**Assignment Questions 1**

**Q1.** Given an array of integer numbers and an integer target, return indices of the two numbers such that they add up to the target.

You may assume that each input would have exactly one solution, and you may not use the same element twice.

You can return the answer in any order.

**Example:** Input: nums = [2,7,11,15], target = 9 Output 0 [0,1]

**Explanation:** Because nums[0] + nums[1] == 9, we return [0, 1][

Solution: import java.util.HashMap;

import java.util.Scanner;

public class TwoSum {

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

// Create a HashMap to store the complement of each element

// and its corresponding index

HashMap<Integer, Integer> map = new HashMap<>();

for (int i = 0; i < nums.length; i++) {

int complement = target - nums[i];

// If the complement exists in the map, we found a pair

if (map.containsKey(complement)) {

// Return the indices of the two numbers

return new int[]{map.get(complement), i};

}

// Otherwise, add the current element and its index to the map

map.put(nums[i], i);

}

// No solution found, return an empty array

return new int[0];

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read the array from the user

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter the elements of the array:");

for (int i = 0; i < n; i++) {

nums[i] = scanner.nextInt();

}

// Read the target value from the user

System.out.print("Enter the target value: ");

int target = scanner.nextInt();

TwoSum solution = new TwoSum();

int[] result = solution.twoSum(nums, target);

System.out.println("Output: [" + result[0] + ", " + result[1] + "]");

}

}

**Q2.** 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 numbers which are not equal to val.

Consider the number of elements in numbers which are not equal to val beck, to get accepted, you need to do the following things:

Change the array numbers such that the first k elements of nums contain the elements which are not equal to val. The remaining elements of nums are not important as well as the size of nums.

Return k.

**Example :** Input: nums = [3,2,2,3], val = 3 Output: 2, nums = [2,2,\*,\*]

**Explanation:** Your function should return k = 2, with the first two elements of nuns being 2. It does not matter what you leave beyond the returned k (hence they are underscores)[

Solution: import java.util.Arrays;

import java.util.Scanner;

public class RemoveElement {

public int removeElement(int[] nums, int val) {

int k = 0; // Number of elements not equal to val

// Iterate through the array

for (int i = 0; i < nums.length; i++) {

// If the current element is not equal to val

if (nums[i] != val) {

// Move the element to the front of the array

nums[k] = nums[i];

k++; // Increment k

}

}

return k;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read the array from the user

System.out.print("Enter the number of elements in the array: ");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter the elements of the array:");

for (int i = 0; i < n; i++) {

nums[i] = scanner.nextInt();

}

// Read the value to be removed from the user

System.out.print("Enter the value to be removed: ");

int val = scanner.nextInt();

RemoveElement solution = new RemoveElement();

int k = solution.removeElement(nums, val);

System.out.println("Output: " + k + ", nums = " + Arrays.toString(nums));

}

}

**Q3.** 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.

You must write an algorithm with O(log n) runtime complexity.

**Example 1:** Input: nums = [1,3,5,6], target = 5

Output: 2

Solution: import java.util.Scanner;

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;

}

}

// Target not found, return the index where it would be inserted

return left;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read the array from the user

System.out.print("Enter the number of elements in the sorted array: ");

int n = scanner.nextInt();

int[] nums = new int[n];

System.out.println("Enter the elements of the sorted array:");

for (int i = 0; i < n; i++) {

nums[i] = scanner.nextInt();

}

// Read the target value from the user

System.out.print("Enter the target value: ");

int target = scanner.nextInt();

SearchInsertPosition solution = new SearchInsertPosition();

int index = solution.searchInsert(nums, target);

System.out.println("Output: " + index);

}

}

**Q4.** You are given a large integer represented as an integer array of digits, where each digit[i] is the nth 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.

Increment the large integer by one and return the resulting array of digits.

**Example 1:** Input: digits = [1,2,3] Output: [1,2,4]

**Explanation:** The array represents the integer 123.

Incrementing by one gives 123 + 1 = 124. Thus, the result should be [1,2,4].

Solution: import java.util.Arrays;

import java.util.Scanner;

public class PlusOne {

public int[] plusOne(int[] digits) {

int n = digits.length;

// Traverse the array in reverse order

for (int i = n - 1; i >= 0; i--) {

// Increment the current digit by one

digits[i] += 1;

// If the current digit becomes 10, set it to 0 and continue to the next digit

if (digits[i] == 10) {

digits[i] = 0;

}

// If the current digit is less than 10, we don't need to carry over, so we can return the array

else {

return digits;

}

}

// If all digits become 0 and we still need to carry over, create a new array with an additional leading 1

int[] result = new int[n + 1];

result[0] = 1;

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read the number of digits from the user

System.out.print("Enter the number of digits: ");

int n = scanner.nextInt();

int[] digits = new int[n];

System.out.println("Enter the digits (from left to right):");

for (int i = 0; i < n; i++) {

digits[i] = scanner.nextInt();

}

PlusOne solution = new PlusOne();

int[] result = solution.plusOne(digits);

System.out.println("Output: " + Arrays.toString(result));

}

}

**Q5.** 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.

Merge nums1 and nums2 into a single array sorted in non-decreasing order.

