Most Frequent Element

```
import java.util.*;

public class Source {

 public static int mostFrequentElement(int[] arr, int n) {

   //write Code
   // 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) {
      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, n));
    }
}
```

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,vertexCount) || !isValidIndex(w,vertexCount)) {
         return;
      }
      adj[v].add(w);
      adj[w].add(v);
   }

   private boolean isValidIndex(int i, int vertexCount) {
      // Write code here
      if(i <= vertexCount){
```

```
        return true;
    }
    else{
        return false;
    }

}

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

}
```

```
public boolean isTree() {
   // Write Code here
    boolean visited[] = new boolean[vertexCount];
   for (int i = 0; i < vertexCount; i++)
      visited[i] = false;

      if (isCyclic(0, visited, -1))
      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");
        }
    }
}
```

Find kth Largest Element in a Stream

```
import java.util.*;


class MinHeap
  {
    final PriorityQueue<Integer> pq;
    final int k;

    public MinHeap(int k)
    {
      this.k = k;
      pq = new PriorityQueue<>(k);
    }

    public int add(int n)
    {
      // if the min-heap's size is less than `k`, push the current element
      // into the min-heap
      if (pq.size() < k) {
        pq.add(n);
      }

      // otherwise, if the current element is more than the smallest element
      // in the min-heap, remove the smallest element from the heap and
      // push the current element
```

```
            else if (pq.peek() < n)
            {
                pq.poll();
                pq.add(n);
            }

            // if the size of the min-heap reaches `k`, return the top element
            if (pq.size() == k) {
                return pq.peek();
            }
            else {
                return Integer.MIN_VALUE;
            }
        }
    }


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
    MinHeap pq = new MinHeap(k);

        //   MinHeap pq = new MinHeap(k);
        for (int i = 0; i < n; i++) {
            try {
                int x = pq.add(stream[i]);
                if (x != Integer.MIN_VALUE) {
                    System.out.println(k + " largest number is " + x);
//                      i++;
                } else
                {
                    System.out.println("None");
                }
            } catch (NoSuchElementException e) {
//                  System.out.println("None");
//                  break;
            }
        }
    }
}
```

Sort Nearly Sorted Array

```
import java.util.*;

public class Source {

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

        // Write code here

        // 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();
        }
    }
        private static void printArray(int[] arr, int n)
    {
        for (int i = 0; i < n; i++)
            System.out.print(arr[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, n);

        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + " ");
        }
    }
}
```

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
      {
       // 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));
    }
}
```

Find All Symmetric Pairs in an Array

```
import java.util.*;

public class Source {

   public static void symmetricPair(int[][] arr) {
      // Write code here

      // 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);
    }
}
```

Find All Common Element in All Rows of Matrix

```
import java.util.*;


public class Source {


   public static void printElementInAllRows(int matrix[][],int row,int col) {
  HashMap<Integer,Integer> h = new HashMap<>();
  List<Integer> list = new ArrayList<Integer>();
  for (int i = 0; i < row; i++){
    for (int j = 0; j < col; j++){
       if (i == 0){
          h.put(matrix[0][j],1);
        }
       if (i > 0 && h.containsKey(matrix[i][j]) && h.get(matrix[i][j]) == i){
          h.put(matrix[i][j],i+1);
          if (i == row -1){
             list.add(matrix[i][j]);
        }
        }
      }



  Collections.sort(list);
  for (Integer l:list){
    System.out.print(l + " ");
  }
```

```
    }
    }

    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);
    }
}
```

Find Itinerary in Order

```
import java.util.*;

 public class Source{



private static void findItinerary(Map<String, String> tickets)
        {
                // 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: tickets.entrySet())
                        reverseMap.put(entry.getValue(), entry.getKey());

                // Find the starting point of itinerary
                String start = null;
                for (Map.Entry<String,String> entry: tickets.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 = tickets.get(start);
                while (to != null)
                {
                System.out.print(start + "->" + to + "\n");
                        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);

}
```

```
}
```

Search Element in a Rotated Array

```
import java.util.*;

public class Source {

    static int search(int arr[], int l, int h, int key)
   {
      if (l > h)
         return -1;

      int mid = (l+h)/2;
      if (arr[mid] == key)
         return mid;

      /* If arr[l...mid] first subarray is sorted */
      if (arr[l] <= arr[mid])
      {
         /* As this subarray is sorted, we
            can quickly check if key lies in
            half or other half */
         if (key >= arr[l] && key <= arr[mid])
            return search(arr, l, 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, h, key);
   }

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

   return search(arr, l, 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 = search(arr, 0, n - 1, key);
   if (i != -1) {
      System.out.println(i);
   } else {
      System.out.println("-1");
   }
}
```

}

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])
        {
            /* Store the prev median */
            m1 = m2;
            m2 = arr1[i];
            i++;
        }
        else
        {
            /* Store the prev median */
            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));
    }
}
```