💡 **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][**

**Program -**

class Solution {

public:

vector<int> twoSum(vector<int>& nums, int target) {

vector<int>ans;

unordered\_map<int,int>m;

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

{

int val = target-nums[i];

if(m.find(val)!=m.end())

{

ans.push\_back(m.find(val)->second);

ans.push\_back(i);

break;

}

m.insert(pair<int,int>(nums[i],i));

}

return ans;

}

};

**Output -  
Input - Nums = [2,7,11,15]**

**Target = 9**

**Output - [0,1]**

**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 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,*\*,*\*]**

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

**Program:**

**class Solution {**

**public:**

**int removeElement(vector<int>& nums, int val) {**

**if(nums.size() == 0){**

**return 0; }**

**int i, j = 0;**

**for(i = 0; i < nums.size() - 1; i++){**

**if(nums[i] != val){**

**nums[j++] = nums[i] }**

**}**

**if(nums[i] != val){**

**nums[j++] = nums[i];**

**}**

**return j; }};**

**Output - Input - nums= [3,2,2,3]**

**Val = 3**

**Output - 2,2**

**💡 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**

**Program:**

**class Solution {**

**public:**

**int searchInsert(vector<int>& nums, int target) {**

**int start = 0;**

**int end = nums.size()-1;**

**while(start <= end){**

**int mid = (start + end)/2;**

**if(target > nums[mid]){**

**start = mid + 1;**

**}else if(target < nums[mid]){**

**end = mid - 1;**

**}else{**

**return mid;**

**}**

**}**

**return start }};**

**OUTPUT :**

**Nums: [1,3,5,6]**

**Target = 5**

**Output : 2**

**💡 Q4. You are given a large integer represented as an integer array digits, where each digit[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]**

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

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

**Program:**

class Solution {

public:

vector<int> plusOne(vector<int>& digits) {

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

++digits[i];

digits[i] %= 10;

if (digits[i] != 0) return digits;

}

digits.insert(digits.begin(), 1);

return digits;

}

**Output: - Digit = [1,2,3]**

**[1,2,4]**

**💡 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.**

**Program**

class Solution {

public:

void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {

int i = m - 1, j = n - 1, k = m

for(; i >=0 && j >=0; k--){

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

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

} else {

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

}

}

while(i >= 0) {

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

}

while(j >= 0) {

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

}

}

};

**Output - nums1 = [1,2,3,0,0,0] , m =3,**

**nums2 = [2,5,6] , n = 3**

**Output - [1,2,2,3,5,6]**

**💡 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: I**

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

**Output: true**

**Program:**

class Solution {

public:

bool containsDuplicate(vector<int>& nums) {

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

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

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

return true;

}

}

}

return false;

}

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

Output - true

**💡 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]**

**Program:**

Class solution {

Public:

Void movezeroes (vector<int>&nums){

Int i = 0 , j = 0 , n = nums.size();

While (i < n)

{

If (nums [i] !=0) {

Swap (nums[i], nums[j]);

i++;

J++;

}

Else

i++;

}

}

};

**OUTPUT: nums = [0,1,0,3,12]**

**Output : [1,3,12,0,0]**

💡 **Q8.** **You have a set of integers, 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]**

**program :**

class Solution {

public:

vector<int> findErrorNums(vector<int>& nums) {

int N = nums.size(), sum = N \* (N + 1) / 2;

vector<int> ans(2);

vector<bool> seen(N+1);

for (int num : nums) {

sum -= num;

};

Output: nums =[1,2,2,4]

Output: [2,3]