1. **Most Frequent Element**

|  |
| --- |
| import java.util.\*; |
|  |  |
|  | public class Source { |
|  |  |
|  | static int mostFrequent(int arr[], int n) |
|  | { |
|  | Arrays.sort(arr); |
|  | int max\_count = 1; |
|  | int 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 (curr\_count > max\_count) |
|  | { |
|  | max\_count = curr\_count; |
|  | res = arr[n - 1]; |
|  | } |
|  | 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(mostFrequent(arr, n)); |
|  | } |
|  | } |

1. **Check if an Undirected Graph is a Tree or Not**

|  |
| --- |
| import java.io.\*; |
|  | import java.util.\*; |
|  |  |
|  |  |
|  | public class Source { |
|  |  |
|  | private int vertexCount; |
|  | private static LinkedList<Integer> adj[]; |
|  |  |
|  |  |
|  | @SuppressWarnings("unchecked") |
|  | Source(int vertexCount) { |
|  | this.vertexCount = vertexCount; |
|  | Source.adj = new LinkedList[vertexCount]; |
|  | for (int i = 0; i < vertexCount; ++i) { |
|  | adj[i] = new LinkedList<Integer>(); |
|  | } |
|  | } |
|  |  |
|  | public void addEdge(int v, int w) { |
|  | adj[v].add(w); |
|  | adj[w].add(v); |
|  | } |
|  |  |
|  | private boolean isValidIndex(int i) { |
|  | return false; |
|  | // Write code here |
|  |  |
|  | } |
|  |  |
|  | private boolean isCyclic(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 (isCyclic(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; |
|  | } |
|  |  |
|  | public boolean isTree() |
|  | { |
|  | // Mark all the vertices as not visited and not part |
|  | // of recursion stack |
|  | boolean visited[] = new boolean[vertexCount]; |
|  | for (int i = 0; i < vertexCount; i++) |
|  | visited[i] = false; |
|  |  |
|  | // The call to isCyclic serves multiple purposes |
|  | // It returns true if graph reachable from vertex 0 |
|  | // is cyclcic. 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 isCyclic(), 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"); |
|  | } |
|  | sc.close(); |
|  | } |
|  | } |

1. **Find kth Largest Element in a Stream**

|  |
| --- |
| import java.util.\*; |
|  |  |
|  | public class Source { |
|  | static PriorityQueue<Integer> min; |
|  | static int k; |
|  |  |
|  | static List<Integer> getAllKthNumber(int arr[]) |
|  | { |
|  | List<Integer> list = new ArrayList<>(); |
|  | for (int val : arr) { |
|  | if (min.size() < k) |
|  | min.add(val); |
|  | else { |
|  | if (val > min.peek()) { |
|  | min.poll(); |
|  | min.add(val); |
|  | } |
|  | } |
|  | if (min.size() >= k) |
|  | list.add(min.peek()); |
|  | else |
|  | list.add(-1); |
|  | } |
|  | return list; |
|  | } |
|  | public static void main(String[] args) { |
|  | 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(); |
|  | } |
|  | sc.close(); |
|  | min = new PriorityQueue<>(); |
|  | List<Integer> res = getAllKthNumber(stream); |
|  | for (int x : res){ |
|  | if(x == -1){ |
|  | System.out.println("None"); |
|  | } |
|  | else{ |
|  | System.out.println( k + " largest number is " + x); |
|  | } |
|  | } |
|  | } |
|  | } |

1. **Sort Nearly Sorted Array**

|  |
| --- |
| import java.util.\*; |
|  |  |
|  | public class Source { |
|  |  |
|  | private static void sortArray(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(); |
|  | } |
|  | } |
|  |  |
|  |  |
|  | 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, n, k); |
|  |  |
|  | for (int i = 0; i < arr.length; i++) { |
|  | System.out.print(arr[i] + " "); |
|  | } |
|  | sc.close(); |
|  | } |
|  | } |

