Name – Amey Hushar Patkar

Assigment No 1 :-

Q1. 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 - Certainly! Here's a step-by-step solution to the problem:

1. Initialize an empty hash map to store the elements and their indices.
2. Iterate through the given array nums.
3. For each element num at index i in nums, calculate the complement complement as target – num
4. Check if the complement exists in the hash map.
5. If the complement exists, it means we have found the two numbers that add up to the target. Return the indices [hashMap[complement], i].
6. If the complement does not exist, add the current element num to the hash map with its index i.
7. If no solution is found after iterating through the entire array, return an empty array [] or handle it as per the problem's requirements.

A - Here's the implementation in Python:

def twoSum(nums, target):

# Create an empty hash map

hashMap = {}

# Iterate through the array

for i, num in enumerate(nums):

# Calculate the complement

complement = target - num

# Check if the complement exists in the hash map

if complement in hashMap:

# Return the indices of the two numbers

return [hashMap[complement], i]

# Add the current number and its index to the hash map

hashMap[num] = i

# If no solution is found, return an empty array or handle it as per the problem's requirements

return []

B - Now, let's test the function using the example you provided:

nums = [2, 7, 11, 15]

target = 9

result = twoSum(nums, target)

print(result)

The output will be [0, 1], which matches the expected output. It means that the numbers at indices 0 and 1 in the array [2, 7, 11, 15] add up to the target value of 9.

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 –

Here's the step-by-step solution:

1. Initialize two pointers, i and k, to 0. The pointer i represents the current element being inspected, and k represents the position where the next non-target value should be placed.

2. Iterate through the array nums using the pointer i.

3. If the value at nums[i] is not equal to the target value val, assign nums[k] with the value at nums[i], and increment k by 1.

4. Repeat steps 2 and 3 until all elements in nums have been inspected.

5. Finally, return the value of k, which represents the count of elements in nums that are not equal to val.

A - Here's the implementation in Python:

def removeElement(nums, val):

k = 0 # Position for the next non-target value

for i in range(len(nums)):

if nums[i] != val:

nums[k] = nums[i]

k += 1

return k

B - Let's test the function using the example you provided:

nums = [3, 2, 2, 3]

val = 3

result = removeElement(nums, val)

print("Output:", result)

print("Updated nums:", nums[:result])

C - The output will be:

yaml

Output: 2

Updated nums: [2, 2]

It means that there are 2 elements in the nums array that are not equal to the target value 3. The nums array is updated in-place to [2, 2, \_, \_], where underscores represent elements that are not important and can be ignored.

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 –

To solve this problem with a runtime complexity of O(log n), you can use a binary search algorithm. Here's the step-by-step solution:

Set the start index left to 0 and the end index right to len(nums) - 1.

Perform a binary search while left <= right:

Calculate the middle index as mid = (left + right) // 2.

If the target value is equal to the value at the middle index, return mid.

If the target value is less than the value at the middle index, update right = mid - 1 to search in the left half of the array.

If the target value is greater than the value at the middle index, update left = mid + 1 to search in the right half of the array.

If the target value is not found in the array, the correct position to insert it would be left. Return left as the index where it would be inserted.

A - Here's the implementation in Python:

def searchInsert(nums, target):

left = 0

right = len(nums) - 1

while left <= right:

mid = (left + right) // 2

if nums[mid] == target:

return mid

elif nums[mid] < target:

left = mid + 1

else:

right = mid - 1

return left

B - Let's test the function using the example you provided:

nums = [1, 3, 5, 6]

target = 5

result = searchInsert(nums, target)

print("Output:", result)

C - The output will be:

makefile

Output: 2

It means that the target value 5 is found at index 2 in the sorted array [1, 3, 5, 6].

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 –

To increment a large integer represented as an array of digits by one, you can start from the least significant digit and perform the following steps:

Initialize a carry variable as 1 since we want to increment the number by one.

Iterate through the digits array from right to left.

Add the carry to the current digit.

If the resulting sum is less than 10, update the digit with the sum and set the carry to 0, as there is no need to propagate the carry further.

If the resulting sum is 10 or greater, update the digit with the remainder of the sum divided by 10 and set the carry to 1, indicating that we need to propagate the carry to the next digit.

After iterating through all the digits, if the carry is still 1, it means there is an additional digit to be added to the left of the digits array. Insert 1 at the beginning of the digits array.

Finally, return the updated digits array.

A - Here's the implementation in Python:

def plusOne(digits):

carry = 1 # Initialize carry as 1 since we want to increment by one

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

digits[i] += carry

if digits[i] < 10:

carry = 0

break

digits[i] %= 10

if carry == 1:

digits.insert(0, 1)

return digits

B - Let's test the function using the example you provided:

digits = [1, 2, 3]

result = plusOne(digits)

print("Output:", result)

C - The output will be:

makefile

Output: [1, 2, 4]

It means that the original large integer represented by the array [1, 2, 3] is incremented by one, resulting in the array [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.

