Assignment 1 Questions - Arrays | DSA

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][

Ans:- **Approach :**

Now I will share two approaches to solving this question :

**1. Brute Force Method**

In this approach I will iterate the array using two loops to find if the sum is equal to the target.for(int i=0;i<nums.size()-1;i++)  
{  
 for(int j = i+1;j<nums.size();j++)  
 {  
 if(nums[i]+nums[j]==target)  
 ...  
 }  
}**Time Complexity** - O(n\*n)  
**Space Complexity** - O(1)

**2. Using Hashmap**

In this approach I will use hashmap and a vector to return the index of the elements if the target is found by addition of two elements.**Time Complexity** - O(n) because I will tarverse the array only once and Hasmap has a time compelexity of O(1) for insertion.   
**Space Complexity** - O(1) Code is as follows :**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())** // in case the second element is found  **{  
 ans.push\_back(m.find(val)->second);  
 ans.push\_back(i);  
 break;  
 }  
 m.insert(pair<int,int>(nums[i],i));** // in case the above criteria is not satisfied I will keep inserting the element in the hashmap  **}  
 return ans;  
 }  
};**

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 :

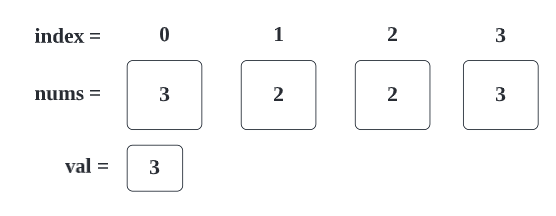
Input: nums = [3,2,2,3], val = 3

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)[

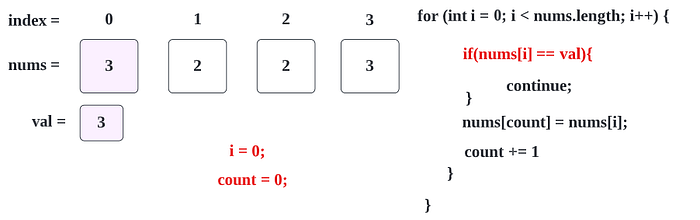
Ans:- solution with an example:

Suppose, we have given following an **nums**array and **val**.

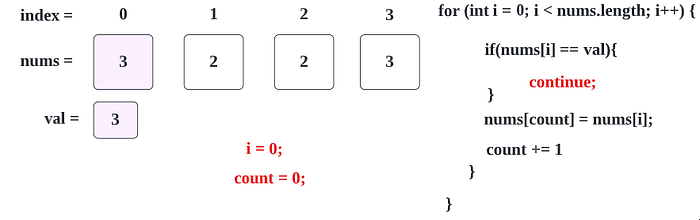


Now we have to remove all element which contain value: **3**from **nums**.

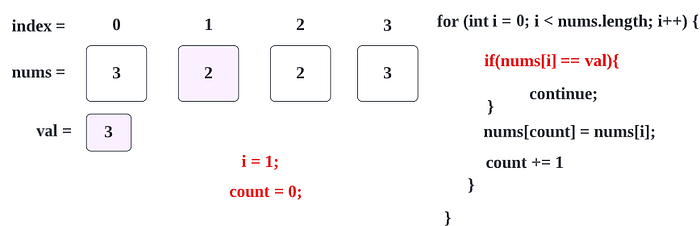
1. We are starting to transverse given array, if element contains value which is same as given **val**, then we skip it.



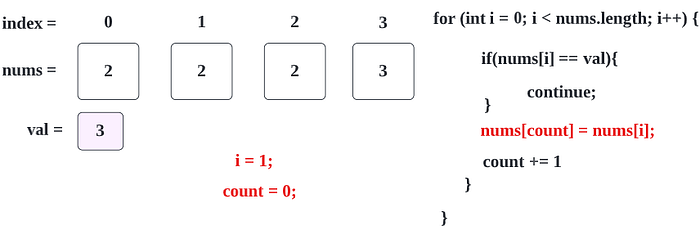
2. Here, **nums[0] → 3** and**val → 3**, both value are matching so, we are skipping below steps.



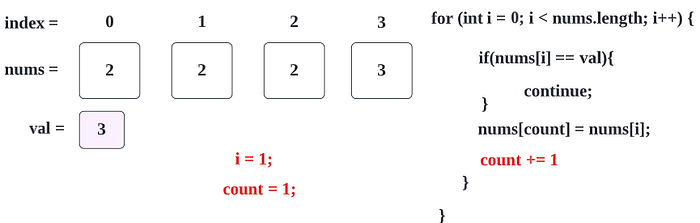
3. Now we are moving forward with **i = 1** case, here, **nums[1] → 2**and**val → 3**, both value are not matching.