**5. Find Sum Between pth and qth Smallest Element**

|  |
| --- |
| import java.util.\*; |
|  |  |
|  | public class Source { |
|  |  |
|  | public static int sumBetweenPthToQthSmallestElement(int[] arr, int p, int q) { |
|  | // Sort the given array |
|  | Arrays.sort(arr); |
|  |  |
|  | // Below code is equivalent to |
|  | 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)); |
|  | sc.close(); |
|  | } |
|  | } |

**6. Find All Symmetric Pairs in an Array**

|  |
| --- |
| import java.util.\*; |
|  |  |
|  | public class Source { |
|  |  |
|  | public static void symmetricPair(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); |
|  | } |
|  | } |
|  |  |
|  | 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 Each Row of Matrix**

|  |
| --- |
| import java.util.\*; |
|  |  |
|  | public class Source { |
|  |  |
|  | public static void printElementInAllRows( int mat[][], int row, int col ) |
|  | { |
|  | int[] arr = new int[row]; |
|  | int count = 0; |
|  | Map<Integer,Integer> mp = new HashMap<>(); |
|  | for (int j = 0; j < col; j++) |
|  | mp.put(mat[0][j],1); |
|  | for (int i = 1; i < row; i++) |
|  | { |
|  | for (int j = 0; j < col; j++) |
|  | { |
|  | if (mp.get(mat[i][j]) != null && mp.get(mat[i][j]) == i) |
|  | { |
|  | mp.put(mat[i][j], i + 1); |
|  | if (i == row - 1){ |
|  | count++; |
|  | 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); |
|  | sc.close(); |
|  | } |
|  | } |

**8. Find Itinerary in Order**

|  |
| --- |
| import java.util.\*; |
|  |  |
|  | public class Source { |
|  |  |
|  | public static void findItinerary(Map<String, String> dataSet) |
|  | { |
|  | Map<String, String> reverseMap = new HashMap<String, String>(); |
|  | for (Map.Entry<String,String> entry: dataSet.entrySet()) |
|  | reverseMap.put(entry.getValue(), entry.getKey()); |
|  | String start = null; |
|  | for (Map.Entry<String,String> entry: dataSet.entrySet()) |
|  | { |
|  | if (!reverseMap.containsKey(entry.getKey())) |
|  | { |
|  | start = entry.getKey(); |
|  | break; |
|  | } |
|  | } |
|  | if (start == null) |
|  | { |
|  | System.out.println("Invalid Input"); |
|  | return; |
|  | } |
|  | String to = dataSet.get(start); |
|  | while (to != null) |
|  | { |
|  | System.out.print(start + "->" + to + "\n"); |
|  | start = to; |
|  | to = dataSet.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); |
|  | sc.close(); |
|  | } |
|  | } |

**9. Search Element in a Rotated Array**

|  |
| --- |
| import java.util.\*; |
|  |  |
|  | public class Source { |
|  |  |
|  | public static int search(int arr[], int t) |
|  | { |
|  | ArrayList<Integer> clist = new ArrayList<>(); |
|  |  |
|  | for (int i : arr) |
|  | clist.add(i); |
|  |  |
|  | return clist.indexOf(t); |
|  | } |
|  |  |
|  | 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, key); |
|  | if (i != -1) { |
|  | System.out.println(i); |
|  | } else { |
|  | System.out.println("-1"); |
|  | } |
|  | sc.close(); |
|  | } |
|  | } |

**10. Find Median After Merging Two Sorted Arrays**

|  |
| --- |
| import java.util.\*; |
|  |  |
|  | public class Source { |
|  |  |
|  | public static int median(int ar1[], int ar2[], int n) |
|  | { |
|  | 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 = ar2[0]; |
|  | break; |
|  | } |
|  | else if (j == n) |
|  | { |
|  | m1 = m2; |
|  | m2 = ar1[0]; |
|  | break; |
|  | } |
|  | if (ar1[i] <= ar2[j]) |
|  | { |
|  | m1 = m2; |
|  | m2 = ar1[i]; |
|  | i++; |
|  | } |
|  | else |
|  | { |
|  | m1 = m2; |
|  | m2 = ar2[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)); |
|  | sc.close(); |
|  | } |
|  | } |