



Experiment-3

Student Name: Sreyash Bhardwaj

UID: 22BET10072

Branch: BE-IT

Section/Group: 22BET-IOT-702-B

Semester: 6th

Date of Performance: 30/01/2025

Subject Name: AP Lab II

Subject Code: 22ITT-314

Problem-1

1. **Aim:** Detect a cycle in a linked list.
2. **Objective:** Given head, the head of a linked list, determine if the linked list has a cycle in it. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to.

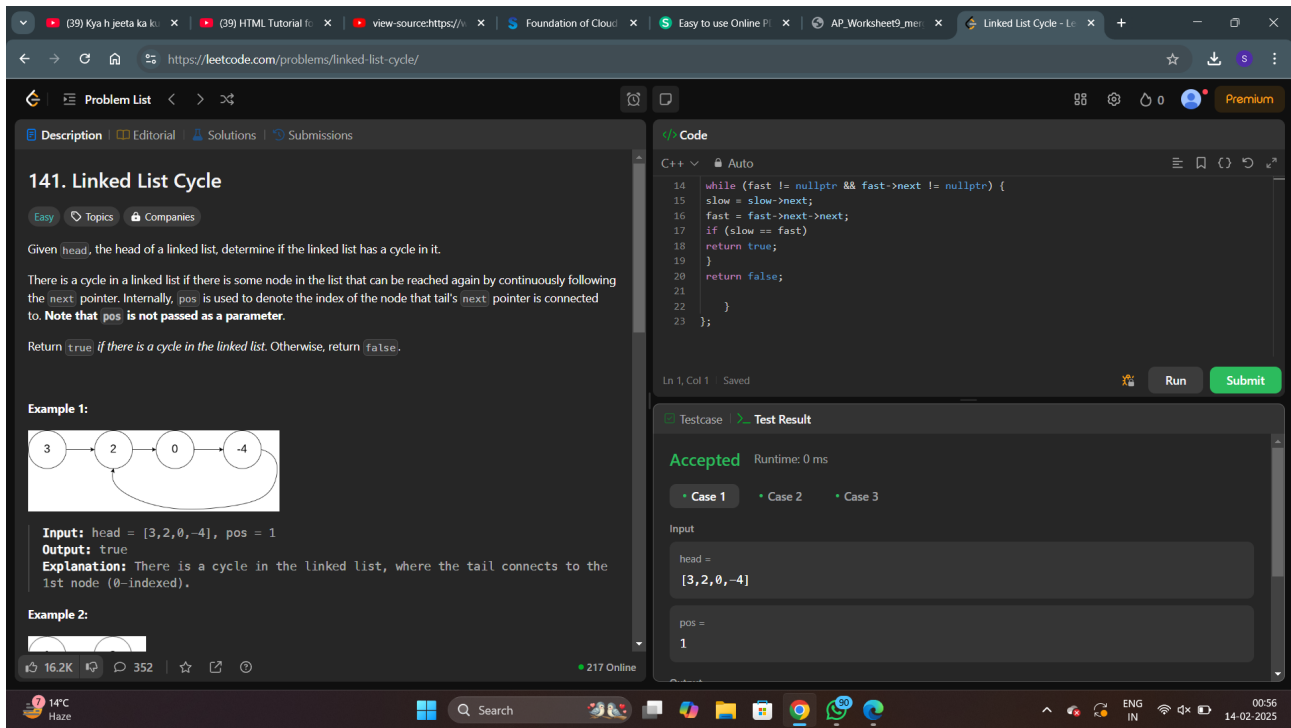
3. Implementation/Code:

```
class Solution {
public:
    bool hasCycle(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head;

        while (fast != nullptr && fast->next != nullptr) {
            slow = slow->next;
            fast = fast->next->next;
            if (slow == fast)
                return true;
        }

        return false;
    }
};
```

4. Output:



The screenshot shows the LeetCode interface for problem 141, "Linked List Cycle". The problem description states: "Given `head`, the head of a linked list, determine if the linked list has a cycle in it. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter.** Return `true` if there is a cycle in the linked list. Otherwise, return `false`."

