Experiment-4(A)

StudentName: Pranav Saini
UID: 22BCS15994
Branch: CSE
Section/Group: 602-A
Semester: 6
DateofPerformance: 13-02-25
SubjectName: AdvancedProgrammingLab-2
SubjectCode: 22CSH-359

1. **Title:** Sorting and Searching (Merge Sorted Array)
   https://leetcode.com/problems/merge-sorted-array/

2. **Objective:** To merge two sorted arrays nums1 and nums2 into one sorted array, while ensuringthat theresult is storedin nums1. The algorithmshould be efficient and merge the arrays in-place.

3. **Algorithm:**
   - Startwiththreepointers:`i`forthelastelementof`nums1`(index`m-1`),`j`for thelastelement of `nums2`(index `n-1`), and `k`for the last position in `nums1`(index `m+n-1`).
   - Comparetheelementsat`nums1[i]`and`nums2[j]`.Placethelargerofthetwoat `nums1[k]`,anddecrement`k`.
   - If`nums1[i]` isgreater,move`i`left.If`nums2[j]` isgreater,move`j`left.
   - Ifanyelementsare left in `nums2`, copythem to `nums1`(since `nums1`already contains the first `m` elements in sorted order).
   - Continuetheprocessuntilallelementsfrombotharraysaremerged.

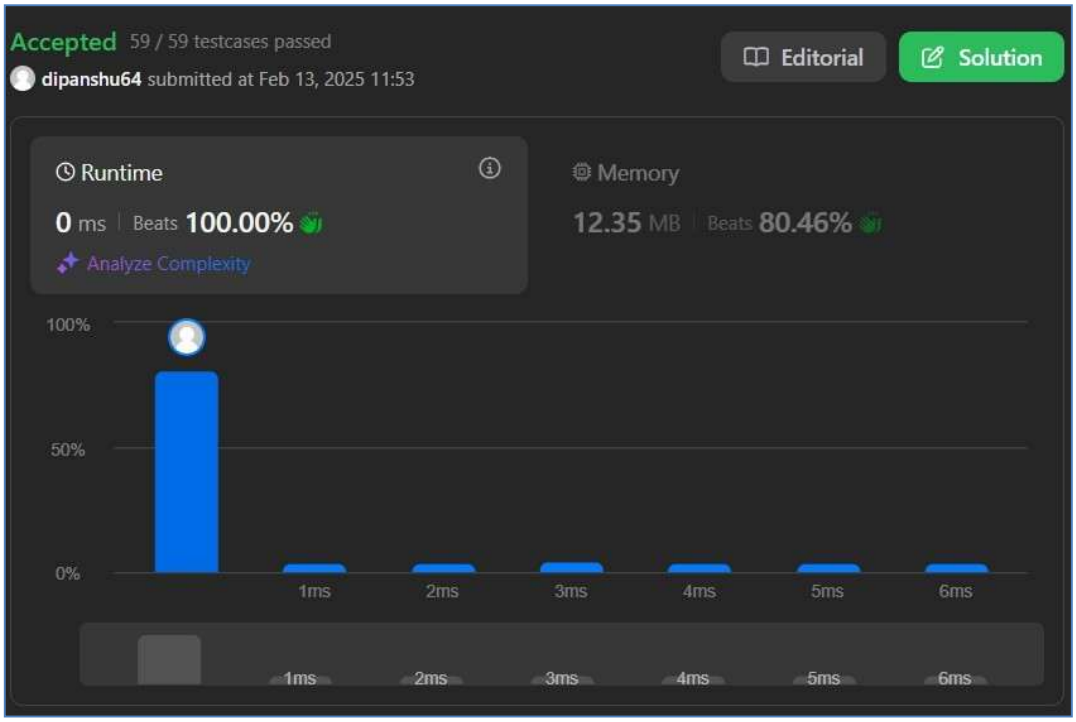
4. **Implementation/Code:**

```
classSolution:
defmerge(self,nums1,m,nums2,n):
    #Startfromtheendofbotharrays i, j, k
    = m- 1, n - 1, m + n- 1

    # Merge in reverse order
    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
```

```
#Ifnums2 isnot fully traversed,copy theremaining elements while j >=
0:
    nums1[k]=nums2[j]
    j-=1
    k-=1
```

