```
public static void main(String[] args) {
// Largest Element in an Array
class Solution {

    public int largest(int arr[], int n)
    {
        int max=arr[0];
        for(int i=0;i<n;i++){
            if(arr[i]>max){max=arr[i];}
        }
        return max;
    }
}
// Second Largest Element in an Array
class Solution {
    int print2largest(int arr[], int n) {
        // code here
        int greatest=-1;
        int second_greatest=-1;
        for(int i=0;i<n;i++){
            if(arr[i]>greatest){
                second_greatest=greatest;
                greatest=arr[i];
            }
            else if(arr[i]>second_greatest&&arr[i]!=greatest){
                second_greatest=arr[i];
            }
        }
        return second_greatest;
    }
}
// Check if the array is sorted
class Solution {
    public boolean check(int[] arr) {
        int n=arr.length;
        for(int i=0;i<n;i++){if(arr[i]>arr[(i+1)%n]){return false}}
        return true;
    }
}
// Remove duplicates from Sorted array
class Solution {
    public int removeDuplicates(int[] nums) {
        int n=nums.length;
        int copy[]=new int[n];
        for(int i=0;i<n;i++){copy[i]=nums[i];}
        Map<Integer,Integer> map=new HashMap<>();
        int ind=0;
        for(int i=0;i<n;i++){
            if(!map.containsKey(copy[i])){
```

```
                map.put(copy[i],1);
                copy[ind++]=copy[i];
            }
        }
        for(int i=0;i<n;i++){nums[i]=copy[i];}
        return ind;
    }
}
// Left Rotate an array by one place
// Left rotate an array by D places
// Move Zeros to end
// Linear Search
class Solution{
    static int searchInSorted(int arr[], int N, int K)
    {

        // Your code here
        for(int i=0;i<N;i++){
            if(arr[i]==K){return 1;}
        }
        return -1;

    }
}
// Find the Union
class Solution
{
    //Function to return a list containing the union of the two arrays.
    public static ArrayList<Integer> findUnion(int arr1[], int arr2[], int n, int m)
    {
        // add your code here
        int i = 0, j = 0;
        ArrayList<Integer> ans = new ArrayList<Integer>();
        while (i < n && j < m)
        {
            while(  i+1<n  &&  arr1[i]==arr1[i+1]  ){ i++;}
            while(  j+1<m  &&  arr2[j]==arr2[j+1]  ){j++;}
            if (arr1[i] < arr2[j]){ans.add(arr1[i++]);}
            else if (arr2[j] < arr1[i]) {ans.add(arr2[j++]);}
            else
            {
                ans.add(arr2[j++]);
                i++;
            }
        }
        while(i < n){
            while(  i+1<n  &&  arr1[i]==arr1[i+1]  ){i++;}
            ans.add(arr1[i++]);
        }
```

```
        //Storing the remaining elements of second array (if there are any).
        while(j < m){
            while(  j+1<m  &&  arr2[j]==arr2[j+1]  ){j++;}
            ans.add(arr2[j++]);
        }
        return ans;
    }
}

// Find missing number in an array
// Maximum Consecutive Ones
class Solution {
    public int findMaxConsecutiveOnes(int[] nums) {
        int start=0,end=0,n=nums.length,ans=0;
        while(end<n){
            if(nums[end]==1){end++;}
            else{start=end;start++;end++;}
            ans=Math.max(end-start,ans);
        }
        return ans;
    }
}
// Subarray with given sum equals K
class Solution {
    public int subarraySum(int[] nums, int k) {
        int n=nums.length,sum=0,cnt=0;
        Map<Integer,Integer> hm=new HashMap<>();
        hm.put(0, 1);
        for(int i=0;i<n;i++){
            sum+=nums[i];
            if(hm.containsKey(sum-k)){
                cnt+= hm.get(sum-k);
            }
            hm.put(sum,hm.getOrDefault(sum,0)+1);
        }
        return cnt;
    }
}
// Find the Missing Number
// Find the number that appears once
class Solution {
    public int singleNumber(int[] nums) {
       int ans=0;
       int n=nums.length;
       for(int i=0;i<n;i++){