Example 1:

Diagram: A linked list with nodes containing values 3, 2, 0, and -4. The node with value 0 has its next pointer connected back to the node with value 2, forming a cycle.

Input: `head = [3,2,0,-4]`, `pos = 1`
Output: `true`
Explanation: There is a cycle in the linked list, where the tail connects to the 1st node (0-indexed).

Example 2:

Diagram: A linked list with nodes containing values 1 and 2. The node with value 2 has its next pointer set to `null`, indicating no cycle.

Code:

```
C++  
while (fast != nullptr && fast->next != nullptr) {  
    slow = slow->next;  
    fast = fast->next->next;  
    if (slow == fast)  
        return true;  
}  
return false;
```

Testcase: Case 1 (Accepted) Runtime: 0 ms

Input:

`head = [3,2,0,-4]`

`pos = 1`

The Windows taskbar at the bottom shows the date and time as 14-02-2025, 00:56.

Problem-2

1. **Aim:** Reverse linked list 2
2. **Objective:** Given the head of a singly linked list and two integers left and right where $\text{left} \leq \text{right}$, reverse the nodes of the list from position left to position right, and return the reversed list.
3. **Implementation/Code:**

```
class Solution {
public:
    ListNode* reverseBetween(ListNode* head, int left, int right) {
        if (left == 1)
            return reverseN(head, right);

        head->next = reverseBetween(head->next, left - 1, right - 1);

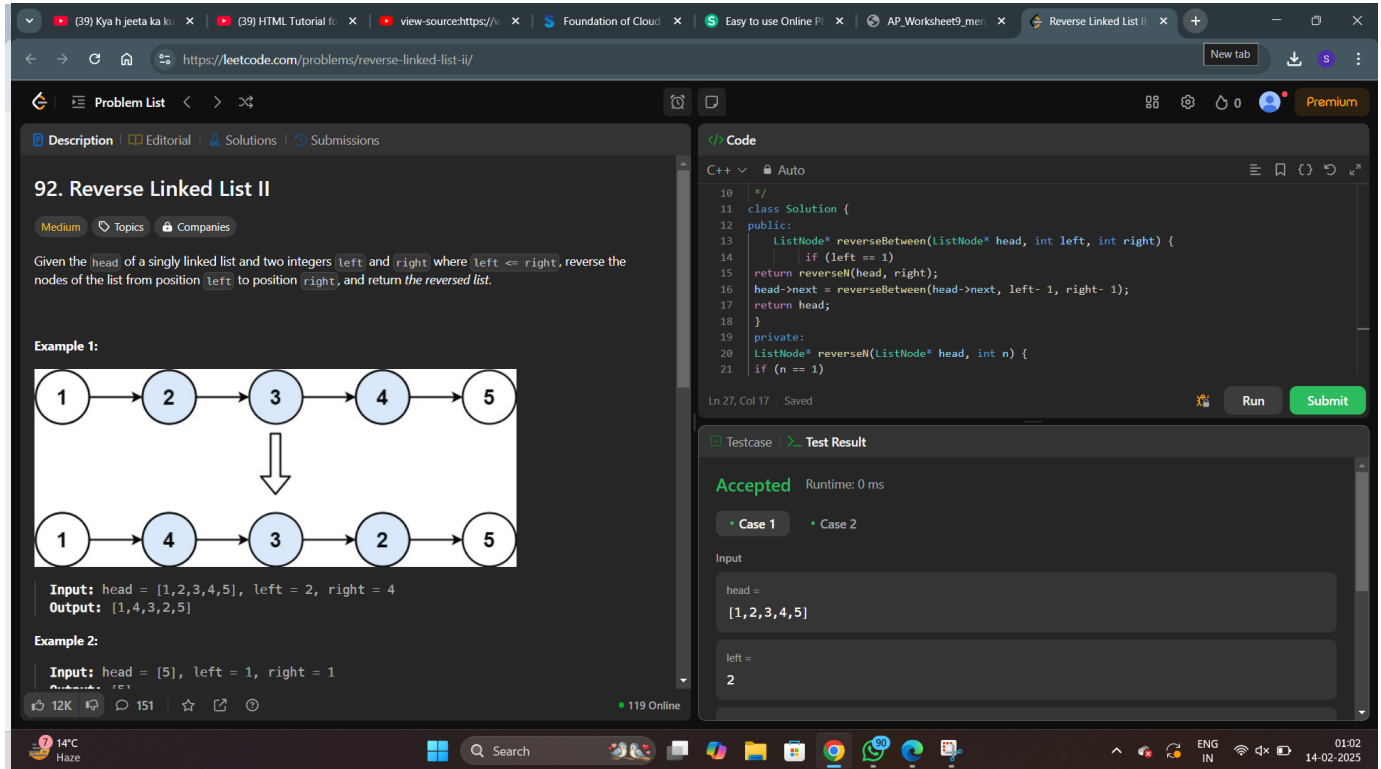
        return head;
    }

private:
    ListNode* reverseN(ListNode* head, int n) {
        if (n == 1)
            return head;

        ListNode* newHead = reverseN(head->next, n - 1);
        ListNode* headNext = head->next;
        head->next = headNext->next;
        headNext->next = head;

        return newHead;
    }
};
```