5. **Output:**



6. **TimeComplexity:**O(m+n)

7. **SpaceComplexity:**O(1)

8. **LearningOutcomes:**

- **ArrayManipulation**:Understandinghowto manipulatearraysin-place.
- **TwoPointerTechnique**:Usingmultiplepointerstotraversearraysefficientlyandmerge them.
- **In-placeSorting**:Learninghowtomergesortedarrayswithoutextra space.
- **Optimization**:Achievinglineartimecomplexitybyprocessingthearraysfromtheend, avoiding unnecessary moves.

Experiment4(B)

1. **Title:** FirstBadColors(https://leetcode.com/problems/first-bad-version/)

2. **Objective:** The task is to find the first bad version in a series of product versions, where all versions after the first bad version are also bad. You should minimize the number of calls to the isBadVersion(version) API.

3. **Algorithm:**
   - Initialize two pointers `left` as 1 and `right` as `n`.
   - Perform a binary search:
     - While `left` is less than `right`, calculate the mid-point `mid = (left + right) // 2`.
     - If `isBadVersion(mid)` is `True`, then the first bad version is at `mid` or to the left of it, so update `right = mid`.
     - If `isBadVersion(mid)` is `False`, then the first bad version must be to the right, so update `left = mid + 1`.
   - When the loop terminates, `left` will point to the first bad version.

4. **Implementation/Code:**

```
classSolution:

    deffirstBadVersion(self,n):

        left, right = 1, n


        whileleft<right:

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

            isBadVersion(mid):

                right=mid#Potentialfirstbadversion else:

                      left=mid+1#Badversionmustbeontheright


        returnleft
```

6. Output:

8. TimeComplexity:O(logn) **9.SpaceComplexity:**O(1)

9. LearningOutcomes:

1. **BinarySearch**:Understandingandapplyingthebinarysearchalgorithmtoreducethenumber of checks (calls to isBadVersion).

2. **Optimization**:MinimizingthenumberofcallstoanexternalAPI,whichisakeyperformance concern in real-world applications.

3. **ProblemSolving**:Effectivelynarrowingdowntherangeforthefirstbadversionusinga divide-and-conquer strategy.

Experiment4(C)

1. **Title:** SortColors(https://leetcode.com/problems/sort-colors/)

2. **Objective:** To sort an array containing three distinct values (0, 1, 2) in place, where 0 represents red, 1 represents white, and 2 represents blue. The array should be sorted in such a way that all 0s come first, followed by all 1s, and then all 2s.

4. **Algorithm:**

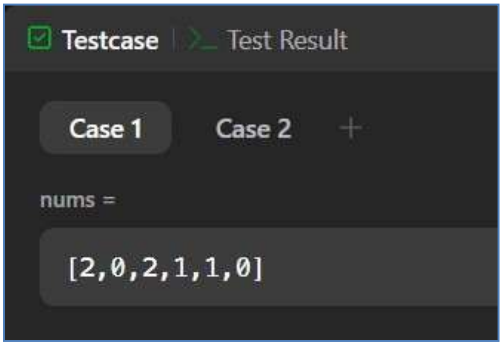
- Initializethreepointers: `low`(starting at the beginning of the array), `mid`(also starting at the beginning), and `high`(starting at the end of the array).
- Traversethearrayusingthe`mid`pointer:
  - If`nums[mid]`is 0,swapitwith`nums[low]`andincrementboth`low`and`mid`.
  - If`nums[mid]`is 1,justincrement`mid`.
  - If`nums[mid]`is2,swapitwith`nums[high]`anddecrement`high`.
- Continuethisprocessuntil`mid`exceeds`high`.

