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]

`Approach` :

- Create an empty hashmap to store the elements and their indices.
- Iterate through the array, using a loop with an index variable i.
- For each element at index i, calculate the complement as target - nums[i].
- Check if the complement exists in the hashmap:
- If it does, return the indices [hashmap[complement], i].
- If it doesn't, continue to the next element.
- If no solution is found after iterating through the entire array, return an empty array.

Time complexity: `O(n)` --> We will iterate thru' the array (len of `n`) only once.
Space Complexity: `O(n)` --> Need to store all the elements of the array into the hashmap!

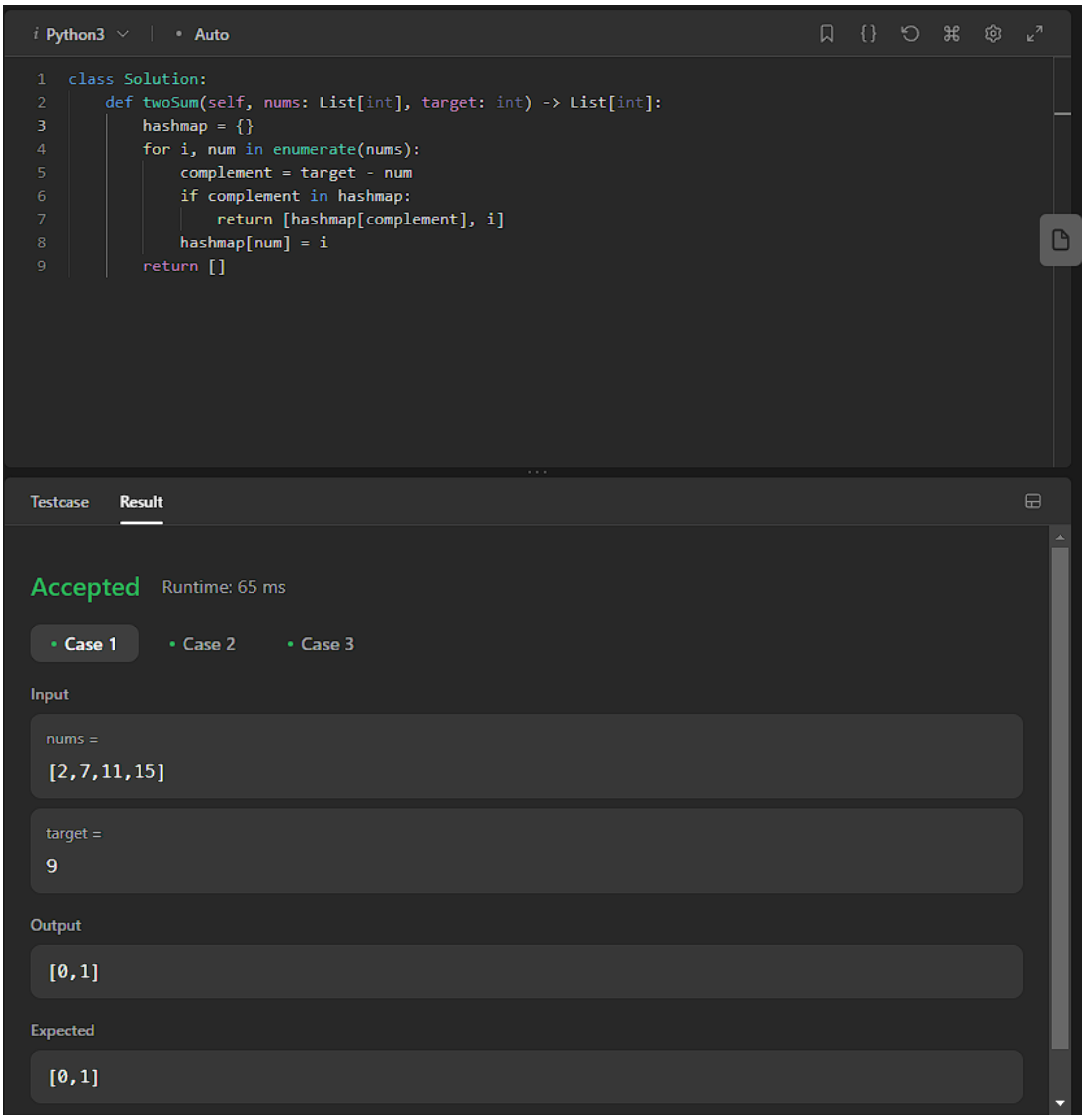
In [ ]:
```python
def twoSum(nums, target):
    # Create a hashmap to store the elements and their indices
    hashmap = {}

    # Iterate through the array
    for i, num in enumerate(nums):
        # Calculate the complement
        complement = target - num

        # Check if the complement is in the hashmap
        if complement in hashmap:
            # Return the indices of the two numbers
            return [hashmap[complement], i]

        # Add the current element to the hashmap
        hashmap[num] = i

    # If no solution is found, return an empty array
    return []
```



```python
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        hashmap = {}
        for i, num in enumerate(nums):
            complement = target - num
            if complement in hashmap:
                return [hashmap[complement], i]
            hashmap[num] = i
        return []
```

