**Q1. Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to 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 Output0 [0,1]**

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

**Prgm:**

class Solution {

static int[] twoSum(int []num , int target)

{

for(int i = 0 ; i < num.length - 1 ; i++)

for(int j = i + 1 ; j < num.length ; j++)

{

if(num[i] + num[j] == target)

return new int[]{i + 1 , j + 1};

}

return new int[]{-1,-1};

}

public static void main(String args[])

{

int [] a = {2, 7, 11, 15};

int target = 9;

for(int x : twoSum(a , target))

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

}

}

**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 nums which are not equal to val.**

**Consider the number of elements in nums which are not equal to val be k, to get accepted, you need to do the following things:**

* **Change the array nums 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,*\*,*\*]**

**Prgm:**

class Solution {

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

int index = 0;

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

if (nums[i] != val) {

nums[index] = nums[i];

index++;

}

}

return index;

}

}

**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**

**Prgm:**

class Solution {

static int searchInsert(int[] a , int target)

{

int n = a.length;

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

{

if(a[i] >= target)

return i;

}

return n;

}

public static void main(String args[])

{

int a[] = {1 , 3 , 5 , 7 , 9} , target = 8;

System.out.println(searchInsert(a , target));

}

}

**Q4. 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.**

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

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

**Prgm:**

class Solution {

for (int i = digits.length - 1; i >= 0; i--) {

if (digits[i] < 9) {

digits[i]++;

return digits;

}

digits[i] = 0;

}

digits = new int[digits.length + 1];

digits[0] = 1;

return digits

}

**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]**

**Prgm:**

class Solution {

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 (j >= 0) {

if (i >= 0 && nums1[i] > nums2[j]) {

nums1[k--] = nums1[i--];

} else {

nums1[k--] = nums2[j--];

}

}

}

}

**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**

**Prgm:**

class Solution {

public boolean containsDuplicate(int[] nums) {

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

for(int j = i + 1; j < nums.length; j++) {

if(nums[i] == nums[j]) {

return true;

}

}

}

return false;

}

public static void main (String[] args) {

Solution sol = new Solution();

int nums[]= {1, 2, 3, 1};

boolean res = sol.containsDuplicate(nums);

System.out.println(res);

}

}

**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]**

**Prgm:**class Solution {

public void moveZeroes(int[] nums) {

int n = nums.length;

int i = 0, j = 0;

while(j<n){

if(nums[j]==0){

j++;

}

else{

int temp = nums[j];

nums[j] = nums[i];

nums[i] = temp;

i++;

j++;

}

}

}

}

**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]**

**Prgm:**

class Solution {

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

int dup = -1, missing = -1;

for (int i = 1; i <= nums.length; i++) {

int count = 0;

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

if (nums[j] == i)

count++;

}

if (count == 2)

dup = i;

else if (count == 0)

missing = i;

}

return new int[] {dup, missing};

}

}