Experiment 4

Student Name: Nitil Jakhar
UID: 22BCS17300
Branch: CSE
Section/Group: NTPP_IOT-602/A
Semester: 6th
Date of Performance: 3/02/2
Subject: AP 2

1. **Aim: Sorting and Searching**
2. **Objective:**
   1. Merge Sorted Array
   2. Search in Rotated Sorted Array
3. **Code:**

1. **Merge Sorted Array:**

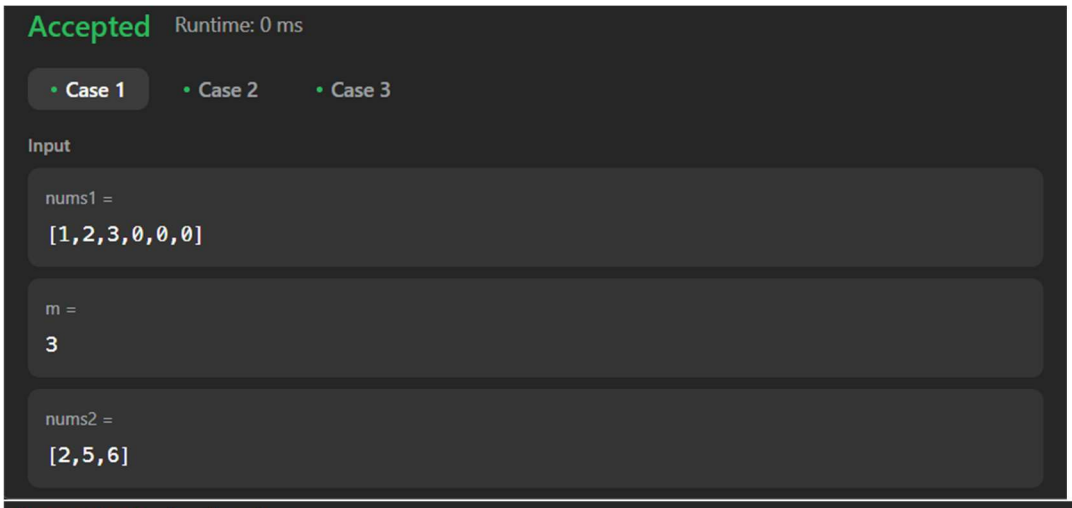
```
class Solution:
    def merge(self, nums1: List[int], m: int, nums2: List[int], n: int) -> None:
        # Start from the end of both nums1 and nums2
        i, j, k = m - 1, n - 1, m + n - 1

        # Compare elements from nums1 and nums2 and place the larger one at the
end of nums1
        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
```