Accepted Runtime: 65 ms

Input

nums =

```
[2,7,11,15]
```

target =

```
9
```

Output

```
[0,1]
```

Expected

```
[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,*,*]

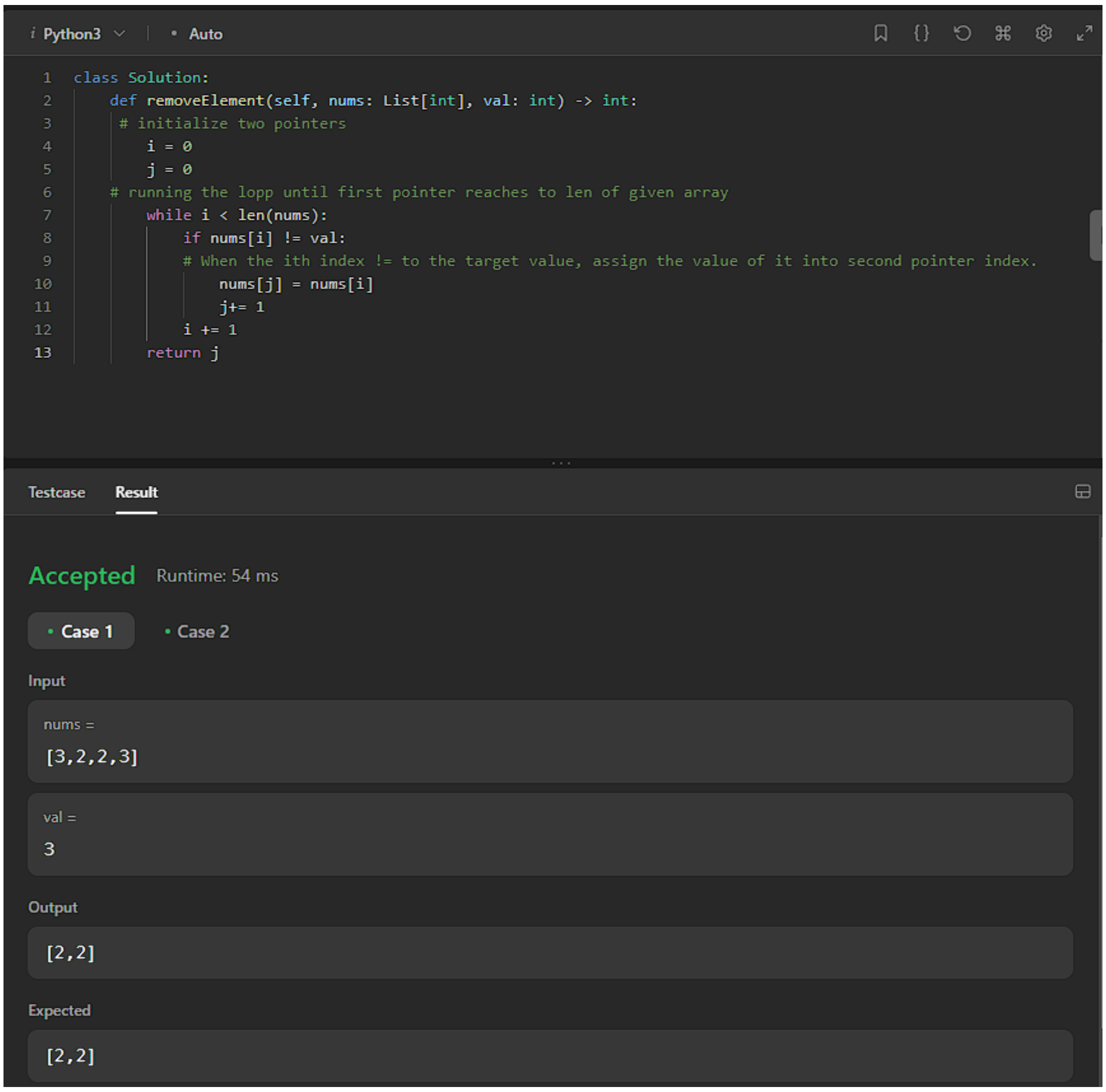
Explanation: Your function should return k = 2, with the first two elements of nums being 2. It does not matter what you leave beyond the returned k (hence they are underscores)

`Approach` :

- Initialize two pointers i and j to 0.
- Iterate through the array nums using the pointer i:
- If the current element nums[i] is not equal to the target value val:
    - Set nums[j] to nums[i].
        - Increment j by 1.
        - Increment i by 1.
- After iterating through the array, j will be the count of elements that are not equal to val.
- Return j as the result.

Time complexity: `O(n)` --> We will iterate thru' the array (len of `n`) only once.
Space Complexity: `O(1)` --> we are modifying the input array in-place

In [ ]:
```python
def removeElement(nums, val):
    # initialize two pointers
    i = 0
    j = 0
    # running the lopp until first pointer reaches to len of given array
    while i < len(nums):
        if nums[i] != val:
            # When the ith index != to the target value, assign the value of it
            nums[j] = nums[i]
            j += 1
        i += 1
    return j
```



