```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Set;

public class Arrays {
    // 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;
        }
        // Sort the array - and - get the last element - O(NlogN);
    }

    // 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;
            // Sort the array and get the last second element - if lse != last ele(NLogN+N)
            // Two passes - first pass get largest and then again to get second

        }
    }

    // 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;
    }

    // M2 --Put in a Set-HashSet
    class Solution {
        public int removeDuplicates(int[] nums) {
            // Set<Integer> st=new HashSet<>();
            int i = 0, n = nums.length;
            for (int j = 1; j < n; j++) {
                if (nums[i] != nums[j]) {
                    nums[i + 1] = nums[j];
                    i++;
                }
            }
            return i + 1;

        }
    }

}

// Left Rotate an array by one place
class Solution {
    class Solution {

        public void LeftRotate(int nums[]) {
            int n = nums.length;
            int temp = nums[0];
            for (int j = 1; j < n - 1; j++) {
                nums[j] = nums[j + 1];
            }
            nums[n - 1] = temp;
        }
    }
}

// Left rotate an array by D places
class Solution {
    class Solution {
        public void rotate(int[] nums, int k) {
            for (int i = 0; i < k; i++) {
                RightRotate(nums);
            }
        }

        public void RightRotate(int nums[]) {
            int n = nums.length;
            int temp = nums[n - 1];
            int curr = nums[0];
            for (int j = n - 2; j >= 0; j--) {
                nums[j + 1] = nums[j];
            }
            nums[0] = temp;
        }
    }

    class Solution {
        public void rotate(int[] nums, int k) {
            int t[] = new int[k];
            int n = nums.length;
            if (n == 1) {
                return;
            }
            k = k % n;
            for (int i = 0; i < k; i++) {
                t[i] = nums[n - k + i];
            }
            for (int i = n - k - 1; i >= 0; i--) {
                nums[i + k] = nums[i];
            }
            for (int i = 0; i < k; i++) {
                nums[i] = t[i];
            }
            return;
        }
    }

    class Solution {
        public void rotate(int[] nums, int k) {
            int n = nums.length;
            k = k % n;
            f(nums, 0, n - k - 1);
            f(nums, n - k, n - 1);
            f(nums, 0, n - 1);
        }

        public void f(int a[], int s, int e) {
            int low = s, high = e;
            while (low <= high) {
                int g = a[low];
                a[low] = a[high];
                a[high] = g;
                low++;
                high--;
            }
        }

        public void print(int a[]) {
            for (int i = 0; i < a.length; i++) {
                System.out.print(a[i] + " ");
            }
            System.out.println();
        }
    }
}

// Move Zeros to end
class Solution {
    public void moveZeroes(int[] nums) {
        int n = nums.length;
        if (n == 0 || n == 1) {
            return;
        }
        int j = -1;
        for (int i = 0; i < n; i++) {
            if (nums[i] == 0) {
                j = i;
                break;
            }
        }
        if (j == -1) {
            return;
        }
        for (int i = j + 1; i < n; i++) {
            if (nums[i] != 0) {

                swap(nums, i, j);
                j++;
            }
        }
    }

    public void swap(int a[], int p, int n) {
        int temp = a[p];
        a[p] = a[n];
        a[n] = temp;
    }
}

// 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;
        }
        // M2--Set

        // Intersection- visited array;
    }

    // Find the Intersection
    class Solution {
        public class Solution {
            public static ArrayList<Integer> findArrayIntersection(ArrayList<Integer> arr1,
int n,
                    ArrayList<Integer> arr2, int m) {
                // Write Your Code Here.
