//Most Frequent Element

```
//Find the most frequent element in an array
import java.util.*;

class Source {

   static int mostFrequent(int arr[], int n) {

      // Sort the array
      Arrays.sort(arr);

      // find the max frequency using linear
      // traversal
      int max_count = 1, res = arr[0];
      int curr_count = 1;

      for (int i = 1; i < n; i++) {
         if (arr[i] == arr[i - 1])
            curr_count++;
         else {
            if (curr_count > max_count) {
               max_count = curr_count;
               res = arr[i - 1];
            }
            curr_count = 1;
         }
      }
```

```
    // If last element is most frequent
    if (curr_count > max_count) {
        max_count = curr_count;
        res = arr[n - 1];
    }

    return res;
}


public static void main(String[] args) {


    Scanner scan = new Scanner(System.in);

    int no = scan.nextInt();

    if (no < 1) {
        System.out.println("-1");
    } else {

        int[] values = new int[no];

        for (int i = 0; i < no; i++) {
            values[i] = scan.nextInt();
        }


        int n = values.length;

        System.out.println(mostFrequent(values, n));
```

```
    }
  }
}
```

//Check Whether an Undirected Graph is a Tree or Not

```
//Program to check whether a graph is tree or not

import java.util.Iterator;
import java.util.LinkedList;
import java.util.Scanner;

// This class represents a directed graph using adjacency list representation
class Source
{
   private int V; // No. of vertices
   private LinkedList<Integer> adj[]; //Adjacency List

   // Constructor

   @SuppressWarnings("unchecked")
   Source(int v)
   {
      V = v;
      adj = new LinkedList[V];
      for (int i=0; i<v; ++i)
         adj[i] = new LinkedList<Integer>();
   }

   // Function to add an edge into the graph
   void addEdge(int v,int w)
   {
      adj[v].add(w);
      adj[w].add(v);
   }
```

```
// A recursive function that uses visited[] and parent
// to detect cycle in subgraph reachable from vertex v.
boolean isCyclicUtil(int v, boolean visited[], int parent)
{
   // Mark the current node as visited
   visited[v] = true;
   Integer i;

   // Recur for all the vertices adjacent to this vertex
   Iterator<Integer> it = adj[v].iterator();
   while (it.hasNext())
   {
      i = it.next();

      // If an adjacent is not visited, then recur for
      // that adjacent
      if (!visited[i])
      {
         if (isCyclicUtil(i, visited, v))
            return true;
      }

      // If an adjacent is visited and not parent of
      // current vertex, then there is a cycle.
      else if (i != parent)
         return true;
   }
   return false;
}
```

```
// Returns true if the graph is a tree, else false.
boolean isTree()
{
   // Mark all the vertices as not visited and not part
   // of recursion stack
   boolean visited[] = new boolean[V];
   for (int i = 0; i < V; i++)
      visited[i] = false;

   // The call to isCyclicUtil serves multiple purposes
   // It returns true if graph reachable from vertex 0
   // is cyclcic. It also marks all vertices reachable
   // from 0.
   if (isCyclicUtil(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 < V; u++)
      if (!visited[u])
         return false;

   return true;
}

// Driver method
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");
        }

    }
}
```

//Find kth Largest Element in a Stream

```
// Java Program for the above approach
import java.util.*;
class Source
{
   static void kthLargest(int stream[], int n, int k)
   {

      // Create a min heap and store first k-1 elements
      // of stream into
      Vector<Integer> pq = new Vector<Integer>(n);

      // Push first k elements and print "_" (k-1) times
      for (int i = 0; i < k - 1; i++)
      {
         pq.add(stream[i]);
         System.out.println("None");
      }
      pq.add(stream[k - 1]);

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

         // We must insert last element before we
         // decide last k-th largest output.
         Collections.sort(pq);
         System.out.println(k+ " largest number is "+pq.get(0));
         if (stream[i] > pq.get(0))
         {
            pq.remove(0);
```

```
            pq.add(stream[i]);
        }
    }

    // Print last k-th largest element (after
    // (inserting last element)
    Collections.sort(pq);
    System.out.println(k+ " largest number is "+pq.get(0));
}

// Driver code
public static void main(String[] args) {


    Scanner scan = new Scanner(System.in);

    int size = scan.nextInt();
    int k = scan.nextInt();


        int[] arr = new int[size];

        for (int i = 0; i < size; i++) {
            arr[i] = scan.nextInt();
        }


    //int arr[] = {10, 20, 11, 70, 50, 40, 100, 55};
    //int k = 3;
    int n = arr.length;
    kthLargest(arr, n, k);
```

```
    }
}
```