```python
class Solution:
    def removeElement(self, nums: List[int], val: int) -> int:
     # initialize two pointers
        i = 0
        j = 0
    # running the lopp until first pointer reaches to len of given array
        while i < len(nums):
            if nums[i] != val:
            # When the ith index != to the target value, assign the value of it into second pointer index.
                nums[j] = nums[i]
                j+= 1
            i += 1
        return j
```

Accepted Runtime: 54 ms

Input

nums =

[3,2,2,3]

val =

3

Output

[2,2]

Expected

[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

`Approach` :

- Initialize two pointers, left and right, pointing to the start and end of the array, respectively.
- While left is less than or equal to right:

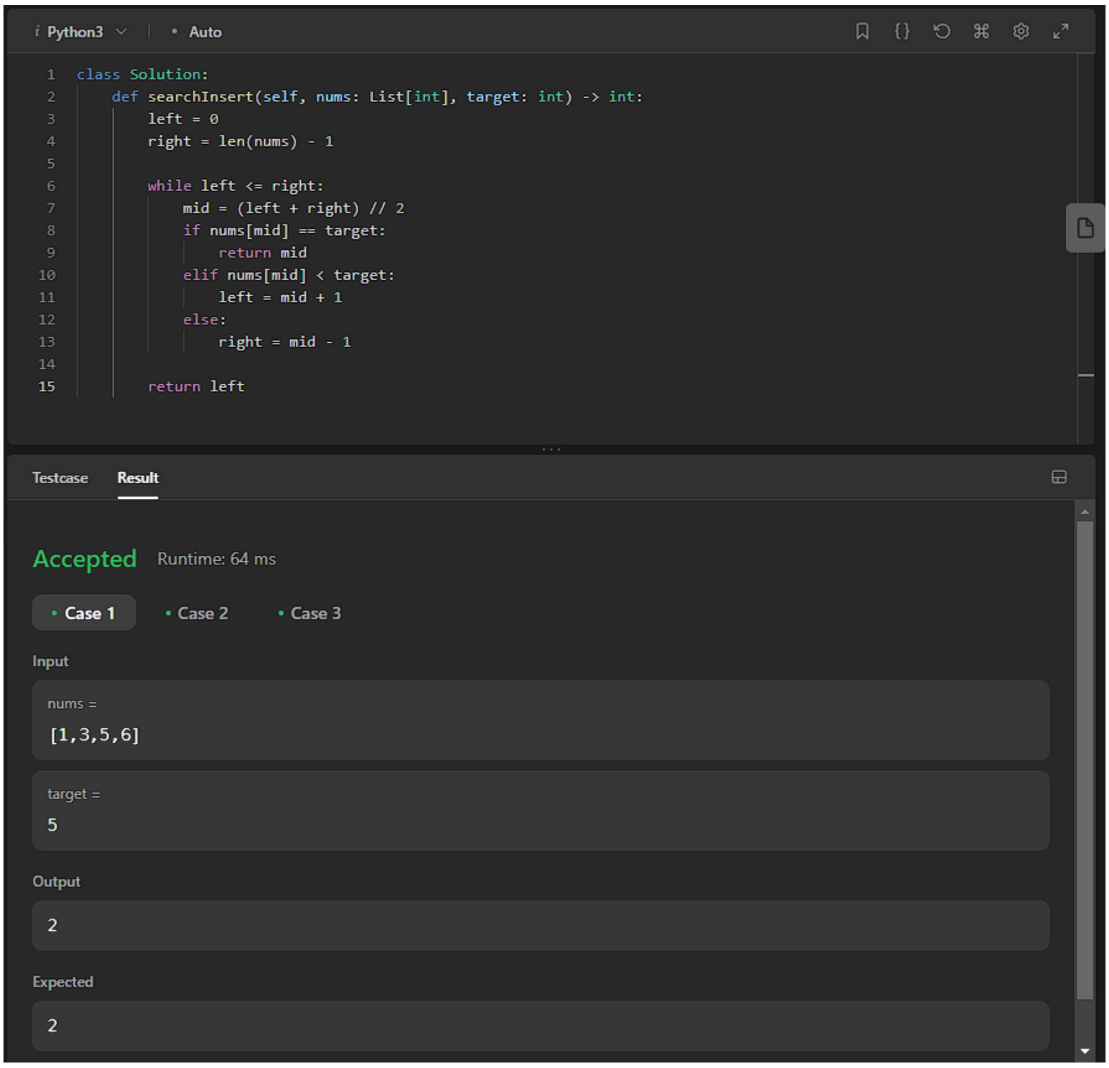
- Calculate the middle index as mid = (left + right) // 2.
- Compare the target value with the middle element nums[mid].
  - If they are equal, return mid as the index of the target value.
  - If the target value is less than nums[mid], update right = mid - 1 to search in the left half.
  - If the target value is greater than nums[mid], update left = mid + 1 to search in the right half.
- If the loop ends without finding the target value, return left as the index where it would be inserted.

Time complexity: `O(n)` --> We will iterate thru' the array (len of `n`) only once.
