1. **Column wise sum of a matrix**

Given a matrix A of size N x M. Find the sum of individual columns in the matrix.

Input Format:

First-line contains an integers 'N'and 'M' which indicates the row and column size of matrix

In the next N lines, you are given M integers.

Output Format:
Display the sum of individual columns
Sample input:

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

Sample Output:

```
12 15 18
```

2. **Sum of even and odd elements in a matrix**

Given a matrix A of size N x M. Find the sum of all even and the sum of all odd elements in the matrix.

Input Format:

First-line contains integers 'N' and 'M' which indicates the row and column size of the matrix

In the next N lines, you are given M integers.

Output Format:
Display the sum of all even elements and sum of all odd elements

Sample input:

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

Sample Output:

```
20 25
```

3. **Maximum and Minimum**

Given an array, find the maximum and minimum values in the array that satisfy the below condition.

Condition:Condition is the element value and the number of times that element is present in the array must be the same.

Note: If no element is satisfying the given condition print -1.

Input Format:
First-line contains an integer 'N' which indicates the length of the Array.
The next line contains 'N' array elements.

Output Format:
Display minimum and maximum element satisfying given condition.

Sample input:
7
1 2 3 4 5 6 2

Sample Output:
1 2

4. N times repeated Elements

Given an array and a number **k**(say), Find the elements that are repeated k times in the given array.
Note: If no element fount print -1.

Input Format:
The first line contains integer 'N' which indicates the length of the Array.
The next line contains 'N' array elements.
The next line contains a number 'K'.

Output Format:
Display elements that are repeated 'K' times in the array.

Sample I/O:

Input:
6
1 1 2 2 3 4
2

Output:
1 2

5. Self Dividing Numbers

A *self-dividing number* is a number that is divisible by every digit it contains.

For example, 128 is a self-dividing number because `128 % 1 == 0`, `128 % 2 == 0`, and `128 % 8 == 0`.

Also, a self-dividing number is not allowed to contain the digit zero.

Given a lower and upper number bound, output a list of every possible self dividing number, including the bounds if possible.

Example 1:

Input:

1

22

Output:

1 2 3 4 5 6 7 8 9 11 12 15 22

Column wise sum of a matrix

```
import java.util.*;
public class ColumnSum
{
   public static void main(String args[])
   {
      Scanner sc=new Scanner(System.in);
      int r=sc.nextInt();
      int c=sc.nextInt();
      int arr[][]=new int[r][c];
      for(int i=0;i<r;i++)
      {
         for(int j=0;j<c;j++)
         {
            arr[i][j]=sc.nextInt();
         }
      }
      for(int i=0;i<c;i++)
      {
         int sum=0;
         for(int j=0;j<r;j++)
         {
           sum+= arr[j][i];
         }
         System.out.print(sum+" ");
      }
   }
}
```

Sum of even and odd elements in a matrix

```
import java.util.*;
public class EvenAndOddSum
{
  public static void main(String args[])
  {
    Scanner sc=new Scanner(System.in);
    int r=sc.nextInt();
    int c=sc.nextInt();
    int arr[][]=new int[r][c];
    int evesum=0;
    int oddsum=0;
    for(int i=0;i<r;i++)
    {
      for(int j=0;j<c;j++)
      {
        arr[i][j]=sc.nextInt();
        if(arr[i][j]%2==0)
        evesum+=arr[i][j];
        else
        oddsum+=arr[i][j];


      }
    }
    System.out.print(evesum+" "+oddsum);
  }
}
```

Maximum and Minimum

```
import java.util.*;
public class MinAndMax
{
    public static void main(String args[])
    {
        Scanner sc=new Scanner(System.in);
        int r=sc.nextInt();
        int arr[]=new int[r];
        int freq[]=new int[20];
        int max=Integer.MIN_VALUE;
        int min=Integer.MAX_VALUE;
        int count=0;
        for(int i=0;i<r;i++)
        {
            arr[i]=sc.nextInt();
            freq[arr[i]]++;
        }
        for(int i=1;i<10;i++)
        {
            if(freq[i]==i)
            {
                if(min>freq[i])
                    min=freq[i];
                    if(max<freq[i])
                    max=freq[i];
                    count++;
            }
        }
        if(count>0)
        System.out.println(min+" "+max);
        else
        System.out.println("-1");
    }
}
```

Explanation:

1. **Input**: The program prompts the user to enter an integer **r**, which represents the number of elements in the array.
2. **Array Initialization**: An integer array **arr** of size **r** is initialized to hold the elements entered by the user. Additionally, an integer array **freq** of size 20 is initialized to store the frequency of each element. This **freq** array assumes that the elements in the input array range from 1 to 9.
3. **Frequency Calculation**: The program reads **r** elements from the user and counts the frequency of each element using the **freq** array.
4. **Minimum and Maximum Initialization**: Variables **max** and **min** are initialized to **Integer.MIN_VALUE** and **Integer.MAX_VALUE** respectively to find the maximum and minimum frequencies.
5. **Finding Minimum and Maximum Frequency**: The program iterates over the **freq** array from index 1 to 9 (inclusive). For each element **i**, if its frequency equals **i**, it updates the **max** and **min** variables accordingly. Additionally, it increments the **count** variable if such elements are found.
6. **Output**: If there are elements in the array with frequency equal to their value, it prints the minimum and maximum frequencies. Otherwise, it prints **-1** to indicate that no such element was found.

N times repeated Elements

```
import java.util.Scanner;

public class NTimesRepeatedK {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
```

```
// Input array length
int r = sc.nextInt();
int arr[] = new int[r];

// Input array elements
for (int i = 0; i < r; i++) {
    arr[i] = sc.nextInt();
}

// Input value of K
int k = sc.nextInt();

// Determine the default size for freq array
int maxPossibleValue = 1000; // Choose a reasonable default size
int freq[] = new int[maxPossibleValue];

// Count frequency of each element
for (int i = 0; i < r; i++) {
    freq[arr[i]]++;
}

boolean found = false;

// Display elements repeated 'k' times
for (int i = 0; i < freq.length; i++) {
    if (freq[i] == k) {
        System.out.print(i + " ");
        found = true;
    }
}
```

```
        // If no element found, print -1
        if (!found) {
            System.out.println("-1");
        }
    }
}
```

Explanation:

1. **Input Format**:
    - The first line contains an integer 'N' indicating the length of the array.
    - The next line contains 'N' array elements.
    - The third line contains a number 'K'.
2. **Output Format**:
    - The program displays elements that are repeated 'K' times in the array.
3. **Logic**:
    - The program reads the length of the array, array elements, and the value of 'K' from the user.
    - It initializes an array **freq** to store the frequency of each element, assuming a reasonable default size.
    - It counts the frequency of each element in the input array using the **freq** array.
    - Then, it iterates through the **freq** array to find elements with a frequency equal to 'K'.
    - If such elements are found, it prints them. Otherwise, it prints "-1" indicating no element is repeated 'K' times in the array.

The program uses an array of size maxPossibleValue to store the frequency of elements.

Note:

This program assumes a maximum possible value for array elements and uses it to determine the size of the frequency array. Adjusting **maxPossibleValue** according to the range of array elements ensures accurate frequency counting.

If 'K' is larger than **maxPossibleValue**, it might not detect elements repeated 'K' times accurately. Adjusting **maxPossibleValue** accordingly can resolve this issue.

```
        }
```

Self Dividing Numbers

```
import java.util.Scanner;

public class SelfDividingNumber {
    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int start = sc.nextInt();
        int end = sc.nextInt();

        for (int i = start; i <= end; i++) {
            if (isSelfDividing(i)) {
                System.out.print(i + " ");
            }
        }
    }

    public static boolean isSelfDividing(int num) {
        int n = num;
        while (n > 0) {
            int digit = n % 10;
            if (digit == 0 || num % digit != 0) {
                return false;
            }
            n /= 10;
        }
        return true;
    }
}
```

Explanation:

1. **Input Format**:
    - The program expects two integers separated by spaces. The first integer represents the starting point of the range, and the second integer represents the ending point of the range.
2. **Output Format**:
    - The program outputs all self-dividing numbers within the given range, separated by spaces.
3. **Logic**:
    - The program iterates through each number in the range from **start** to **end**.
    - For each number, it checks if it is self-dividing by calling the **isSelfDividing()** method.
    - The **isSelfDividing()** method checks whether a number is self-dividing by iterating through its digits and checking divisibility as per the conditions specified.
    - If a number is self-dividing, it is printed; otherwise, it is skipped.

where N is the number of numbers in the given range (**end - start**) and M is the number of digits in each number. The program iterates through each number in the range and checks each digit.

Note:

- Self-dividing numbers are those numbers that are divisible by each of their digits. They cannot contain the digit 0.
- The **isSelfDividing()** method iterates through the digits of a number and checks if the number is divisible by each digit. If any digit is 0 or the number is not divisible by any digit, the method returns false. Otherwise, it returns true.