Advanced Pragramming

ASSIGNMENT 06

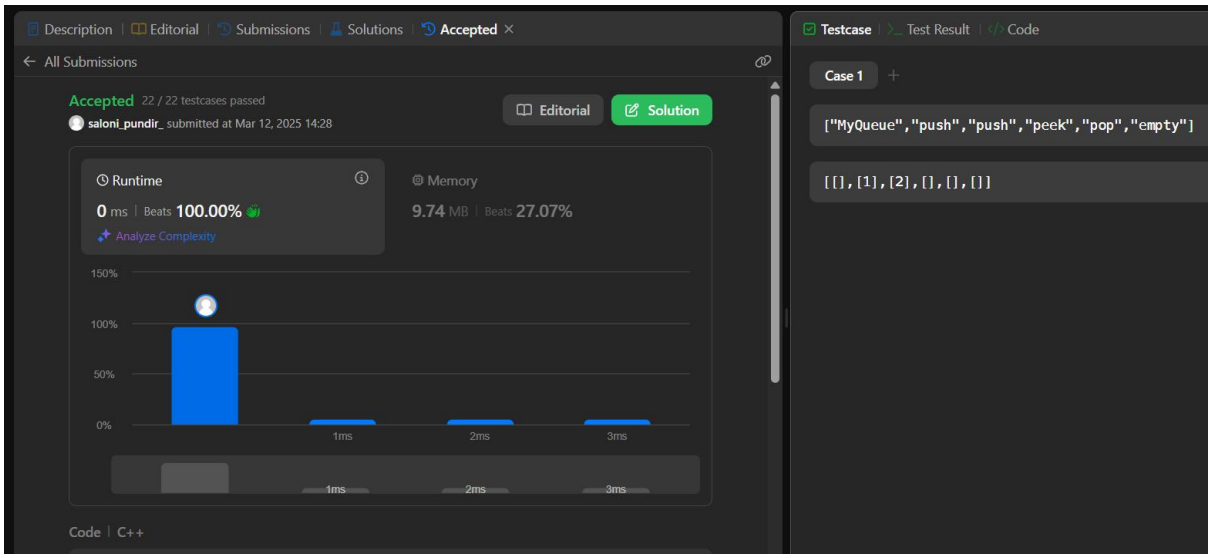
Stack-Based Implementations

Q1. 1.Implement Queue using Stack

Code:

```
class MyQueue {
private:
    stack<int> input;
    stack<int> output;

public:
    MyQueue() {}

    void push(int x) {
        input.push(x);
    }

    int pop() {
        peek();
        int val = output.top();
        output.pop();
        return val;
    }

    int peek() {
        if (output.empty()) {
            while (!input.empty()) {
                output.push(input.top());
                input.pop();
            }
        }
        return output.top();
    }
```

Output:



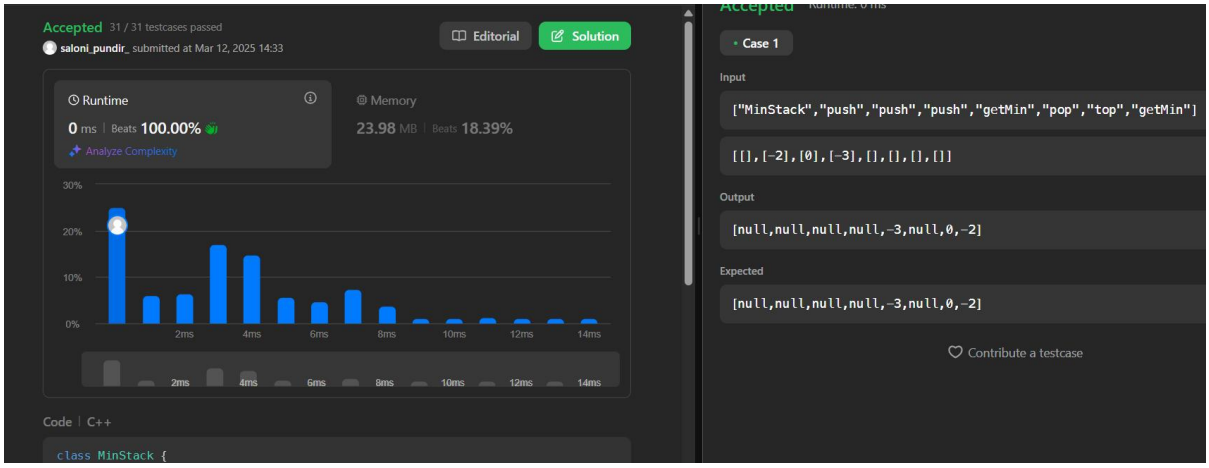
Q2. Implement Min Stack using Two Stacks

Code:

```cpp
class MinStack {
private:
    vector<vector<int>> st;

public:
    MinStack() {

    }

    void push(int val) {
        int min_val = getMin();
        if (st.empty() || min_val > val) {
            min_val = val;
        }
        st.push_back({val, min_val});
    }

    void pop() {
        st.pop_back();
    }

    int top() {
        return st.empty() ? -1 : st.back()[0];
    }

    int getMin() {
        return st.empty() ? -1 : st.back()[1];
    }
};
```