Space Complexity: `O(1)` --> It uses a constant amount of additional space for the pointers left, right, and mid.

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
```



```python
class Solution:
    def searchInsert(self, nums: List[int], target: int) -> int:
        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
```

```
Accepted   Runtime: 64 ms
Case 1   Case 2   Case 3
Input
nums =
[1,3,5,6]
target =
5
Output
2
Expected
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].

`Approach`:

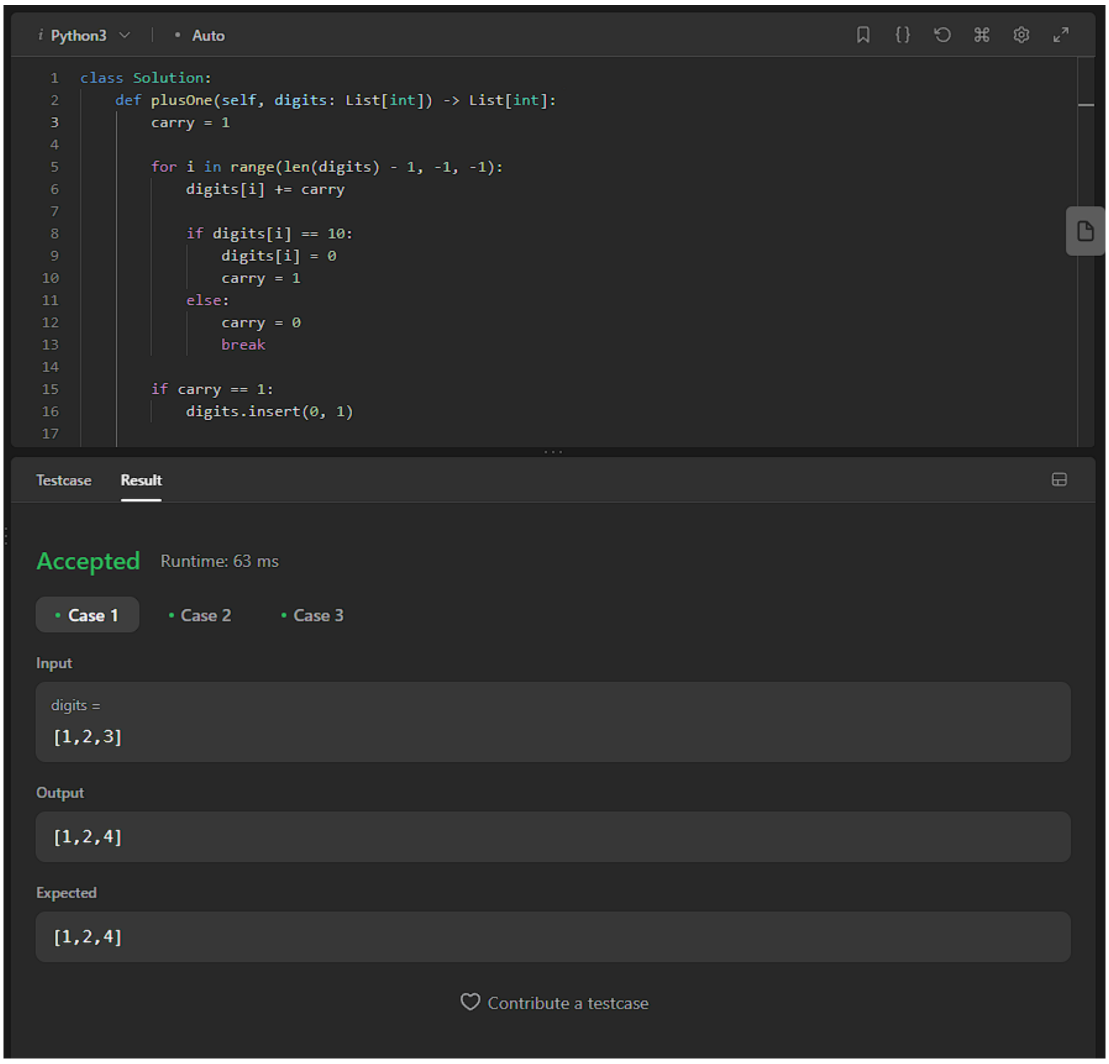
- Initialize a carry variable to 1.

- Iterate through the digits from right to left:
  - Add the carry to the current digit.
    - If the digit becomes 10:
    - Set the digit to 0.
  - Update the carry to 1.
- Otherwise, break out of the loop.
- If the carry is still 1 after the loop, it means we need to add an additional digit at the beginning of the array. Append 1 to the front of the digits array.
- Return the updated digits array.

Time complexity: `O(n)` --> We will iterate thru' the array (len of `n`) only once.
Space Complexity: `O(1)` --> It uses a constant amount of additional space for the pointers left, right, and mid.

In [ ]:
```python
def plusOne(digits):
    # initialize the carry to 1
    c = 1
    for i in range(len(digits) - 1, -1, -1):
        digits[i] += c

