

AP Assignment 3

1. Print Linked List: <https://www.geeksforgeeks.org/problems/print-linked-list-elements/0>

The screenshot shows the GeeksforGeeks website interface for the 'Print Linked List' problem. The 'Output Window' on the left indicates 'Problem Solved Successfully' with 1112/1112 test cases passed, 1/1 attempts, 100% accuracy, and 0.1 time taken. The code editor on the right shows a C++ solution for printing a linked list.

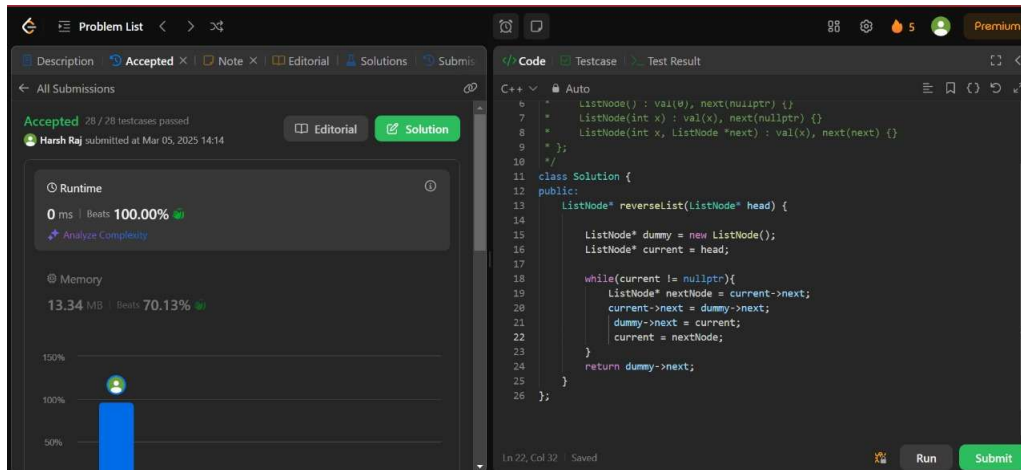
```
24
25 Node(int x) {
26     data = x;
27     next = nullptr;
28 }
29 };
30 */
31 /*
32 Print elements of a linked list on console
33 Head pointer input could be NULL as well for empty list
34 */
35
36 class Solution {
37 public:
38     // Function to display the elements of a linked list in same line
39     void printList(Node *head) {
40         // your code goes here
41         while(head != nullptr){
42             cout<<head->data<<" ";
43             head = head->next;
44         }
45     }
46 };
47
48 // Driver Code Ends
```

2. Remove duplicates from a sorted list: <https://leetcode.com/problems/remove-duplicates-from-sorted-list/description/>

The screenshot shows the LeetCode website interface for the 'Remove Duplicates from Sorted List' problem. The 'Runtime' and 'Memory' sections show performance metrics: 0 ms (100.00% beats) and 16.01 MB (90.09% beats). The code editor on the right shows a C++ solution for removing duplicates from a sorted linked list.

```
7 * ListNode(int x) : val(x), next(nullptr) {}
8 * ListNode(int x, ListNode *next) : val(x), next(next) {}
9 * };
10 */
11 class Solution {
12 public:
13     ListNode* deleteDuplicates(ListNode* head) {
14
15         ListNode* current = head;
16
17         while(current != nullptr && current->next != nullptr){
18             if(current->val == current->next->val){
19                 current->next = current->next->next;
20             }
21             else{
22                 current = current->next;
23             }
24         }
25         return head;
26     }
27 };
28
```

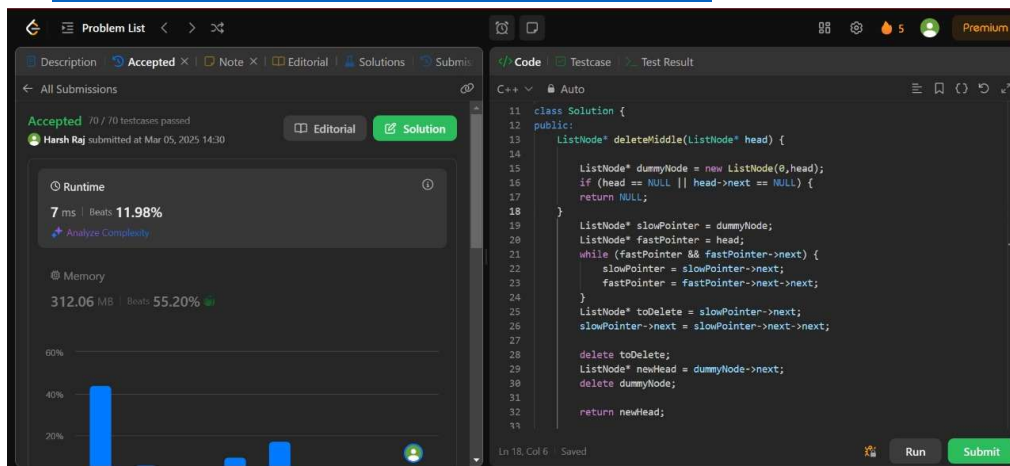
3. Reverse a linked list: <https://leetcode.com/problems/reverse-linked-list/description/>



