

- Get the number from the user.
- Find the closest two integers whose product equals num + 1 or num + 2.
- Return the two integers in any order.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class ClosestDivisor {
    public static int[] closestDivisors(int num) {
        int[] divisors1 = findDivisors(num + 1);
        int[] divisors2 = findDivisors(num + 2);
        return Math.abs(divisors1[0] - divisors1[1])
            < Math.abs(divisors2[0] - divisors2[1])
                ? divisors1
                : divisors2;
    }

    public static int[] findDivisors(int num) {
        int[] divisors = new int[2];
        for (int i = (int) Math.sqrt(num); i > 0; i--) {
            if (num % i == 0) {
                divisors[0] = i;
                divisors[1] = num / i;
                break;
            }
        }
```

```
        return divisors;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number: ");
        int num = sc.nextInt();
        int[] result = closestDivisors(num);
        System.out.println(
            "The Closest Divisor for "+ num
            +" is [" + result[0] + "," + result[1] + "]"
        );
    }
}
```

Output

```
Enter the number:
123
The Closest Divisor for 123 is [5,25]
```

---

7. Special Numbers

<h3> Problem statement :</h3> Number is a special number if it's digits only consist 0, 1, 2, 3, 4 or 5. Given a number N and we have to find N-th Special Number.

Example 1:

```
Input:  N = 6
Output: 5
```

> Explanation: First 6 numbers are (0, 1, 2, 3, 4, 5)

Example 2:

```
Input: N = 7
Output: 10
```

> Explanation: First 7 numbers are (0, 1, 2, 3, 4, 5, 10)

Analysis:

- Get the number from the user.
- Find the N-th special number.
- Return the N-th special number.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class SpecialNumbers {
    public static int findSpecialNumber(int N) {
        int count = 0, i = 0;
        while (count <h3 N) {
            String num = String.valueOf(i);
```

```
            boolean isSpecial = true;
            for (char c : num.toCharArray()) {
                if (c > '5') {
                    isSpecial = false;
                    break;
                }
            }
            if (isSpecial) {
                count++;
            }
            i++;
        }
        return i - 1;
    }

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

Output

```
Enter any number:
9
12
```

---

8. NCr

Problem statement :

Given two integers n and r, find nCr. Since the answer may be very large, calculate the answer modulo 10^9+7. ######
Example 1: Input: n = 3, r = 2 Output: 3 > Explaination: 3C2 = 3.
Example 2:

```
Input: n = 2, r = 4
Output: 0
```

> Explaination: r is greater than n.

Constraints:

```
1 ≤ n ≤ 1000
1 ≤ r ≤ 800
```

Analysis:

- Get the values of n and r from the user.
- Calculate the value of nCr.
- Return the value of nCr.

Code:

```
package com.hands_on;
```

```java
import java.util.Scanner;

public class NCR {
    public static int nCr(int n, int r){
        int ncrValue = (factorial(n)/ (factorial(r)* factorial(n-r)) );
        return ncrValue;
    }
    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.println("Enter the value of n: ");
        int n = sc.nextInt();
        System.out.println("Enter the value of r: ");
        int r = sc.nextInt();
        System.out.println("The value of "+n+"C"+r+" is "+ nCr(n,r));

    }
}
```

Output

```
Enter the value of n:
5
Enter the value of r:
```

```
2
The value of 5C2 is 10
```

9. Void of diamond

Problem statement :

You are given an integer ‘N’, ‘N’ will always be an odd integer. Your task is to print a pattern with the following description: 1. The pattern will consist of ‘N’ lines. 2. The pattern will consist of ‘ ‘ (space) and ‘*’ characters only. 3. The pattern will be a “Void of Diamond” pattern. 4. A “Void of Diamond” pattern is a pattern ‘N’ * ‘N’ cells and ‘ ‘ characters make a diamond shape and ‘*’ fill all other points. 5. For a better understanding of the “Void of Diamond” pattern refer to example and sample input-output.

For example:

```
If ‘N’ is 5 then the pattern will be-
*****
** **
*   *
** **
*****
```

Sample Input 1:

```
5
```

Sample Output 1:

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

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

Analysis:

- Get the number from the user.
- Print the Void of Diamond pattern.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class VoidDiamond {
    public static void voidDiamond(int n) {
        for(int i=1; i<=n/2+1; i++)
        {
            for(int j=1; j<=n/2+1-i+1; j++)
            {
                System.out.print("*");
            }
            for(int k=1; k<=2*i-2; k++)
            {
                System.out.print(" ");
            }
            for(int j=1; j<=(n/2+1)-i+1; j++)
            {
                System.out.print("*");
```

```
            }
            System.out.println();
        }
        for(int i=(n/2+1)-1; i>=1; i--)
        {
            for(int j=i; j<=n/2+1; j++)
            {
                System.out.print("*");
            }
            for(int k=1; k<=(2*i)-2; k++)
            {
                System.out.print(" ");
            }
            for(int j=i; j<=n/2+1; j++)
            {
                System.out.print("*");
            }
            System.out.println();
        }
    }

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

Output

```
Enter the number:
5
*****
** **
*   *
** **
*****
```

---

10. Pyramid star pattern

Problem Statement:

Print a pattern of stars in the shape of a rightangled triangle which resembles a pyramid , with the base increasing in size as shown below. The number of rows in the triangle should be determined by user input.

Example:

```
Input 1: N=5
Output 1:
*
* *
* * *
* * * *
* * * * *
```

Constraints:

```
1<=N<= 10
```

Analysis:

- Get the number from the user.
- Print the Pyramid star pattern.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class PyramidStarPattern {
    public static void pyramidStarPattern(int n) {
        for (int i = 1; i <= n; i++) {
            for (int j = 1; j <= i; j++) {
                System.out.print("* ");
            }
            System.out.println();
        }
    }

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

11. Triangle of numbers

Problem statement :

You are given a pattern. Now you need to print the same pattern for any given ‘N’ number of rows.

For example, Pattern for ‘N’ = 4 will be.