        if digits[i] == 10:
            digits[i] = 0
            c = 1
        else:
            c = 0
            break
    if c == 1:
        digits.insert(0, 1)
    return digits
```



```python
class Solution:
    def plusOne(self, digits: List[int]) -> List[int]:
        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
                break

        if carry == 1:
            digits.insert(0, 1)

```

Accepted Runtime: 63 ms

Input

```
digits =
[1,2,3]
```

Output

```
[1,2,4]
```

Expected

```
[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]

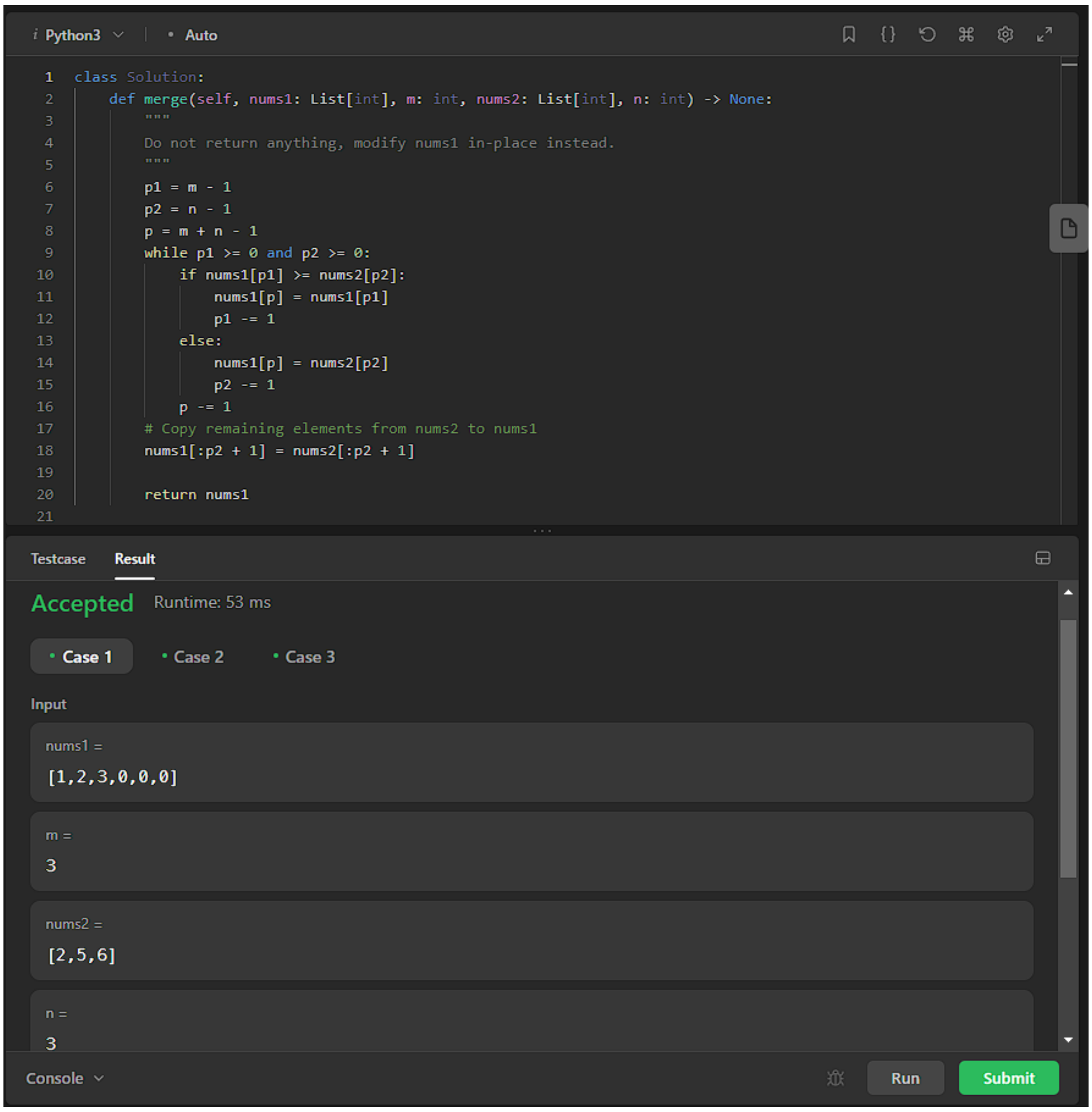
Explanation: The arrays we are merging are [1,2,3] and [2,5,6]. The result of the merge is [1,2,2,3,5,6] with the underlined elements coming from nums1.

`Approach` :

- Initialize a carry variable to 1.
- Iterate through the digits from right to left:
    - Add the carry to the current digit.
        - If the digit becomes 10:
        - Set the digit to 0.
    - Update the carry to 1.
- Otherwise, break out of the loop.
- If the carry is still 1 after the loop, it means we need to add an additional digit at the beginning of the array. Append 1 to the front of the digits array.
- Return the updated digits array.

Time complexity: `O(n)` --> We will iterate thru' the array (len of `n`) only once.
Space Complexity: `O(1)` --> It uses a constant amount of additional space for the pointers left, right, and mid.

In [ ]:
```
def merge(n1, n2, a, b):
    x1 = a - 1
    x2 = b - 1
    pointer = a + b - 1
    while x1 >= 0 and x2 >= 0:
        if n1[x1] >= n2[x2]:
            n1[pointer] = n1[x1]
            x1 -= 1
        else:
            n1[pointer] = n2[x2]
            x2 -= 1
        pointer -= 1
    # Copy remaining elements from 2nd arr to 1st arr.
    n1[:x2 + 1] = n2[:x2 + 1]
    return n1
```



```python
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        """
        Do not return anything, modify nums1 in-place instead.
        """
        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
        # Copy remaining elements from nums2 to nums1
        nums1[:p2 + 1] = nums2[:p2 + 1]

        return nums1