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.

**Example 1:** Input: nums1 = [1,2,3,0,0,0], m = 3, nums2 = [2,5,6], n = 3 Output: [1,2,2,3,5,6]

**Explanation:** The arrays we are merging are [1,2,3] and [2,5,6]. The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

Solution: import java.util.Arrays;

import java.util.Scanner;

public class MergeSortedArrays {

public static void merge(int[] nums1, int m, int[] nums2, int n) {

int i = m - 1; // Index of last element in nums1

int j = n - 1; // Index of last element in nums2

int k = m + n - 1; // Index of last element in the merged array

// Merge elements from the end of the arrays

while (i >= 0 && j >= 0) {

if (nums1[i] > nums2[j]) {

nums1[k] = nums1[i];

i--;

} else {

nums1[k] = nums2[j];

j--;

}

k--;

}

// Copy remaining elements from nums2 to nums1 (if any)

while (j >= 0) {

nums1[k] = nums2[j];

j--;

k--;

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read the size and elements of nums1

System.out.print("Enter the size of nums1: ");

int m = scanner.nextInt();

int[] nums1 = new int[m];

System.out.print("Enter the elements of nums1 (sorted in non-decreasing order): ");

for (int i = 0; i < m; i++) {

nums1[i] = scanner.nextInt();

}

// Read the size and elements of nums2

System.out.print("Enter the size of nums2: ");

int n = scanner.nextInt();

int[] nums2 = new int[n];

System.out.print("Enter the elements of nums2 (sorted in non-decreasing order): ");

for (int i = 0; i < n; i++) {

nums2[i] = scanner.nextInt();

}

// Create nums1 with space for merging nums2

int[] mergedArray = new int[m + n];

System.arraycopy(nums1, 0, mergedArray, 0, m);

// Merge arrays

merge(mergedArray, m, nums2, n);

// Print the merged array

System.out.println("Merged array: " + Arrays.toString(mergedArray));

}

}

**Q6.** 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.

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

Output: true

Solution: import java.util.HashSet;

import java.util.Scanner;

import java.util.Set;

public class ContainsDuplicate {

public static 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) {

Scanner scanner = new Scanner(System.in);

// Read the size and elements of the array

System.out.print("Enter the size of the array: ");

int size = scanner.nextInt();

int[] nums = new int[size];

System.out.print("Enter the elements of the array: ");

for (int i = 0; i < size; i++) {

nums[i] = scanner.nextInt();

}

// Check if the array contains any duplicate

boolean containsDuplicate = containsDuplicate(nums);

// Print the result

System.out.println("Output: " + containsDuplicate);

}

}

**Q7.** Given an integer array nums, move all 0's to the end of it while maintaining the relative order of the nonzero elements.

Note that you must do this in-place without making a copy of the array.

**Example 1:** Input: nums = [0,1,0,3,12] Output: [1,3,12,0,0]

Solution: import java.util.Scanner;

public class MoveZeroes {

public static void moveZeroes(int[] nums) {

int index = 0; // Index to keep track of the non-zero elements

// Move non-zero elements to the front of the array

for (int num : nums) {

if (num != 0) {

nums[index] = num;

index++;

}

}

// Fill the remaining elements with zeroes

while (index < nums.length) {

nums[index] = 0;

index++;

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read the size and elements of the array

System.out.print("Enter the size of the array: ");

int size = scanner.nextInt();

int[] nums = new int[size];

System.out.print("Enter the elements of the array: ");

for (int i = 0; i < size; i++) {

nums[i] = scanner.nextInt();

}

// Move zeros to the end of the array

moveZeroes(nums);

// Print the modified array

System.out.print("Output: ");

for (int num : nums) {

System.out.print(num + " ");

}

System.out.println();

}

}

**Q8.** 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.

You are given an integer array nums representing the data status of this set after the error.

Find the number that occurs twice and the number that is missing and return them in the form of an array.

**Example 1:** Input: nums = [1,2,2,4] Output: [2,3]

Solution: import java.util.Arrays;

import java.util.HashSet;

import java.util.Scanner;

import java.util.Set;

public class FindErrorNums {

public static int[] findErrorNums(int[] nums) {

int[] result = new int[2];

Set<Integer> set = new HashSet<>();

int n = nums.length;

int duplicate = -1;

int missing = -1;

for (int num : nums) {

if (set.contains(num)) {

duplicate = num;

} else {

set.add(num);

}

}

// Find the missing number

for (int i = 1; i <= n; i++) {

if (!set.contains(i)) {

missing = i;

break;

}

}

result[0] = duplicate;

result[1] = missing;

return result;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Read the size and elements of the array

System.out.print("Enter the size of the array: ");

int size = scanner.nextInt();

int[] nums = new int[size];

System.out.print("Enter the elements of the array: ");

for (int i = 0; i < size; i++) {

nums[i] = scanner.nextInt();

}

int[] errorNums = findErrorNums(nums);

// Print the result

System.out.println("Output: " + Arrays.toString(errorNums));

}

}