4. Output:



The screenshot displays a web browser window with the LeetCode problem "92. Reverse Linked List II" open. The problem description states: "Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return the reversed list." Example 1 shows a linked list [1, 2, 3, 4, 5] being reversed from index 2 to 4, resulting in [1, 4, 3, 2, 5]. Example 2 shows a linked list [5] being reversed from index 1 to 1, resulting in [5].

The code editor shows a C++ solution:

```
10  */
11  class Solution {
12  public:
13      ListNode* reverseBetween(ListNode* head, int left, int right) {
14          if (left == 1)
15              return reverseN(head, right);
16          head->next = reverseBetween(head->next, left-1, right-1);
17          return head;
18      }
19  private:
20      ListNode* reverseN(ListNode* head, int n) {
21          if (n == 1)
```

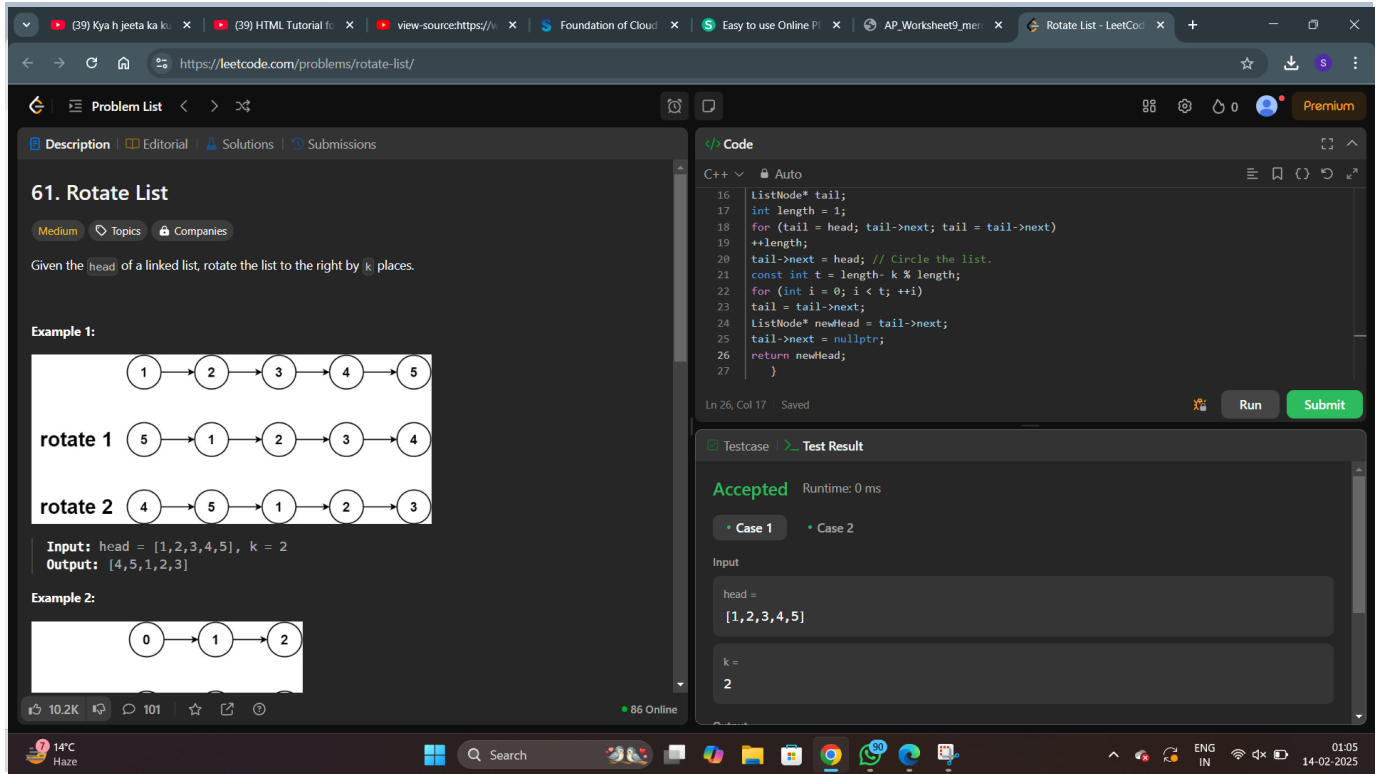
The test results section shows "Accepted" with a runtime of 0 ms. The input for Case 1 is head = [1, 2, 3, 4, 5] and left = 2, resulting in the output [1, 4, 3, 2, 5].

Problem-3

1. **Aim:** Rotate a list
2. **Objective:** Given the head of a linked list, rotate the list to the right by k places.
3. **Code:**

```
class Solution {  
  
    public:  
  
    ListNode* rotateRight(ListNode* head, int k) {  