```

Accepted Runtime: 53 ms

Input

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

m =
3

nums2 =
[2,5,6]

n =
3

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

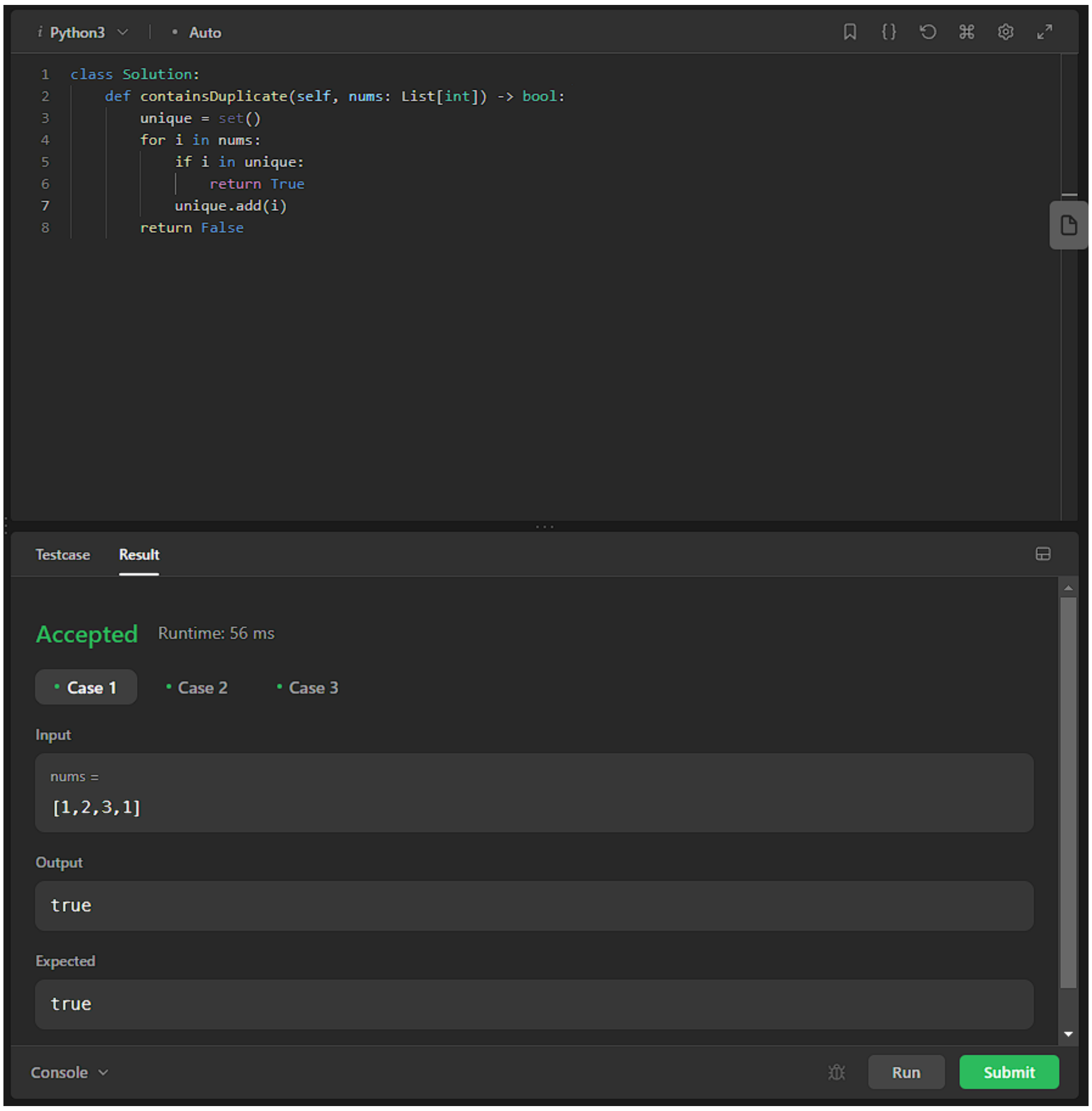
`Approache` :

- Initialize an empty set.
- Iterate through each element num in the nums array.
- If num is already in the set, return True as a duplicate value is found.
- Otherwise, add num to the set.

- If the loop ends without finding any duplicates, return False.

Time complexity: `O(n)` --> We will iterate thru' the array (len of `n`) only once.
Space Complexity: `O(1)` --> Due to the hashing-based implementation of sets in Python

In [ ]:
```python
def containsDuplicate(nums):
    unique = set()
    for i in nums:
        if i in unique:
            return True
        unique.add(i)
    return False
```



```python
class Solution:
    def containsDuplicate(self, nums: List[int]) -> bool:
        unique = set()
        for i in nums:
            if i in unique:
                return True
            unique.add(i)
        return False
```

