1. Most Frequent Element

```
import java.util.*;
import java.util.HashMap;
import java.util.Map;
import java.util.Map.Entry;

public class Source {

   public static int mostFrequentElement(int[] arr) {

      // Write code here
      HashMap<Integer, Integer> hash = new HashMap<>();
      int n = arr.length;
      for (int i = 0; i < n; i++) {
         int key = arr[i];
         if (hash.containsKey(key)) {
            int freq = hash.get(key);
            freq++;
            hash.put(key, freq);
         } else {
            hash.put(key, 1);
         }
      }
      int max_count = 0 ,res = -1;
      for(Entry<Integer,Integer> val : hash.entrySet()){
         if(max_count < val.getValue()) {
            res = val.getKey();
            max_count = val.getValue();
         }
```

```
        }
        return res;
    }

    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));
    }
}
```


2. 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 i < this.vertexCount;

   }

   private boolean isCyclic(int v, boolean visited[], int parent) {
```

```
    // Write code here
    visited[v] = true;
    Integer i;

    Iterator <Integer> iterator = adj[v].iterator();
    while (iterator.hasNext()) {
        i =iterator.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;

    // The call to isCyclicUtil serves multiple purposes
    // It returns true if graph reachable from vertex 0
    // is cyclic. It also marks all vertices reachable
    // from 0.
    if (isCyclic(0, visited, -1))
        return false;
```

```
        // If we find a vertex which is not reachable from 0
        // (not marked by isCyclicUtil(), then we 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");
        }
    }
}
```

3. 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();

        }


// Write code here
         PriorityQueue<Integer> heap = new PriorityQueue<>(k);

        for (int i = 0; i < n; i++) {
            if (heap.size() < k) {
                heap.offer(stream[i]);
            } else if (heap.peek() < stream[i]) {
                heap.poll();
                heap.offer(stream[i]);
            }

            if (heap.size() == k) {
```

```
            System.out.println(k + " largest number is " + heap.peek());
        } else {
            System.out.println("None");
        }
    }
  }
}
```


4. Sort Nearly Sorted Array

```
import java.util.*;

public class Source {

   private static void sortArray(int[] arr, int k) {
```

```
    // Write code here
    if (arr == null || arr.length == 0) {
        return;
    }
    PriorityQueue<Integer> minHeap = new PriorityQueue<>();
     // the array
    int minCount = Math.min(arr.length, k + 1);
    // add first k + 1 items to the min heap
    for (int i = 0; i < minCount; i++) {
        minHeap.add(arr[i]);
    }

    int index = 0;
    for (int i = k + 1; i < arr.length; i++) {
        arr[index++] = minHeap.peek();
        minHeap.poll();
        minHeap.add(arr[i]);
    }

    while (!minHeap.isEmpty()) {
        arr[index++] = minHeap.poll();
    }
}


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] + " ");
        }
    }
}
```

5. 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));
    }
}
```

6. Find All Symmetric Pairs in an Array

```
import java.util.*;

public class Source {

    public static void symmetricPair(int[][] arr) {
        // Write code here
        Set<String> set = new HashSet<>();
        for (int[] pair : arr) {
        int x = pair[1], y = pair[0];
```

```
      if (set.contains(y + " " + x)) {
        System.out.println(x + " " + y);
      } else {
        set.add(x + " " + y);
      }
    }
  }

  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);
  }
}
```

7. Find All Common Element in All Rows of Matrix

```
import java.util.*;

public class Source {

   public static void printElementInAllRows(int mat[][]) {

      // Write code here
      HashMap <Integer,Integer> countMap = new HashMap<>();
      int m = mat.length;
      int n = mat[0].length;
      for(int i = 0; i < m; i++) {
```

```
        HashSet <Integer> rowSet = new HashSet<>();
        for(int j = 0; j < n; j++) {
            if(rowSet.contains(mat[i][j])){
                continue;
            }
            rowSet.add(mat[i][j]);
            countMap.put(mat[i][j], countMap.getOrDefault(mat[i][j],0)+1);
        }
    }
    List<Integer> res = new ArrayList<>();
    for(Map.Entry<Integer,Integer> entry : countMap.entrySet()) {
        if(entry.getValue() == m) {
            res.add(entry.getKey());
        }
    }
    Collections.sort(res);
    for (int num : res) {
        System.out.print(num + " ");
    }
}

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);
   }
}
```

8. 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);
    }
}
```

9. 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
      if (left > right)
         return -1;

      int mid = (left + right) / 2;
      if (arr[mid] == key)
         return mid;

      /* If arr[l...mid] first subarray is sorted */
      if (arr[left] <= arr[mid]) {
```

```
        /* As this subarray is sorted, we
           can quickly check if key lies in
           half or other half */
        if (key >= arr[left] && key <= arr[mid])
            return search(arr, left, mid - 1, key);
        /*If key not lies in first half subarray,
       Divide other half  into two subarrays,
       such that we can quickly check if key lies
       in other half */
        return search(arr, mid + 1, right, key);
    }

    /* If arr[l..mid] first subarray is not sorted,
       then arr[mid... h] must be sorted subarray*/
    if (key >= arr[mid] && key <= arr[right])
        return search(arr, mid + 1, right, key);

    return search(arr, left, mid - 1, key);
}
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");
        }
    }
}
```

10. 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;
        int j = 0;
        int count;
        int m1 = -1, m2 = -1;

        for (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));
    }
}
```