```
.
   1
  232
 34545
4567654
```

Analysis:

- Get the number from the user.
- Print the Triangle of numbers pattern.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class TriangleOfNumbers {
    public static void triangleOfNumbers(int n) {
        for (int i = 1; i <= n; i++) {
            int k = i;
```

```
            for (int j = 1; j <= n - i; j++) {
                System.out.print(" ");
            }
            for (int j = 1; j <= i; j++) {
                System.out.print(k++);
            }
            k -= 2;
            for (int j = 1; j < i; j++) {
                System.out.print(k--);
            }
            System.out.println();
        }
    }

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

Output

```
Enter the number:
4
   1
  232
 34543
4567654
```

12. Circle and Rectangle Overlapping

Problem statement :

You are given a circle represented as (radius, xCenter, yCenter) and an axis-aligned rectangle represented as (x1, y1, x2, y2), where (x1, y1) are the coordinates of the bottom-left corner, and (x2, y2) are the coordinates of the top-right corner of the rectangle. Return true if the circle and rectangle are overlapped otherwise return false. In other words, check if there is any point (xi, yi) that belongs to the circle and the rectangle at the same time.

Example 1:

```
Input:
    radius = 1
    xCenter = 0
    yCenter = 0
    x1 = 1
    y1 = -1
    x2 = 3
    y2 = 1
Output:
    true
```

> Explanation: Circle and rectangle share the point (1,0).

Example 2:

```
Input:
    radius = 1
    xCenter = 1
```

```
    yCenter = 1
    x1 = 1
    y1 = -3
    x2 = 2
    y2 = -1
Output:
    false
```

Constraints:

```
1 <= radius <= 2000
-104 <= xCenter, yCenter <= 104
-104 <= x1 < x2 <= 104
-104 <= y1 < y2 <= 104
```

Analysis:

- Get the values of radius, xCenter, yCenter, x1, y1, x2, y2 from the user.
- Check if the circle and rectangle are overlapped.
- Return true if the circle and rectangle are overlapped otherwise return false.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class CircleRectangleOverlap {
    public static boolean checkOverlap(
```

```
        int radius,
        int xCenter,
        int yCenter,
        int x1,
        int y1,
        int x2,
        int y2
    ) {
        int x = Math.max(x1, Math.min(xCenter, x2));
        int y = Math.max(y1, Math.min(yCenter, y2));
        int dx = x - xCenter;
        int dy = y - yCenter;
        return (dx * dx + dy * dy) <= radius * radius;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the radius: ");
        int radius = sc.nextInt();
        System.out.println("Enter the xCenter: ");
        int xCenter = sc.nextInt();
        System.out.println("Enter the yCenter: ");
        int yCenter = sc.nextInt();
        System.out.println("Enter the x1: ");
        int x1 = sc.nextInt();
        System.out.println("Enter the y1: ");
        int y1 = sc.nextInt();
        System.out.println("Enter the x2: ");
        int x2 = sc.nextInt();
        System.out.println("Enter the y2: ");
        int y2 = sc.nextInt();
        if(checkOverlap(radius, xCenter, yCenter, x1, y1, x2, y2)){
            System.out.println("We can overlap circle with rectangle");
```

```
        } else {
            System.out.println("No we can't over lap circle with rectangle");
        }
    }
}
```

Output

```
Enter the radius:
1
Enter the xCenter:
0
Enter the yCenter:
0
Enter the x1:
1
Enter the y1:
-1
Enter the x2:
1
Enter the y2:
3
We can overlap circle with rectangle
```

---

13. Oneful Pairs

Problem statement :

Chef defines a pair of positive integers (a,b) to be a Oneful PairOneful Pair, if a+b+(a·b)=111 For example, (1,55)(1,55) is a Oneful PairOneful Pair, since

1+55+(1·55)=56+55=1111+55+(1·55)=56+55=111.

But (1,56)(1,56) is not a Oneful PairOneful Pair, since

1+56+(1·56)=57+56=113≠1111+56+(1·56)=57+56 =113 not equal to 111.

```
Input 1: 1 55
Output 1: Yes
```

> Explanation: (1,55)(1,55) is a Oneful PairOneful Pair, since 1+55+(1·55)=56+55=1111+55+(1·55)=56+55=111.

```
Input 2: 1 56
Output 2: No
```

> Explanation: (1,56)(1,56) is not a Oneful PairOneful Pair, since 1+56+(1·56)=57+56=113≠1111+56+(1·56)=57+56=113 not equal to 111

Costraints

```
1≤a,b≤1000
```

Analysis:

- Get the values of a and b from the user.
- Check if the pair is a Oneful Pair.