Accepted Runtime: 56 ms

Case 1 Case 2 Case 3

Input

nums =
```
[1,2,3,1]
```

Output
```
true
```

Expected
```
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]

`Approache` :

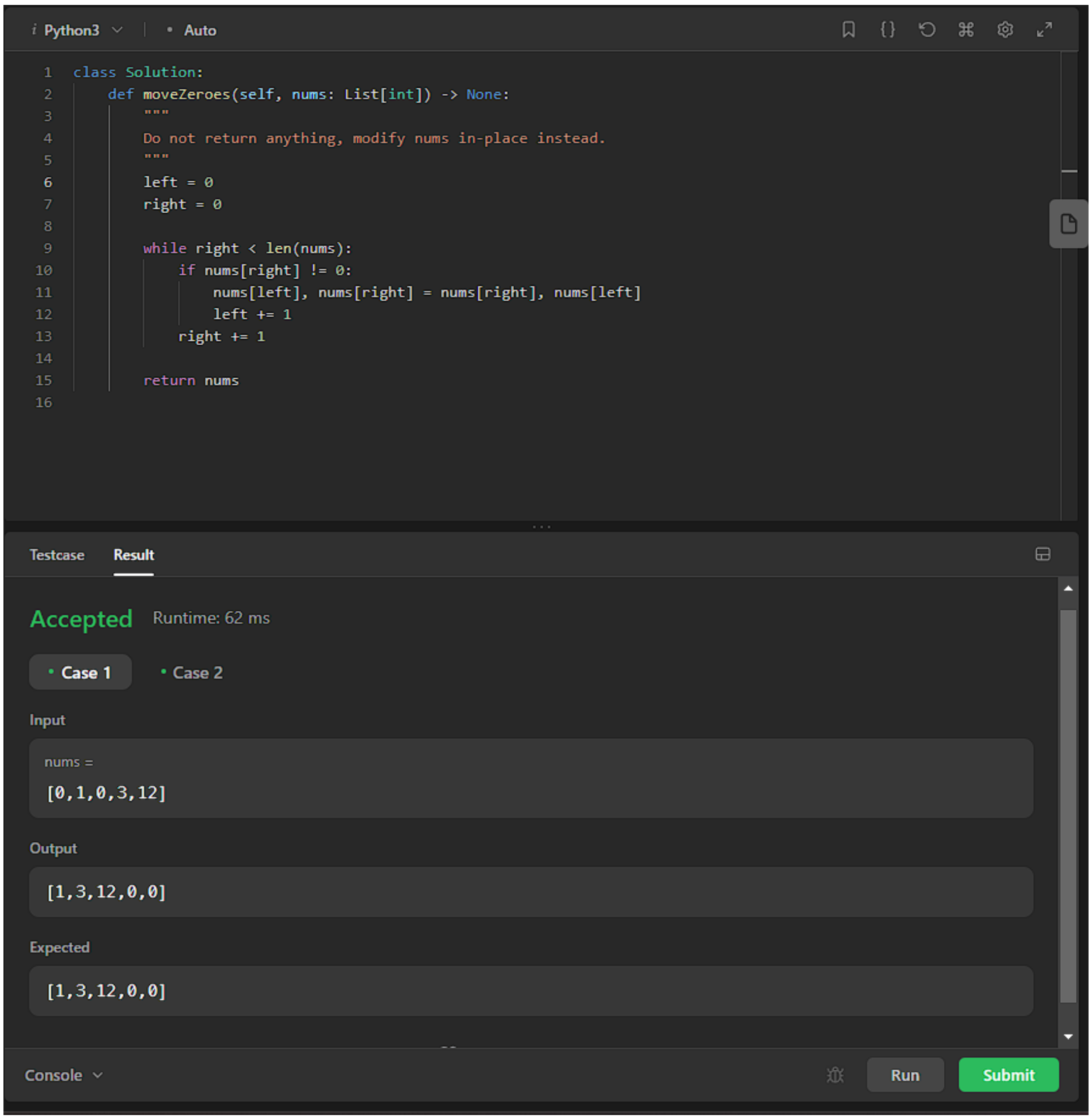
- Initialize two pointers, left and right, both initially pointing to the start of the array.
- Iterate through the array using the right pointer:
- If the element at nums[right] is non-zero:
- Swap the element at nums[left] with the element at nums[right].
- Increment both left and right pointers.
- After the iteration, all non-zero elements are now placed towards the left side of the array, with all zeros towards the right side.
- Return the modified nums array

Time complexity: `O(n)` --> We will iterate thru' the array (len of `n`) only once.
Space Complexity: `O(1)` --> This algorithms perform the operations in-place.

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

    return nums
```



```python
class Solution:
    def moveZeroes(self, nums: List[int]) -> None:
        """
        Do not return anything, modify nums in-place instead.
        """
        left = 0
        right = 0

        while right < len(nums):
            if nums[right] != 0:
                nums[left], nums[right] = nums[right], nums[left]
                left += 1
            right += 1

        return nums
```