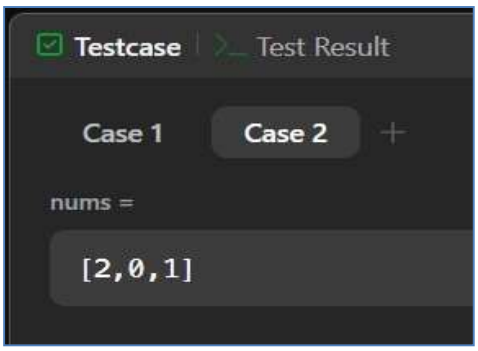
5. **Implementation/Code:**

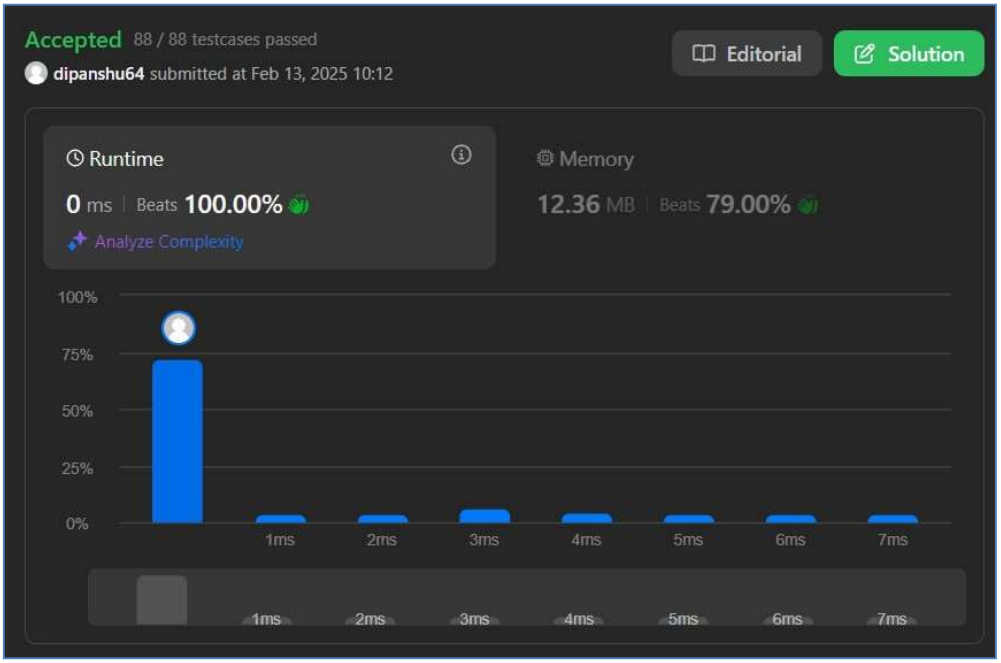
```
classSolution:
    defsortColors(self,nums):
        low,mid,high=0,0,len(nums)-1

        whilemid<=high:
            ifnums[mid]==0:
                nums[low],nums[mid]=nums[mid],nums[low] low
                += 1
                mid+=1
            elifnums[mid]==1: mid
                += 1
            else:
                nums[mid],nums[high]=nums[high],nums[mid]
                high -= 1
```

6. **Output:**







8.TimeComplexity:O(N)

9.SpaceComplexity:O(1)

10. LearningOutcomes:

1. **Three-wayPartitioning**:Understandinghowtopartitionanarrayintothreegroupsbasedona specific condition (here, 0, 1, 2).
2. **In-placeSorting**:Sorting withoutusingextraspaceforanotherarray.
3. **TwoPointerTechnique**:Efficientuseofmultiplepointers(low,mid,high)totraversethe array in a single pass.
4. **Optimization**:Solvingtheprobleminlineartime(O(n))andconstantspace(O(1)).