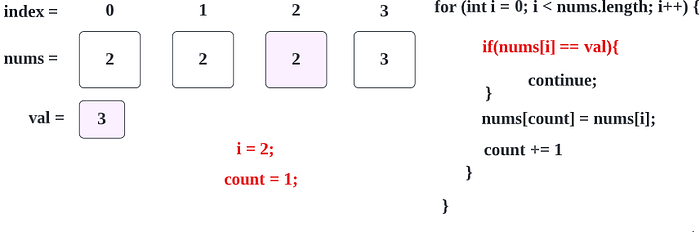
4. So we are going to replace **num’s 0th** index (as count → 0) value (which is **3**) with **num’s 1st** index value (which is **2**), so result will be like below.



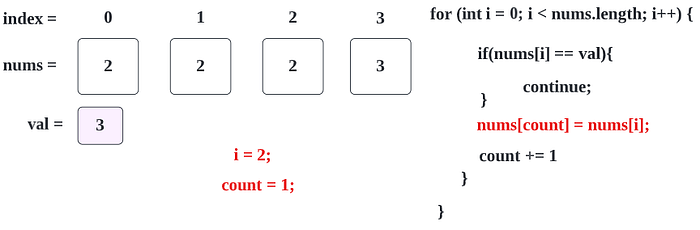
5. Now we have replaced value, so we are incrementing **count**variable by **1.**



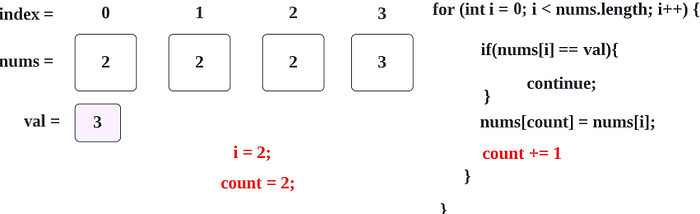
6. Now, next for loop, **i** will be incremented by **1**, and we repeat the step 1.



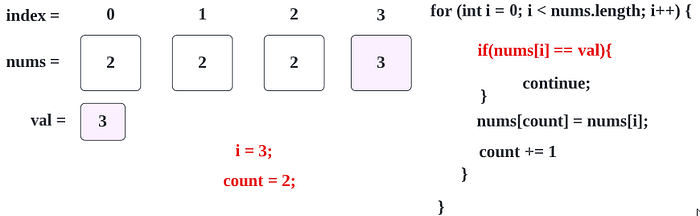
7. As the step 3, we are moving forward with **i = 2**case, here, **nums[2] → 2**and**val → 3**, both value are not matching.



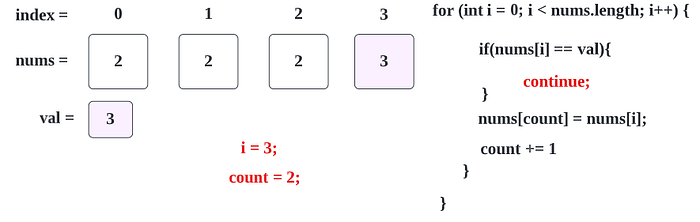
8. As the step 4 and 5, we are going to replace **num’s 1st** index (as **count → 1**) value (which is 2) with **num’s 3rd** index value (which is also **2)**, and then we increment value of **count**with **1,**so resultant will be like below.



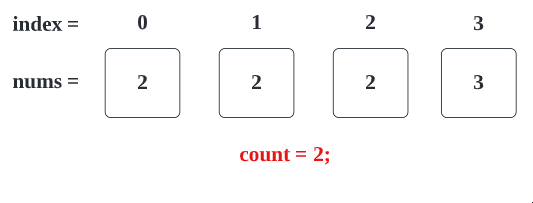
9. As the step 6, for the next for loop, **i** will be incremented by **1**, and we repeat the step 1.



10. As the step 2, here, **nums[3] → 3** and**val → 3**, both value are matching so, we are skipping below steps.

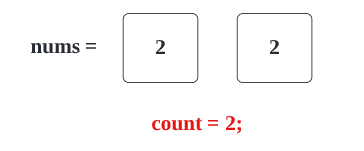


11. Now **i**will be 4 and for loop condition **(i < nums.length)** → **(4 < 4)** will be failed and resultant value **count**→ 2 and **array**will be like below



12. Here we will return **count**which mean**2** as an answer.

By returning **2** means, resultant **array’s first 2 element** only will be counted.



So, here you can see we got desired result, in this array there is no **val → 3.**

**Code (Java): →**class Solution {

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

int count = 0;

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

if(nums[i] == val){

continue;

}

nums[count] = nums[i];

count++;

}

return count;

}

}

Code (Python): →class Solution(object):

def removeElement(self, nums, val):

"""

:type nums: List[int]

:type val: int

:rtype: int

"""

count = 0

for i in range(len(nums)):

if nums[i] == val:

continue

nums[count] = nums[i]

count += 1

return count

# Time Complexity

We are scanning the array once, hence the time complexity will be **O(n)**.

# Space Complexity

Since we have no used any extra array, the space complexity will be **O(1)**.

