// Q 1. Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target.

```
import java.util.HashMap;
import java.util.Map;

public class TwoSum {
    public int[] twoSum(int[] nums, int target) {

        Map<Integer, Integer> complementMap = new HashMap<>();

        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];



            if (complementMap.containsKey(complement)) {
                return new int[]{complementMap.get(complement), i};
            }



            complementMap.put(nums[i], i);
        }

        throw new IllegalArgumentException("No two numbers add up to the target.");
    }

    public static void main(String[] args) {
        TwoSum solution = new TwoSum();

        int[] nums = {2, 7, 11, 15};
        int target = 9;
```

```
        int[] result = solution.twoSum(nums, target);
        System.out.println("Indices: [" + result[0] + ", " + result[1] + "]");
    }
}
```

Q 2. Given an integer array nums and an integer val, remove all occurrences of val in nums in-place. The order of the elements may be changed. Then return the number of elements in nums which are not equal to val.

```
public class RemoveElement {
    public int removeElement(int[] nums, int val) {
        int k = 0;

        for (int i = 0; i < nums.length; i++) {
            if (nums[i] != val) {
                nums[k] = nums[i];
                k++;
            }
        }

        return k;
    }

    public static void main(String[] args) {
        RemoveElement solution = new RemoveElement();

        int[] nums = {3, 2, 2, 3};
        int val = 3;

        int count = solution.removeElement(nums, val);
        System.out.println("Count: " + count);
```

```
        System.out.print("Updated Array: [");
        for (int i = 0; i < count; i++) {
            System.out.print(nums[i]);
            if (i < count - 1) {
                System.out.print(", ");
            }
        }
        System.out.println("]");
    }
}
```

Q 3.

Given a sorted array of distinct integers and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

```
public class SearchInsertPosition {
    public int searchInsert(int[] nums, int target) {
        int left = 0;
        int right = nums.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (nums[mid] == target) {
                return mid;
            } else if (nums[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
```

```
        return left;
    }

    public static void main(String[] args) {
        SearchInsertPosition solution = new SearchInsertPosition();

        int[] nums = {1, 3, 5, 6};
        int target = 5;

        int index = solution.searchInsert(nums, target);
        System.out.println("Index: " + index);
    }
}
```

Q 4. You are given a large integer represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

ANS -

```
 public class PlusOne {
    public int[] plusOne(int[] digits) {
        int n = digits.length;
            for (int i = n - 1; i >= 0; i--) {
            digits[i]++;



            if (digits[i] == 10) {
                digits[i] = 0;
            } else {
```

```
            return digits;
        }
    }

    int[] result = new int[n + 1];
    result[0] = 1;
    return result;
  }

  public static void main(String[] args) {
    PlusOne solution = new PlusOne();

    int[] digits = {1, 2, 3};
    int[] result = solution.plusOne(digits);

    System.out.print("Result: ");
    for (int digit : result) {
      System.out.print(digit + " ");
    }
  }
}
```

Q You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively.

The final sorted array should not be returned by the function, but instead be stored inside the array nums1. To accommodate this, nums1 has a length of m + n, where the first m elements denote the

elements that should be merged, and the last n elements are set to 0 and should be ignored. nums2 has a length of n.

```
public class MergeSortedArrays {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int i = m - 1;
        int j = n - 1;
        int k = m + n - 1;



        while (i >= 0 && j >= 0) {
            if (nums1[i] > nums2[j]) {
                nums1[k] = nums1[i];
                i--;
            } else {
                nums1[k] = nums2[j];
                j--;
            }
            k--;
        }



        while (j >= 0) {
            nums1[k] = nums2[j];
            j--;
            k--;
        }
    }

    public static void main(String[] args) {
```

```
        MergeSortedArrays solution = new MergeSortedArrays();

        int[] nums1 = {1, 2, 3, 0, 0, 0};
        int m = 3;
        int[] nums2 = {2, 5, 6};
        int n = 3;

        solution.merge(nums1, m, nums2, n);

        System.out.print("Merged Array: ");
        for (int num : nums1) {
            System.out.print(num + " ");
        }
    }
}
```

Q 6 Given an integer array nums, return true if any value appears at least twice in the array, and return false if every element is distinct.

Input: nums = [1,2,3,1]

Output: true

ans -

```
import java.util.HashSet;
import java.util.Set;

public class ContainsDuplicate {
```

```
    public boolean containsDuplicate(int[] nums) {
        Set<Integer> set = new HashSet<>();
        for (int num : nums) {
            if (set.contains(num)) {
                return true;
            }
            set.add(num);
        }
        return false;
    }

    public static void main(String[] args) {
        ContainsDuplicate solution = new ContainsDuplicate();

        int[] nums = {1, 2, 3, 1};
        boolean result = solution.containsDuplicate(nums);

        System.out.println("Contains Duplicate: " + result);
    }
}
```

Q 7. Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the nonzero elements.

Ans -

```
public class MoveZeroes {
    public void moveZeroes(int[] nums) {
        int i = 0;
```

```
        for (int num : nums) {
            if (num != 0) {
                nums[i] = num;
                i++;
            }
        }

        while (i < nums.length) {
            nums[i] = 0;
            i++;
        }
    }

    public static void main(String[] args) {
        MoveZeroes solution = new MoveZeroes();

        int[] nums = {0, 1, 0, 3, 12};
        solution.moveZeroes(nums);

        System.out.print("Result: ");
        for (int num : nums) {
            System.out.print(num + " ");
        }
    }
}
```

Q 8. You have a set of integers s, which originally contains all the numbers from 1 to n. Unfortunately, due to some error, one of the numbers in s got duplicated to another number in the set, which results in repetition of one number and loss of another number.

ANS - import java.util.Arrays;

```
public class FindErrorNums {
    public int[] findErrorNums(int[] nums) {
        int[] result = new int[2];



        Arrays.sort(nums);

        int duplicate = -1;
        int missing = 1;

        for (int i = 1; i < nums.length; i++) {
            if (nums[i] == nums[i - 1]) {
                duplicate = nums[i];
            } else if (nums[i] > nums[i - 1] + 1) {
                missing = nums[i - 1] + 1;
            }
        }

        if (nums[nums.length - 1] != nums.length) {
            missing = nums.length;
        }

        result[0] = duplicate;
        result[1] = missing;

        return result;
    }

    public static void main(String[] args) {
```

```
        FindErrorNums solution = new FindErrorNums();

        int[] nums = {1, 2, 2, 4};
        int[] result = solution.findErrorNums(nums);

        System.out.println("Duplicate: " + result[0]);
        System.out.println("Missing: " + result[1]);
    }
}
```