Problem Solving - I

1. Calculate Delayed Arrival Time

Problem statement:

You are given a positive integer arrivalTime denoting the arrival time of a train in hours, and another positive integer delayedTime denoting the amount of delay in hours. Return the time when the train will arrive at

> Note that the time in this problem is in 24-hours format.

Example 1:

```
Input: arrivalTime = 15, delayedTime = 5
Output: 20
```

> **Explanation:** Arrival time of the train was 15:00 hours. It is delayed by 5 hours. Now it will reach at 15+5 = 20 (20:00 hours).

Example 2:

```
Input: arrivalTime = 13, delayedTime = 11
Output: 0
```

> **Explanation:** Arrival time of the train was 13:00 hours. It is delayed by 11 hours. Now it will reach at 13+11=24 (Which is denoted by 00:00 in 24 hours format so return 0).

Constraints:

```
1 <= arrivaltime < 24
1 <= delayedTime <= 24
```

Analysis

- The problem is simple and requires only one line of code to solve it.
- We need to add the arrival time and delayed time and return the result.
- The time is in 24-hours format so we need to take care of the overflow condition.
- We can solve this problem in O(1) time complexity.

Code

```
package com.self_practice;
```

```java
import java.util.Scanner;

public class DelayedArrivalTime {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the arrival time of the train: ");
        int arrivalTime = sc.nextInt();
        System.out.println("Enter the delayed time of the train: ");
        int delayedTime = sc.nextInt();
        System.out.println("The train will arrive at: " + delayedArrivalTime(arrivalTime, delayedTime));
    }

    public static int delayedArrivalTime(int arrivalTime, int delayedTime) {
        return (arrivalTime + delayedTime) % 24;
    }
}
```

Output

```
Enter the arrival time of the train:
15
Enter the delayed time of the train:
5
The train will arrive at: 20
```

---

2. Recycling Pens

Problem statement:

You have 'N' empty pens whose refills have been used up. You have 'R' rupees in your pocket. You have two choices of operations that you can perform each time.

1. Recycle 1 empty pen and get 'K' rupees as a reward.
2. Buy 1 refill for 'C' rupees and combine it with 1 empty pen to make one usable pen.

Your task is to find the maximum number of usable pens that you can get. For example if you have 'N' = 5 , 'R' = 10 , 'K' = 2 , 'C' = 3. You can recycle one pen and get 2 rupees as a reward so you will have a total of 12 r with 4 pens to make it usable. So your answer is 4.

Sample Input 1 :

```
3
10 10 5 5
15 11 3 5
3 20 20 2
```

Sample Output 1 :

```
6
7
3
```

> **Explanation for Sample input 1 :**
>
> In the first test case, you will sell 4 empty pens and you will get 20 rupees so your total money will be 10+20 = 30 and from that, you will buy 6 refills and make 6 usable pens.
>
> In second test case you will sell 8 pens and you will get 8*3=24 rupees and your total money will be 24+11 = 35 and from that, you will buy 7 refills and make 7 usable pens.
>
> In this test case, you have a lot of money to buy but you have only 3 empty pens so you buy 3 refills for 6 rupees and make 3 usable pens.

Sample Input 2 :

```
3
10 10 1 10
5 0 5 5
6 0 4 2
```

Sample Output 2 :

```
1
2
4
```

> **Explanation of Sample Input 2 :**
>
> In the first test case you can buy 1 refill from the money you have and make 1 usable pen.
>
> In the second test case you can sell 2 empty pens and get 10 rupees and from it, you can buy 2 refills and make 2 usable pens.
>
> In the third test case you can sell 2 empty pens and get 8 rupees and from it, you can buy 4 refills and make 4 usable pens.

Constraints :

```
1 <= T <= 10^5
1 <= N <= 10^9
0 <= R <= 10^9
1 <= K <= 10^9
1 <= C <= 10^9
```

Analysis

- The problem is simple and requires some basic mathematical calculations.
- We need to find the maximum number of usable pens that we can get.