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

## Ans:- Solution: →

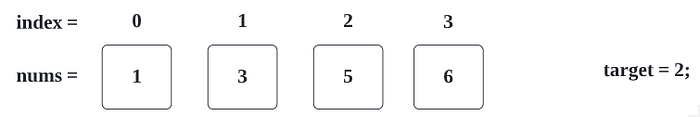
Here we can solve this problem by traversing whole array, but for that time complexity will be **O(n).**Here it is clearly mentioned that we must write an algorithm with O(log n) runtime complexity.

*o(log n) → Binary Search → Divide and conquer*

**Binary Search : →** In this we will find desired number on a sorted array of numbers, we divide (decrease) the array of numbers(search space), conquer the sub problems by recursively looking at the middle point for our target and splitting until we find our target the base case condition.

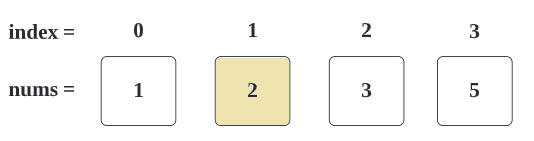
Now, lets understand it by example:

Suppose, we have given below array of **nums**and **target.**

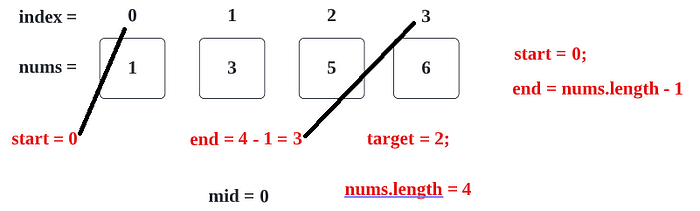


In above case, we can determine that we need to fill target value to 1st index.

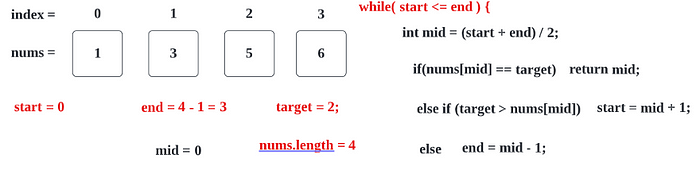
So, the resultant array would be like (just imagine),



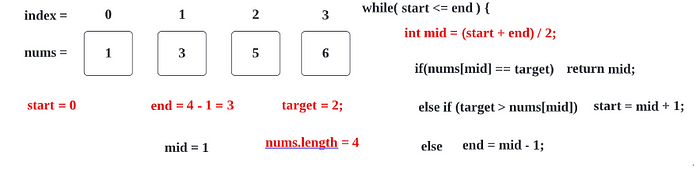
1. Here, first set **start** and **end** as **0** and **N — 1**, where the **start** and **end** variables denote the starting and ending bound of the search space respectively.



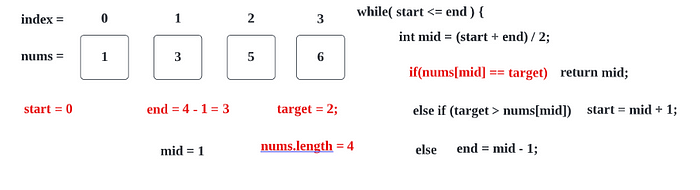
2. Now in next step, we are proceeding with while loop that will help us to traverse an array.



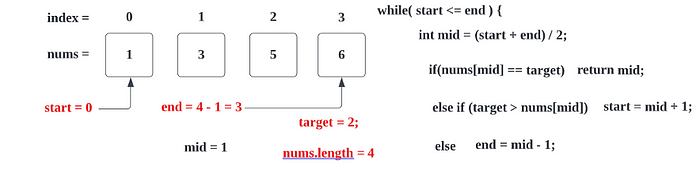
3. Here mid we will get is 1 (mid => (0 + 3)/2 => 1.5 => (int)1.5 => **1.**



4. First if condition will be failed as nums[1] => **1 != 2.**

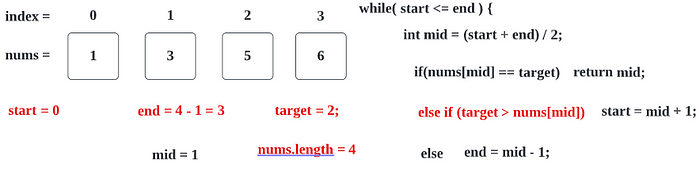


Before moving to another step, I am just explaining like **start**and **end**position.