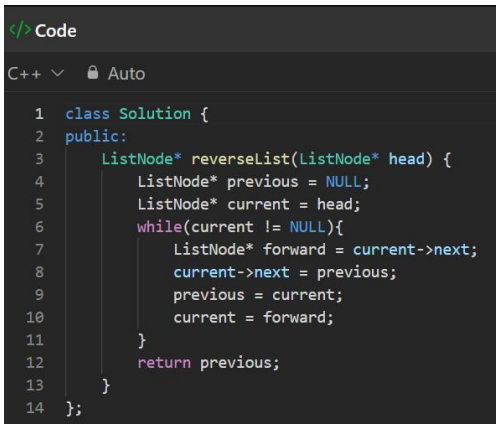
Saved

Output:

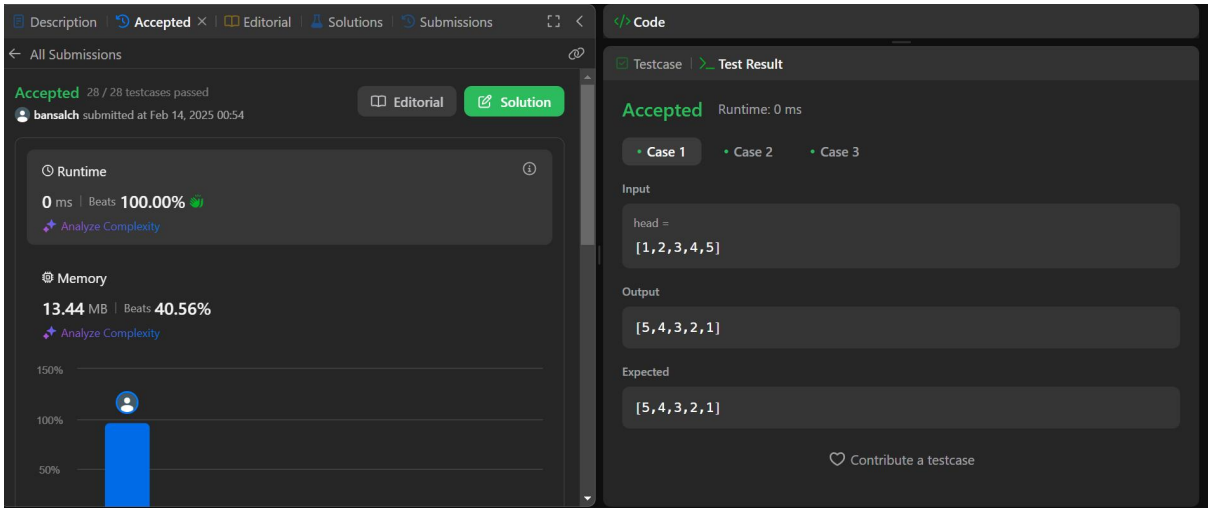


Q3. Implement Max Stack using Two Stacks

Code:



Output:



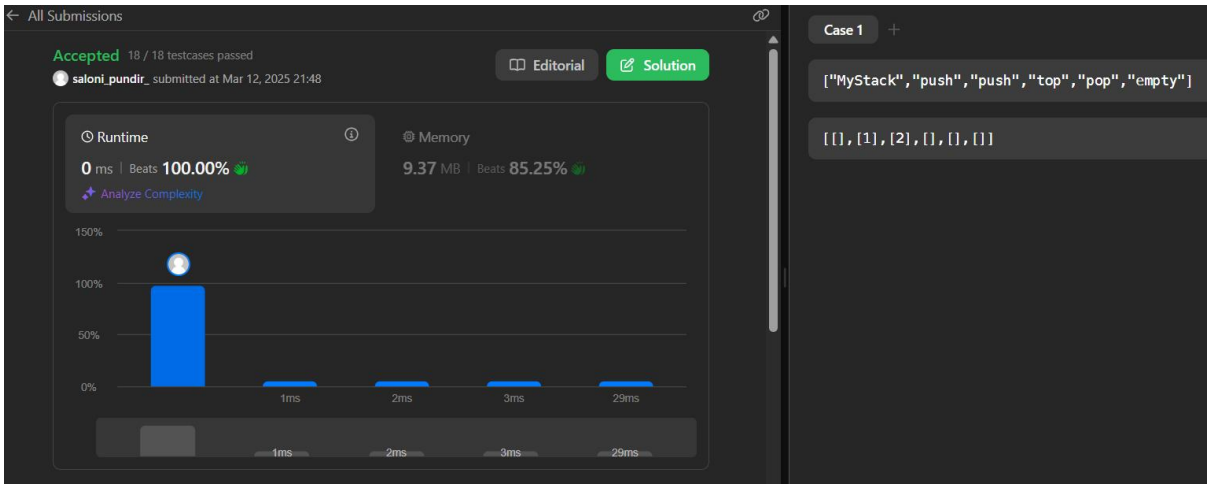
Queue-Based Implementations

Q4. Implement Stack using Queues

Code:

```cpp
class MyStack {
private:
    std::queue<int> q;

public:
    MyStack() {}

    void push(int x) {
        q.push(x);
        for (int i = 0; i < q.size() - 1; i++) {
            q.push(q.front());
            q.pop();
        }
    }

    int pop() {
        int top = q.front();
        q.pop();
        return top;
    }

    int top() {
        return q.front();
    }

    bool empty() {
        return q.empty();
    }
};
```