- We can solve this problem by following the below steps:
  - First, we will calculate the number of pens we can get by recycling the empty pens.
  - Then we will calculate the number of refills we can buy from the money we have.
  - We will take the minimum of both the above values as our answer.
- We can solve this problem in O(1) time complexity.

Code

```java
package com.self_practice;

import java.util.Scanner;

public class RecyclingPens {
    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();
        for (int i = 0; i < t; i++) {
            System.out.println("Enter the number of empty pens, rupees, reward, and cost: ");
            int n = sc.nextInt();
            int r = sc.nextInt();
            int k = sc.nextInt();
            int c = sc.nextInt();
            System.out.println("The maximum number of usable pens that you can get: " + maxUsablePens(n, r, k, c));
        }
    }

    public static int maxUsablePens(int n, int r, int k, int c) {
        int pensByRecycling = r / k;
        int pensByBuyingRefills = n / c;
        return Math.min(pensByRecycling, pensByBuyingRefills);
    }
}
```

Output

```
Enter the number of test cases:
3
Enter the number of empty pens, rupees, reward, and cost:
10 10 5 5
The maximum number of usable pens that you can get: 6

Enter the number of empty pens, rupees, reward, and cost:
15 11 3 5
The maximum number of usable pens that you can get: 7
```

```
Enter the number of empty pens, rupees, reward, and cost:
3 20 20 2
The maximum number of usable pens that you can get: 3
```

---

3. Water Bottles

Problem statement:

There are numBottles water bottles that are initially full of water. You can exchange numExchange empty water bottles from the market with one full water bottle. The operation of drinking a full water bottle turns i numBottles and numExchange, return the maximum number of water bottles you can drink.

Example 1:

```
Input: numBottles = 9, numExchange = 3
Output: 13
```

> **Explanation:** You can exchange 3 empty bottles to get 1 full bottle. Number of full bottles you can drink = 9 + 3 = 12. You can exchange 3 empty bottles to get 1 full bottle. Number of full bottles you can drink = 1 + + 1 = 13.

Example 2:

```
Input: numBottles = 15, numExchange = 4
Output: 19
```

> **Explanation:** You can exchange 4 empty bottles to get 1 full bottle. Number of full bottles you can drink = 15 + 3 = 18. You can exchange 4 empty bottles to get 1 full bottle. Number of full bottles you can drink = 1 18 + 1 = 19.

Constraints:

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

Analysis

- The problem is simple and requires some basic mathematical calculations.
- We need to find the maximum number of water bottles we can drink.
- We can solve this problem by following the below steps:
    - First, we will calculate the number of full bottles we can drink.
    - Then we will calculate the number of empty bottles we can exchange.
    - We will repeat the above two steps until we can exchange the empty bottles.

- We can solve this problem in O(log(numBottles)) time complexity.

Code

```java
package com.self_practice;

import java.util.Scanner;

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

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

Output

```
Enter the number of water bottles and empty bottles required to exchange:
9 3
The maximum number of water bottles you can drink: 13
```

---

4. Water Bottles

Problem statement:

There are numBottles water bottles that are initially full of water. You can exchange numExchange empty water bottles from the market with one full water bottle. The operation of drinking a full water bottle turns i
numBottles and numExchange, return the maximum number of water bottles you can drink. determined by the user input. The program should prompt the user to enter the size of the square and then print the cor

Example 1:

```
Input: Size of square: 3
Output:
    ***
    * *
    ***
```

Example 2:

```
Input: Size of square: 5
Output:
    *****
    *   *
    *   *
    *   *
    *****
```

Analysis

- The problem is simple and requires some basic mathematical calculations.
- We need to print the hollow square pattern based on the user input.
- We can solve this problem by following the below steps:
    - First, we will print the first row of the square.
    - Then we will print the middle rows of the square.
    - Finally, we will print the last row of the square.

Code

```java
package com.self_practice;

import java.util.Scanner;

public class HollowSquarePattern {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of the square: ");
        int size = sc.nextInt();
        printHollowSquarePattern(size);
```