                ArrayList<Integer> ls = new ArrayList<>();
                int i = 0, j = 0;
                while (i < n && j < m) {
                    if (arr1.get(i) < arr2.get(j)) {
                        i++;
                    } else if (arr1.get(i) > arr2.get(j)) {
                        j++;
                    } else {
                        ls.add(arr1.get(i));
                        i++;
                        j++;
                    }
                }
                return ls;
            }
        }
    }
```

```
// Find missing number in an array
class Compute {

    public static int missingNumber(int A[], int N) {
        // Your code goes here
        int reqd = N * (N + 1) / 2;
        int sum = 0;
        for (int i = 0; i < N; i++) {
            sum += A[i];
        }
        return reqd - sum;
    }
}

// Maximum Consecutive Ones
class Solution {
    public int findMaxConsecutiveOnes(int[] nums) {
        int cnt = 0;
        int maxcnt = 0;
        for (int i = 0; i < nums.length; i++) {
            if (nums[i] == 1) {
                cnt++;
            } else {
                cnt = 0;
            }
            maxcnt = Math.max(cnt, maxcnt);
        }
        return maxcnt;
    }
}

// Subarray with given sum
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;
    }

    // M2 generate all subarrays with two for loops
    class Solution {

        // Function for finding maximum and value pair
        public static int lenOfLongSubarr(int nums[], int N, int K) {
            // Complete the function
            int n = nums.length, sum = 0, maxlen = 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)) {
                    maxlen = Math.max(i - hm.get(sum - K), maxlen);
                }
                if (hm.get(sum) == null) {
                    hm.put(sum, i);
                }
            }
            return maxlen;
        }

    }

}

// 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
// Find the row with maximum number of…
// 2 Sum Problem
class Solution {
    public static String twoSum(int n, int[] arr, int target) {
        for (int i = 0; i < n; i++) {
            for (int j = i + 1; j < n; j++) {
                if (arr[i] + arr[j] == target)
                    return "YES";
            }
        }
        return "NO";
    }
```

```
public static int[] twoSum(int n, int[] arr, int target) {
    int[] ans = new int[2];
    ans[0] = ans[1] = -1;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (arr[i] + arr[j] == target) {
                ans[0] = i;
                ans[1] = j;
                return ans;
            }
        }
    }
    return ans;
}

public static String twoSum(int n, int[] arr, int target) {
    HashMap<Integer, Integer> mpp = new HashMap<>();
    for (int i = 0; i < n; i++) {
        int num = arr[i];
        int moreNeeded = target - num;
        if (mpp.containsKey(moreNeeded)) {
            return "YES";
        }

        mpp.put(arr[i], i);
    }
    return "NO";
}

public static int[] twoSum(int n, int[] arr, int target) {
    int[] ans = new int[2];
    ans[0] = ans[1] = -1;
    HashMap<Integer, Integer> mpp = new HashMap<>();
    for (int i = 0; i < n; i++) {
        int num = arr[i];
        int moreNeeded = target - num;
        if (mpp.containsKey(moreNeeded)) {
            ans[0] = mpp.get(moreNeeded);
            ans[1] = i;
            return ans;
        }

        mpp.put(arr[i], i);
    }
    return ans;
}

public static String twoSum(int n, int[] arr, int target) {
        Arrays.sort(arr);
        int left = 0, right = n - 1;
        while (left < right) {
            int sum = arr[left] + arr[right];
            if (sum == target) {
                return "YES";
            } else if (sum < target)
                left++;
            else
                right--;
        }
        return "NO";
    }
}

// Sortan array of 0's 1's and 2
class Solution {
    public void sortColors(int[] nums) {
        int mid = 0, low = 0, high = nums.length - 1;
        while (mid <= high) {
            if (nums[mid] == 0) {
                int temp = nums[low];
                nums[low] = nums[mid];
                nums[mid] = temp;
                mid++;
                low++;
            } else if (nums[mid] == 1) {
                mid++;
            } else {
                int temp = nums[high];
                nums[high] = nums[mid];
                nums[mid] = temp;
                high--;

            }
        }
    }
}

// Majority Element (>n/2 times)
class Solution {
    public int majorityElement(int[] nums) {
        int count = 0;
        int ele = 0;
        for (int n : nums) {
            if (count == 0) {
                ele = n;
            }
            if (n == ele) {
                count++;
            } else {
                count--;
            }
        }
        return ele;
    }
    // Two Loops
    // Hash Map
}

// 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 sum
class Solution {
    public int maxSubArray(int[] nums) {
        int sum = 0;
        int maxSum = nums[0];
        int ansStart = -1, ansEnd = -1, start = -1;
        for (int i = 0; i < nums.length; i++) {
            if (sum == 0) {
                start = i;
            }
            sum += nums[i];
            if (sum > maxSum) {
                maxSum = sum;
                ansStart = start;
                ansEnd = i;
            }
            if (sum < 0) {
                sum = 0;
            }
        }
        return maxSum;
    }
}

// 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++) {