The screenshot shows the LeetCode interface for the problem 'Reverse a linked list'. On the left, the 'Accepted' status is confirmed with 28/28 testcases passed. The runtime is 0 ms (beats 100.00%) and memory usage is 13.34 MB (beats 70.13%). The code on the right is a C++ implementation of the reverse linked list algorithm. It uses a dummy node and iterates through the list, reversing the next pointers.

```
6  * ListNode() : val(0), next(nullptr) {}
7  * ListNode(int x) : val(x), next(nullptr) {}
8  * ListNode(int x, ListNode *next) : val(x), next(next) {}
9  */
10
11 class Solution {
12 public:
13     ListNode* reverseList(ListNode* head) {
14
15         ListNode* dummy = new ListNode();
16         ListNode* current = head;
17
18         while(current != nullptr){
19             ListNode* nextNode = current->next;
20             current->next = dummy->next;
21             dummy->next = current;
22             current = nextNode;
23         }
24         return dummy->next;
25     }
26 };
```

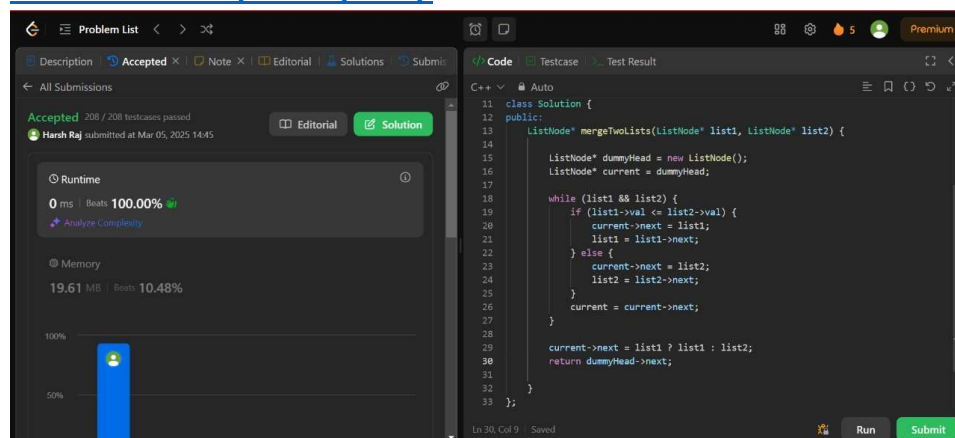
4. Delete middle node of a list: <https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list/description/>



The screenshot shows the LeetCode interface for the problem 'Delete the middle node of a linked list'. The 'Accepted' status is confirmed with 70/70 testcases passed. The runtime is 7 ms (beats 11.98%) and memory usage is 312.06 MB (beats 55.20%). The code on the right is a C++ implementation that finds the middle node and bypasses it by connecting the previous node's next pointer to the next node's next pointer.

```
11 class Solution {
12 public:
13     ListNode* deleteMiddle(ListNode* head) {
14
15         ListNode* dummyNode = new ListNode(0, head);
16         if (head == NULL || head->next == NULL) {
17             return NULL;
18         }
19
20         ListNode* slowPointer = dummyNode;
21         ListNode* fastPointer = head;
22         while (fastPointer && fastPointer->next) {
23             slowPointer = slowPointer->next;
24             fastPointer = fastPointer->next->next;
25         }
26
27         ListNode* toDelete = slowPointer->next;
28         slowPointer->next = slowPointer->next->next;
29         delete toDelete;
30         ListNode* newHead = dummyNode->next;
31         delete dummyNode;
32         return newHead;
33     }
34 };
```

