```
public class Striver_Array_Series {
    // BASIC

    // 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);
    }

    // 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;
    }
    }

    // Second Largest Element in an Array without sorting
    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

        }
    }

    // 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;

    }
}
// Left Rotate an array by one place

// EASY

// Maximum Consecutive Ones
// 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;
    }
}
// Left rotate an array by D places

// 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;

        }
    }

}

// Find missing number in an array
// Find the number that appears once, and other numbers twice.
// Find the Union and intersection of two sorted arrays
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;
}

// 2Sum 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";
    }
}
// MEDIUM
// Search in a 2d Matrix
// Stock Buy and Sell
// Rearrange the array in alternating positive and negative items
// Find the duplicate in an array of N+1 integers.
// Kadane's Algorithm, maximum subarray sum
// Print the subarray with maximum sum
// Grid Unique Paths
// Sort an array of 0's 1's and 2's
// Pascal's Triangle
// Leaders in an Array problem
// Print the matrix in spiral manner
// Rotate Matrix by 90 degrees

// HARD
// Majority Element (>n/2 times)
// Majority Element (n/3 times)
// Merge Overlapping Subintervals
// Merge two sorted arrays without extra space
// Longest Consecutive Sequence in an Array
// Longest subarray with given sum(Positives)
// Longest subarray with given sum(Positives + Negatives)
// Find number of subarrays with sum K
// Count number of subarrays with given xor K
// Next Permutation
// Set Matrix Zeros

// EXPERT
// 3-Sum Problem
// 4-Sum Problem
// Find the repeating and missing number
// Maximum Product Subarray
// Merge Sort
```

```
    // Count Inversions
    // Reverse Pairs
}
```