Experiment - 1(A)

Student Name: Ankur
UID: 22BCS16091
Branch: CSE
Section/Group: NTPP_602-A
Semester: 6
Date of Performance: 17-02-25
Subject Name: Advanced Programming Lab-2
Subject Code: 22CSH-359

1. **Title:** Arrays and logic building (Remove duplicates from a sorted array)
   https://leetcode.com/problems/remove-duplicates-from-sorted-array/

2. **Objective:** The goal is to modify the input sorted array nums by removing the duplicates in-place, such that each unique element appears only once, and return the number of unique elements. The relative order of the elements should be kept the same.

3. **Algorithm:**

- **Initial Check:**

- If the array is empty, return 0, as there are no elements to process.

- **Pointer Initialization:**

- Use two pointers: i for the index where the next unique element should be placed, and j for iterating through the array.

- **Traverse the Array:**

- Start with j = 1, and compare each element nums[j] with the element at index i.
- If nums[i] is not equal to nums[j], it means a new unique element is found. Increment i and assign nums[i] = nums[j].

- **Return the Count of Unique Elements:**

- After the loop finishes, return i + 1, as i will be the index of the last unique element.

4. **Implementation/Code:**

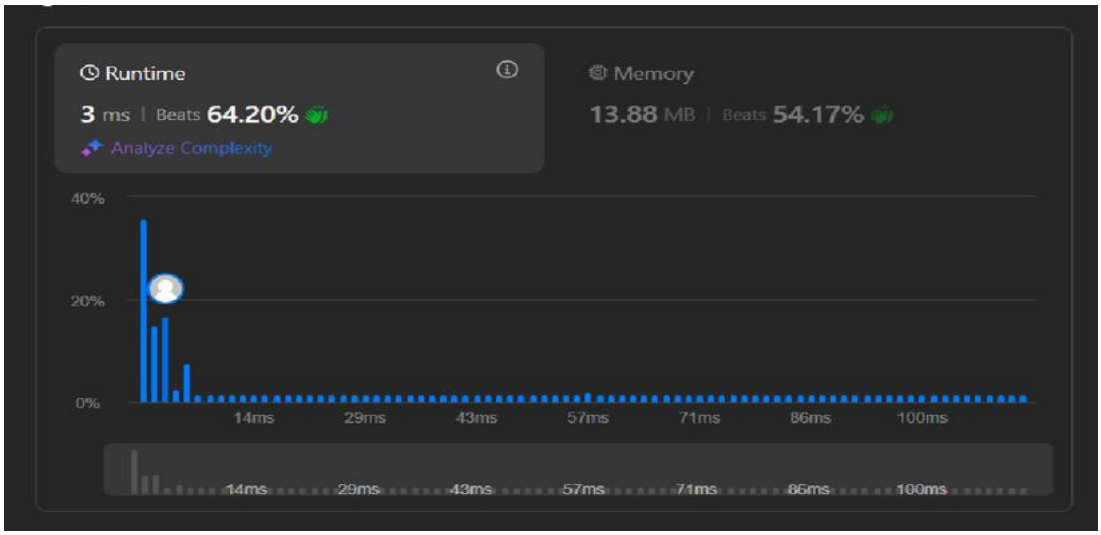
```
class Solution:
    def removeDuplicates(self, nums):
        # If the array is empty
        if not nums:
            return 0
```

```
        # Pointer for the place to insert the next unique element
        i = 0

        # Traverse the array starting from index 1
        for j in range(1, len(nums)):
            if nums[i] != nums[j]:
                i += 1
                nums[i] = nums[j]

        # The number of unique elements is i + 1
        return i + 1
```

5. **Output:**



6. **Time Complexity:** O(n)

7. **Space Complexity:** O(1)

8. **Learning Outcomes:**

- **In-Place Modification:** Understanding how to modify an array in-place while maintaining its integrity, which is a key concept in optimizing space complexity.

- **Two Pointer Technique:** Learn how to use the two-pointer technique to solve problems where elements need to be moved or compared based on certain conditions.