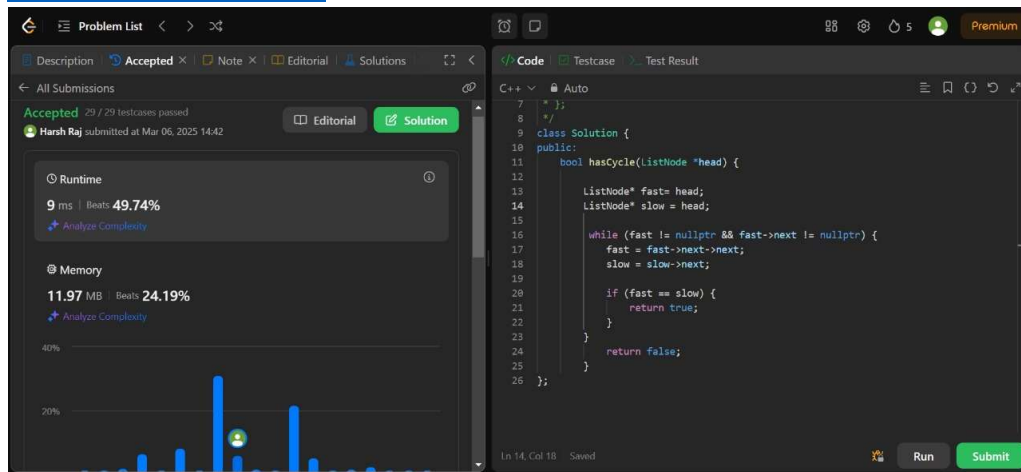
5. Merge two sorted linked lists: <https://leetcode.com/problems/merge-two-sorted-lists/description/>



The screenshot shows the LeetCode interface for the problem 'Merge two sorted linked lists'. The 'Accepted' status is confirmed with 208/208 testcases passed. The runtime is 0 ms (beats 100.00%) and memory usage is 19.61 MB (beats 10.48%). The code on the right is a C++ implementation that merges two sorted linked lists by comparing the values of the nodes and linking them in order.

```
11 class Solution {
12 public:
13     ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
14
15         ListNode* dummyHead = new ListNode();
16         ListNode* current = dummyHead;
17
18         while (list1 && list2) {
19             if (list1->val <= list2->val) {
20                 current->next = list1;
21                 list1 = list1->next;
22             } else {
23                 current->next = list2;
24                 list2 = list2->next;
25             }
26             current = current->next;
27         }
28
29         current->next = list1 ? list1 : list2;
30         return dummyHead->next;
31     }
32 };
33
```

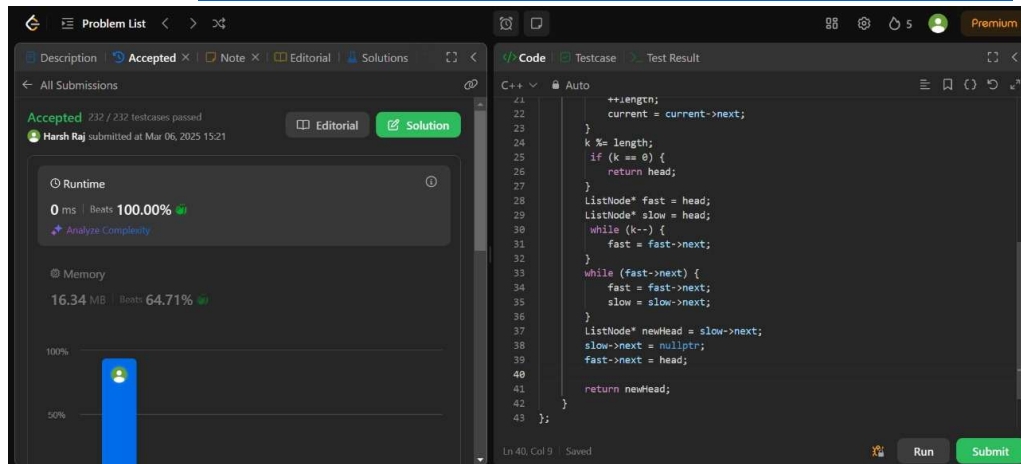
6. Detect a cycle in a linked list: <https://leetcode.com/problems/linked-list-cycle/description/>



The screenshot shows the LeetCode interface for the problem "Detect a cycle in a linked list". The submission is accepted, with 29/29 testcases passed. The runtime is 9 ms, beating 49.74% of submissions. The memory usage is 11.97 MB, beating 24.19% of submissions. The code is written in C++ and implements Floyd's Cycle-Finding algorithm (slow and fast pointers).

```
C++  
7  *;  
8  */  
9  class Solution {  
10 public:  
11     bool hasCycle(ListNode *head) {  
12  
13         ListNode* fast = head;  
14         ListNode* slow = head;  
15  
16         while (fast != nullptr && fast->next != nullptr) {  
17             fast = fast->next->next;  
18             slow = slow->next;  
19  
20             if (fast == slow) {  
21                 return true;  
22             }  
23         }  
24         return false;  
25     }  
26 };
```