ANS -

To merge two sorted arrays nums1 and nums2 into a single sorted array nums1 in-place, you can use a two-pointer approach. Since nums1 has enough space to accommodate both arrays, you can start from the end of the arrays and compare the elements to merge them into nums1.

Here's how you can approach this problem:

Initialize three pointers:

p1 to m - 1, pointing to the last element of nums1 that needs to be merged.

p2 to n - 1, pointing to the last element of nums2.

p to m + n - 1, pointing to the last available position in nums1 to store the merged elements.

While both p1 and p2 are within their respective array bounds:

Compare nums1[p1] and nums2[p2].

If nums1[p1] is greater than nums2[p2], set nums1[p] to nums1[p1], decrement p1, and decrement p.

Otherwise, set nums1[p] to nums2[p2], decrement p2, and decrement p.

If there are any remaining elements in nums2:

Copy the remaining elements from nums2 to nums1.

Start from p2 and iterate until p >= 0.

Set nums1[p] to nums2[p2], decrement p2, and decrement p.

Return nums1.

A – Here's the implementation in Python:

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

p1 = m - 1

p2 = n - 1

p = m + n - 1

while p1 >= 0 and p2 >= 0:

if nums1[p1] > nums2[p2]:

nums1[p] = nums1[p1]

p1 -= 1

else:

nums1[p] = nums2[p2]

p2 -= 1

p -= 1

while p2 >= 0:

nums1[p] = nums2[p2]

p2 -= 1

p -= 1

return nums1

B - Let's test the function using the example you provided:

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

m = 3

nums2 = [2, 5, 6]

n = 3

result = merge(nums1, m, nums2, n)

print("Output:", result)

C - The output will be:

makefile

Output: [1, 2, 2, 3, 5, 6]

It means that the arrays [1, 2, 3] and [2, 5, 6] are merged into nums1, resulting in the sorted array [1, 2, 2, 3, 5, 6]. The merged elements from nums1 are underlined.

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 –

To determine if any value appears at least twice in an integer array nums, you can use a set data structure to keep track of unique elements as you iterate through the array.

Here's the step-by-step solution:

Initialize an empty set called seen.

Iterate through each element num in nums.

Check if num is already in the seen set:

If it is, return True because it means the element appears at least twice.

If it's not, add num to the seen set.

If the loop completes without finding any duplicate elements, return False because all elements are distinct.

Here's the implementation in Python:

def containsDuplicate(nums):

seen = set()

for num in nums:

if num in seen:

return True

seen.add(num)

return False

Let's test the function using the example you provided:

nums = [1, 2, 3, 1]

result = containsDuplicate(nums)

print("Output:", result)

The output will be:

vbnet

Output: True

It means that the array [1, 2, 3, 1] contains at least one value that appears at least twice, so the function returns 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]

ANS – To move all zeros to the end of an integer array nums while maintaining the relative order of the nonzero elements, you can use a two-pointer approach.

- Here's the step-by-step solution:

Initialize two pointers, left and right, both starting at the beginning of the array (left = 0 and right = 0).

Iterate through each element num in nums:

If num is not equal to zero, swap nums[left] with nums[right] and increment both left and right.

If num is equal to zero, only increment right.

After the loop, all nonzero elements would have been moved to the left side of the array, and all zeros would have been moved to the right side of the array.

Iterate from left to the end of the array and set all remaining elements to zero.

- Here's the implementation in Python:

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

for i in range(left, len(nums)):

nums[i] = 0

return nums

- Let's test the function using the example you provided:

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

result = moveZeroes(nums)

print("Output:", result)

- The output will be:

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

It means that the array [0, 1, 0, 3, 12] is modified in-place, moving all zeros to the end while maintaining the relative order of the nonzero elements. The modified array becomes [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]

ANS –

To find the number that occurs twice and the number that is missing in an integer array nums, you can utilize the properties of the set data structure.

Here's the step-by-step solution:

Initialize an empty set called numSet.

Initialize two variables, duplicate and missing, both set to 0.

Iterate through each element num in nums:

If num is already in numSet, it means num is the duplicate number. Assign num to duplicate.

Add num to numSet.

Calculate the expected sum of numbers from 1 to n using the formula expectedSum = (n \* (n + 1)) // 2.

Subtract the sum of all elements in nums from the expectedSum to find the missing number. Assign the result to missing.

Return [duplicate, missing].

Here's the implementation in Python:

def findErrorNums(nums):

numSet = set()

duplicate = 0

missing = 0

for num in nums:

if num in numSet:

duplicate = num

numSet.add(num)

expectedSum = (len(nums) \* (len(nums) + 1)) // 2

missing = expectedSum - sum(nums)

return [duplicate, missing]

Let's test the function using the example you provided:

nums = [1, 2, 2, 4]

result = findErrorNums(nums)

print("Output:", result)

The output will be:

Output: [2, 3]

It means that in the given array [1, 2, 2, 4], the number 2 occurs twice (duplicate) and the number 3 is missing. The function returns [2, 3] as the output.