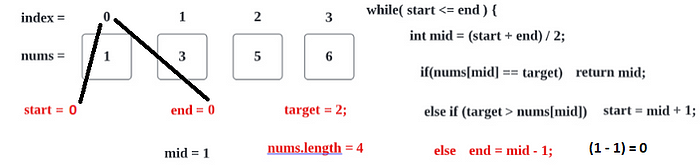
Now, with second **else if**condition:

(2 > nums[1]) => 2 > 3 => **FALSE**



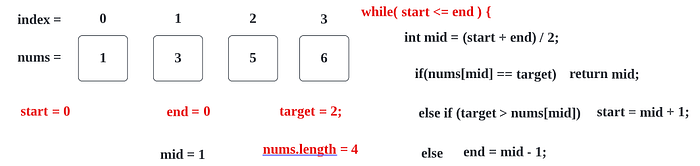
Now for the last else condition, **end**value will be changed

end = mid-1 => 1–1 => 0



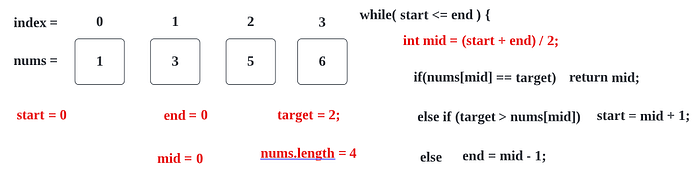
Now, we are going to next iteration of while loop and checking condition

start ≤ end => 0≤0 => **TRUE**

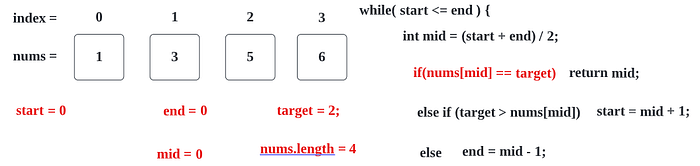


So, now we will proceed further and check **mid**value.

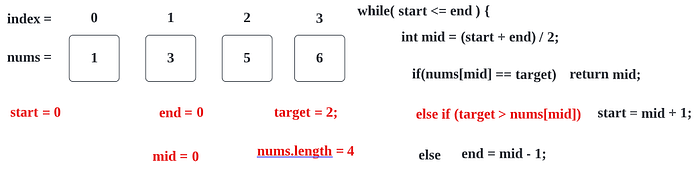
**mid**= (0 + 0)/2 = **0**



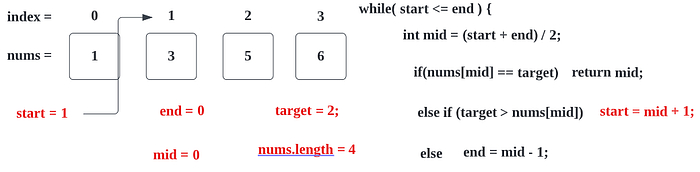
(nums[mid] == target) => nums[0] == 2 => 1 ==2 => **FALSE**



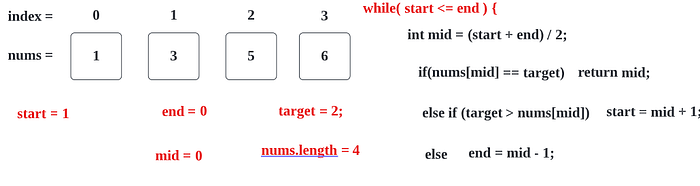
**target**> nums[mid] => 2 > num[0] => 2 >1 => **TRUE**



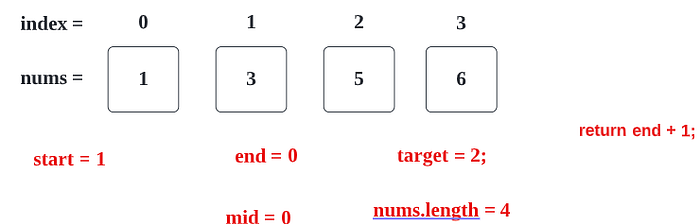
**start**= mid +1 => 0 + 1 => **1**



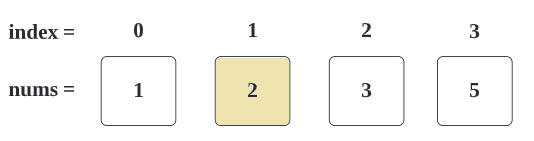
start ≤ end => 1≤0 => **FALSE**



at the end we will simply return **end + 1 => 0 + 1 => 1**(which will be insertion position (index))



which you can verify with below image, that **target**value which is **2**, should be placed in 1st index.



Code: (Java): → class Solution {

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

// Last and First indexes

int start = 0;

int end = nums.length - 1;

// Traverse an array

while( start <= end ) {

int mid = (start + end) / 2;

//if target value found.

if(nums[mid] == target) {

return mid;

}

// If target value is greater then mid elements's value

else if (target > nums[mid]) {

start = mid + 1;

}

//otherwise target value is less,

else {

end = mid - 1;

}

}

// Return the insertion position

return end + 1;

}

}

Code: (Python) → class Solution(object):

def searchInsert(self, nums, target):

"""

:type nums: List[int]

:type target: int

:rtype: int

"""

# Last and First indexes

start = 0

end = len(nums) - 1

# Traverse an array

while (start <= end):

mid = (start + end)/2

# if target value found.

if nums[mid] == target:

return mid

# If target value is greater then mid elements's value

elif target > nums[mid]:

start = mid + 1

# otherwise target value is less,

else:

end = mid -1

# Return the insertion position

return end + 1

# Time Complexity

We are traversing an array using binary search so time complexity would be O(log N).