```
    }

    public static void printHollowSquarePattern(int size) {
        for (int i = 1; i <= size; i++) {
            for (int j = 1; j <= size; j++) {
                if (i == 1 || i == size || j == 1 || j == size) {
                    System.out.print("*");
                } else {
                    System.out.print(" ");
                }
            }
            System.out.println();
        }
    }
}
```

Output

```
Enter the size of the square:
5
*****
*   *
*   *
*   *
*****
```

---

5. Reverse Bits

Problem Statement:

Given a 32 bit number x, reverse its binary form and return the answer in decimal.

Example 1:

```
Input:
    x = 1
Output:
    2147483648
```

> Explanation:
>
> Binary of 1 in 32 bits representation 00000000000000000000000000000001
>
> Reversing the binary form we get, 10000000000000000000000000000000,

> whose decimal value is 2147483648.

Example 2:

```
Input:
    x = 5
Output:
    2684354560
```

> Explanation:
>
> Binary of 5 in 32 bits representation00000000000000000000000000000101 Reversing the binary form weget, 10100000000000000000000000000000, whose decimal value is 2684354560.

Constraints:

```
1<= x < 2^32
```

Analysis

- The problem is simple and requires some basic mathematical calculations.
- We need to reverse the binary form of the given number and return the answer in decimal.
- We can solve this problem by following the below steps:
  - First, we will convert the given number into binary form.
  - Then we will reverse the binary form.
  - Finally, we will convert the reversed binary form into decimal.
- We can solve this problem in O(1) time complexity.

Code

```java
package com.self_practice;

import java.util.Scanner;

public class ReverseBits {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number: ");
        int x = sc.nextInt();
        System.out.println("The reversed bits of the number: " + reverseBits(x));
    }

    public static int reverseBits(int x) {
        int reversed = 0;
        for (int i = 0; i < 32; i++) {
            reversed = (reversed << 1) | (x & 1);
```

```
            x >>= 1;
        }
        return reversed;
    }
}
```

Output

```
Enter the number:
5
The reversed bits of the number: 2684354560
```

---

6. XOR Operation in an Array

Problem Statement:

You are given an integer n and an integer start. Define an array nums where nums[i] = start + 2 * i (0- indexed) and n == nums.length. Return the bitwise XOR of all elements of nums.

Example 1:

```
Input: n = 5, start = 0
Output: 8
```

> **Explanation:** Array nums is equal to [0, 2, 4, 6, 8] where (0 ^ 2 ^ 4 ^ 6 ^ 8) = 8.
>
> Where "^" corresponds to bitwise XOR operator.

Example 2:

```
Input: n = 4, start = 3
Output: 8
```

> **Explanation:** Array nums is equal to [3, 5, 7, 9] where (3 ^ 5 ^ 7 ^ 9) = 8.

Constraints:

```
1 <= n <= 1000
 0 <= start <= 1000
 n == nums. Length
```

Analysis

- The problem is simple and requires some basic mathematical calculations.
- We need to find the bitwise XOR of all elements of the array.
- We can solve this problem by following the below steps:
  - First, we will create the array nums.
  - Then we will find the bitwise XOR of all elements of the array.

Code

```java
package com.self_practice;

import java.util.Scanner;

public class XOROperationInArray {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the value of n and start: ");
        int n = sc.nextInt();
        int start = sc.nextInt();
        System.out.println("The bitwise XOR of all elements of the array: " + xorOperation(n, start));
    }

    public static int xorOperation(int n, int start) {
        int result = 0;
        for (int i = 0; i < n; i++) {
            result ^= (start + 2 * i);
        }
        return result;
    }
}
```

Output

```
Enter the value of n and start:
5 0
The bitwise XOR of all elements of the array: 8
```

---

7. Party of Couple

Problem Statement:

You are given an integer array arr[] of size n, representing n number of people in a party, each person is denoted by an integer. Couples are represented by the same number ie: two people have the same integer val single person in the party of couples. ##### ###### Example 1: Input: n = 5 arr = {1, 2, 3, 2, 1} Output: 3 > **Explanation:** Only the number 3 is single.

Example 2:

```
Input:
    n = 11
    arr = {1, 2, 3, 5, 3, 2, 1, 4, 5, 6, 6}
Output:
    4
```

> Explanation: 4 is the only single.

Constraints:

```
1 ≤ n ≤ 10^4
1 ≤ arr[i] ≤ 10^6
```

Analysis

- The problem is simple and requires some basic mathematical calculations.
- We need to find the only single person in the party of couples.
- We can solve this problem by following the below steps:
  - First, we will create a hash map to store the frequency of each person.
  - Then we will iterate through the hash map and find the only single person.
- We can solve this problem in O(n) time complexity.

Code

```java
package com.self_practice;

import java.util.Scanner;

public class PartyOfCouple {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of people in the party: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println("The only single person in the party of couples: " + singlePerson(arr));
    }
```

```
    public static int singlePerson(int[] arr) {
        int singlePerson = 0;
        for (int i : arr) {
            singlePerson ^= i;
        }
        return singlePerson;
    }
}
```

Output

```
Enter the number of people in the party:
5
Enter the elements of the array:
1 2 3 2 1
The only single person in the party of couples: 3
```

---

8. Transpose Matrix

Problem Statement :

Given a 2D integer array matrix, return the transpose of matrix. The transpose of a matrix is the matrix flipped over its main diagonal, switching the matrix's row and column indices.

Example 1:

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

Example 2:

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

Constraints:

```
m == matrix.length
n == matrix[i].length
1 <= m, n <= 1000
1 <= m * n <= 10^5
-109 <= matrix[i][j] <= 10^9
```

Analysis

- The problem is simple and requires some basic mathematical calculations.
- We need to find the transpose of the given matrix.
- We can solve this problem by following the below steps:
    - First, we will create a new matrix with the dimensions of the transpose matrix.
    - Then we will iterate through the given matrix and fill the transpose matrix.
    - Finally, we will return the transpose matrix.

Code

```java
package com.self_practice;

import java.util.Scanner;

public class TransposeMatrix {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of rows and columns: ");
        int m = sc.nextInt();
        int n = sc.nextInt();
        int[][] matrix = new int[m][n];
        System.out.println("Enter the elements of the matrix: ");
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                matrix[i][j] = sc.nextInt();
            }
        }
        int[][] transposeMatrix = transpose(matrix);
        System.out.println("The transpose of the matrix: ");
        for (int i = 0; i < transposeMatrix.length; i++) {
            for (int j = 0; j < transposeMatrix[i].length; j++) {
                System.out.print(transposeMatrix[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static int[][] transpose(int[][] matrix) {
        int m = matrix.length;
        int n = matrix[0].length;
        int[][] transposeMatrix = new int[n][m];
        for (int i = 0; i < m; i++) {
            for (int j = 0; j < n; j++) {
                transposeMatrix[j][i] = matrix[i][j];
            }
```

```
        }
        return transposeMatrix;
    }
}
```

Output

```
Enter the number of rows and columns:
3 3
Enter the elements of the matrix:
1 2 3
4 5 6
7 8 9
The transpose of the matrix:
1 4 7
2 5 8
3 6 9
```

---

9. Valid Mountain Array

Problem statement :

Given an array of integers arr, return true if and only if it is a valid mountain array. Recall that arr is a mountain array if and only if:

- arr.length >= 3
- There exists some i with 0 < i < arr.length - 1 such that:
- arr[0] < arr[1] < ... < arr[i - 1] < arr[i]
- arr[i] > arr[i + 1] > ... > arr[arr.length - 1]

Example 1:

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

Example 2:

```
Input: arr = [3,5,5]
Output: false
```

Example 3:

```
Input: arr = [0,3,2,1]
```

```
Output: true
```

Constraints:

```
1 <= arr.length <= 10^4
0 <= arr[i] <= 10^4
```

Analysis

- The problem is simple and requires some basic mathematical calculations.
- We need to check whether the given array is a valid mountain array or not.
- We can solve this problem by following the below steps:
  - First, we will check the length of the array.
  - Then we will find the peak of the mountain.
  - Finally, we will check the conditions for a valid mountain array.

Code

