1) Most Frequent Element

```
import java.util.*;


public class Source {


 public static int mostFrequentElement(int[] arr) {
    int n = arr.length;
    int maxcount = 0;
   int element_having_max_freq = 0;
   for (int i = 0; i < n; i++) {
     int count = 0;
     for (int j = 0; j < n; j++) {
       if (arr[i] == arr[j]) {
         count++;
       }
     }


     if (count > maxcount) {
       maxcount = count;
       element_having_max_freq = arr[i];
     }
   }


   return element_having_max_freq;


 }
```

```
 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) {
  return true; // Write code here
}

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

public boolean isTree() {
   boolean visited[] = new boolean[vertexCount];
                for (int k = 0; k < vertexCount; k++)
                        visited[k] = false;
                if (isCyclic(0, visited, -1))
                        return false;

                for (int k = 0; k < vertexCount; k++)
                        if (!visited[k])
                                return false;

                return true;// Write Code here
}

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 {
    static PriorityQueue<Integer> minheap;
    static int k;

     static List<Integer> findMaximum(int arr[])
     {
            List<Integer> list = new ArrayList<>();

        // one by one adding values to the min heap
        for (int val : arr) {

            // if the heap size is less than k , we add to the heap
            if (minheap.size() < k)
                minheap.add(val);
```

```
/*
otherwise ,
first we  compare the current value with the
min heap TOP value

if TOP val > current element , no need to
remove TOP , bocause it will be the largest kth
element anyhow

else  we need to update the kth largest element
by removing the top lowest element
*/

else {
   if (val > minheap.peek()) {
      minheap.poll();
      minheap.add(val);
   }
}

// if heap size >=k we add
// kth largest element
// otherwise -1

if (minheap.size() >= k)
   list.add(minheap.peek());
else
   System.out.println("None");
```

```
        }
        return list;
    }
    public static void main(String[] args) {
     minheap = new PriorityQueue<>();
        Scanner sc = new Scanner(System.in);

        int n = sc.nextInt();

         k = sc.nextInt();

        int stream[] = new int[n];

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

            stream[i] = sc.nextInt();

        }
         List<Integer> result = findMaximum(stream);

        for (int x : result)
            System.out.println(k+" largest number is "
                               + x);


    }
}
```

4)Sort Nearly Sorted Array

```
import java.util.*;


public class Source {


    private static void sortArray(int[] arr, int k) {
      int n=arr.length;
        if (arr == null || arr.length == 0) {
            return;
        }
        // min heap
        PriorityQueue<Integer> priorityQueue = new PriorityQueue<>();
        // if there are less than k + 1 elements present in 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++) {
            priorityQueue.add(arr[i]);
        }


        int index = 0;
        // here even if size=k then n will be n=k,so i<n works fine
        for (int i = k + 1; i < n; i++) {
            arr[index++] = priorityQueue.peek();
            priorityQueue.poll();
            priorityQueue.add(arr[i]);
        }
```

```
        Iterator<Integer> itr = priorityQueue.iterator();

        while (itr.hasNext()) {
            arr[index++] = priorityQueue.peek();
            priorityQueue.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) {
   Arrays.sort(arr);
   int size=arr.length;
   int sum=0;
   if(1<=p && q<size){
   for(int i=p;i<q-1;i++)
   {
     sum=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) {
        Map<Integer,Integer> map=new HashMap<Integer,Integer>();
        for(int i=0;i<arr.length;i++)
        {
        // First and second elements of current pair
            int first = arr[i][0];
            int sec   = arr[i][1];

            // Look for second element of this pair in hash
            Integer val = map.get(sec);

            // If found and value in hash matches with first
            // element of this pair, we found symmetry
            if (val != null && val == first)
              System.out.println(sec + " " + first );
```

```
            else  // Else put sec element of this pair in hash
                map.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);
        }
}
```

7)Find All Common Element in All Rows of Matrix

```
import java.util.*;

public class Source {

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

        // Create a HashMap to store the count of each element in the matrix
        Map<Integer, Integer> elementCount = new HashMap<>();
```

```
    // Traverse the first row of the matrix and add the elements to the HashMap
    for(int j=0; j<mat[0].length; j++) {
        int element = mat[0][j];
        elementCount.put(element, 1);
    }

    // Traverse the remaining rows of the matrix and update the count of each element in the HashMap
    for(int i=1; i<mat.length; i++) {
        for(int j=0; j<mat[i].length; j++) {
            int element = mat[i][j];
            if(elementCount.containsKey(element) && elementCount.get(element) == i) {
                // Increment the count of the element if it has already been encountered in the previous
rows
                elementCount.put(element, i+1);
            }
        }
    }

    // Print the elements that have occurred in all the rows of the matrix
    List<Integer> commonElements = new ArrayList<>();
    for(Map.Entry<Integer, Integer> entry : elementCount.entrySet()) {
        if(entry.getValue() == mat.length) {
            commonElements.add(entry.getKey());
        }
    }

    // Sort the common elements in ascending order
    Collections.sort(commonElements);
```

```
        // Print the common elements
        for(int element : commonElements) {
            System.out.print(element + " ");
        }
    }

    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) {
      Map<String, String> updatedlist = new HashMap<String, String>();

for (Map.Entry<String,String> entry: tickets.entrySet())
updatedlist.put(entry.getValue(), entry.getKey());

String startcity = null;

for (Map.Entry<String,String> entry: tickets.entrySet())
{
if (!updatedlist.containsKey(entry.getKey()))
{
startcity = entry.getKey();
break;
}
}

if (startcity == null)
{
System.out.println("Invalid Input");
return;
}

String dstcity = tickets.get(startcity);
while (dstcity != null)
{

System.out.print(startcity + "->" + dstcity + "\n");
```

```
startcity = dstcity;
dstcity = tickets.get(dstcity);
} // Write code here
    }

    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) {
        int pivot = findPivot(arr, left, right);


        if (pivot == -1)
```

```
        return binarySearch(arr, left, right, key);



    if (arr[pivot] == key)
        return pivot;
    if (arr[0] <= key)
        return binarySearch(arr, left, pivot - 1, key);
    return binarySearch(arr, pivot + 1, right, key);
}

static int findPivot(int arr[], int low, int high)
{
    // base cases
    if (high < low)
        return -1;
    if (high == low)
        return low;

    /* low + (high - low)/2; */
    int mid = (low + high) / 2;
    if (mid < high && arr[mid] > arr[mid + 1])
        return mid;
    if (mid > low && arr[mid] < arr[mid - 1])
        return (mid - 1);
    if (arr[low] >= arr[mid])
        return findPivot(arr, low, mid - 1);
    return findPivot(arr, mid + 1, high);
}
```

```
  static int binarySearch(int arr[], int low, int high, int key)
  {
     if (high < low)
        return -1;
     int mid = (low + high) / 2;
     if (key == arr[mid])
        return mid;
     if (key > arr[mid])
        return binarySearch(arr, (mid + 1), high, key);
     return binarySearch(arr, low, (mid - 1), key);
// Write code here
  }

  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){


     int[] result = new int[arr1.length + arr2.length];
      int i = 0, j = 0, k = 0;
      while (i < arr1.length && j < arr2.length) {
         if (arr1[i] < arr2[j])
            result[k++] = arr1[i++];
         else
            result[k++] = arr2[j++];
      }
      while (i < arr1.length)
         result[k++] = arr1[i++];
      while (j < arr2.length)
         result[k++] = arr2[j++];


      return (result[n - 1] + result[n]) / 2;   // Write code here
   }


   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));
    }
}
```