# Space Complexity

Since we have no used any extra array, the space complexity will be **O(1)**.

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]

Explanation:The array represents the integer 123.

Incrementing by one gives 123 + 1 = 124.

Thus, the result should be [1,2,4].

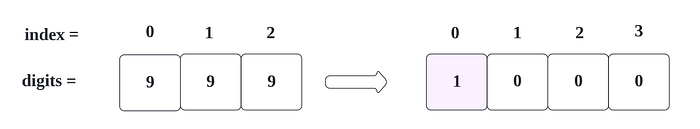
# Ans:- Solution: →

This is a little tricky problem. The interface requires to return **int[]**, but you are not sure what’s the length for the returning array. Given input array,**int[]** digits, it could be **digits.length** or **digits.length + 1**.

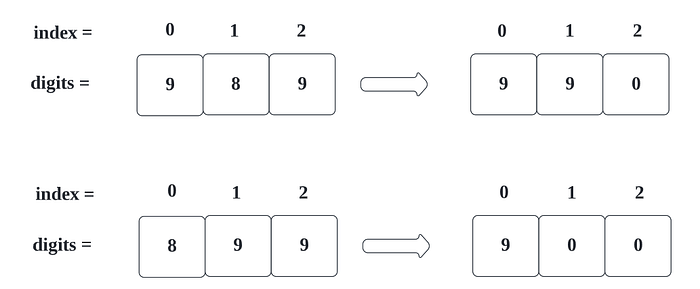
To get this solution, lets go step by step.

1. First, lets find out in what condition we will require **digits.length + 1.**

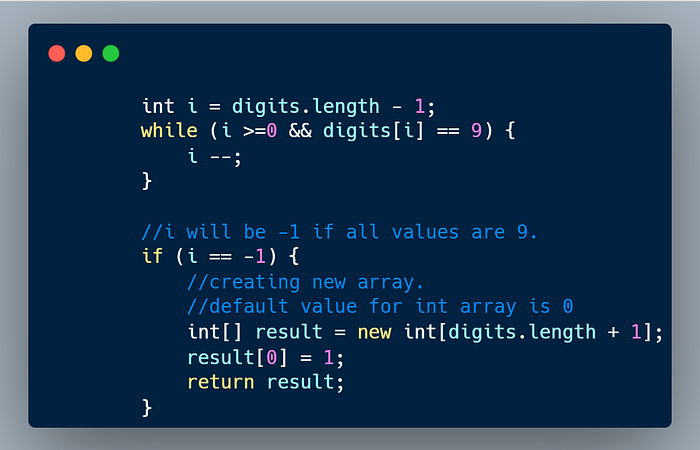
Here, by checking different values, we can see that if**all digits are 9**, in that case only we require **digits.length + 1**



If any digit value, other then 9 then we don’t need to take an array with size **digits.length + 1.**



For above condition, we can use following logic,

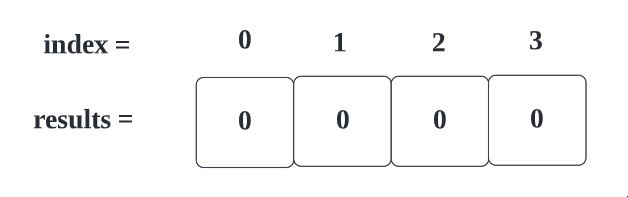


Here, we are moving from Right To Left, as if we find last element’s value **9**, then it will be **0** as 9+1 = 10.

So, with above logic, below result will be achieved **if all values are 9**.

1. New array will be created with total **digits.length + 1** size.

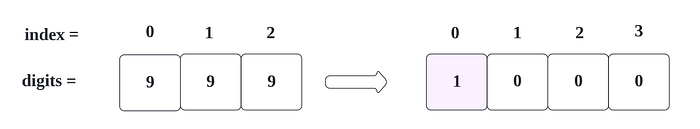
https://miro.medium.com/v2/resize:fit:363/1*PzhmnwZ4PWFdhtGYXvHuzw.png



2. We will assign 1, to our first index.

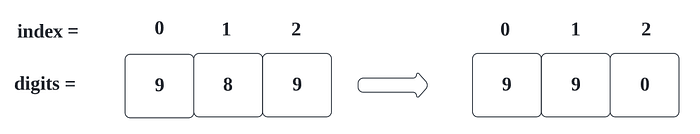
https://miro.medium.com/v2/resize:fit:131/1*A0xTSQqDSlc4DJdWv3La1A.png

So, result will be,

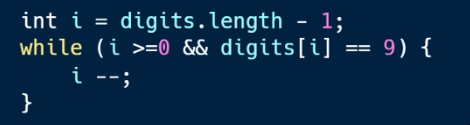


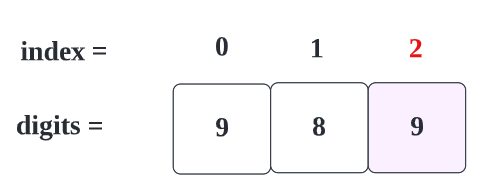
Now, moving forward to other conditions where array size is not getting incremented.