        if (!head || !head->next || k == 0)  
            return head;  
  
        ListNode* tail;  
        int length = 1;  
  
        for (tail = head; tail->next; tail = tail->next)  
            ++length;  
        tail->next = head; // Circle the list.  
  
        const int t = length - k % length;  
        for (int i = 0; i < t; ++i)  
            tail = tail->next;  
        ListNode* newHead = tail->next;  
        tail->next = nullptr;  
  
        return newHead;  
    }  
};
```

4. Output:



The screenshot shows the LeetCode interface for problem 61, "Rotate List". The problem description states: "Given the head of a linked list, rotate the list to the right by k places." Example 1 shows a linked list [1, 2, 3, 4, 5] being rotated 1 place to become [5, 1, 2, 3, 4], and rotated 2 places to become [4, 5, 1, 2, 3]. The input is head = [1, 2, 3, 4, 5], k = 2, and the output is [4, 5, 1, 2, 3]. Example 2 shows a linked list [0, 1, 2]. The code editor shows a C++ solution that calculates the length of the list, finds the new head, and rotates the list. The test result shows "Accepted" with a runtime of 0 ms.

61. Rotate List

Medium

Given the head of a linked list, rotate the list to the right by k places.

Example 1:

rotate 1

rotate 2

Input: head = [1,2,3,4,5], k = 2
Output: [4,5,1,2,3]