- Return Yes if the pair is a Oneful Pair otherwise return No.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class OnefulPairs {
    public static String isOnefulPair(int a, int b) {
        return a + b + (a * b) == 111 ? "Yes" : "No";
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the value of a: ");
        int a = sc.nextInt();
        System.out.println("Enter the value of b: ");
        int b = sc.nextInt();
        System.out.println(isOnefulPair(a, b));
    }
}
```

Output

```
Enter the value of a:
1
Enter the value of b:
```

```
55
Yes
```

---

14. Water Bottles II

Problem statement :

You are given two integers numBottles and numExchange. numBottles represents the number of full water bottles that you initially have. In one operation, you can perform one of the following

> operations:
>
> Drink any number of full water bottles turning them into empty bottles.
>
> Exchange numExchange empty bottles with one full water bottle.
>
> Then, increase numExchange by one.
>
> Note that you cannot exchange multiple batches of empty bottles for the same value of numExchange. For example, if numBottles == 3 and numExchange == 1, you cannot exchange 3 empty water bottles for 3 full bottles.
>
> Return the maximum number of water bottles you can drink.

Example 1:

```
Input: numBottles = 13, numExchange = 6
Output: 15
```

> Explanation: The table above shows the number of full water bottles, empty water bottles, the value of numExchange, and the number of bottles drunk.

Example 2:

```
Input: numBottles = 10, numExchange = 3
Output: 13
```

> Explanation: The table above shows the number of full water bottles, empty water bottles, the value of numExchange, and the number of bottles drunk.

Constraints:

```
1 <= numBottles <= 100
1 <= numExchange <= 100
```

Analysis:

- Get the values of numBottles and numExchange from the user.
- Calculate the maximum number of water bottles you can drink.
- Return the maximum number of water bottles you can drink.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class WaterBottles {
```

```
    public static int maxWaterBottles(int numBottles, int numExchange) {
        int total = numBottles;
        while(numBottles >= numExchange){
            int newBottles = numBottles / numExchange;
            total += newBottles;
            numBottles = newBottles + numBottles % numExchange;
        }
        return total;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of bottles: ");
        int numBottles = sc.nextInt();
        System.out.println("Enter the number of exchange: ");
        int numExchange = sc.nextInt();
        System.out.println(
            "The maximum number of water bottles you can drink is "
            + maxWaterBottles(numBottles, numExchange)
        );
    }
}
```

Output

```
Enter the number of bottles:
13
Enter the number of exchange:
6
The maximum number of water bottles you can drink is 15
```

15. Decode XORed Permutation

Problem Statement:

There is an integer array perm that is a permutation of the first n positive integers, where n is always odd. It was encoded into another integer array encoded of length n - 1, such that encoded[i] = perm[i] XOR perm[i + 1]. For example, if perm = [1,3,2], then encoded = [2,1].Given the encoded array, return the original array perm. It is guaranteed that the answer exists and is unique.

Example 1:

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

> Explanation: If perm = [1,2,3], then encoded = [1 XOR 2,2 XOR 3] = [3,1]

Example 2:

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

Constraints:

```
3 <= n < 10^5
n is odd.
encoded.length == n - 1
```

Analysis:

- Get the encoded array from the user.
- Find the original array perm.
- Return the original array perm.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class DecodeXORedPermutation {
    public static int[] decode(int[] encoded) {
        int n = encoded.length + 1;
        int[] perm = new int[n];
        int total = 0;
        for (int i = 1; i <= n; i++) {
            total ^= i;
        }
        int encodedTotal = 0;
        for (int i = 1; i < n - 1; i += 2) {
            encodedTotal ^= encoded[i];
        }
        perm[0] = total ^ encodedTotal;
        for (int i = 1; i < n; i++) {
            perm[i] = perm[i - 1] ^ encoded[i - 1];
        }
        return perm;
    }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] encoded = new int[n - 1];
        System.out.println("Enter the elements: ");
        for (int i = 0; i < n - 1; i++) {
            encoded[i] = sc.nextInt();
        }
        int[] perm = decode(encoded);
        System.out.println("The original array is: ");
        for (int i = 0; i < n; i++) {
            System.out.print(perm[i] + " ");
        }
    }
}
```

Output

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

---

16. Single Number II

Problem Statement:

Given an integer array nums where every element appears three times except for one, which appears exactly once. Find the single element and return it.You must implement a solution with a linear runtime complexity and use only constant extra space.

Example 1:

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

Example 2:

```
Input: nums = [0,1,0,1,0,1,99]
Output: 99
```

Constraints:

- 1 <= nums.length <= 3 * 10^4
- -2^31 <= nums[i] <= 2^31 - 1
- Each element in nums appears exactly three times except for one element which appears once

Analysis:

- Get the integer array nums from the user.
- Find the single element that appears exactly once.
- Return the single element.

Code:

```
package com.hands_on;

import java.util.Scanner;

public class SingleNumberII {
    public static int singleNumber(int[] nums) {
        int ones = 0, twos = 0;
        for (int num : nums) {
            ones = (ones ^ num) & ~twos;
            twos = (twos ^ num) & ~ones;
        }
        return ones;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter the elements: ");
        for (int i = 0; i <h3 n; i++) {
            nums[i] = sc.nextInt();
        }
        System.out.println("The single element is: " + singleNumber(nums));
    }
}
```

Output

```
Enter the number of elements:
7
Enter the elements:
0 1 0 1 0 1 99
The single element is: 99
```

---

17. Total Hamming Distance

Problem Statement:

The Hamming distance between two integers is the number of positions at which the corresponding bits are different. Given an integer array nums, return the sum of Hamming distances between all the pairs of the integers in nums.

Example 1:

```
Input: nums = [4,14,2]
Output: 6
```

> Explanation: In binary representation, the 4 is 0100, 14 is 1110, and 2 is 0010 (just showing the four bits relevant in this case).
>
> The answer will be:
>
> HammingDistance(4, 14) + HammingDistance(4, 2) + HammingDistance(14, 2) = 2 + 2 + 2 = 6.

Example 2:

```
Input: nums = [4,14,4]
```

```
Output: 4
```

Constraints:

- 1 <= nums.length <= 10^4
- 0 <= nums[i] <= 10^9
- The answer for the given input will fit in a 32-bit integer.

Analysis:

- Get the integer array nums from the user.
- Find the sum of Hamming distances between all the pairs of the integers in nums.
- Return the sum of Hamming distances.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class TotalHammingDistance {
    public static int totalHammingDistance(int[] nums) {
        int total = 0;
        for (int i = 0; i < 32; i++) {
            int count = 0;
            for (int num : nums) {
                count += (num >> i) & 1;
            }
            total += count * (nums.length - count);
        }
```

```
        return total;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter the elements: ");
        for (int i = 0; i <h4 n; i++) {
            nums[i] = sc.nextInt();
        }
        System.out.println(
            "The total Hamming distance is: "
            + totalHammingDistance(nums)
        );
    }
}
```

Output

```bash
Enter the number of elements:
3
Enter the elements:
4 14 4
The total Hamming distance is: 4
```

---

18. XOR Queries of a Subarray

Problem Statement:

You are given an array arr of positive integers. You are also given the array queries where queries[i] = [lefti, righti].For each query i compute the XOR of elements from lefti to righti (that is, arr[lefti] XOR arr[lefti + 1] XOR ... XOR arr[righti]).Return an array answer where answer[i] is the answer to the ith query.

Example 1:

```
Input: arr = [1,3,4,8], queries = [[0,1],[1,2],[0,3],[3,3]]
Output: [2,7,14,8]
```

> Explanation: The binary representation of the elements in the array are:
>
> 1 = 0001
>
> 3 = 0011
>
> 4 = 0100
>
> 8 = 1000
>
> The XOR values for queries are:
>
> [0,1] = 1 xor 3 = 2
>
> [1,2] = 3 xor 4 = 7
>
> [0,3] = 1 xor 3 xor 4 xor 8 = 14
>
> [3,3] = 8

Example 2:

```
Input: arr = [4,8,2,10], queries = [[2,3],[1,3],[0,0],[0,3]]
Output: [8,0,4,4]
```

Constraints:

- 1 <= arr.length, queries.length <= 3 * 10^4
- 1 <= arr[i] <= 10^9
- queries[i].length == 2
- 0 <= left <= right < arr.length

Analysis:

- Get the integer array arr and queries from the user.
- Compute the XOR of elements from left to right for each query.
- Return an array answer where answer[i] is the answer to the ith query.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class XORQueriesOfSubarray {
    public static int[] xorQueries(int[] arr, int[][] queries) {
        int n = arr.length;
        int[] xor = new int[n + 1];
        for (int i = 0; i < n; i++) {
            xor[i + 1] = xor[i] ^ arr[i];
        }
```

```
        int[] answer = new int[queries.length];
        for (int i = 0; i < queries.length; i++) {
            answer[i] = xor[queries[i][0]] ^ xor[queries[i][1] + 1];
        }
        return answer;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println("Enter the number of queries: ");
        int q = sc.nextInt();
        int[][] queries = new int[q][2];
        System.out.println("Enter the queries: ");
        for (int i = 0; i < q; i++) {
            queries[i][0] = sc.nextInt();
            queries[i][1] = sc.nextInt();
        }
        int[] answer = xorQueries(arr, queries);
        System.out.println("The answer is: ");
        for (int i = 0; i < q; i++) {
            System.out.print(answer[i] + " ");
        }
    }
}
```

Output

```
Enter the number of elements:
4
Enter the elements:
1 3 4 8
Enter the number of queries:
4
Enter the queries:
0 1
1 2
0 3
3 3
The answer is:
2 7 14 8
```

---

19. XOR DARE

Problem Statement:

Ninja and his friends are playing a truth or dare game. Ninja has chosen dare, and his friends have given him a problem to solve. Now ninja is unable to solve the problem, so he asks for your help. In the problem, you are given an integer array ‘ARR’ consisting of ‘N’ elements and an integer number ‘K’. You need to find the maximum possible strength among all the subsets of array ‘ARR’. The strength of a set of elements is bitwise XOR of all the elements present in the set and ‘K’. > Note: For an empty subset, You can consider the XOR of its elements as 0.

Sample Input 1:

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

Sample Output 1:

```
7
11
```

Explanation of sample input 1:

For the first test case:

The subset [3] will produce maximum strength of (3 XOR 4) = 7

For the second test case:

The subset [1, 2] will produce maximum strength of (1 XOR 2 XOR 8)= 11

Sample Input 2:

```
2
2
4 3
3 3 3 3
3 7
1 2 8
```

Sample Output 2:

```
3
15
```

Constraints:

```
1 <= N <= 1000
0 <= ARR[i] <=1000 for ‘i’ in range 0 to N-1
0 <= K <= 1000
```

Analysis:

- Get the number of test cases from the user.
- Get the integer array arr and k from the user.
- Find the maximum possible strength among all the subsets of array arr.
- Return the maximum possible strength.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class XORDare {
    public static int maxStrength(int[] arr, int k) {
        int n = arr.length;
        int maxStrength = 0;
        for (int i = 0; i < (1 << n); i++) {
```

```
            int strength = 0;
            for (int j = 0; j < n; j++) {
                if ((i & (1 << j)) != 0) {
                    strength ^= arr[j];
                }
            }
            maxStrength = Math.max(maxStrength, strength ^ k);
        }
        return maxStrength;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of test cases: ");
        int t = sc.nextInt();
        while (t-- > 0) {
            System.out.println("Enter the number of elements and k: ");
            int n = sc.nextInt();
            int k = sc.nextInt();
            int[] arr = new int[n];
            System.out.println("Enter the elements: ");
            for (int i = 0; i < n; i++) {
                arr[i] = sc.nextInt();
            }
            System.out.println("The maximum strength is: " + maxStrength(arr, k));
        }
    }
}
```

Output

```
Enter the number of test cases:
2
Enter the number of elements and k:
4 3
Enter the elements:
5 4 1 2
The maximum strength is: 7
Enter the number of elements and k:
3 8
Enter the elements:
1 2 8
The maximum strength is: 11
```

---

20. Count Total Setbits

Problem Statement:

You are given a number N. Find the total number of setbits in the numbers from 1 to N.

Example 1:

```
Input: N = 3
Output: 4
```

> Explanation:
>
> 1 -> 01, 2 -> 10 and 3 -> 11.
>
> So total 4 setbits.

Example 2:

```
Input: N = 4
Output: 5
```

Constraints:

```
1 ≤ N ≤ 10^6
```

Analysis:

- Get the number from the user.
- Find the total number of setbits in the numbers from 1 to N.
- Return the total number of setbits.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class CountTotalSetbits {
    public static int countSetBits(int n) {
        int bitCount = 0;
        for (int i = 1; i <= n; i++)
            bitCount += countSetBitsUtil(i);
        return bitCount;
    }
```

```
    public static int countSetBitsUtil(int x)
    {
        if (x <= 0)
            return 0;
        return (x % 2 == 0 ? 0 : 1)
                + countSetBitsUtil(x / 2);
    }

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

Output

```
Enter any number:
7
The total setbit count is 12
```

---

21. Reverse Bits

Problem Statement:

Reverse the bits of an 32 bit unsigned integer A.

Example Input

```
Input 1:
0
Input 2:
3
```

Example Output

```
Output 1:
0
Output 2:
3221225472
```

Analysis:

- Get the number from the user.
- Reverse the bits of the number.
- Return the reversed number.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class ReverseBits {
    public static long reverseBits(long A) {
        long rev = 0;
```

```
        for (int i = 0; i < 32; i++) {
            rev = rev << 1;
            if ((A & (1 << i)) != 0) {
                rev |= 1;
            }
        }
        return rev;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter any number: ");
        long num = sc.nextLong();
        System.out.println("The reversed number is "+ reverseBits(num));
    }
}
```

Output

```
Enter any number:
3
The reversed number is 3221225472
```

---

22. Majority Element

Problem Statement:

Given an array A of N elements. Find the majority element in the array. A majority element in an array A of size N is an element that appears strictly more than N/2 times in the array. ###### Example 1: Input: N = 3 A[] = {1,2,3} Output: -1 > Explanation: Since, each element in {1,2,3} appears only once so there is no majority element.
Example 2:

```
Input:
    N = 5
    A[] = {3,1,3,3,2}
Output:
    3
```

> Explanation: Since, 3 is present more than N/2 times, so it is the majority element.

Constraints:

```
1 ≤ N ≤ 10^7
0 ≤ Ai ≤ 10^6
```

Analysis:

- Get the number of elements from the user.
- Get the integer array A from the user.
- Find the majority element in the array.
- Return the majority element.

Code:

```
package com.hands_on;
```

```java
import java.util.Scanner;

public class MajorityElement {
    public static int majorityElement(int[] A) {
        int count = 0, candidate = -1;
        for (int num : A) {
            if (count == 0) {
                candidate = num;
            }
            count += (num == candidate) ? 1 : -1;
        }
        count = 0;
        for (int num : A) {
            if (num == candidate) {
                count++;
            }
        }
        return count > A.length / 2 ? candidate : -1;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] A = new int[n];
        System.out.println("Enter the elements: ");
        for (int i = 0; i < n; i++) {
            A[i] = sc.nextInt();
        }
        System.out.println("The majority element is: " + majorityElement(A));
    }
}
```

Output

```
Enter the number of elements:
5
Enter the elements:
3 1 3 3 2
The majority element is: 3
```

---

23. Find Missing and Repeating

Problem Statement:

Given an unsorted array Arr of size N of positive integers. One number 'A' from set {1, 2,....,N} is missing and one number 'B' occurs twice in array. Find these two numbers.

Example 1:

```
Input:
    N = 2
    Arr[] = {2, 2}
Output:
    2 1
```

> Explanation: Repeating number is 2 and smallest positive missing number is 1.

Example 2:

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

> Explanation: Repeating number is 3 and smallest positive missing number is 2.

Constraints:

```
2 ≤ N ≤ 10^5
1 ≤ Arr[i] ≤ N
```

Analysis:

- Get the number of elements from the user.
- Get the integer array Arr from the user.
- Find the missing and repeating numbers in the array.
- Return the missing and repeating numbers.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class FindMissingAndRepeating {
    public static int[] findTwoElement(int[] Arr, int N) {
        int[] ans = new int[2];
        for (int i = 0; i < N; i++) {
```

```
            if (Arr[Math.abs(Arr[i]) - 1] > 0) {
                Arr[Math.abs(Arr[i]) - 1] = -Arr[Math.abs(Arr[i]) - 1];
            } else {
                ans[0] = Math.abs(Arr[i]);
            }
        }
        for (int i = 0; i < N; i++) {
            if (Arr[i] > 0) {
                ans[1] = i + 1;
                break;
            }
        }
        return ans;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] Arr = new int[n];
        System.out.println("Enter the elements: ");
        for (int i = 0; i < n; i++) {
            Arr[i] = sc.nextInt();
        }
        int[] ans = findTwoElement(Arr, n);
        System.out.println("The missing and repeating numbers are: ");
        for (int i = 0; i <h3 2; i++) {
            System.out.print(ans[i] + " ");
        }
    }
}
```

Output

```
Enter the number of elements:
3
Enter the elements:
1 3 3
The missing and repeating numbers are:
3 2
```

---

24. Kadane's Algorithm

Problem Statement:

Given an array Arr[] of N integers. Find the contiguous sub-array (containing at least one number) which has the maximum sum and return its sum. ###### Example 1: Input: N = 5 Arr[] = {1,2,3,-2,5} Output: 9 > Explanation: Max subarray sum is 9 of elements (1, 2, 3, -2, 5) which is a contiguous subarray.

Example 2:

```
Input:
N = 4
Arr[] = {-1,-2,-3,-4}
Output:
-1
```

> Explanation: Max subarray sum is -1 of element (-1)

Constraints:

```
1 ≤ N ≤ 10^6
-107 ≤ A[i] ≤ 10^7
```

Analysis:

- Get the number of elements from the user.
- Get the integer array Arr from the user.
- Find the contiguous sub-array which has the maximum sum and return its sum.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class KadanesAlgorithm {
    public static int maxSubarraySum(int[] Arr, int N) {
        int maxSum = Integer.MIN_VALUE, sum = 0;
        for (int i = 0; i < N; i++) {
            sum += Arr[i];
            maxSum = Math.max(maxSum, sum);
            if (sum < 0) {
                sum = 0;
            }
        }
        return maxSum;
    }

    public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] Arr = new int[n];
        System.out.println("Enter the elements: ");
        for (int i = 0; i <h3 n; i++) {
            Arr[i] = sc.nextInt();
        }
        System.out.println("The maximum sum is: " + maxSubarraySum(Arr, n));
    }
}
```

Output

```
Enter the number of elements:
5
Enter the elements:
1 2 3 -2 5
The maximum sum is: 9
```

---

25. Sort an array of 0s, 1s and 2s

Problem Statement:

Given an array of size N containing only 0s, 1s, and 2s; sort the array in ascending order.

Example 1:

```
Input:
N = 5
arr[]= {0 2 1 2 0}
Output:
0 0 1 2 2
```

> Explanation: 0s 1s and 2s are segregated into ascending order.

Example 2:

```
Input:
N = 3
arr[] = {0 1 0}
Output:
0 0 1
```

> Explanation: 0s 1s and 2s are segregated into ascending order.

Constraints:

```
1 <= N <= 10^6
0 <= A[i] <= 2
```

Analysis:

- Get the number of elements from the user.
- Get the integer array arr from the user.
- Sort the array in ascending order.
- Return the sorted array.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class SortArrayOf0s1s2s {
    public static void sort012(int[] arr, int n) {
        int low = 0, mid = 0, high = n - 1;
        while (mid <= high) {
            if (arr[mid] == 0) {
                int temp = arr[low];
                arr[low] = arr[mid];
                arr[mid] = temp;
                low++;
                mid++;
            } else if (arr[mid] == 1) {
                mid++;
            } else {
                int temp = arr[mid];
                arr[mid] = arr[high];
                arr[high] = temp;
                high--;
            }
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
```

```
        System.out.println("Enter the elements: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        sort012(arr, n);
        System.out.println("The sorted array is: ");
        for (int i = 0; i < n; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Output

```
Enter the number of elements:
5
Enter the elements:
0 2 1 2 0
The sorted array is:
0 0 1 2 2
```

---

26. Count Primes

Problem Statement:

Given an integer n, return the number of prime numbers that are strictly less than n.

Example 1:

```
Input: n = 10
Output: 4
```

> Explanation: There are 4 prime numbers less than 10, they are 2, 3, 5, 7.

Example 2:

```
Input: n = 0
Output: 0
```

Constraints:

```
0 <= n <= 5 * 10^6
```

Analysis:

- Get the number from the user.
- Find the number of prime numbers that are strictly less than n.
- Return the number of prime numbers.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class CountPrimes {
    public static int countPrimes(int n) {
```

```
        boolean[] prime = new boolean[n];
        for (int i = 2; i < n; i++) {
            prime[i] = true;
        }
        for (int i = 2; i * i < n; i++) {
            if (prime[i]) {
                for (int j = i * i; j < n; j += i) {
                    prime[j] = false;
                }
            }
        }
        int count = 0;
        for (int i = 2; i < n; i++) {
            if (prime[i]) {
                count++;
            }
        }
        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.println("The number of prime numbers is "+ countPrimes(num));
    }
}
```