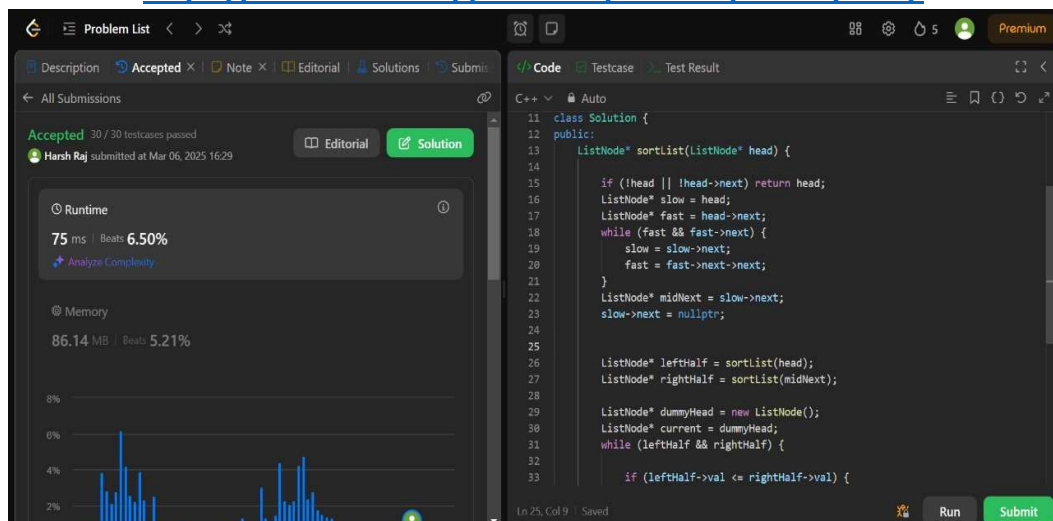
7. Rotate a list: <https://leetcode.com/problems/rotate-list/description/>



The screenshot shows the LeetCode interface for the problem "Rotate a list". The submission is accepted, with 232/232 testcases passed. The runtime is 0 ms, beating 100.00% of submissions. The memory usage is 16.34 MB, beating 64.71% of submissions. The code is written in C++ and implements a rotation algorithm by finding the tail and connecting it to the head.

```
C++  
21  
22     ++length;  
23     current = current->next;  
24 }  
25 k %= length;  
26 if (k == 0) {  
27     return head;  
28 }  
29 ListNode* fast = head;  
30 ListNode* slow = head;  
31 while (k--) {  
32     fast = fast->next;  
33 }  
34 while (fast->next) {  
35     fast = fast->next;  
36     slow = slow->next;  
37 }  
38 ListNode* newHead = slow->next;  
39 slow->next = nullptr;  
40 fast->next = head;  
41  
42 return newHead;  
43 };
```

8. Sort List: <https://leetcode.com/problems/sort-list/description/>



The screenshot shows the LeetCode interface for the problem "Sort List". The submission is accepted, with 30/30 testcases passed. The runtime is 75 ms, beating 6.50% of submissions. The memory usage is 86.14 MB, beating 5.21% of submissions. The code is written in C++ and implements a merge sort algorithm for a linked list.

```
C++  
11 class Solution {  
12 public:  
13     ListNode* sortList(ListNode* head) {  
14  
15         if (!head || !head->next) return head;  
16         ListNode* slow = head;  
17         ListNode* fast = head->next;  
18         while (fast && fast->next) {  
19             slow = slow->next;  
20             fast = fast->next->next;  
21         }  
22         ListNode* midNext = slow->next;  
23         slow->next = nullptr;  
24  
25         ListNode* leftHalf = sortList(head);  
26         ListNode* rightHalf = sortList(midNext);  
27  
28         ListNode* dummyHead = new ListNode();  
29         ListNode* current = dummyHead;  
30         while (leftHalf && rightHalf) {  
31             if (leftHalf->val <= rightHalf->val) {  
32  
33
```

9. Merge k sorted lists: <https://leetcode.com/problems/merge-k-sorted-lists/description/>

The screenshot displays a LeetCode submission interface for the problem "Merge k sorted lists". The submission is marked as "Accepted" with 134 out of 134 test cases passed. The user, Harsh Raj, submitted the solution on March 06, 2025, at 16:37. The solution is written in C++ and uses a min-heap to merge k sorted lists.

Runtime: 3 ms | Beats 64.88%
Memory: 18.48 MB | Beats 66.07%

```
11 class Solution {
12 public:
13     ListNode* mergeKLists(vector<ListNode*> lists) {
14
15         auto compare = [](ListNode* a, ListNode* b) {
16             return a->val > b->val;
17         };
18
19         priority_queue<ListNode*, vector<ListNode*>, decltype(compare)> min_heap(compare);
20
21         for (ListNode* list_head : lists) {
22             if (list_head) {
23                 min_heap.push(list_head);
24             }
25         }
26
27         ListNode* dummy_head = new ListNode();
28         ListNode* current_node = dummy_head;
29
30         while (!min_heap.empty()) {
31             ListNode* min_node = min_heap.top();
32             min_heap.pop();
33         }
34     }
35 }
```