           ans=ans^nums[i];
       }
       return ans;
    }
```

```
}
// Search an element in a 2D matrix
class Solution {
    public int floor(int [][]matrix,int target){
        int ans=-1;
        int start=0;
        int end=matrix.length-1;

        while(start<=end){
            int mid=(start+end)/2;
            if(matrix[mid][0]==target){return mid;}
            else if(matrix[mid][0]<target){
                ans=mid;
                start=mid+1;
                }
            else{end=mid-1;}
        }
        return ans;
    }
    public boolean binarysearch(int [][]matrix,int target,int row){
        int start=0;
        int end=matrix[row].length-1;

        while(start<=end){
            int mid=(start+end)/2;
            if(matrix[row][mid]==target){return true;}
            else if(matrix[row][mid]>target){end=mid-1;}
            else{start=mid+1;}
        }
        return false;
    }
    public boolean searchMatrix(int[][] matrix, int target) {
        int rows=matrix.length;
        int cols=matrix[0].length;
        int R=floor(matrix,target);
        if(R==-1){
            return false;
        }
        return binarysearch(matrix,target,R);
    }
    public boolean searchMatrix(int[][] matrix, int target) {
        int rows=matrix.length;
        int cols=matrix[0].length;
        int start=0;
        int end=(rows*cols)-1;
        while(start<=end){
            int mid=(start+end)/2;
            int r=mid/cols;
            int c=mid%cols;
```

```
            if(matrix[r][c]==target){return true;}
            else if(matrix[r][c]>target){end=mid-1;}
            else{start=mid+1;}
        }
        return false;
    }
}
// Find the row with maximum number of 1s
class Solution {
    int rowWithMax1s(int arr[][], int n, int m) {
        // code here
        int max_row_index=-1,max=-1;
        int index;
        for (int i=0;i<n;i++) {
            index=first(arr[i],0,m-1);
            if(index!=-1&&m-index>max){
                max=m-index;
                max_row_index=i;
            }
        }
        return max_row_index;
    }
     int first(int arr[],int low,int high){
        if(high>=low){
            int mid=low+(high-low)/2;
            if ((mid==0||(arr[mid-1]==0))&&arr[mid]==1){return mid;}
            else if(arr[mid] == 0){return first(arr,(mid+1),high);}
            else {return first(arr,low,(mid-1));}
        }
        return -1;
    }
}
// Majority Element (>n/2 times)

// Kadane's Algorithm, maximum subarray sum
class Solution {
    public int maxSubArray(int[] nums) {
        int sum=0;
        int maxSum=nums[0];
        for(int i=0;i<nums.length;i++){
            sum+=nums[i];
            if(sum>maxSum){
                maxSum=sum;
            }
            if(sum<0){
                sum=0;
            }
        }
        return maxSum;
```

```
    }
}
// Print subarray with maximum subarray
// Stock Buy and Sell
class Solution {
    public int maxProfit(int[] prices) {
        int minPrice=Integer.MAX_VALUE;
        int maxProfit=0;
        for(int i=0;i<prices.length;i++){
            if(prices[i]<minPrice){minPrice=prices[i];}
            if(prices[i]-minPrice>maxProfit){maxProfit=prices[i]-minPrice;}
        }
        return maxProfit;
    }
}
// Rearrange the array in alternating …
// Next Permutation
class Solution {
    public void nextPermutation(int[] A) {
        if(A==null||A.length<=1){return;}
        int i=A.length-2;
        while(i>=0&&A[i]>=A[i+1]){i--;}
        if(i>=0){
            int j=A.length-1;
            while(A[j]<=A[i]){j--;}
            swap(A,i,j);
        }
        reverse(A,i+1,A.length-1);
}

public void swap(int[] A, int i, int j) {
    int tmp = A[i];
    A[i] = A[j];
    A[j] = tmp;
}

public void reverse(int[] A, int i, int j) {
    while(i < j) {swap(A, i++, j--);}
}
}
// Leaders in an Array problem
class Solution{
    //Function to find the leaders in the array.
