**217. Contains Duplicate**

Easy

3867926Add to ListShare

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

Solutions :

class Solution {

public boolean containsDuplicate(int[] nums) {

HashSet<Integer> checkDuplicate = new HashSet<>();

for(Integer element : nums)

{

if(checkDuplicate.add(element) == false)

return true;

}

return false;

}

}

**53. Maximum Subarray**

Easy

18551882Add to ListShare

Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return *its sum*.

A **subarray** is a **contiguous** part of an array.

**Example 1:**

**Input:** nums = [-2,1,-3,4,-1,2,1,-5,4]

**Output:** 6

**Explanation:** [4,-1,2,1] has the largest sum = 6.

class Solution {

public int maxSubArray(int[] nums) {

int sum = 0;

int max = Integer.MIN\_VALUE;

for(int element : nums){

if(sum < 0)

sum = element;

else

sum +=element;

if(sum > max)

max = sum;

}

return max;

}

}

**Longest Substring Without Repeating Characters**

Medium

21501961Add to ListShare

Given a string s, find the length of the **longest substring** without repeating characters.

**Example 1:**

**Input:** s = "abcabcbb"

**Output:** 3

**Explanation:** The answer is "abc", with the length of 3.

class Solution {

public int lengthOfLongestSubstring(String s) {

int length = 0, max = 0;

char[] stringInChar = s.toCharArray();

Queue<Character> output = new LinkedList<>();

for(char element : stringInChar){

if(output.contains(element)){

output = removeElement(output,element);

length = output.size();

}

output.add(element);

length++;

if(length > max)

max = length;

}

return max;

}

public Queue<Character> removeElement( Queue<Character> a,char b){

a.poll();

if(a.contains(b))

return removeElement(a,b);

return a;

}

}

**88. Merge Sorted Array**

Easy

3479341Add to ListShare

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.

class Solution {

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

int a = 0;

for(int element : nums2){

if(m ==0)

nums1[a] = element;

else

nums1[m+a] = element;

a++;

}

Arrays.sort(nums1);

}}

**Two Sum**

Easy

29464935Add to ListShare

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 1:**

**Input:** nums = [2,7,11,15], target = 9

**Output:** [0,1]

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

class Solution {

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

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

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

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

return new int [] {i,j};

}

}

return new int [] {0,0};

}

}

**704. Binary Search**

Easy

351291Add to ListShare

Given an array of integers nums which is sorted in ascending order, and an integer target, write a function to search target in nums. If target exists, then return its index. Otherwise, return -1.

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

**Example 1:**

**Input:** nums = [-1,0,3,5,9,12], target = 9

**Output:** 4

**Explanation:** 9 exists in nums and its index is 4

class Solution {

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

return binarySearch(nums,0,nums.length-1,target);

}

int binarySearch(int[] arr,int l, int r, int x){

if(l<r){

int mid = (l + r-1) /2;

if(arr[mid] == x)

return mid;

else if(arr[mid] > x)

return binarySearch(arr,1,mid-1,x);

else

return binarySearch(arr,mid+1,r,x);

}

return -1; }}

**278. First Bad Version**

Easy

42531583Add to ListShare

You are a product manager and currently leading a team to develop a new product. Unfortunately, the latest version of your product fails the quality check. Since each version is developed based on the previous version, all the versions after a bad version are also bad.

Suppose you have n versions [1, 2, ..., n] and you want to find out the first bad one, which causes all the following ones to be bad.

You are given an API bool isBadVersion(version) which returns whether version is bad. Implement a function to find the first bad version. You should minimize the number of calls to the API.

**Example 1:**

**Input:** n = 5, bad = 4

**Output:** 4

**Explanation:**

call isBadVersion(3) -> false

call isBadVersion(5) -> true

call isBadVersion(4) -> true

Then 4 is the first bad version.

/\* The isBadVersion API is defined in the parent class VersionControl.

boolean isBadVersion(int version); \*/

public class Solution extends VersionControl {

public int firstBadVersion(int n) {

int l=1,r = n;

while(l<=r){

int mid = (l+r) /2;

boolean result = isBadVersion(mid);

if(result == false)

l = mid+1;

else{

boolean result1 = isBadVersion(mid-1);

if(result1 == false)

return mid;

r = mid -1;

}

}

return 0;

}

}