Day-1

1. Move All Zeros to the End

Problem: Move all 0s to the end of the array while maintaining the order of non-zero elements.

Examples:

- Input: [0,1,0,3,12] → Output: [1,3,12,0,0]
- Input: [1,0,2,0,0,3] → Output: [1,2,3,0,0,0]

Java Code:

```
public class MoveZeros {
    public static void moveZeros(int[] nums) {
        int index = 0; // Pointer for non-zero elements
        for (int num : nums) {
            if (num != 0) {
                nums[index++] = num;
            }
        }
        while (index < nums.length) {
            nums[index++] = 0;
        }
    }

    public static void main(String[] args) {
        int[] arr1 = {0, 1, 0, 3, 12};
        moveZeros(arr1);
        for (int num : arr1) System.out.print(num + " "); // 1 3 12 0 0
        System.out.println();

        int[] arr2 = {1, 0, 2, 0, 0, 3};
        moveZeros(arr2);
        for (int num : arr2) System.out.print(num + " "); // 1 2 3 0 0 0
    }
}
```

2. Left Rotate an Array by D Places

Problem: Rotate the array to the left by D elements.

Examples:

- Input: [1,2,3,4,5,6,7], D=2 → Output: [3,4,5,6,7,1,2]
- Input: [10,20,30,40,50], D=3 → Output: [40,50,10,20,30]

Java Code:

```
public class LeftRotateByD {

    public static void rotateLeft(int[] arr, int d) {
        d = d % arr.length;  // In case d > n
        reverse(arr, 0, d - 1);
        reverse(arr, d, arr.length - 1);
        reverse(arr, 0, arr.length - 1);
    }

    private static void reverse(int[] arr, int start, int end) {
        while (start < end) {
            int temp = arr[start];
            arr[start++] = arr[end];
            arr[end--] = temp;
        }
    }

    public static void main(String[] args) {
        int[] arr1 = {1, 2, 3, 4, 5, 6, 7};
        rotateLeft(arr1, 2);
        for (int num : arr1) System.out.print(num + " "); // 3 4 5 6 7 1 2
        System.out.println();

        int[] arr2 = {10, 20, 30, 40, 50};
        rotateLeft(arr2, 3);
        for (int num : arr2) System.out.print(num + " "); // 40 50 10 20 30
    }
}
```

Day-2

1. Remove Duplicates from a Sorted Array

Problem: Remove duplicates from a sorted array and return the new length (in-place modification).

Examples:

- Input: [1,1,2,2,3,3,4] → Output: [1,2,3,4], Length = 4
- Input: [0,0,1,1,1,2,2,3,3,4] → Output: [0,1,2,3,4], Length = 5

Java Code:

```
public class RemoveDuplicates {
    public static int removeDuplicates(int[] nums) {
        if (nums.length == 0) return 0;
        int index = 1;
        for (int i = 1; i < nums.length; i++) {
            if (nums[i] != nums[i - 1]) {
                nums[index++] = nums[i];
            }
        }
        return index;  // New length
    }

    public static void main(String[] args) {
        int[] arr1 = {1, 1, 2, 2, 3, 3, 4};
        int len1 = removeDuplicates(arr1);
        for (int i = 0; i < len1; i++) System.out.print(arr1[i] + " "); // 1 2 3 4
        System.out.println();

        int[] arr2 = {0, 0, 1, 1, 1, 2, 2, 3, 3, 4};
        int len2 = removeDuplicates(arr2);
        for (int i = 0; i < len2; i++) System.out.print(arr2[i] + " "); // 0 1 2 3 4
    }
}
```

2. Find the Missing Number in an Array

Problem: An array contains n-1 numbers from 1 to n with one number missing. Find the missing number.

Examples:

- Input: [1, 2, 4, 5] → Output: 3
- Input: [3, 7, 1, 2, 8, 4, 5, 6] → Output: 9

Java Code:

```
public class MissingNumber {
    public static int findMissing(int[] arr, int n) {
        int total = n * (n + 1) / 2;
        int sum = 0;
        for (int num: arr) {
            sum += num;
        }
        return total - sum;
    }
    public static void main(String[] args) {
        int[] arr1 = {1, 2, 4, 5};
        System.out.println("Missing Number: " + findMissing(arr1, 5)); // 3

        int[] arr2 = {3, 7, 1, 2, 8, 4, 5, 6};
        System.out.println("Missing Number: " + findMissing(arr2, 9)); // 9
    }
}
```

3. Find the Number That Appears Once, Others Twice

Problem: In an array where every number appears twice except one, find the number that appears only once.