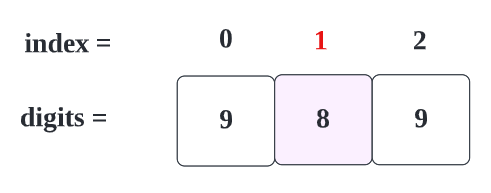
For ex,



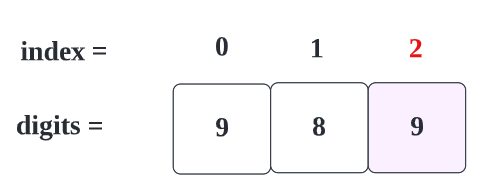
In this case also, we need to pass trough first condition where we are checking **9**.



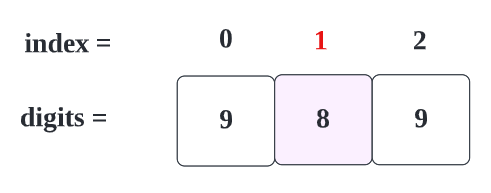




For the **first**iteration, **i** will be **2,**the condition is TRUE.

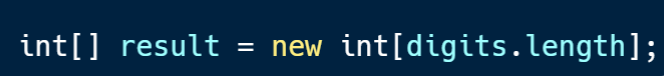


For the **second**iteration, **i** will be 1**,**the condition is FALSE.

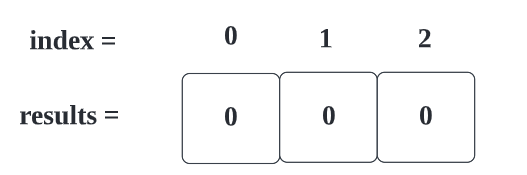


Keep note, **i** is now **1**.

Now, we are creating an new array with same size of digits. (we are creating new array to prevent un-necessary calculations.

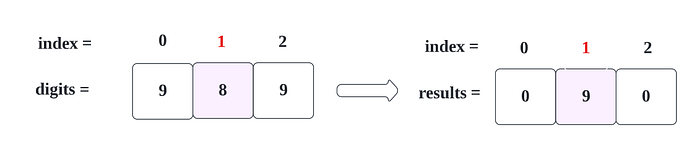


which is,

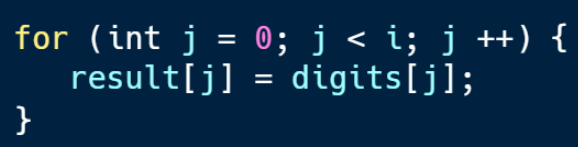


Now, we are incrementing **index i’s**value with 1.

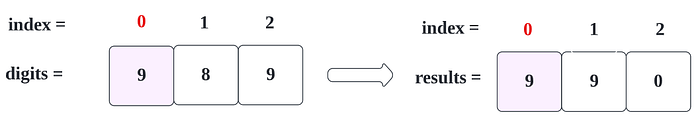
https://miro.medium.com/v2/resize:fit:515/1*BCGxn0Mit8w7j4gNMypcGA.png



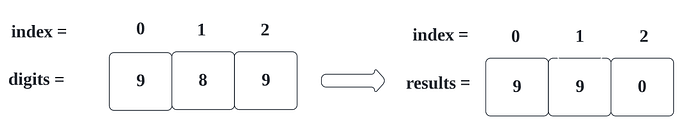
Now, other values we are storing as it is.



which is like below



Now, we got desired result,



Lets see whole code

**Code (Java): →** class Solution {

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

int i = digits.length - 1;

while (i >=0 && digits[i] == 9) {

i --;

}

if (i == -1) {

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

result[0] = 1;

return result;

}

int[] result = new int[digits.length];

result[i] = digits[i] + 1;

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

result[j] = digits[j];

}

return result;

}

}

**Time Complexity**

We are traversing an whole array (from right to left) until we find space, so time complexity will be **O(n)**.

**Space Complexity**

Since we have used an extra array with +1 more space, the space complexity will be ***O(n+1)***.

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.

## Ans:- Approach

From the given constraints, we can say one thing which is length of nums1 is greater than length of nums2.  
So one of the straight forward approach will be to store all the elements to an extra space like ArrayList, sort that ArrayList and then put back all the elements to nums1. In this case, we are actually applying sorting which makes \*\*NlogN \*\*complexity where N = m + n.  
But let us again recheck, do we actually need an extra space and do additional sorting as we already have 2 sorted array.  
More optimised way will be to use 3 variables say i, j, k where

i = m - 1

j = n - 1

k = m + n - 1

Now just have a traversal until i greater than or equal to 0 and j greater than or equal to 0 and do a check for the following :

-> if nums1[i] is greater than nums2[j], store nums1[i] to nums1[k] and decrease i and k.

-> else store nums2[j] to nums1[k] and decrease j and k.

