1. Most Frequent Element

```
import java.util.*;

public class Source {

 public static int mostFrequentElement(int[] arr) {

   // Write code here
   int n=arr.length;
   int maxFreq = 0;
        int mostFreqElement = -1;
        Map<Integer, Integer> freqMap = new HashMap<>();
        for (int i = 0; i < n; i++) {
            int element = arr[i];
            int freq = freqMap.getOrDefault(element, 0) + 1;
            freqMap.put(element, freq);
            if (freq > maxFreq) {
                maxFreq = freq;
                mostFreqElement = element;
            }
        }
         return mostFreqElement;
 }


 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 >= 0 && i < vertexCount);
    }

    private boolean isCyclic(int v, boolean visited[], int parent) {
        // Write code here
         visited[v] = true;

    // Recur for all the vertices adjacent to this vertex
    for (int i : adj[v]) {
        // If an adjacent vertex is not visited, then recur for it
        if (!visited[i]) {
            if (isCyclic(i, visited, v)) {
                return true;
            }
        }
        // If an adjacent vertex is visited and not parent of current vertex, then
there is a cycle
        else if (i != parent) {
            return true;
        }
    }

    return false;
    }

    public boolean isTree() {
        // Write Code here
        boolean visited[] = new boolean[vertexCount];
    Arrays.fill(visited, false);

    // Check if the graph contains a cycle
    if (isCyclic(0, visited, -1)) {
        return false;
    }
```

```
    // Check if all the vertices are visited
    for (int i = 0; i < vertexCount; i++) {
        if (!visited[i]) {
            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> minHeap = new PriorityQueue<>();

        for (int i = 0; i < n; i++) {
            if (minHeap.size() < k) {
                minHeap.add(stream[i]);
            } else {
                if (stream[i] > minHeap.peek()) {
                    minHeap.poll();
                    minHeap.add(stream[i]);
                }
            }
            if (minHeap.size() == k) {
                System.out.println(k + " largest number is " + minHeap.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


        PriorityQueue<Integer> minHeap = new PriorityQueue<>();
         int[] result = new int[arr.length];

         for (int i = 0; i <= k; i++) {
        minHeap.offer(arr[i]);
    }

    int index = 0;
    for (int i = k+1; i < arr.length; i++) {
        result[index++] = minHeap.poll();
        minHeap.offer(arr[i]);
```

```
    }

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

    for (int i = 0; i < arr.length; i++) {
        arr[i] = result[i];
    }
    }


    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 pthSmallest = arr[p - 1];
        int qthSmallest = arr[q - 1];
        int sum = 0;
        for (int i = 0; i < arr.length; i++) {
            if (arr[i] > pthSmallest && arr[i] < qthSmallest) {
                sum += arr[i];
            }
        }
        return sum;
```

```
    }

    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
         Map<Integer, Integer> map = new HashMap<>();
    for (int i = 0; i < arr.length; i++) {
        int a = arr[i][0];
        int b = arr[i][1];
        if (map.containsKey(b) && map.get(b) == a) {
            System.out.println(b + " " + a);
        } else {
            map.put(a, b);
        }
    }
    }

    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
         Set<Integer> commonElements = new HashSet<>();

        // Add all the elements of the first row to the HashSet
        for (int i = 0; i < mat[0].length; i++) {
            commonElements.add(mat[0][i]);
        }

        // Iterate over the remaining rows and remove any element from the HashSet
that is not present in the current row
        for (int i = 1; i < mat.length; i++) {
            Set<Integer> currentRowElements = new HashSet<>();
            for (int j = 0; j < mat[i].length; j++) {
                currentRowElements.add(mat[i][j]);
            }
            commonElements.retainAll(currentRowElements);
        }

        // Print the common elements in ascending order
        List<Integer> sortedElements = new ArrayList<>(commonElements);
        Collections.sort(sortedElements);
        for (int i = 0; i < sortedElements.size(); i++) {
            System.out.print(sortedElements.get(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);
    }
}
```

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> reverseTickets = new HashMap<>();
        for (Map.Entry<String, String> entry : tickets.entrySet()) {
            reverseTickets.put(entry.getValue(), entry.getKey());
        }

        String startCity = null;
        for (String city : tickets.keySet()) {
            if (!reverseTickets.containsKey(city)) {
                startCity = city;
                break;
            }
        }
        if (startCity == null) {
            System.out.println("Invalid Input");
            return;
        }

        ArrayList<String> itinerary = new ArrayList<>();
        itinerary.add(startCity);

        while (itinerary.size() < tickets.size() + 1) {
            String lastCity = itinerary.get(itinerary.size() - 1);
            String nextCity = tickets.get(lastCity);
            itinerary.add(nextCity);
        }

        for (int i = 0; i < itinerary.size() - 1; i++) {
            System.out.println(itinerary.get(i) + "->" + itinerary.get(i + 1));
        }

    }

    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 (right >= left) {
            int mid = left + (right - left) / 2;

            if (arr[mid] == key)
                return mid;

            if (arr[mid] < key)
                return search(arr, left, mid - 1, key);

            return search(arr, mid + 1, right, key);
        }

        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");
        }
    }
}
```

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[] merged = new int[2*n];
        int i = 0, j = 0, k = 0;
        while (i < n && j < n) {
            if (arr1[i] < arr2[j]) {
                merged[k++] = arr1[i++];
            } else {
                merged[k++] = arr2[j++];
            }
        }
        while (i < n) {
            merged[k++] = arr1[i++];
        }
        while (j < n) {
            merged[k++] = arr2[j++];
        }
        if (2*n % 2 == 1) {
            return merged[2*n/2];
        } else {
            return (merged[2*n/2] + merged[2*n/2-1]) / 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));
    }
}
```