            minPrice = Math.min(minPrice, prices[i]);
            int currentprofit = prices[i] - minPrice;
            if (currentprofit > maxProfit) {
                maxProfit = prices[i] - minPrice;
            }
        }
        return maxProfit;
    }
}

// Rearrange the array in alternating manner
class Solution {
    public int[] rearrangeArray(int[] nums) {
        int n = nums.length;
        int ans[] = new int[n];
        int posi = 0, negi = 1;
        for (int i = 0; i < n; i++) {
            if (nums[i] < 0) {
                ans[negi] = nums[i];
                negi += 2;
            } else {
                ans[posi] = nums[i];
                posi += 2;
            }
        }
        return ans;
    }
}

// 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
class Solution {
    public void setZeroes(int[][] matrix) {
        int val = -1;
        int n = matrix.length;
        int m = matrix[0].length;
        int rowSpace[] = new int[n];
        int colSpace[] = new int[m];
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (matrix[i][j] == 0) {
                    rowSpace[i] = 1;
                    colSpace[j] = 1;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            if (rowSpace[i] == 1) {
                FillRow(i, matrix, m);
            }
        }
        for (int i = 0; i < m; i++) {
            if (colSpace[i] == 1) {
                FillCol(i, matrix, n);
            }
        }
    }

    public void FillRow(int row, int matrix[][], int m) {
        for (int i = 0; i < m; i++) {
            matrix[row][i] = 0;
        }
    }
    public void FillCol(int col, int matrix[][], int n) {
        for (int i = 0; i < n; i++) {
            matrix[i][col] = 0;
        }
    }
}

// Rotate Matrix by 90 degrees
class Solution {
    // a[i][j]=image[j][n-i-1 ]
    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;
            }
        }
    }
}

// Count subarrays with given sum
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;
    }
}

// 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;
    }
}

// Hard
// Pascal'sTriangle
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 {
    class TUF {
        static ArrayList<ArrayList<Integer>> threeSum(int[] num) {
            Arrays.sort(num);
            ArrayList<ArrayList<Integer>> res = new ArrayList<>();
            for (int i = 0; i < num.length - 2; i++) {
                if (i == 0 || (i > 0 && num[i] != num[i - 1])) {
                    int lo = i + 1, hi = num.length - 1, sum = 0 - num[i];
                    while (lo < hi) {
                        if (num[lo] + num[hi] == sum) {
                            ArrayList<Integer> temp = new ArrayList<>();
                            temp.add(num[i]);
                            temp.add(num[lo]);
                            temp.add(num[hi]);
                            res.add(temp);
                            while (lo < hi && num[lo] == num[lo + 1]){lo++;}
                            while (lo < hi && num[hi] == num[hi - 1]){hi--;}
                            lo++;
                            hi--;
                        } else if (num[lo] + num[hi] < sum)
                            lo++;
                        else
                            hi--;
                    }
                }
            }
            return res;
        }
    }

    // 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 Solution {
    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 given XOR of k
class Solution {
    public class tUf {
        public static int subarraysWithXorK(int[] a, int k) {
            int n = a.length; // size of the given array.
            int xr = 0;
            Map<Integer, Integer> mpp = new HashMap<>(); // declaring the map.
            mpp.put(xr, 1); // setting the value of 0.
            int cnt = 0;
            for (int i = 0; i < n; i++) {
                xr = xr ^ a[i];
                int x = xr ^ k;
                if (mpp.containsKey(x)) {
                    cnt += mpp.get(x);
                }
                if (mpp.containsKey(xr)) {
                    mpp.put(xr, mpp.get(xr) + 1);
                } else {
                    mpp.put(xr, 1);
                }
            }
            return cnt;
        }
    }
}

// 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 extra space
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        merge(nums1, nums2, m, n);
        int k = 0;
        for (int i = m; i < n + m; i++) {
            nums1[i] = nums2[k++];
        }
    }

    public void merge(int[] arr1, int[] arr2, int n, int m) {
        int len = n + m;
        int gap = (len / 2) + (len % 2);
        while (gap > 0) {
            int left = 0;
            int right = left + gap;
            while (right < len) {
                if (left < n && right >= n) {
                    swapIfGreater(arr1, arr2, left, right - n);
                } else if (left >= n) {
                    swapIfGreater(arr2, arr2, left - n, right - n);
                } else {
                    swapIfGreater(arr1, arr1, left, right);
                }
                left++;
                right++;
            }
            if (gap == 1)
                break;
            gap = (gap / 2) + (gap % 2);
        }
    }

    public void swapIfGreater(int[] arr1, int[] arr2, int ind1, int ind2) {
        if (arr1[ind1] > arr2[ind2]) {
            int temp = arr1[ind1];
            arr1[ind1] = arr2[ind2];
            arr2[ind2] = temp;
        }
    }
}

// Find the repeating and missing number
class Solution {
    class Solve {
        int[] findTwoElement(int arr[], int n) {
