Q1: Most Frequent Element

```
import java.util.*;

import static java.lang.Math.min;

public class Source {

 public static int mostFrequentElement(int[] arr) {

   // Write code here

        int n= arr.length;

       int maxCount = 0;

       int ans = 0;

       if(arr[0] == 0){

           ans = -1;

       }

       else {

           for(int i=0;i<n;i++){

               int currCount = 1;

               for(int j=i+1;j<n;j++){

                   if(arr[i] == arr[j]){

                       currCount++;

                   }

               }

               if(currCount > maxCount){

                   maxCount = currCount;

                   ans = arr[i];

               }else if(maxCount == currCount){

                   ans = min(ans,arr[i]);

               }

           }

       }

       return ans;
 }
 public static void main(String[] args) {
   int n;
   Scanner sc = new Scanner(System.in);
   n = sc.nextInt();
   int arr[] = new int[n];
   for(int i = 0; i < n; i++){
      arr[i] = sc.nextInt();
   }
   System.out.println(mostFrequentElement(arr));
 }
}
```

Q2: Check Whether an Undirected Graph is a Tree or Not

```
import java.util.*;

public class Source {

   private int vertexCount;
   private static LinkedList<Integer> adj[];

   Source(int vertexCount) {
      this.vertexCount = vertexCount;
      this.adj = new LinkedList[vertexCount];
      for (int i = 0; i < vertexCount; ++i) {
         adj[i] = new LinkedList<Integer>();
      }
   }
```

```
public void addEdge(int v, int w) {
   if (!isValidIndex(v) || !isValidIndex(w)) {
      return;
   }
   adj[v].add(w);
   adj[w].add(v);
}

private boolean isValidIndex(int i) {
   // Write code here
   return true;
}

private boolean isCyclic(int v, boolean visited[], int parent) {
   // Write code here
   visited[v] = true;
    Integer i;
    Iterator<Integer> it = adj[v].iterator();
    while (it.hasNext())
    {
    i = it.next();
    if (!visited[i])
    {
    if (isCyclic(i, visited, v))
    return true;
    }
    else if (i != parent)
    return true;
    }
    return false;
}
```

```
public boolean isTree() {
   // Write Code here
   boolean visited[] = new boolean[vertexCount];
    for (int i = 0; i < vertexCount; i++)
    visited[i] = false;

    if (isCyclic(0, visited, -1))
    return false;
    for (int u = 0; u < vertexCount; u++)
    if (!visited[u])
    return false;
    return true;
}

public static void main(String args[]) {
   Scanner sc = new Scanner(System.in);
   // Get the number of nodes from the input.
   int noOfNodes =  sc.nextInt();
    // Get the number of edges from the input.
   int noOfEdges = sc.nextInt();

   Source graph = new Source(noOfNodes);
   // Adding edges to the graph
   for (int i = 0; i <noOfEdges; ++i) {
      graph.addEdge(sc.nextInt(),sc.nextInt());
   }
   if (graph.isTree()) {
      System.out.println("Yes");
   } else {
      System.out.println("No");
```

```
        }
    }
}
```

Q3: Find kth Largest Element in a Stream

```
import java.util.*;
public class Source {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int k = sc.nextInt();
        int stream[] = new int[n];
        for (int i = 0; i < n; i++) {
            stream[i] = sc.nextInt();
        }
        int[] v = kthLargest(k, stream, n);
        for (int it : v)
            if (it == -1) {
                System.out.println("None");
            } else {
                System.out.println(k + " largest number is " + it);
            }
}

// Write code here
static int[] kthLargest(int k, int arr[], int n) {
        int[] ans = new int[n];
        PriorityQueue<Integer> pq = new PriorityQueue<>((a, b) -> a - b);
        for (int i = 0; i < n; i++) {
            if (pq.size() < k)
                pq.add(arr[i]);
            else {
                if (arr[i] > pq.peek()) {
                    pq.remove();
                    pq.add(arr[i]);
                }
            }
            if (pq.size() < k)
                ans[i] = -1;
            else
                ans[i] = pq.peek();
        }
        return ans;
    }
}
```

Q4: Sort Nearly Sorted Array

```
import java.util.*;
public class Source {
    private static void sortArray(int[] arr, int k) {
        // Write code here
        // Sort the array using
         // inbuilt function
         Arrays.sort(arr);
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int k = sc.nextInt();
        int arr[] = new int[n];

        for(int i = 0; i < n; i++){
            arr[i] = sc.nextInt();
        }
        sortArray(arr, k);

        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

Q5: Find Sum Between pth and qth Smallest Elements

```
import java.util.*;

public class Source {
    public static int sumBetweenPthToQthSmallestElement(int[] arr, int p, int q) {
      // Write code here
        Arrays.sort(arr);
         int result = 0;
         for (int i = p; i < q - 1; i++)
         result += arr[i];
         return result;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int arr[] = new int[n];
        for(int i = 0; i < n; i++){
            arr[i] = sc.nextInt();
        }
        int p = sc.nextInt();
        int q = sc.nextInt();
        System.out.println(sumBetweenPthToQthSmallestElement(arr, p, q));
    }
}
```

Q6: Find All Symmetric Pairs in an Array