Example 2:

```
ListNode* tail;
int length = 1;
for (tail = head; tail->next; tail = tail->next)
    ++length;
tail->next = head; // Circle the list.
const int t = length- k % length;
for (int i = 0; i < t; ++i)
    tail = tail->next;
ListNode* newHead = tail->next;
tail->next = nullptr;
return newHead;
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

head = [1,2,3,4,5]

k = 2

14°C Haze 01:05 14-02-2025

Problem-4

1. **Aim:** Merge k sorted lists
2. **Objective:** You are given an array of k linked-lists lists, each linked-list is sorted in ascending order. Merge all the linked-lists into one sorted linked-list and return it.

3. Implementation/Code:

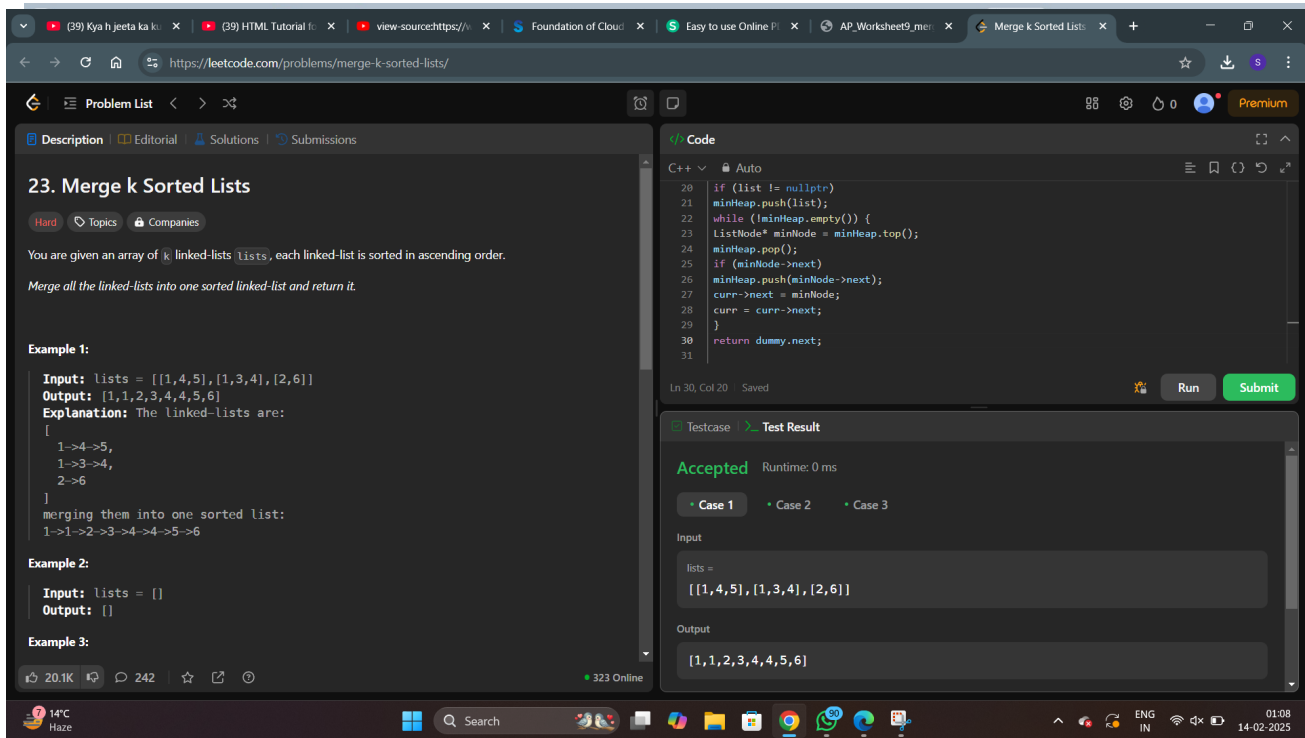
```
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        ListNode dummy(0);
        ListNode* curr = &dummy;
        auto compare = [](ListNode* a, ListNode* b) { return a->val > b->val; };
        priority_queue<ListNode*, vector<ListNode*>, decltype(compare)> minHeap(
            compare);

        for (ListNode* list : lists)
            if (list != nullptr)
                minHeap.push(list);

        while (!minHeap.empty()) {
            ListNode* minNode = minHeap.top();
            minHeap.pop();
            if (minNode->next)
                minHeap.push(minNode->next);
            curr->next = minNode;
            curr = curr->next;
        }

        return dummy.next;
    }
};
```

4. Output:



The screenshot shows the LeetCode interface for problem 23, "Merge k Sorted Lists". The problem description states: "You are given an array of k linked-lists lists, each linked-list is sorted in ascending order. Merge all the linked-lists into one sorted linked-list and return it." Example 1 shows input lists = [[1,4,5],[1,3,4],[2,6]] and output [1,1,2,3,4,4,5,6]. Example 2 shows input lists = [] and output [].

The code editor shows a C++ solution using a min-heap to merge the lists. The code is as follows:

```
20 if (list != nullptr)
21     minHeap.push(list);
22 while (!minHeap.empty()) {
23     ListNode* minNode = minHeap.top();
24     minHeap.pop();
25     if (minNode->next)
26         minHeap.push(minNode->next);
27     curr->next = minNode;
28     curr = curr->next;
29 }
30 return dummy->next;
31
```

The test result shows "Accepted" with a runtime of 0 ms. The input is lists = [[1,4,5],[1,3,4],[2,6]] and the output is [1,1,2,3,4,4,5,6].

Problem-5

1. Aim: Sort List

2. Objective: Given the head of a linked list, return the list after sorting it in ascending order.

3. Implementation/Code:

```
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        const int length = getLength(head);
        ListNode dummy(0, head);

        for (int k = 1; k < length; k *= 2) {
            ListNode* curr = dummy.next;
            ListNode* tail = &dummy;
            while (curr != nullptr) {
                ListNode* l = curr;
                ListNode* r = split(l, k);
                curr = split(r, k);
                auto [mergedHead, mergedTail] = merge(l, r);
                tail->next = mergedHead;
                tail = mergedTail;
            }
        }

        return dummy.next;
    }
private:
    int getLength(ListNode* head) {
        int length = 0;
        for (ListNode* curr = head; curr; curr = curr->next)
            ++length;
        return length;
    }
}
```

```
ListNode* split(ListNode* head, int k) {
    while (--k && head)
        head = head->next;

    ListNode* rest = head ? head->next : nullptr;
    if (head != nullptr)
        head->next = nullptr;
    return rest;
}

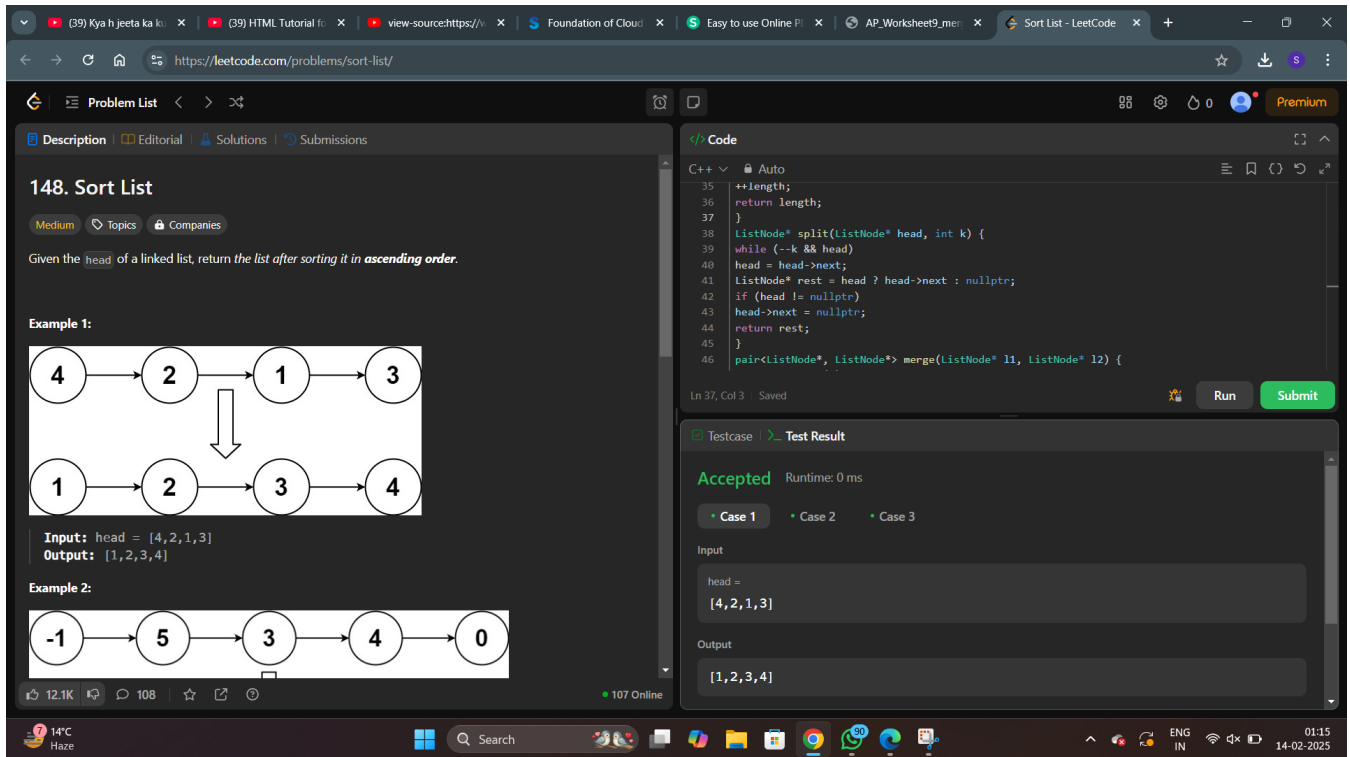
pair<ListNode*, ListNode*> merge(ListNode* l1, ListNode* l2) {
    ListNode dummy(0);
    ListNode* tail = &dummy;