Output

```
Enter any number:
10
The number of prime numbers is 4
```

---

27. Watering Plants

Problem Statement:

You want to water n plants in your garden with a watering can. The plants are arranged in a row and are labeled from 0 to n - 1 from left to right where the ith plant is located at x = i. There is a river at x = -1 that you can refill your watering can at. Each plant needs a specific amount of water. You will water the plants in the following way: ● Water the plants in order from left to right. ● After watering the current plant, if you do not have enough water to completely water the next plant, return to the river to fully refill the watering can. ● You cannot refill the watering can early. You are initially at the river (i.e., x = -1). It takes one step to move one unit on the x-axis. Given a 0-indexed integer array plants of n integers, where plants[i] is the amount of water the ith plant needs, and an integer capacity representing the watering can capacity, return the number of steps needed to water all the plants.

Example 1:

```
Input: plants = [2,2,3,3], capacity = 5
Output: 14
```

> Explanation: Start at the river with a full watering can:

- Walk to plant 0 (1 step) and water it. Watering can has 3 units of water.
- Walk to plant 1 (1 step) and water it. Watering can has 1 unit of water.
- Since you cannot completely water plant 2, walk back to the river to refill (2 steps).
- Walk to plant 2 (3 steps) and water it. Watering can has 2 units of water.

- Since you cannot completely water plant 3, walk back to the river to refill (3 steps).
- Walk to plant 3 (4 steps) and water it. Steps needed = 1 + 1 + 2 + 3 + 3 + 4 = 14.

Example 2:

```
Input: plants = [1,1,1,4,2,3], capacity = 4
Output: 30
```

> Explanation: Start at the river with a full watering can:

- Water plants 0, 1, and 2 (3 steps). Return to river (3 steps).
- Water plant 3 (4 steps). Return to river (4 steps).
- Water plant 4 (5 steps). Return to river (5 steps).
- Water plant 5 (6 steps). Steps needed = 3 + 3 + 4 + 4 + 5 + 5 + 6 = 30.

Constraints:

```
n == plants.length
1 <= n <= 1000
1 <= plants[i] <= 10^6
max(plants[i]) <= capacity <= 10^9
```

Analysis:

- Get the integer array plants and capacity from the user.
- Find the number of steps needed to water all the plants.
- Return the number of steps needed.

Code:

```
package com.hands_on;

import java.util.Scanner;

public class WateringPlants {
    public static int minSteps(int[] plants, int capacity) {
        int n = plants.length;
        int steps = 0, water = 0;
        for (int i = 0; i < n; i++) {
            if (plants[i] > water) {
                steps += 2 * i;
                water = capacity;
            }
            water -= plants[i];
        }
        return steps + n;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of plants: ");
        int n = sc.nextInt();
        int[] plants = new int[n];
        System.out.println("Enter the amount of water each plant needs: ");
        for (int i = 0; i <h3 n; i++) {
            plants[i] = sc.nextInt();
        }
        System.out.println("Enter the watering can capacity: ");
        int capacity = sc.nextInt();
        System.out.println(
```

```
            "The number of steps needed is: "
            + minSteps(plants, capacity)
        );
    }
}
```

Output

```
Enter the number of plants:
4
Enter the amount of water each plant needs:
2 2 3 3
Enter the watering can capacity:
5
The number of steps needed is: 14
```

---

28. Trapping Rain Water

Problem Statement:

Given an array arr[] of N non-negative integers representing the height of blocks. If width of each block is 1, compute how much water can be trapped between the blocks during the rainy season.

Example 1:

```
Input:
N = 6
arr[] = {3,0,0,2,0,4}
```

```
Output:
10
```

Example 2:

```
Input:
N = 4
arr[] = {7,4,0,9}
Output:
10
```

> Explanation: Water trapped by above block of height 4 is 3 units and above block of height 0 is 7 units. So, the total unit of water trapped is 10 units.

Example 3:

```
Input:
N = 3
arr[] = {6,9,9}
Output:
0
```

> Explanation: No water will be trapped.

Constraints:

```
3 < N < 10^6
0 < Ai < 10^8
```

Analysis:

- Get the number of elements from the user.
- Get the integer array arr from the user.
- Compute how much water can be trapped between the blocks during the rainy season.
- Return the amount of water trapped.

Code:

```
package com.hands_on;

import java.util.Scanner;

public class TrappingRainWater {
    public static int trap(int[] arr, int n) {
        int left = 0, right = n - 1, leftMax = 0, rightMax = 0, water = 0;
        while (left < right) {
            if (arr[left] < arr[right]) {
                if (arr[left] >= leftMax) {
                    leftMax = arr[left];
                } else {
                    water += leftMax - arr[left];
                }
                left++;
            } else {
                if (arr[right] >= rightMax) {
                    rightMax = arr[right];
                } else {
                    water += rightMax - arr[right];
                }
```

```
                right--;
            }
        }
        return water;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements: ");
        for (int i = 0; i <h3 n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println("The amount of water trapped is: " + trap(arr, n));
    }
}
```

Output

```
Enter the number of elements:
6
Enter the elements:
3 0 0 2 0 4
The amount of water trapped is: 10
```

---

29. Minimize Maximum Pair Sum in Array

Problem Statement :

The pair sum of a pair (a,b) is equal to a + b. The maximum pair sum is the largest pair sum in a list of pairs. For example, if we have pairs (1,5), (2,3), and (4,4), the maximum pair sum would be max(1+5, 2+3, 4+4) = max(6, 5, 8) = 8. Given an array nums of even length n, pair up the elements of nums into n / 2 pairs such that: - Each element of nums is in exactly one pair, and - The maximum pair sum is minimized. Return the minimized maximum pair sum after optimally pairing up the elements.