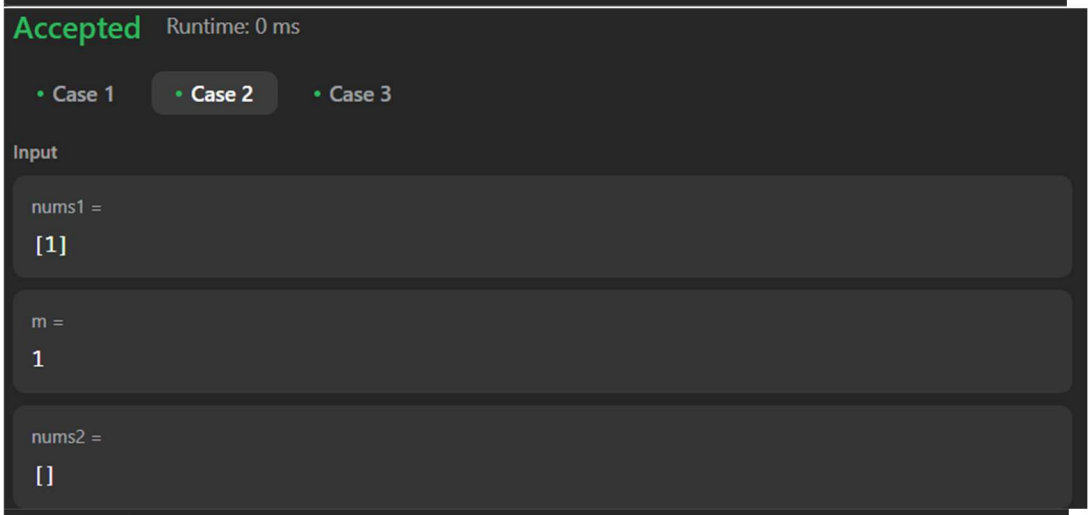
2. **Search in Rotated Sorted Array:**

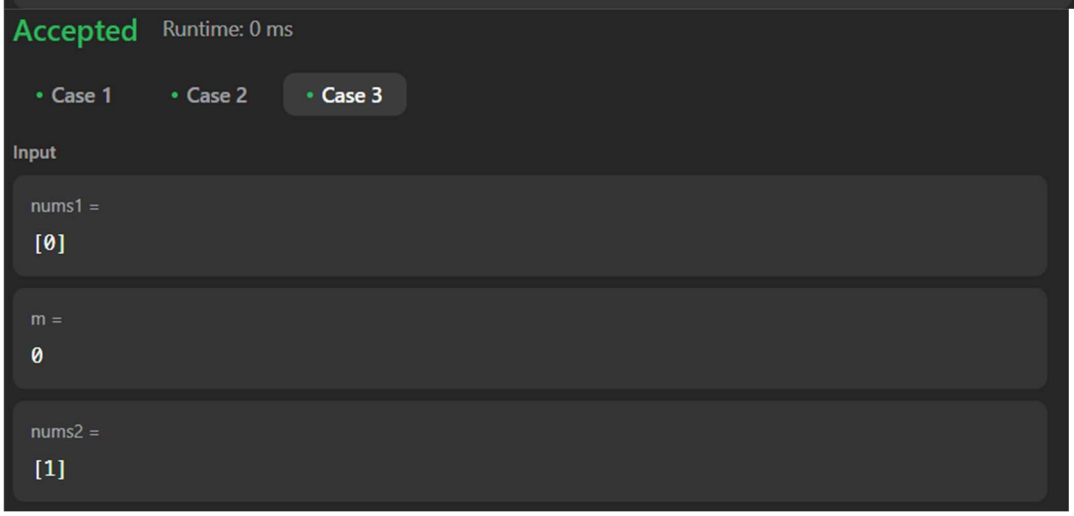
```
from typing import List

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

```
    left, right = 0, len(nums) - 1

    while left <= right:
        mid = (left + right) // 2

        # If the target is found, return its index
        if nums[mid] == target:
            return mid

        # Check if the left half is sorted
        if nums[left] <= nums[mid]:
            # If target is in this sorted range
            if nums[left] <= target < nums[mid]:
                right = mid - 1  # Search in left half
            else:
                left = mid + 1   # Search in right half

        # Otherwise, the right half must be sorted
        else:
            # If target is in this sorted range
            if nums[mid] < target <= nums[right]:
                left = mid + 1   # Search in right half
            else:
                right = mid - 1  # Search in left half