There can be cases where there are some left out elements in nums2. So just make sure we cover all elements with an extra while loop until j ≥ 0 and store those elements to nums[k] as those will be the largest numbers that are remaining with us to be merged.

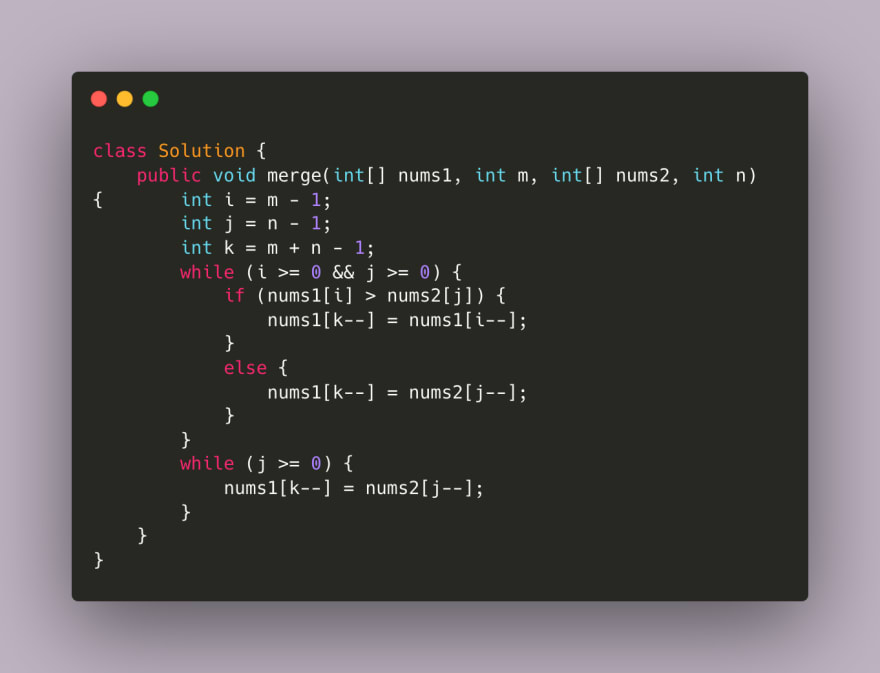
## Time and Space Complexity

Here we are traversing through the arrays of length m and n. So our time complexity will be **O(m + n)**.

We are not making use of any extra space. So our space complexity will be **O(1)**.

Please correct me if I made any mistake.

## Code



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

# Ans:- Approach

* First Sort the Array.
* Then Compare elements one by one.
* We Only have to return a boolean value.

**Complexity**

* Time complexity: O(nlog(n))
* Space complexity: O(log(n))

**Code**

class Solution {

public boolean containsDuplicate(int[] nums) {

Arrays.sort(nums);

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

if (nums[i] == nums[i+1]) {

return true;

}

}

return false;

}

}

**Complexity using a HashMap**

* Time complexity: O(n)
* Space complexity: O(n)

**Code**

class Solution {

public boolean containsDuplicate(int[] nums) {

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

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

if (map.containsKey(nums[i])) {

return true;

}

map.put(nums[i],1);

}

return false;

}

}

**Complexity using a HashSet**

* Time complexity: O(n)
* Space complexity: O(n)

**Code**

class Solution {

public boolean containsDuplicate(int[] nums) {

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

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

if (set.contains(nums[i])) {

return true;

}

set.add(nums[i]);

}

return false;

}

}

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]

Ans:- The goal of this solution is to move all the zeroes in the input array nums to the end of the array while maintaining the relative order of the non-zero elements

**Approach**

1. Initialize a variable m to -1, which will keep track of the position of the first zero encountered.
2. Traverse the array nums from left to right: a. If the current element is zero, and m is -1 or the element at position m is non-zero, set m to the current index. b. If the current element is non-zero and m is not -1, swap the current element with the element at position m and increment m.
3. After the traversal, all the zeroes will be at the end of the array, and the relative order of the non-zero elements will be maintained.

**Complexity**

* Time complexity:

O(n), where n is the length of the input array nums. The algorithm iterates through the array once

* Space complexity:

O(1), as the algorithm only uses a constant amount of extra space to store the variable m and the temporary variable temp used for swapping elements.

**Code**

class Solution {

public void moveZeroes(int[] nums) {

int m=-1;

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

{

if(nums[i]==0)

{

if(m==-1 || nums[m]!=0)

{

m=i;

}

}

else

{

if(m!=-1)

{

int temp = nums[i];

nums[i]=nums[m];

nums[m]=temp;

m++;

}

}

}

}

}

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]

Ans:- Since we are supposed to have all the numbers from 1 to n, we can make use of an extra array of sizes n. Let us initialize this array with “-1”.

We will solve the given problem statement in two steps. First, we will look for the duplicate number. We will traverse the array, take each number and place it in its respective index. If the number in that index is not -1, this means it’s a duplicate number and hence we will include it in our output array.

As the second step, we will traverse through the extra array, check for the index that has -1, this means it is missing. Hence we will append this value to our output array.

