**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][

**Answer:** To solve this problem, you can use a hash table (dictionary in Python) to store the elements of the array as you iterate through it. As you process each element, you check if the complement (target minus the current element) is already present in the hash table. If it is, that means you have found the two numbers that add up to the target.

Here's the implementation in Python:

```python

def twoSum(nums, target):

# Create a dictionary to store the elements and their indices

num\_dict = {}

# Iterate through the array

for i, num in enumerate(nums):

# Calculate the complement

complement = target - num

# Check if the complement is in the dictionary

if complement in num\_dict:

# Return the indices of the two numbers

return [num\_dict[complement], i]

# Add the current number and its index to the dictionary

num\_dict[num] = i

# If no solution is found, return an empty list

return []

# Example usage

nums = [2, 7, 11, 15]

target = 9

result = twoSum(nums, target)

print(result) # Output: [0, 1]

```

In this implementation, we iterate through the `nums` array, and for each element, we calculate its complement (`target - num`) and check if it exists in the `num\_dict`. If it does, we return the indices of the two numbers that add up to the target. Otherwise, we add the current number and its index to the dictionary.

The time complexity of this solution is O(n) because we iterate through the array once, and the space complexity is also O(n) because in the worst case, we need to store all the elements of the array in the dictionary.

**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)[

**Answer:** To solve this problem, you can use the two-pointer approach. Initialize two pointers, `i` and `j`, initially pointing to the start of the `nums` array. Iterate through the array with the `j` pointer, and whenever you encounter a number that is not equal to `val`, update the value at the `i` pointer with that number and increment both `i` and `j`. This way, the `i` pointer always points to the next position where a non-val element should be placed.

Here's the implementation in Python:

```python

def removeElement(nums, val):

i = 0 # Pointer to track the position of non-val elements

for j in range(len(nums)):

if nums[j] != val:

nums[i] = nums[j]

i += 1

return i

# Example usage

nums = [3, 2, 2, 3]

val = 3

result = removeElement(nums, val)

print(result) # Output: 2

print(nums) # Output: [2, 2, \_, \_]

```

In this implementation, we maintain two pointers, `i` and `j`. The `i` pointer tracks the position where non-val elements should be placed, and the `j` pointer iterates through the array. Whenever we encounter a number that is not equal to `val`, we update the value at the `i` pointer with that number and increment both pointers. Finally, we return the value of `i`, which represents the number of elements in `nums` that are not equal to `val`.

The time complexity of this solution is O(n) because we iterate through the `nums` array once. The space complexity is O(1) because we modify the array in-place without using any additional data structures.

**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

**Answer:** def searchInsert(nums, target):

left = 0

right = len(nums) - 1

while left <= right:

mid = left + (right - left) // 2

if nums[mid] == target:

return mid

elif nums[mid] < target:

left = mid + 1

else:

right = mid - 1

return left

# Example usage

nums = [1, 3, 5, 6]

target = 5

result = searchInsert(nums, target)

print(result) # Output: 2

**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].

**Answer:**

def plusOne(digits):

carry = 1

for i in range(len(digits) - 1, -1, -1):

digits[i] += carry

if digits[i] == 10:

digits[i] = 0

carry = 1

else:

carry = 0

if carry == 1:

digits.insert(0, 1)

return digits

# Example usage

digits = [1, 2, 3]

result = plusOne(digits)

print(result) # Output: [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

**Answer:**

def merge(nums1, m, nums2, n):

i = m - 1 # Pointer for nums1

j = n - 1 # Pointer for nums2

k = m + n - 1 # Pointer for merged array

while i >= 0 and j >= 0:

if nums1[i] > nums2[j]:

nums1[k] = nums1[i]

i -= 1

else:

nums1[k] = nums2[j]

j -= 1

k -= 1

# If there are any remaining elements in nums2, copy them to nums1

while j >= 0:

nums1[k] = nums2[j]

j -= 1

k -= 1

# Example usage

nums1 = [1, 2, 3, 0, 0, 0]

m = 3

nums2 = [2, 5, 6]

n = 3

merge(nums1, m, nums2, n)

print(nums1) # 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:

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

Output: true

**Answer:**

def containsDuplicate(nums):

num\_set = set()

for num in nums:

if num in num\_set:

return True

num\_set.add(num)

return False

# Example usage

nums = [1, 2, 3, 1]

result = containsDuplicate(nums)

print(result) # 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]

**Answer:**

def moveZeroes(nums):

left = 0

right = 0

while right < len(nums):

if nums[right] != 0:

nums[left], nums[right] = nums[right], nums[left]

left += 1

right += 1

while left < len(nums):

nums[left] = 0

left += 1

# Example usage

nums = [0, 1, 0, 3, 12]

moveZeroes(nums)

print(nums) # Output: [1, 3, 12, 0, 0]

**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]

**Answer:**

def findErrorNums(nums):

numSet = set()

duplicate = 0

missing = 0

for num in nums:

if num in numSet:

duplicate = num

else:

numSet.add(num)

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

if i not in numSet:

missing = i

return [duplicate, missing]

# Example usage

nums = [1, 2, 2, 4]

result = findErrorNums(nums)

print(result) # Output: [2, 3]