    static ArrayList<Integer> leaders(int arr[], int n){
        // Your code here
        int greatest=Integer.MIN_VALUE;
        ArrayList<Integer> leaders=new ArrayList<>();
        for(int i=n-1;i>=0;i--){
            if(arr[i]>=greatest){
```

```
                leaders.add(0,arr[i]);
                greatest=arr[i];
            }
        }
        return leaders;
    }
}

// Longest Consecutive Sequence in an array
class Solution {
    public int longestConsecutive(int[] nums) {
        Set<Integer> num_set=new HashSet<Integer>();
        for(int num : nums){num_set.add(num);}
        int longestStreak = 0;
        for(int num : num_set){
            if(!num_set.contains(num-1)) {
                int currentNum = num;
                int currentStreak = 1;
                while(num_set.contains(currentNum+1)) {
                    currentNum+=1;
                    currentStreak+=1;
                }
                longestStreak=Math.max(longestStreak,currentStreak);
            }
        }
        return longestStreak;
    }
}
// Set Matrix Zeros
// Rotate Matrix by 90 degrees
class Solution {
    public void rotate(int[][] matrix) {
        transpose(matrix);
        print(matrix);
        reverse(matrix);
    }
    public void print(int a[][]){
        for(int i=0;i<a.length;i++){
            for(int j=0;j<a.length;j++){
                System.out.print(a[i][j]+" ");
            }
        System.out.println();
        }
    }
    public void transpose(int mat[][]){
        int n=mat.length;
        int m=mat[0].length;
        for(int i=0;i<n;i++){
            for(int j=i;j<n;j++){
```

```
                int temp=mat[i][j];
                mat[i][j]=mat[j][i];
                mat[j][i]=temp;
            }
        }
    }
    public void reverse(int mat[][]){
        int n=mat.length;
        for(int i=0;i<n;i++){reverserow(mat[i]);}
    }
    public void reverserow(int mat[]){
        int n=mat.length;
        for(int i=0;i<n/2;i++){
            int temp=mat[i];
            mat[i]=mat[n-1-i];
            mat[n-1-i]=temp;
        }
    }
}
// Print the matrix in spiral manner
class Solution {
    public List<Integer> spiralOrder(int[][] matrix) {
        List<Integer> result = new ArrayList<>();
        if (matrix==null||matrix.length==0){return result;}
        int rows=matrix.length,cols=matrix[0].length;
        int left=0,right=cols-1,top=0,bottom=rows-1;
        while(left<=right&&top<=bottom){
            for (int i=left;i<=right;i++){result.add(matrix[top][i]);}
            top++;
            for (int i=top;i<=bottom;i++){result.add(matrix[i][right]);}
            right--;
            if(top<=bottom) {
                for(int i=right;i>=left;i--){result.add(matrix[bottom][i]);}
                bottom--;
            }
            if(left<=right){
                for (int i=bottom;i>=top;i--){result.add(matrix[i][left]);}
                left++;
            }
        }

        return result;
    }
}
// Pascal's Triangle
class Solution {
    public List<List<Integer>> generate(int numRows) {
      List<List<Integer>> res = new ArrayList<List<Integer>>();
                List<Integer> row, pre = null;
```

```
            for(int i=0;i<numRows;++i){
                row=new ArrayList<Integer>();
                for(int j=0;j<=i;++j){
                                        if(j==0||j==i){row.add(1);}
                    else{row.add(pre.get(j-1)+pre.get(j));}
                                    }
                pre=row;
                res.add(row);
            }
            return res ;
    }
}

// Majority Element (n/3 times)
class Solution {
    public List<Integer> majorityElement(int[] nums) {
        int num1=Integer.MAX_VALUE,num2=Integer.MAX_VALUE,count1=0,count2=0,
len=nums.length;
        for(int i=0;i<len;i++){
            if(nums[i]==num1){
                count1++;
            }
            else if(nums[i]==num2){
                count2++;
            }
            else if(count1==0){
                num1=nums[i];
                count1=1;
            }
            else if(count2==0){
                num2=nums[i];
                count2=1;
            }
            else{