Let us look into the code snippet.

class ErrorNumsFinder:  
 def findErrorNums(self, nums: List[int]) -> List[int]:  
 numbers = [-1 for i in range(len(nums)+1)]  
 output = []  
 for num in nums:  
 if numbers[num] == -1:  
 numbers[num] = num  
 else:  
 output.append(num)  
   
 for i in range(1,len(numbers)):  
 if numbers[i] == -1:  
 output.append(i)  
 return output

**Complexity analysis.**

**Time Complexity**

Since we are traversing through the array twice the time complexity is O(2N) which is equal to O(N).

**Space Complexity**

Since we are an extra array to store our elements, the space complexity is O(N).

**Another solution:-**

**1.**

class Solution(object):

def findErrorNums(self, nums):

N = len(nums)

nset = set(nums)

missing = N \* (N + 1) // 2 - sum(nset)

duplicated = sum(nums) - sum(nset)

return [duplicated, missing]

2.

from collections import Counter

class Solution(object):

def findErrorNums(self, nums):

\_nums = Counter(nums)

duplicate = int([i for i in \_nums if \_nums[i] > 1][0])

\_diff = sum([i for i in range(len(nums)+1)]) - sum(nums)

#print ("duplicate = " + str(duplicate))

return [duplicate, duplicate+\_diff]

3.

from collections import Counter

class Solution(object):

def findErrorNums(self, nums):

cnt = Counter(nums)

\_len = len(nums)

\_nums = [x+1 for x in range(\_len)]

n\_sum = sum([x+1 for x in range(\_len)])

repeat = [x for x in list(cnt.keys()) if cnt[x] == 2][0]

miss = n\_sum - (sum(nums) - repeat)

# print ("repeat = " + str(repeat))

# print ("miss = " + str(miss))

return [repeat, miss]

3.

class Solution(object):

def findErrorNums(self, nums):

"""

:type nums: List[int]

:rtype: List[int]

"""

N = len(nums)

nset = set(nums)

missing = N \* (N + 1) // 2 - sum(nset)

duplicated = sum(nums) - sum(nset)

return [duplicated, missing]

### Test case :

s=Solution()

assert s.findErrorNums([1,2,2,4]) == [2,3]

assert s.findErrorNums([1,1,3,4]) == [1,2]

assert s.findErrorNums([]) == [0,0]

assert s.findErrorNums([1,2,3]) == [0,0]

1. HASH TABLE

class Solution(object):

def findErrorNums(self, nums):

"""

:type nums: List[int]

:rtype: List[int]

"""

hashs = [0] \* len(nums)

missing = -1

for i in range(len(nums)):

hashs[nums[i] - 1] += 1

return [hashs.index(2) + 1, hashs.index(0) + 1]

5.

class Solution(object):

def findErrorNums(self, nums):

return [sum(nums) - sum(set(nums)), sum(range(1, len(nums)+1)) - sum(set(nums))]

6.

class Solution:

def findErrorNums(self, nums):

x = sum(nums) - sum(set(nums))

y = sum(range(len(nums)+1))-(sum(nums)-x)

return [x,y]

7.

class Solution:

def findErrorNums(self, nums):

count = [0] \* (len(nums)+1)

for x in nums:

count[x] += 1

twice = z = 0

for x in range(1, len(nums)+1):

if count[x] == 2:

twice = x

if count[x] == 0:

z = x

return [twice, z]

# Time: O(n)

# Space: O(1)

8.

class Solution(object):

def findErrorNums(self, nums):

"""

:type nums: List[int]

:rtype: List[int]

"""

x\_xor\_y = 0

for i in range(len(nums)):

x\_xor\_y ^= nums[i] ^ (i+1)

bit = x\_xor\_y & ~(x\_xor\_y-1)

result = [0] \* 2

for i, num in enumerate(nums):

result[bool(num & bit)] ^= num

result[bool((i+1) & bit)] ^= i+1

if result[0] not in nums:

result[0], result[1] = result[1], result[0]

return result

# Time: O(n)

# Space: O(1)

9.

class Solution2(object):

def findErrorNums(self, nums):

"""

:type nums: List[int]

:rtype: List[int]

"""

result = [0] \* 2

for i in nums:

if nums[abs(i)-1] < 0:

result[0] = abs(i)

else:

nums[abs(i)-1] \*= -1

for i in range(len(nums)):

if nums[i] > 0:

result[1] = i+1

else:

nums[i] \*= -1

return result

# Time: O(n)

# Space: O(1)

10.

class Solution3(object):

def findErrorNums(self, nums):

"""

:type nums: List[int]

:rtype: List[int]

"""

N = len(nums)

x\_minus\_y = sum(nums) - N\*(N+1)//2

x\_plus\_y = (sum(x\*x for x in nums) - N\*(N+1)\*(2\*N+1)/6) // x\_minus\_y

return (x\_plus\_y+x\_minus\_y) // 2, (x\_plus\_y-x\_minus\_y) // 2