//Sort Nearly Sorted Array

```
//Program to sort a nearly sorted array

import java.util.Iterator;
import java.util.PriorityQueue;
import java.util.Scanner;

class Source {
   private static void kSort(int[] arr, int n, int k)
   {

      // min heap
      PriorityQueue<Integer> priorityQueue
            = new PriorityQueue<>();

      // add first k + 1 items to the min heap
      for (int i = 0; i < k + 1; i++) {
         priorityQueue.add(arr[i]);
      }

      int index = 0;
      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();
    }
}

// A utility function to print the array
private static void printArray(int[] arr, int n)
{
    for (int i = 0; i < n; i++)
        System.out.print(arr[i] + " ");
}

// Driver Code
public static void main(String[] args)
{
    Scanner scan = new Scanner(System.in); //MAKE SCANNER

    int size = scan.nextInt(); //FIRST SCAN THE ARRAY SIZE
    int k = scan.nextInt();    //SECOND SCAN SOME OTHER VARIABLE


    int[] arr = new int[size]; //MAKE ARRAY OF SIZE SIZE FROM USER

    for (int i = 0; i < size; i++) { //INPUT FROM USER FOR ARRAY OF SIZE SIZE
        arr[i] = scan.nextInt();
    }



    kSort(arr, size, k);
    printArray(arr, size);
```

```
    }
}
```

//Find Sum Between pth and qth Smallest Elements

```
import java.util.Arrays;
import java.util.Scanner;

class Source {

   // Returns sum between two kth smallest element of array
   static int sumBetweenTwoKth(int arr[],
                               int k1, int k2)
   {
      // Sort the given array
      Arrays.sort(arr);

      // Below code is equivalent to
      int result = 0;

      for (int i = k1; i < k2 - 1; i++)
         result += arr[i];

      return result;
   }

   // Driver code
   public static void main(String[] args)
   {

     // int arr[] = { 20, 8, 22, 4, 12, 10, 14 };
     // int k1 = 3, k2 = 6;
     // int n = arr.length;
```

```
        Scanner scan = new Scanner(System.in); //MAKE SCANNER

        int size = scan.nextInt(); //FIRST SCAN THE ARRAY SIZE



        int[] arr = new int[size]; //MAKE ARRAY OF SIZE SIZE FROM USER

        for (int i = 0; i < size; i++) { //INPUT FROM USER FOR ARRAY OF SIZE SIZE
           arr[i] = scan.nextInt();
        }

        int k1 = scan.nextInt();    //THIRD SCAN SOME OTHER VARIABLE
        int k2 = scan.nextInt();    //FOURTH SCAN SOME OTHER VARIABLE

        System.out.print(sumBetweenTwoKth(arr,
              k1, k2));
    }
}
```

//Find All Symmetric Pairs in an Array

```
import java.util.HashMap;
import java.util.Scanner;

class Source {

   // Print all pairs that have a symmetric counterpart
   static void findSymPairs(int arr[][])
   {
      // Creates an empty hashMap hM
      HashMap<Integer, Integer> hM = new HashMap<Integer, Integer>();

      // Traverse through the given array
      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 = hM.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
```

```
            hM.put(first, sec);
        }
    }

    // Driver method
    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();
            }
        }
        findSymPairs(arr);
    }



    }
```

//Find All Common Element in All Rows of Matrix

```
import java.util.*;

class Source
{




   // prints common element in all rows of matrix
   static void printCommonElements(int[][] mat, int row, int col) {

      // Specify number of rows and columns
      int M = row;
      int N =col;

      Map<Integer, Integer> mp = new HashMap<>();
      Set<Integer> hash_Set = new TreeSet<>();

      // initalize 1st row elements with value 1
         for (int j = 0; j < N; j++)
         mp.put(mat[0][j], 1);

     // int size=M*N;

      // traverse the matrix
   //  int[] arr = new int[size];//MAKE ARRAY OF SIZE SIZE FROM USER
```

```
        int n=0;



        for (int i = 1; i < M; i++) {
           for (int j = 0; j < N; j++) {
              // If element is present in the map and
              // is not duplicated in current row.
              if (mp.get(mat[i][j]) != null && mp.get(mat[i][j]) == i) {
                 // we increment count of the element
                 // in map by 1

                //  arr[n] = mat[i][j];
                 mp.put(mat[i][j], i + 1);
                 hash_Set.add(mat[i][j]);
                 n++;

                 // If this is last row
                 if (i == M - 1) {

                   // System.out.print(mp.get(mat[i][j]));

                    /*
                    for (int n = 0; n < size; n++) { //INPUT FROM USER FOR ARRAY OF SIZE SIZE
                       arr[n] = mat[i][j];

                    }
*/
                   // for (int l = 0; l < arr.length; l++) {

                       // Print array element present at index i
                    //   System.out.print(arr[l] + " ");
```

```
                // }
                //System.out.print(hash_Set);


                 //  System.out.println("hi");

                hash_Set.forEach( element ->{
                  System.out.print(element+" ");});
                 break;
            }
        }
    }
}
}



// Driver code
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();
            }
        }

        // System.out.println(matrix);
        printCommonElements(matrix,row,col);
    }
}
```