                count1--;
                count2--;
            }
        }
        List<Integer> mon=new ArrayList<>();
        int c1=0;
        int c2=0;
        for(int i=0;i<len;i++){
            if(nums[i]==num1){
                c1++;
            }
            if(nums[i]==num2){
                c2++;
            }
        }
```

```
        if(c1>len/3){
            mon.add(num1);
        }
        if(c2>len/3){
            mon.add(num2);
        }
        return mon;
    }
}
// 3-Sum Problem
class Solution {
    public List<List<Integer>> threeSum(int[] nums) {
    List<List<Integer>> result=new ArrayList<>();
    Arrays.sort(nums);
    for(int i=0;i<nums.length-2;i++){
        if(i==0||(i>0 && nums[i]!=nums[i-1])) {
        int low=i+1,high=nums.length-1;
        int sum=0-nums[i];
        while(low<high){
            if(nums[low]+nums[high]==sum){
                result.add(Arrays.asList(nums[i],nums[low],nums[high]));
                while(low<high && nums[low]==nums[low+1]){low++;}
                while(low<high && nums[high]==nums[high-1]){high--;}
                low++;
                high--;
            }
            else if(nums[low]+nums[high]<sum){low++;}
            else{high--;}
        }
        }
    }
        return result;
    }
}
// 4-Sum Problem
class Solution {
    public List<List<Integer>> fourSum(int[] nums, int target) {
        List<List<Integer>> ans=new ArrayList<>();
        int n=nums.length;
        Arrays.sort(nums);
        for(int i=0;i<n;i++){
            for(int j=i+1;j<n;j++){
                long target2=(long)target-(long)nums[i]-(long)nums[j];
                int lo=j+1,hi=n-1;
                while(lo<hi){
                    int twoSum=nums[lo]+nums[hi];
                    if(twoSum<target2){lo++;}
                    else if(twoSum>target2){hi--;}
                    else{
```

```
                    List<Integer> quad=Arrays.asList(nums[i],nums[j],
                     nums[lo],nums[hi]);
                    ans.add(quad);

                    while(lo<hi&&nums[lo]==quad.get(2)){lo++;}
                    while(lo<hi&&nums[hi]==quad.get(3)){hi--;}
                }
            }
            while (j+1<n&&nums[j]==nums[j + 1]){j++;}
        }
        while(i+1<n&&nums[i]==nums[i+1]){i++;}
    }
    return ans;
  }
}
// Largest Subarray with 0 Sum
class GfG
{
  int maxLen(int nums[], int n)
  {
    HashMap<Integer,Integer> hm=new HashMap<>();
    int max=0,sum=0;
    hm.put(0,-1);
    for(int i=0;i<n;i++){
      sum+=nums[i];
      if(hm.containsKey(sum)==true){
        max=Math.max(i-hm.get(sum),max);
      }
      if(hm.get(sum)==null){hm.put(sum,i);}
    }
    return max;
  }
}
// Count number of subarrays with give…
// Merge Overlapping Subintervals
class Solution {
  public int[][] merge(int[][] intervals) {
    List<int[]> res=new ArrayList<>();
    if(intervals.length==0||intervals==null){
      return res.toArray(new int[0][]);
    }
    Arrays.sort(intervals,(a,b) -> a[0]-b[0]);
    int start=intervals[0][0];
    int end=intervals[0][1];

    for(int[] i: intervals){
      if(i[0]<=end){
        end=Math.max(end,i[1]);
      }
```

```
            else{
                res.add(new int[]{start,end});
                start=i[0];
                end=i[1];
            }

        }
         res.add(new int[]{start,end});
         return res.toArray(new int[0][]);
    }
}
// Merge two sorted arrays without ext…
// Find the repeating and missing numb…
// Count Inversions
// Reverse Pairs
// Maximum Product Subarray
class Solution {
    public int maxProduct(int[] nums) {
        int ans = nums[0];
        int dpMin = nums[0];
        int dpMax = nums[0];

        for(int i=1;i<nums.length;++i){
            final int num = nums[i];
            final int prevMin = dpMin;
            final int prevMax = dpMax;
            if(num<0){
                dpMin=Math.min(prevMax*num,num);
                dpMax=Math.max(prevMin*num,num);
            }
            else{
                dpMin=Math.min(prevMin*num,num);
                dpMax=Math.max(prevMax*num,num);
            }
        ans = Math.max(ans, dpMax);
        }
        return ans;
    }
}
}
```