    # Target is not found
    return -1
```

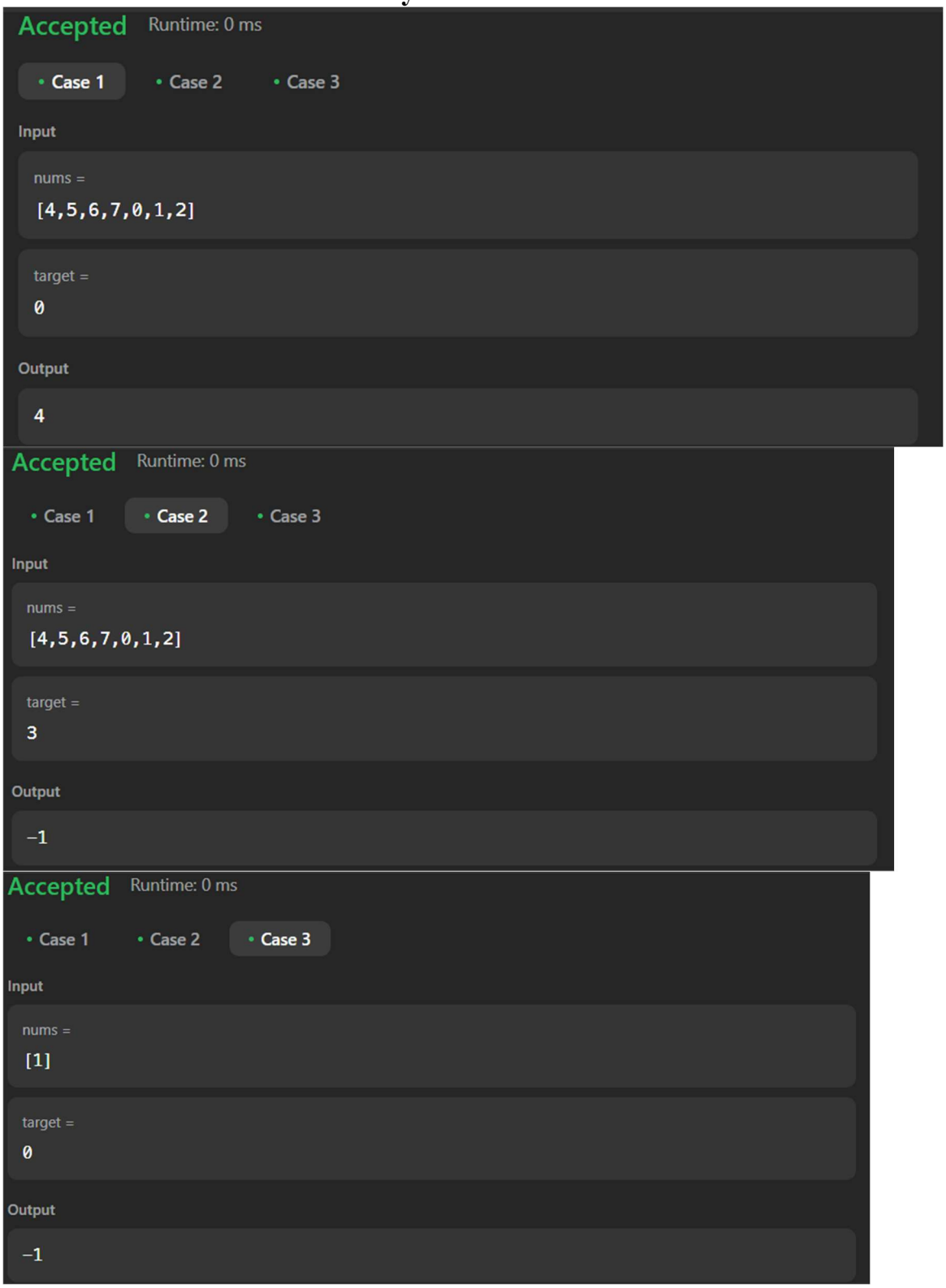
4. **Output:**
    1. **Merge Sorted Array:**


Accepted
Runtime: 0 ms
• Case 1
• Case 2
• Case 3
Input
nums1 =
[1,2,3,0,0,0]
m =
3
nums2 =
[2,5,6]


Accepted
Runtime: 0 ms
• Case 1
• Case 2
• Case 3
Input
nums1 =
[1]
m =
1
nums2 =
[]


Accepted
Runtime: 0 ms
• Case 1
• Case 2
• Case 3
Input
nums1 =
[0]
m =
0
nums2 =
[1]

2. Search in Rotated Sorted Array:



5. Learning Outcome

1) Learned how to apply binary search efficiently to a rotated sorted array.
2) Gained insight into detecting which half of the array is sorted to narrow down the search space.
3) Developed the ability to search in **O(log n)** time complexity instead of **O(n)** linear search.
4) Learned to handle cases like no rotation, single-element arrays, and targets not in the array.
5) Improved logical thinking by implementing conditions to adjust search space dynamically.