Examples:

- Input: [2, 3, 5, 4, 5, 3, 4] → Output: 2
- Input: [1, 2, 2, 1, 3] → Output: 3

Java Code (Using XOR):

```
public class SingleNumber {
    public static int findSingle(int[] arr) {
        int result = 0;
        for (int num : arr) {
            result ^= num;  // XOR cancels out duplicates
        }
        return result;
    }

    public static void main(String[] args) {
        int[] arr1 = {2, 3, 5, 4, 5, 3, 4};
        System.out.println("Single Number: " + findSingle(arr1)); // 2

        int[] arr2 = {1, 2, 2, 1, 3};
        System.out.println("Single Number: " + findSingle(arr2)); // 3
    }
}
```

Day-3

1. Find the Union of Two Arrays

Problem: Find the union of two arrays (without duplicates).

Examples:

- Input: arr1 = [1, 2, 3], arr2 = [2, 3, 4, 5] → Output: [1, 2, 3, 4, 5]

- Input: arr1 = [7, 1, 5], arr2 = [2, 3, 1] ⟶ Output: [1, 2, 3, 5, 7]

Java Code (Using Set):

```
import java.util.*;

public class ArrayUnion {
   public static void findUnion(int[] arr1, int[] arr2) {
      Set<Integer> set = new TreeSet<>(); // TreeSet keeps it sorted
      for (int num : arr1) set.add(num);
      for (int num : arr2) set.add(num);
      System.out.println("Union: " + set);
   }

   public static void main(String[] args) {
      int[] arr1 = {1, 2, 3};
      int[] arr2 = {2, 3, 4, 5};
      findUnion(arr1, arr2); // [1, 2, 3, 4, 5]

      int[] arr3 = {7, 1, 5};
      int[] arr4 = {2, 3, 1};
      findUnion(arr3, arr4); // [1, 2, 3, 5, 7]
   }
}
```

2. Two Sum Problem

Problem: Find two numbers in an array that add up to a target sum.

Examples:

- Input: arr = [2, 7, 11, 15], target = 9 ⟶ Output: [0, 1] (2 + 7)
- Input: arr = [3, 2, 4], target = 6 ⟶ Output: [1, 2] (2 + 4)

Java Code (Using HashMap):

```
import java.util.*;


public class TwoSum {
   public static int[] findTwoSum(int[] nums, int target) {
      Map<Integer, Integer> map = new HashMap<>();
      for (int i = 0; i < nums.length; i++) {
         int complement = target - nums[i];
         if (map.containsKey(complement)) {
                return new int[] { map.get(complement), i };
            }
            map.put(nums[i], i);
        }
        return new int[] { -1, -1 }; // If not found
    }

    public static void main(String[] args) {
        int[] result1 = findTwoSum(new int[] {2, 7, 11, 15}, 9);
        System.out.println("Indexes: " + Arrays.toString(result1)); // [0, 1]


        int[] result2 = findTwoSum(new int[] {3, 2, 4}, 6);
        System.out.println("Indexes: " + Arrays.toString(result2)); // [1, 2]
    }
}
```

Day-4

1 Remove Duplicates from Sorted Array

Given a sorted array, remove the duplicates in place such that each element appear only once and return the new length. Do not allocate extra space for another array, you must do this in place with constant memory.

For example, given input array A = [1,1,2], your function should return length = 2, and A is now [1,2].

1.1 Analysis

The problem is pretty straightforward. It returns the length of the array with unique elements, but the original array need to be changed also. This problem is similar to Remove Duplicates from Sorted Array II

Java Solution

```
public static int removeDuplicates(int[] A) {
if (A.length < 2)
return A.length;
int j = 0;
int i = 1;
while (i < A.length) {
if (A[i] != A[j]) {
j++;
A[j] = A[i];
}
i++;
}
return j + 1;
}
```

2 Remove Duplicates from Sorted Array II

Follow up for "Remove Duplicates": What if duplicates are allowed at most twice?

For example, given sorted array A = [1,1,1,2,2,3], your function should return length = 5, and A is now [1,1,2,2,3].

So this problem also requires in-place array manipulation.

Java Solution 1

We can not change the given array’s size, so we only change the first k elements of the array which has duplicates removed.

```
public int removeDuplicates(int[] nums) {
if(nums==null){
return 0;
}
if(nums.length<3){
return nums.length;
}
int i=0;
int j=1;
while(j<nums.length){
if(nums[j]==nums[i]){
if(i==0){
i++;
j++;
}else if(nums[i]==nums[i-1]){
```

```
j++;
}else{
i++;
nums[i]=nums[j];
j++;
}
}else{
i++;
nums[i]=nums[j];

j++; } }
return i+1;
}
```

Java Solution 2

```
public int removeDuplicates(int[] nums) {
if(nums==null){
return 0;
}
if (nums.length <= 2){
return nums.length;
}
/*
1,1,1,2,2,3
i j
*/
int i = 1; // point to previous
int j = 2; // point to current
while (j < nums.length) {
if (nums[j] == nums[i] && nums[j] == nums[i - 1]) {
j++;
} else {
i++;
nums[i] = nums[j];
j++;
}
}
return i + 1;
}
```