Accepted Runtime: 62 ms

Input

nums =

```
[0,1,0,3,12]
```

Output

```
[1,3,12,0,0]
```

Expected

```
[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]

`Approache` :

- Calculate the sum of all numbers in nums using sum(nums).
- Calculate the sum of all unique numbers in nums using sum(set(nums)). This sum will exclude the duplicate number and only include unique numbers.
- The difference between the two sums, sum(nums) - sum(set(nums)), gives the value of the duplicate number, which is assigned to toRemove.
- Calculate the sum of all numbers from 1 to len(nums) using sum(range(1, len(nums)+1)).
- Calculate the sum of all unique numbers in nums using sum(set(nums)). This sum will exclude the duplicate number and only include unique numbers.
- The difference between the two sums, sum(range(1, len(nums)+1)) - sum(set(nums)), gives the value of the missing number, which is assigned to actualMissing.
- Return the list [toRemove, actualMissing] as the result.

Time complexity: `O(n)` --> As Calculating the sum of nums and unique nums will be O(n)+O(n) => O(n).
Space Complexity: `O(1)` --> Set(nums) creates a set that can potentially contain all the unique numbers in nums, resulting in a space complexity of O(n).

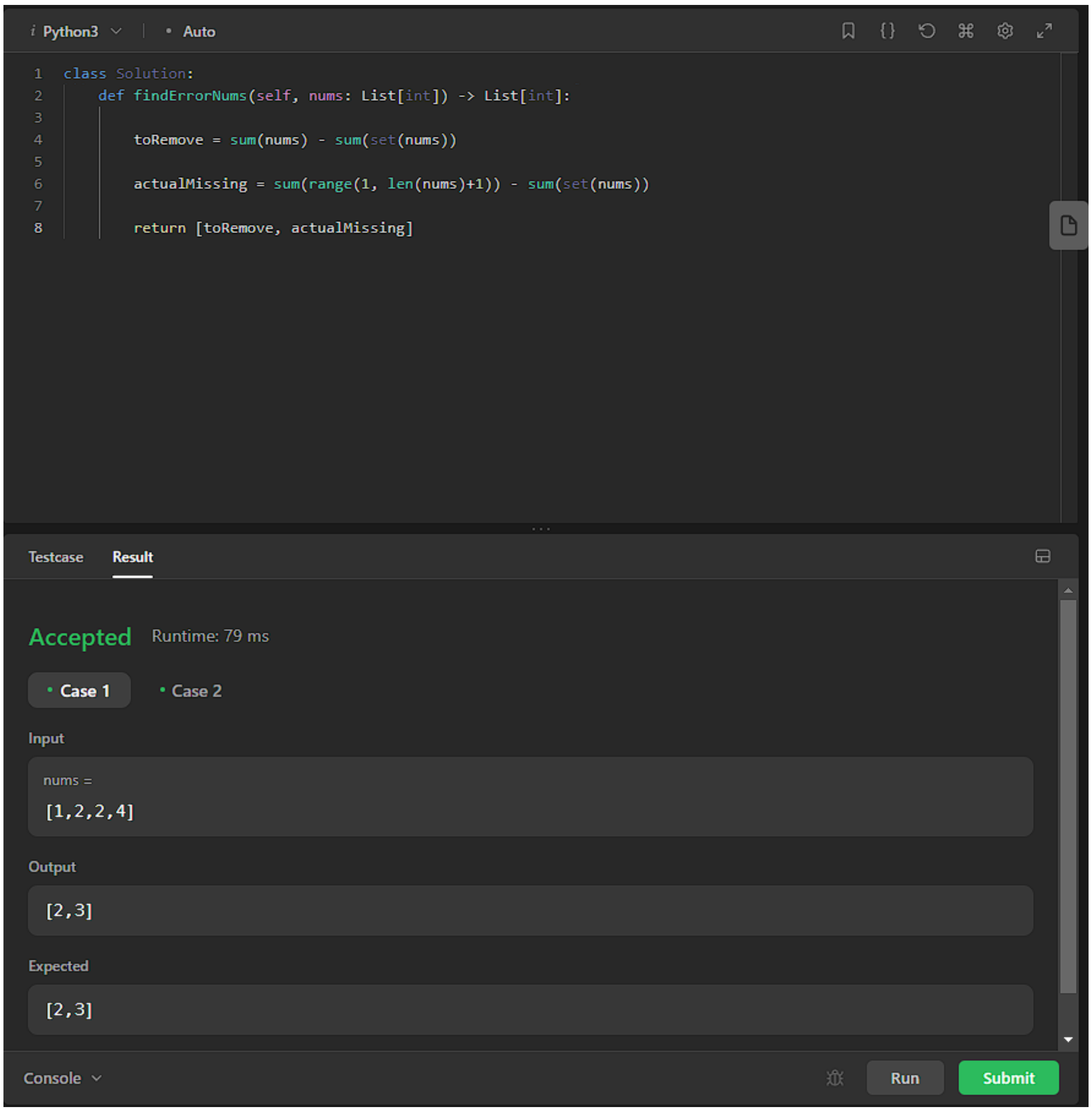
In [ ]:
```python
def findErrorNums(nums):
    xor = 0
    for i, num in enumerate(nums):
        xor ^= num ^ (i + 1)

    # finding the least significant bit
    rightmostSetBit = xor & -xor

    missing = 0
    duplicate = 0
    for num in nums:
        if num & rightmostSetBit:
            # XOR operation on num and initialize it with duplicate variable
            duplicate ^= num
        else:
            missing ^= num

    for i in range(1, len(nums) + 1):
        if i & rightmostSetBit:
            duplicate ^= i
        else:
            missing ^= i

    return sorted([duplicate, missing])
```



```python
class Solution:
    def findErrorNums(self, nums: List[int]) -> List[int]:

        toRemove = sum(nums) - sum(set(nums))

        actualMissing = sum(range(1, len(nums)+1)) - sum(set(nums))

        return [toRemove, actualMissing]
```

Accepted Runtime: 79 ms

Case 1 Case 2

Input

nums =

```
[1,2,2,4]
```

Output

```
[2,3]
```

Expected

```
[2,3]
```