```java
package com.self_practice;

import java.util.Scanner;

public class ValidMountainArray {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of the array: ");
        int n = sc.nextInt();
        int[] arr = new int[n];
        System.out.println("Enter the elements of the array: ");
        for (int i = 0; i < n; i++) {
            arr[i] = sc.nextInt();
        }
        System.out.println("Is the given array a valid mountain array: " + validMountainArray(arr));
    }

    public static boolean validMountainArray(int[] arr) {
        int n = arr.length;
        int i = 0;
        while (i + 1 < n && arr[i] < arr[i + 1]) {
            i++;
        }
        if (i == 0 || i == n - 1) {
            return false;
        }
        while (i + 1 < n && arr[i] > arr[i + 1]) {
```

```
            i++;
        }
        return i == n - 1;
    }
}
```

Output

```
Enter the size of the array:
4
Enter the elements of the array:
0 3 2 1
Is the given array a valid mountain array: true
```

---

10. Find Odd Occurrence Element

Problem statement :

You are given an array of 'N' elements. In this given array, each element appears an even number of times except one element which appears odd no. of times. Your task is to find the element which occurs an odd n

For example :

> Input array [5,5,6,4,6],If we look at the frequency of different elements in this array.
>
> We can see,4 appears an odd number of times, so our answer will be 4.

Sample input 1 :

```
4
5
2 7 7 7 2
3
9 9 9
7
1 2 3 3 2 1 4
5
11 20 5 5 20
```

Sample output 1 :

```
7
9
```

```
4
11
```

> Explanation for sample output 1 :
>
> (i) For the first array, element 7 appears 3 times.
>
> (ii) For the second array, element 9 appears 3 times.
>
> (iii) For the third array, element 4 appears 1 time.
>
> (iv) For the fourth array, element 11 appears 1 time.

Sample input 2 :

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

Sample output 2 :

```
1
2
6
6
10
```

> Explanation for sample output 2: (i) For the first array, element 1 appears 3 times. (ii) For the second array, element 2 appears 1 time. (iii) For the third array, element 6 appears 1 time. (iv) For the fourth array, ele is only 1 element, so the answer is 10.

Constraints:

```
1<= T <=100
1 <= N <= 10000
1 <= ARR[i] <= 10^8
```

> Where 'T' denotes the number of test cases, 'N' denotes the number of elements in the array, and 'ARR[i]' denotes the 'i'th' element of the array 'ARR'.

Analysis

- The problem is simple and requires some basic mathematical calculations.
- We need to find the element which occurs an odd number of times in the given array.
- We can solve this problem by following the below steps:
  - First, we will create a hash map to store the frequency of each element.
  - Then we will iterate through the hash map and find the element which occurs an odd number of times.
- We can solve this problem in O(n) time complexity.

Code

```java
package com.self_practice;

import java.util.HashMap;

import java.util.Scanner;

public class FindOddOccurrenceElement {
    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();
        for (int i = 0; i < t; i++) {
            System.out.println("Enter the size of the array: ");
            int n = sc.nextInt();
            int[] arr = new int[n];
            System.out.println("Enter the elements of the array: ");
            for (int j = 0; j < n; j++) {
                arr[j] = sc.nextInt();
            }
            System.out.println("The element which occurs an odd number of times: " + findOddOccurrenceElement(arr));
        }
    }

    public static int findOddOccurrenceElement(int[] arr) {
        HashMap<Integer, Integer> map = new HashMap<>();
        for (int i : arr) {
            map.put(i, map.getOrDefault(i, 0) + 1);
        }
        for (int i : map.keySet()) {
            if (map.get(i) % 2 != 0) {
                return i;
            }
        }
        return -1;
```

```
    }
}
```

Output

```
Enter the number of test cases:
4
Enter the size of the array:
5
Enter the elements of the array:
1 1 2 2 1
The element which occurs an odd number of times: 1
```
