1) A celebrity is a person who is known to all but does not know anyone at a party. If you go to a party of N people, find if there is a celebrity in the party or not.
A square NxN matrix M[][] is used to represent people at the party such that if an element of row i and column j is set to 1 it means ith person knows jth person. Here M[i][i] will always be 0.
Return the index of the celebrity, if there is no celebrity return -1.
Note: Follow 0-based indexing.
Follow Up: Can you optimize it to O(N)

Example 1:
Input:
N = 3
```
M[][] = {{0 1 0},
         {0 0 0},
         {0 1 0}}
```
Output: 1
Explanation: 0th and 2nd person both
know 1. Therefore, 1 is the celebrity.

Example 2:
Input:
N = 2
```
M[][] = {{0 1},
         {1 0}}
```
Output: -1
Explanation: The two people at the party both
know each other. None of them is a celebrity.

Your Task:
You don't need to read input or print anything. Complete the function celebrity() which takes the matrix M and its size N as input parameters and returns the index of the celebrity. If no such celebrity is present, return -1.

Expected Time Complexity: O(N^2)
Expected Auxiliary Space: O(1)

Constraints:
2 <= N <= 3000
0 <= M[][] <= 1

Sol:

```java
public class CelebrityProblem {
    static boolean knows(int[][] M, int i, int j) {
        return M[i][j] == 1;
    }

    static int findCelebrity(int[][] M, int N) {
        int candidate = 0;

        for (int i = 1; i < N; i++) {
            if (knows(M, candidate, i))
                candidate = i;
        }

        for (int i = 0; i < N; i++) {
            if (i != candidate && (knows(M, candidate, i) || !knows(M, i,
candidate))) {
                return -1;
            }
        }

        return candidate;
    }

    public static void main(String[] args) {
        int[][] M1 = {{0, 1, 0},
                {0, 0, 0},
                {0, 1, 0}};
        int N1 = 3;
        System.out.println(findCelebrity(M1, N1));

        int[][] M2 = {{0, 1},
                {1, 0}};
        int N2 = 2;
        System.out.println(findCelebrity(M2, N2));
    }
}
```

2)Write a program to Validate an IPv4 Address.
According to Wikipedia, IPv4 addresses are canonically represented in dot-decimal notation, which consists of four decimal numbers, each ranging from 0 to 255, separated by dots, e.g., 172.16.254.1 .
A valid IPv4 Address is of the form x1.x2.x3.x4 where 0 <= (x1, x2, x3, x4) <= 255.
Thus, we can write the generalized form of an IPv4 address as (0-255).(0-255).(0-255).(0-255).
Note: Here we are considering numbers only from 0 to 255 and any additional leading zeroes will be considered invalid.

Your task is to complete the function isValid which returns 1 if the given IPv4 address is valid else returns 0. The function takes the IPv4 address as the only argument in the form of a string.

Example 1:

Input:
IPv4 address = 222.111.111.111
Output: 1
Explanation: Here, the IPv4 address is as per the criteria mentioned and also all four decimal numbers lie in the mentioned range.

Example 2:

Input:
IPv4 address = 5555..555
Output: 0
Explanation: 5555..555 is not a valid IPv4 address, as the middle two portions are missing.

Your Task:
Complete the function isValid() which takes the address in the form of string s as an input parameter and returns 1 if this is a valid address otherwise returns 0.

Expected Time Complexity: O(N), N = length of the string.
Expected Auxiliary Space: O(1)

Constraints:
1<=length of string <=50

Note: The Input/Output format and Example given are used for the system's internal purpose, and should be used by a user for Expected Output only. As it is a function problem, hence a user should not read any input from stdin/console. The task is to complete the function specified, and not to write the full code.

Sol:

```java
public class IPv4Validation {
   public static int isValid(String s) {
       String[] parts = s.split("\\.");

       if (parts.length != 4)
           return 0;

       for (String part : parts) {
           if (part.isEmpty() || part.length() > 3)
               return 0;

           for (char c : part.toCharArray()) {
               if (!Character.isDigit(c))
                   return 0;
           }

           if (Integer.parseInt(part) < 0 || Integer.parseInt(part) > 255 ||
(part.length() > 1 && part.charAt(0) == '0'))
               return 0;
       }

       return 1;
   }

   public static void main(String[] args) {
       String address1 = "222.111.111.111";
       System.out.println(isValid(address1));

       String address2 = "5555..555";
       System.out.println(isValid(address2));
   }
}
```

3) The union of two arrays can be defined as the common and distinct elements in the two arrays.
Given two sorted arrays of size n and m respectively, find their union.

Example 1:

Input:
n = 5, arr1[] = {1, 2, 3, 4, 5}
m = 3, arr2 [] = {1, 2, 3, 6, 7}

Output:
1 2 3 4 5 6 7
Explanation:
Distinct elements including both the arrays are: 1 2 3 4 5 6 7.

Example 2:
Input:
n = 5, arr1[] = {2, 2, 3, 4, 5}
m = 5, arr2[] = {1, 1, 2, 3, 4}
Output:
1 2 3 4 5
Explanation:
Distinct elements including both the arrays are: 1 2 3 4 5.
Example 3:

Input:
n = 5, arr1[] = {1, 1, 1, 1, 1}
m = 5, arr2[] = {2, 2, 2, 2, 2}
Output:
1 2
Explanation:
Distinct elements including both the arrays are: 1 2.

Your Task:
You do not need to read input or print anything. Complete the function findUnion() that takes two arrays arr1[], arr2[], and their size n and m as input parameters and returns a list containing the union of the two arrays.

Expected Time Complexity: O(n+m).
Expected Auxiliary Space: O(n+m).

Constraints:
1 <= n, m <= 105
-109 <= arr1[i], arr2[i] <= 109

Sol:
```
=import java.util.*;

public class UnionOfArrays {
   public static ArrayList<Integer> findUnion(int arr1[], int arr2[], int n,
int m) {
       ArrayList<Integer> union = new ArrayList<>();

       int i = 0, j = 0;

```

```java
        while (i < n && j < m) {
            if (arr1[i] < arr2[j]) {
                union.add(arr1[i]);
                i++;
            }
            else if (arr1[i] > arr2[j]) {
                union.add(arr2[j]);
                j++;
            }
            else {
                union.add(arr1[i]);
                i++;
                j++;
            }
        }

        while (i < n) {
            union.add(arr1[i]);
            i++;
        }

        while (j < m) {
            union.add(arr2[j]);
            j++;
        }

        return union;
    }

    public static void main(String[] args) {
        int[] arr1 = {1, 2, 3, 4, 5};
        int[] arr2 = {1, 2, 3, 6, 7};
        int n = arr1.length;
        int m = arr2.length;

        ArrayList<Integer> union = findUnion(arr1, arr2, n, m);
        System.out.println(union);
    }
}
```