            // code here
            long sum = 0, sum_sq = 0;
            for (int i = 0; i < n; i++) {
                sum += arr[i];
                sum_sq += arr[i] * arr[i];

            }
            long req_sum = n * (n + 1) / 2;
            long req_sum_sq = (n * (n + 1) * ((2 * n) + 1)) / 6;
            long eqn1 = sum - req_sum;
            long eqn2 = sum_sq - req_sum_sq;
            eqn2 = eqn2 / eqn1;
            long val1 = (eqn1 + eqn2) / 2;
            long val2 = val1 - eqn1;
            return new int[] { (int) val1, (int) val2 };
        }
    }

    class Solve {
        int[] findTwoElement(int arr[], int n) {
            // code here
            int xorele = 0;
            int xorini = 0;
            for (int i = 0; i < n; i++) {
                xorele = xorele ^ arr[i];
                xorini = xorini ^ (i + 1);
            }
            int res = xorele ^ xorini;
            int getbit = f(res);
            int zeroclub = 0, oneclub = 0;
            for (int i = 0; i < n; i++) {
                if ((arr[i] & (1 << getbit)) != 0) {
                    oneclub = oneclub ^ arr[i];
                } else {
                    zeroclub = zeroclub ^ arr[i];

                }
            }
            for (int i = 1; i <= n; i++) {
                if (((1 << getbit) & (i)) != 0) {
                    oneclub = oneclub ^ i;
                } else {
                    zeroclub = zeroclub ^ i;
                }
            }
            int cnt = 0;
            for (int i = 0; i < n; i++) {
                if (oneclub == arr[i]) {
                    cnt++;
                }
            }
            if (cnt == 2) {
                return new int[] { oneclub, zeroclub };

            } else
                return new int[] { zeroclub, oneclub };
        }

        public int f(int a) {
            for (int i = 0; i < 32; i++) {
                if ((a & (1 << i)) != 0) {
                    return i;
                }
            }
            return -1;
        }
    }
}

// Count Inversions
class Solution {
    // Function to count inversions in the array.
    static long inversionCount(long arr[], long n) {
        long temp[] = new long[(int) n];
        return _mergeSort(arr, temp, 0, n - 1);
    }

    static long _mergeSort(long arr[], long temp[], long left, long right) {
        long mid, inv_count = 0;
        if (right > left) {
            mid = (right + left) / 2;
            inv_count = _mergeSort(arr, temp, left, mid);
            inv_count += _mergeSort(arr, temp, mid + 1, right);
            inv_count += merge(arr, temp, left, mid + 1, right);
        }
        return inv_count;
    }

    static long merge(long arr[], long temp[], long left, long mid, long right) {
        long i, j, k;
        long inv_count = 0;
        i = left;
        j = mid;
        k = left;
        while ((i <= mid - 1) && (j <= right)) {
            if (arr[(int) i] <= arr[(int) j]) {
                temp[(int) k++] = arr[(int) i++];
            } else {
                temp[(int) k++] = arr[(int) j++];
                inv_count = inv_count + (mid - i);
            }
        }
        while (i <= mid - 1) {
            temp[(int) k++] = arr[(int) i++];
        }
        while (j <= right) {
            temp[(int) k++] = arr[(int) j++];
        }
        for (i = left; i <= right; i++) {
            arr[(int) i] = temp[(int) i];
        }
        return inv_count;
    }
}

// Reverse Pairs
class Solution {
    public int reversePairs(int[] nums) {
        return f(nums);
    }

    int merge(int[] nums, int low, int mid, int high) {
        int cnt = 0;
        int j = mid + 1;
        for (int i = low; i <= mid; i++) {
            while (j <= high && nums[i] > (2 * (long) nums[j])) {
                j++;
            }
            cnt += (j - (mid + 1));
        }

        ArrayList<Integer> temp = new ArrayList<>();
        int left = low, right = mid + 1;
        while (left <= mid && right <= high) {
            if (nums[left] <= nums[right]) {
                temp.add(nums[left++]);
            } else {
                temp.add(nums[right++]);
            }
        }

        while (left <= mid) {
            temp.add(nums[left++]);
        }
        while (right <= high) {
            temp.add(nums[right++]);
        }

        for (int i = low; i <= high; i++) {
            nums[i] = temp.get(i - low);
        }
        return cnt;
    }

    int mergeSort(int[] nums, int low, int high) {
        if (low >= high)
            return 0;
        int mid = (low + high) / 2;
        int inv = mergeSort(nums, low, mid);
        inv += mergeSort(nums, mid + 1, high);
        inv += merge(nums, low, mid, high);
        return inv;
    }

    int f(int[] nums) {
        return mergeSort(nums, 0, nums.length - 1);
    }

}

// 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;
        }

        class Solution {
            public int maxProduct(int[] nums) {
                int prefix = 1;
                int suffix = 1;
                int n = nums.length;
                int res = Integer.MIN_VALUE;
                for (int i = 0; i < n; i++) {
                    if (prefix == 0) {
                        prefix = 1;
                    }
                    if (suffix == 0) {
                        suffix = 1;
                    }
                    prefix = prefix * nums[i];
                    suffix = suffix * nums[n - i - 1];
                    res = Math.max(res, Math.max(prefix, suffix));
                }
                return res;
            }
        }
    }
}
```