```
import java.util.*;

public class Source {
    public static void symmetricPair(int[][] arr) {
        // Write code here
        HashMap<Integer, Integer> hM = new HashMap<Integer, Integer>();
        for (int i = 0; i < arr.length; i++)
        {
            int first = arr[i][0];
            int sec = arr[i][1];
            Integer val = hM.get(sec);
            if (val != null && val == first)
                System.out.println(sec + " " + first );
            else
                hM.put(first, sec);
        }
    }
    public static void main(String arg[]) {
        Scanner sc = new Scanner(System.in);
        int row = sc.nextInt();
        int arr[][] = new int[row][2];
        for(int i = 0 ; i < row ; i++){
            for(int j = 0 ; j < 2 ; j++){
                arr[i][j] = sc.nextInt();
            }
        }
        symmetricPair(arr);
    }
}
```

Q7: Find All Common Element in All Rows of Matrix

```
import java.util.*;
public class Source {
    public static void printElementInAllRows(int mat[][], int M, int N) {
        // Write code here
        int arr[] = new int[M*N];
        int count=0;
        Map<Integer,Integer> mp = new HashMap<>();
        for (int i = 0; i < N; i++)
            mp.put(mat[0][i],1);
        for (int i = 1; i < M; i++) {
            for (int j = 0; j < N; j++) {
                if (mp.get(mat[i][j]) != null && mp.get(mat[i][j]) == i) {
                    mp.put(mat[i][j], i + 1);
                    if (i == M - 1)
                        arr[count++]= mat[i][j];
                }
            }
        }
        Arrays.sort(arr);
        for(int i=0;i<arr.length;i++)
            if(arr[i]!=0)
                System.out.print(arr[i]+" ");
    }

    public static void main(String[] args) {
        Scanner sc =  new Scanner(System.in);
        int row = sc.nextInt();
        int col = sc.nextInt();

        int matrix[][] = new int[row][col];
        for(int i = 0 ; i < row ; i++){
            for(int j = 0 ; j < col ; j++){
                matrix[i][j] = sc.nextInt();
            }
        }

        printElementInAllRows(matrix, row, col);
    }
}
```

Q8: Find Itinerary in Order

```
import java.util.*;


public class Source {


    public static void findItinerary(Map<String, String> tickets) {
        // Write code here
        Map<String, String> reverseMap = new HashMap<String, String>();
 for (Map.Entry<String,String> entry: tickets.entrySet())
 reverseMap.put(entry.getValue(), entry.getKey());
 String start = null;
 for (Map.Entry<String,String> entry: tickets.entrySet())
 {
 if (!reverseMap.containsKey(entry.getKey()))
 {
 start = entry.getKey();
 break;
 }
 }
 if (start == null)
 {
 System.out.println("Invalid Input");
 return;
 }
 String to = tickets.get(start);
 while (to != null)
 {
 System.out.println(start + "->" + to);
 start = to;
 to = tickets.get(to);
 }
    }

    public static void main(String[] args) {
        Map<String, String> tickets = new HashMap<String, String>();
        Scanner sc = new Scanner(System.in);
        int n  = sc.nextInt();
        for(int i = 0 ; i < n ; i++){
            tickets.put(sc.next(),sc.next());
        }
        findItinerary(tickets);
    }
}
```

Q9: Search Element in a Rotated Array

```
import java.util.*;

public class Source {
    public static int search(int arr[], int left, int right, int key) {
        // Write code here
         int count=-1;
         boolean flag=false;
         for(int i=0;i<arr.length;i++) {
         if(arr[i]==key) {
         flag = true;
         count = i;
         break;
         }
        else count = i;
        }
        if(flag)
        return count;
        else return -1;
    }

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        int arr[] = new int[n];
        for(int i = 0 ; i < n ; i++){
            arr[i] = sc.nextInt();
        }
        int key = sc.nextInt();
        int i = search(arr, 0, n - 1, key);
        if (i != -1) {
            System.out.println(i);
        } else {
            System.out.println("-1");
        }
    }
}
```

Q10: Find Median After Merging Two Sorted Arrays

```
import java.util.*;

public class Source {
    public static int median(int[] arr1, int[] arr2 , int n){
```

```
    // Write code here
     int i = 0, j = 0, m1 = -1, m2 = -1;
     for (int count = 0; count <= n; count++) {
     if (i == n) {
     m1 = m2;
     m2 = arr2[0];
     break;
     } else if (j == n) {
     m1 = m2;
     m2 = arr1[0];
     break;
     }
     if (arr1[i] <= arr2[j]) {
     m1 = m2;
     m2 = arr1[i];
     i++;
     }
     else {
     m1 = m2;
     m2 = arr2[j];
     j++;
     }
     }
     return (m1 + m2)/2;
  }

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


    int arr1[] = new int[n];
```

```
        int arr2[] = new int[n];


        for(int i = 0 ; i < n ; i++){
            arr1[i] = sc.nextInt();
        }


        for(int i = 0 ; i < n ; i++){
            arr2[i] = sc.nextInt();
        }
        System.out.println(median(arr1, arr2, n));
    }
}
```