//Find Itinerary in Order

```
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class Source
{
   // Driver function
   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());
      }
      printResult((tickets));
   }




   // This function populates 'result' for given input 'dataset'
   private static void printResult(Map<String, String> dataSet)
   {
      // To store reverse of given map
      Map<String, String> reverseMap = new HashMap<String, String>();
```

```
// To fill reverse map, iterate through the given map
for (Map.Entry<String,String> entry: dataSet.entrySet())
   reverseMap.put(entry.getValue(), entry.getKey());

// Find the starting point of itinerary
String start = null;
for (Map.Entry<String,String> entry: dataSet.entrySet())
{
   if (!reverseMap.containsKey(entry.getKey()))
   {
      start = entry.getKey();
      break;
   }
}

// If we could not find a starting point, then something wrong
// with input
if (start == null)
{
   System.out.println("Invalid Input");
   return;
}

// Once we have starting point, we simple need to go next, next
// of next using given hash map
String to = dataSet.get(start);
while (to != null)
{
   System.out.println(start + "->" + to);
   start = to;
```

```
            to = dataSet.get(to);
        }
    }
}
```

//Search Element in a Rotated Array

```
import java.util.Scanner;

class Source {

    /* Searches an element key in a
    pivoted sorted array arrp[]
    of size n */
    static int pivotedBinarySearch(int arr[], int n, int key)
    {
        int pivot = findPivot(arr, 0, n - 1);

        // If we didn't find a pivot, then
        // array is not rotated at all
        if (pivot == -1)
            return binarySearch(arr, 0, n - 1, key);

        // If we found a pivot, then first
        // compare with pivot and then
        // search in two subarrays around pivot
        if (arr[pivot] == key)
            return pivot;
        if (arr[0] <= key)
            return binarySearch(arr, 0, pivot - 1, key);
        return binarySearch(arr, pivot + 1, n - 1, key);
    }

    /* Function to get pivot. For array
```

```
3, 4, 5, 6, 1, 2 it returns
3 (index of 6) */
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);
}




/* Standard Binary Search function */
static int binarySearch(int arr[], int low, int high, int key)
{
   if (high < low)
      return -1;

   /* low + (high - low)/2; */
```

```
    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);
}




// main function
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 = pivotedBinarySearch(arr, n, key);
    if (i != -1) {
        System.out.println(i);
    } else {
        System.out.println("-1");
    }



}
```

}

//Find Median After Merging Two Sorted Arrays

```
import java.util.Scanner;


class Source
{
   // function to calculate median
   static int getMedian(int ar1[], int ar2[], int n)
   {
      int i = 0;
      int j = 0;
      int count;
      int m1 = -1, m2 = -1;


                  /* Since there are 2n elements, median will
                  be average of elements at index n-1 and
                  n in the array obtained after merging ar1
                  and ar2 */
      for (count = 0; count <= n; count++)
      {
                              /* Below is to handle case where all
                              elements of ar1[] are smaller than
                              smallest(or first) element of ar2[] */
         if (i == n)
         {
            m1 = m2;
            m2 = ar2[0];
            break;
         }


                              /* Below is to handle case where all
```

```
                        elements of ar2[] are smaller than
                        smallest(or first) element of ar1[] */
    else if (j == n)
    {
        m1 = m2;
        m2 = ar1[0];
        break;
    }
                        /* equals sign because if two
                        arrays have some common elements */
    if (ar1[i] <= ar2[j])
    {
        /* Store the prev median */
        m1 = m2;
        m2 = ar1[i];
        i++;
    }
    else
    {
        /* Store the prev median */
        m1 = m2;
        m2 = ar2[j];
        j++;
    }
  }

  return (m1 + m2)/2;
}
```

```
    /* Driver program to test above function */
    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(getMedian(arr1, arr2, n));

    }
}
```