Example 1:

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

> Explanation: The elements can be paired up into pairs (3,3) and (5,2). The maximum pair sum is max(3+3, 5+2) = max(6, 7) = 7.

Example 2:

```
Input: nums = [3,5,4,2,4,6]
Output: 8
```

> Explanation: The elements can be paired up into pairs (3,5), (4,4), and (6,2). The maximum pair sum is max(3+5, 4+4, 6+2) = max(8, 8, 8) = 8.

Constraints:

```
n == nums.length
2 <= n <= 10^5
n is even.
1 <= nums[i] <= 10^5
```

Analysis:

- Get the integer array nums from the user.
- Pair up the elements of nums into n / 2 pairs such that the maximum pair sum is minimized.
- Return the minimized maximum pair sum.

Code:

```
package com.hands_on;

import java.util.Scanner;

public class MinimizeMaximumPairSum {
    public static int minPairSum(int[] nums) {
        Arrays.sort(nums);
        int n = nums.length, maxSum = 0;
        for (int i = 0; i < n / 2; i++) {
            maxSum = Math.max(maxSum, nums[i] + nums[n - i - 1]);
        }
        return maxSum;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] nums = new int[n];
        System.out.println("Enter the elements: ");
        for (int i = 0; i <h3 n; i++) {
            nums[i] = sc.nextInt();
        }
```

```
        System.out.println(
            "The minimized maximum pair sum is: "
            + minPairSum(nums)
        );
    }
}
```

Output

```
Enter the number of elements:
4
Enter the elements:
3 5 2 3
The minimized maximum pair sum is: 7
```

---

30. Minimum Absolute Sum Difference

Problem Statement:

You are given two positive integer arrays nums1 and nums2, both of length n.The absolute sum difference of arrays nums1 and nums2 is defined as the sum of |nums1[i] - nums2[i]| for each 0 <= i < n (0-indexed).You can replace at most one element of nums1 with any other element in nums1 to minimize the absolute sum difference.Return the minimum absolute sum difference after replacing at most one element in the array nums1. Since the answer may be large, return it modulo 109 + 7. |x| is defined as: - x if x >= 0, or - -x if x < 0.

Example 1:

```
Input: nums1 = [1,7,5], nums2 = [2,3,5]
Output: 3
```

> Explanation: There are two possible optimal solutions:

- Replace the second element with the first: [1,7,5] => [1,1,5], or
- Replace the second element with the third: [1,7,5] => [1,5,5]. Both will yield an absolute sum difference of |1-2| + (|1-3| or |5-3|)
- |5-5| = 3.

Example 2:

```
Input: nums1 = [2,4,6,8,10], nums2 = [2,4,6,8,10]
Output: 0
```

> Explanation: nums1 is equal to nums2 so no replacement is needed. This will result in an absolute sum difference of 0.

Constraints:

```
n == nums1.length
n == nums2.length
1<= n <= 10^5
1 <= nums1[i], nums2[i] <= 10^5
```

Analysis:

- Get the integer arrays nums1 and nums2 from the user.

- Replace at most one element of nums1 with any other element in nums1 to minimize the absolute sum difference.
- Return the minimum absolute sum difference.

Code:

```java
package com.hands_on;

import java.util.Scanner;

public class MinimumAbsoluteSumDifference {
    public static int minAbsoluteSumDiff(int[] nums1, int[] nums2) {
        int n = nums1.length, mod = 1000000007;
        int[] sortedNums1 = nums1.clone();
        Arrays.sort(sortedNums1);
        int maxDiff = 0, sumDiff = 0;
        for (int i = 0; i < n; i++) {
            int diff = Math.abs(nums1[i] - nums2[i]);
            sumDiff = (sumDiff + diff) % mod;
            int idx = Arrays.binarySearch(sortedNums1, nums2[i]);
            if (idx < 0) {
                idx = -idx - 1;
            }
            if (idx < n) {
                maxDiff = Math.max(maxDiff, diff
                        - Math.abs(sortedNums1[idx] - nums2[i]));
            }
            if (idx > 0) {
                maxDiff = Math.max(
                    maxDiff,
                    diff - Math.abs(sortedNums1[idx - 1] - nums2[i])
                );
            }
```

```
        }
        return (sumDiff - maxDiff + mod) % mod;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements: ");
        int n = sc.nextInt();
        int[] nums1 = new int[n];
        int[] nums2 = new int[n];
        System.out.println("Enter the elements of nums1: ");
        for (int i = 0; i < n; i++) {
            nums1[i] = sc.nextInt();
        }
        System.out.println("Enter the elements of nums2: ");
        for (int i = 0; i < n; i++) {
            nums2[i] = sc.nextInt();
        }
        System.out.println(
            "The minimum absolute sum difference is: " +
            minAbsoluteSumDiff(nums1, nums2)
        );
    }
}
```

Output

```
Enter the number of elements:
3
Enter the elements of nums1:
1 7 5
```

```
Enter the elements of nums2:
2 3 5
The minimum absolute sum difference is: 3
```
