Experiment-2(A)

StudentName: Om Ankur Prajapati **UID:**22BCS15926
Branch: CSE **Section/Group:**602-A
Semester:6 **DateofPerformance:**17-02-25
SubjectName: AdvancedProgrammingLab-2 **SubjectCode:**22CSH-359

1. **Title:**Linked Lists (Remove duplicates from a sorted list)
   https://leetcode.com/problems/remove-duplicates-from-sorted-list

2. **Objective:**Given the headof a sortedlinkedlist,the taskisto removeallduplicatessuchthat each element appears only once. Return the modified linked list, which is still sorted.

3. **Algorithm:**

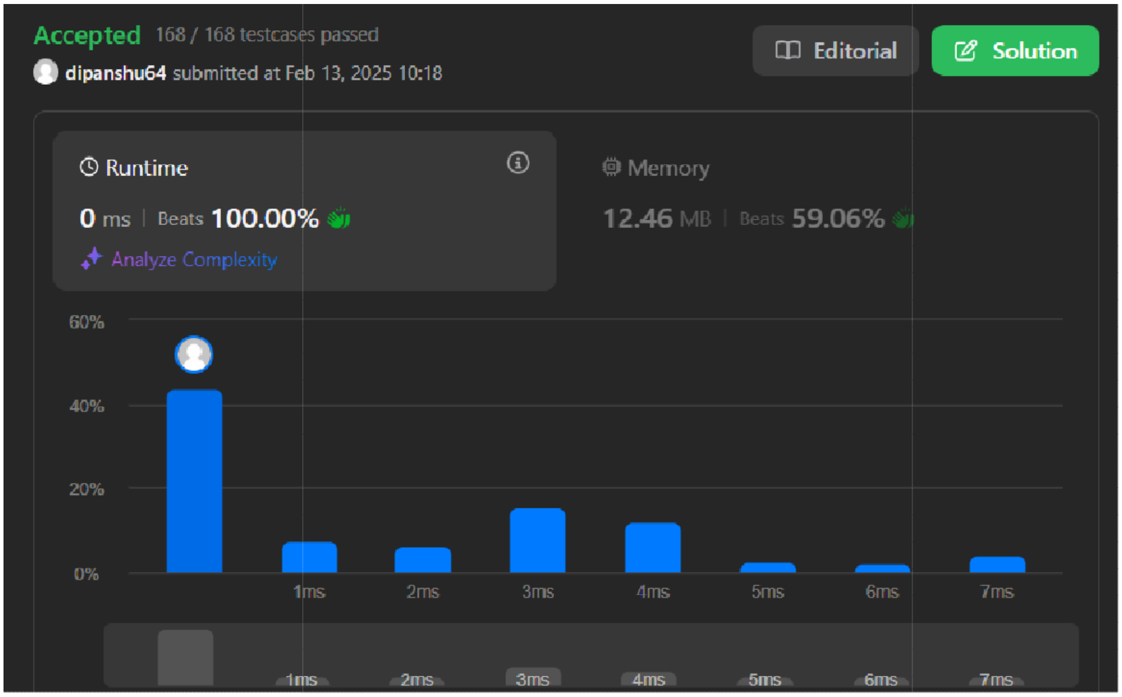
- **IterateThroughtheLinkedList:**
- Startwiththeheadofthelist.
- Traversethelistusingapointercurrentstartingfromthehead.
- **Checkfor Duplicates:**
- Foreachnode,comparethevalueofthecurrentnodewiththevalueofthenextnode.
- If the valuesare equal (i.e., a duplicate), update the current node'snext pointer to skip the next node.
- Ifthevaluesarenotequal,simplymovethecurrentpointertothenextnode.
- **Endof List:**
- Continuethisprocessuntilyoureachtheendofthelist(i.e.,current.nextisNone).
- **ReturnModifiedList:**
- Thelinkedlistwillbemodifiedin-placewiththeduplicatesremoved.
- Returnthemodifiedheadofthelinkedlist.

4. **Implementation/Code:**

```
classSolution:
def deleteDuplicates(self, head):
    current = head
    whilecurrentandcurrent.next:
        #Ifcurrentnode'svalueisequaltothenextnode'svalue
```

```
          ifcurrent.val==current.next.val:
              current.next=current.next.next#Skiptheduplicatenode else:
              current = current.next# Move to the next node return
     head
```

5. **Output:**



Accepted 168 / 168 testcases passed

dipanshu64 submitted at Feb 13, 2025 10:18

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

12.46 MB | Beats 59.06%

6. **TimeComplexity:**O(n) 7.**SpaceComplexity:**O(1)

7. **LearningOutcomes:**

- **In-placeModificationsinLinkedLists:**
  - Understandhowtoperformoperationsonlinkedlistswithoutusingadditionalmemory (i.e., modifying the list in-place).
- **Traversing Linked Lists:**
  - Gainexperiencewithtraversingalinkedlistandmanipulatingthenextpointers.
- **HandlingEdgeCases:**
  - The solution works even for edge cases where the linked list is empty (i.e.,head is None) or has only one node.