    while (l1 && l2) {
        if (l1->val > l2->val)
            swap(l1, l2);
        tail->next = l1;
        l1 = l1->next;
        tail = tail->next;
    }

    tail->next = l1 ? l1 : l2;
    while (tail->next != nullptr)
        tail = tail->next;

    return {dummy.next, tail};
};
```

4. Output:



The screenshot shows the LeetCode 'Sort List' problem page. The problem description states: "Given the head of a linked list, return the list after sorting it in ascending order." Example 1 shows an input linked list [4, 2, 1, 3] being sorted to [1, 2, 3, 4]. Example 2 shows an input linked list [-1, 5, 3, 4, 0] being sorted to [-1, 3, 4, 5, 0]. The C++ code implements a recursive merge sort algorithm. The test result shows 'Accepted' with a runtime of 0 ms.

148. Sort List

Medium

Given the head of a linked list, return the list after sorting it in ascending order.

Example 1:

Input: head = [4, 2, 1, 3]
Output: [1, 2, 3, 4]

Example 2:

Input: head = [-1, 5, 3, 4, 0]
Output: [-1, 3, 4, 5, 0]

```
C++  
class Solution {  
public:  
    int length;  
    ListNode* split(ListNode* head, int k) {  
        while (--k && head)  
            head = head->next;  
        ListNode* rest = head->next ? head->next : nullptr;  
        if (head != nullptr)  
            head->next = nullptr;  
        return rest;  
    }  
    pair<ListNode*, ListNode*> merge(ListNode* l1, ListNode* l2) {  
        if (!l1) return {l2, nullptr};  
        if (!l2) return {l1, nullptr};  
        if (l1->val < l2->val) {  
            l1->next = merge(l1->next, l2);  
            return {l1, l1->next};  
        } else {  
            l2->next = merge(l1, l2->next);  
            return {l2, l2->next};  
        }  
    }  
    ListNode* sortList(ListNode* head) {  
        if (!head) return head;  
        int len = 0;  
        for (ListNode* p = head; p; p = p->next) len++;  
        return merge(head, split(head, len/2)).first;  
    }  
};
```

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

head = [4, 2, 1, 3]

Output

[1, 2, 3, 4]



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.