Experiment 1(B)

1. **Title:** Contains duplicate (https://leetcode.com/problems/contains-duplicate/description/)

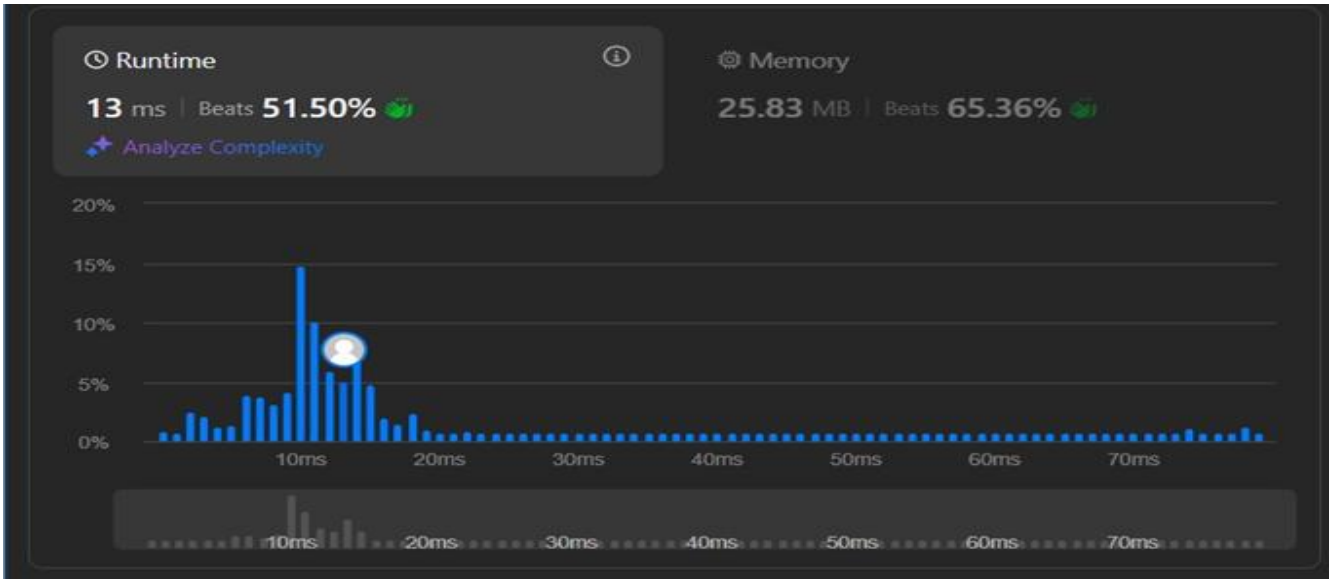
2. **Objective:** The goal is to determine if any value appears at least twice in the given integer array nums, and return true if any value is duplicated, or false if all elements are distinct.

3. **Algorithm:**
   - **Convert the Array to a Set:**
     - A set data structure automatically removes duplicate elements. By converting the input list `nums` to a set, we can easily determine if any duplicates existed.
   - **Compare Lengths:**
     - If the length of the set is smaller than the length of the array `nums`, it means some elements were duplicates (since the set removed them). In this case, return `true`.
     - If the lengths are the same, then there were no duplicates, and we return `false`.

4. **Implementation/Code:**

```
class Solution:
    def containsDuplicate(self, nums):
        # Compare the length of the array and the set of the array
        return len(nums) != len(set(nums))
```

6. **Output:**



Experiment 1(C)

1. **Title:** Two Sum (https://leetcode.com/problems/two-sum/)

2. **Objective:** Given an array of integers nums and a target integer target, the task is to return the indices of the two numbers such that they add up to the target. The solution should be efficient and must not use the same element twice.

3. **Algorithm:**

- **Use a Hash Map (Dictionary):**
  - Create a dictionary seen to store each number and its corresponding index as we iterate through the array.
  - For each element in the array:
    - Calculate the **complement** as target - current number.
    - Check if the complement is already in the dictionary:
      - If it is, return the indices of the complement and the current number.
      - If not, store the current number along with its index in the dictionary.
  - This allows us to find the solution in a single pass through the array.

4. **Implementation/Code:**

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

5. **Output:**