Experiment2(B)

1. **Title:**Reversealinkedlist(https://leetcode.com/problems/reverse-linked-list/)

2. **Objective:**Giventheheadofasinglylinkedlist,thetaskistoreversethelistandreturnthe reversed list.

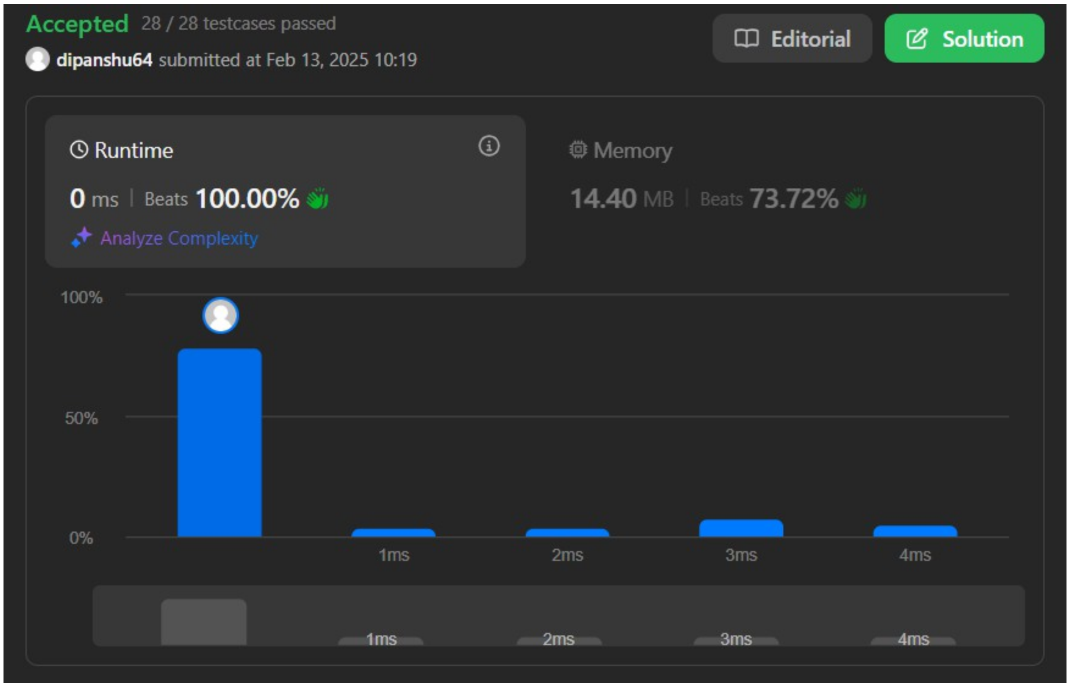
3. **Algorithm:**

- **Initialization:**
  - Startwiththreepointers:
    - previnitiallysettoNone.
    - currentinitializedtohead.
    - next_nodewillhelpinkeepingtrackofthenextnodeinthelist.
- **TraversetheList:**
  - Foreachnodeinthelinkedlist:
    - Savethenextnode:next_node= current.next.
    - Reversethedirectionofthecurrentnode'snextpointer:current.next=prev.
    - Moveprevtothecurrentnode:prev= current.
    - Movecurrenttothenextnode:current= next_node.
- **Endof List:**
  - When currentbecomesNone, thelistis reversed, and prev will be the newhead of the reversed list.
- **Returnthe ReversedList:**
  - Thenewheadofthereversedlinkedlistisprev.

4. **Implementation/Code:**

```
classListNode:
    definit(self,val=0,next=None):
        self.val = val
        self.next=next
classSolution:
    defreverseList(self,head): prev
        = None
        current=head
        whilecurrent:
            next_node=current.next#Savenextnode
            current.next=prev#Reversethecurrentnode'spointer prev =
            current# Move prev to current node
            current=next_node#Movecurrenttonextnode
        returnprev#Returnthenewhead
```

5. Output:



Accepted 28 / 28 testcases passed

dipanshu64 submitted at Feb 13, 2025 10:19

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

14.40 MB | Beats 73.72%

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

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

8. LearningOutcomes:

- **Reversinga LinkedList:**
  - Learn how to reverse the direction of pointers in a singly linked list, both iteratively and recursively.
- **RecursivevsIterativeApproaches:**
  - Understand the trade-offs between iterative and recursive approaches for solving linked list problems.
- **LinkedListManipulation:**
  - Gain experience in manipulating linked list nodes and pointers to achieve desired outcomes (such as reversal).

Experiment2(C)

1. **Title:**Delete middle node of a list (https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list)

2. **Objective:**Given the head of a linked list, the task is to delete the middle node and return the head of the modified list. The middle node is the ⌊n / 2⌋th node, where n is the length of the list, and ⌊x⌋ represents the largest integer less than or equal to x.

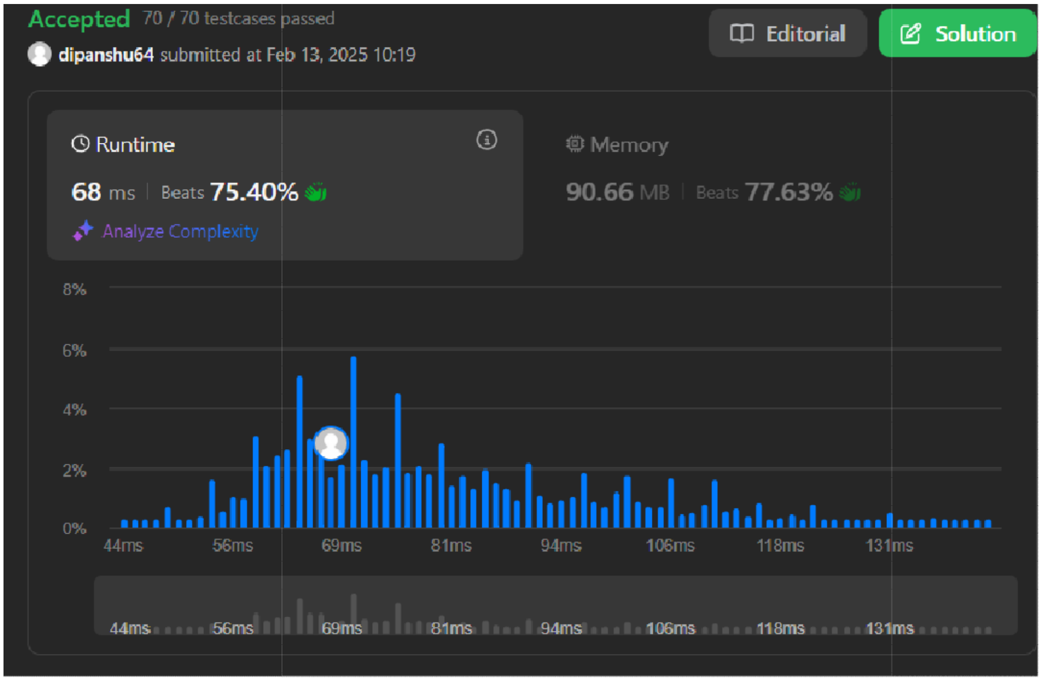
3. Algorithm:

- **FindtheLengthoftheList:**
  - Traversethelinkedlistandcalculatethelengthn.
- **DeterminetheMiddleNode:**
  - Themiddlenodeisattheindexn//2.
- **HandleEdgeCaseforSmallLists:**
  - Ifthelistcontainsonlyonenode(n== 1),returnNone(thelistbecomesempty).
- **TraversetotheNodeBeforetheMiddleNode:**
  - Useapointertotraversetothenodejustbeforethemiddlenode.
- **DeletetheMiddleNode:**
  - Modifythenextpointerofthenodebeforethemiddlenodetopointtothenodeafterthe middle node.
- **ReturntheModifiedList:**
  - Returnthemodifiedheadofthelinkedlistafterthemiddlenodeisremoved.

4. Implementation/Code:

```
classListNode:
    def init(self, val=0, next=None):
        self.val = val
        self.next=next

classSolution:
    defdeleteMiddle(self,head):
        #Ifthelistcontainsonlyonenode,returnNone(emptylist after removal)
        if not head or not head.next:
            return None

        # Initialize two pointers: slow (to find the middle) and fast (to
find the end)
        slow = head
        fast = head
        prev=None
        # Traverse the list with fast moving two steps at a time and slow
moving one step
        while fast and fast.next:
            fast = fast.next.next
            prev = slow
            slow=slow.next

        #Removethemiddlenodebyconnectingprevtoslow.next if prev:
            prev.next=slow.next

        returnhead
```

5. **Output:**



Accepted 70 / 70 testcases passed

dipanshu64 submitted at Feb 13, 2025 10:19

Runtime

68 ms | Beats 75.40%

Memory

90.66 MB | Beats 77.63%

9. **TimeComplexity:**O(n) 7.**SpaceComplexity:**O(1)

10. LearningOutcomes:

- **In-placeModificationsinLinkedLists:**
  - Learnhow tomodifylinkedlistsinplace,withoutcreatingnew nodesorarrays.
- **TwoPointer Technique:**
  - Understandthe usageof theslowand fastpointersto findspecific nodes in a linked list (in this case, the middle node).