Output:

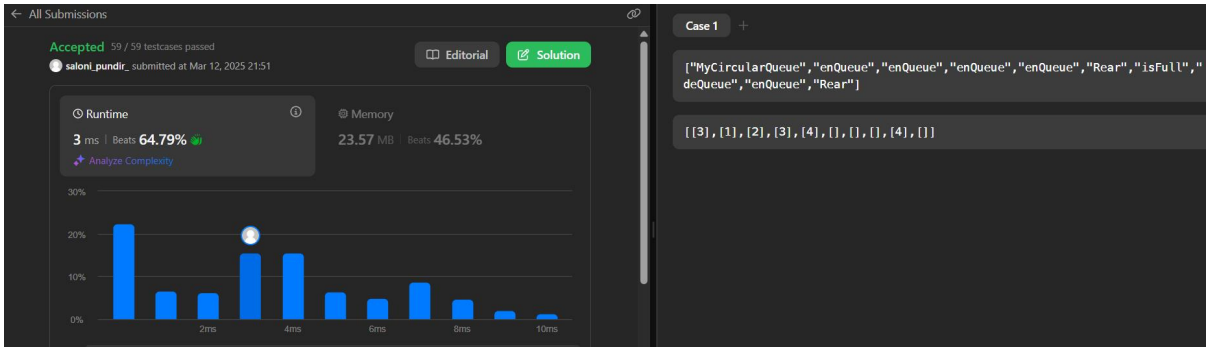


Q5. Implement Circular Queue using Queue

Code:

```cpp
class MyCircularQueue {
private:
    vector<int> v;
    int start = 0, len = 0;
public:
    MyCircularQueue(int k): v(k) {}
    bool enQueue(int value) {
        if (isFull()) return false;
        v[(start + len++) % v.size()] = value;
        return true;
    }
    bool deQueue() {
        if (isEmpty()) return false;
        start = (start + 1) % v.size();
        --len;
        return true;
    }
    int Front() {
        if (isEmpty()) return -1;
        return v[start];
    }
    int Rear() {
        if (isEmpty()) return -1;
        return v[(start + len - 1) % v.size()];
    }
    bool isEmpty() {
        return !len;
    }
    bool isFull() {
```

Output:



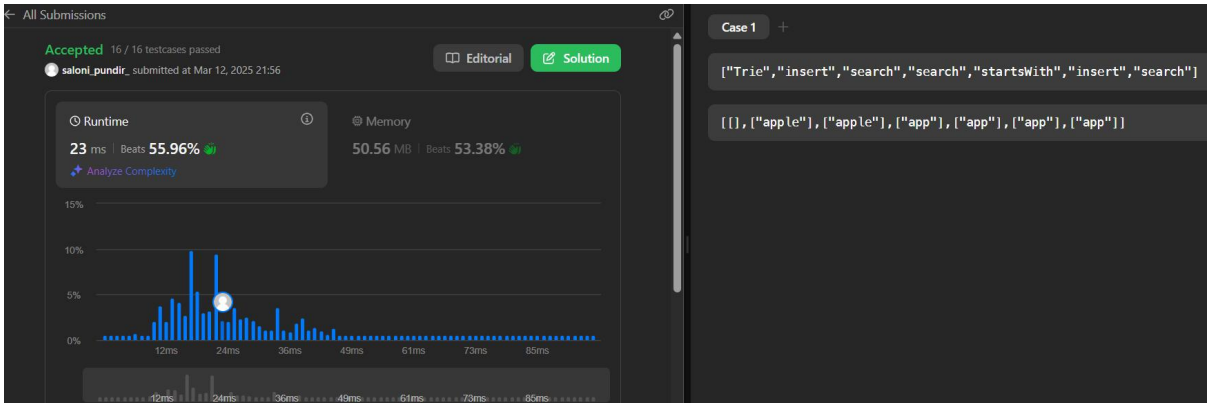
Array-Based Implementations

Q6. Implement Trie (Prefix Tree)

Code:

```cpp
class TrieNode {
public:
    TrieNode *child[26];
    bool isWord;
    TrieNode() {
        isWord = false;
        for (auto &a : child) a = nullptr;
    }
};
class Trie {
    TrieNode* root;
public:
    Trie() {
        root = new TrieNode();
    }
    void insert(string s) {
        TrieNode *p = root;
        for (auto &a : s) {
            int i = a - 'a';
            if (!p->child[i]) p->child[i] = new TrieNode();
            p = p->child[i];
        }
        p->isWord = true;
    }
    bool search(string key, bool prefix=false) {
        TrieNode *p = root;
        for (auto &a : key) {
            int i = a - 'a';
            if (!p->child[i]) return false;
```

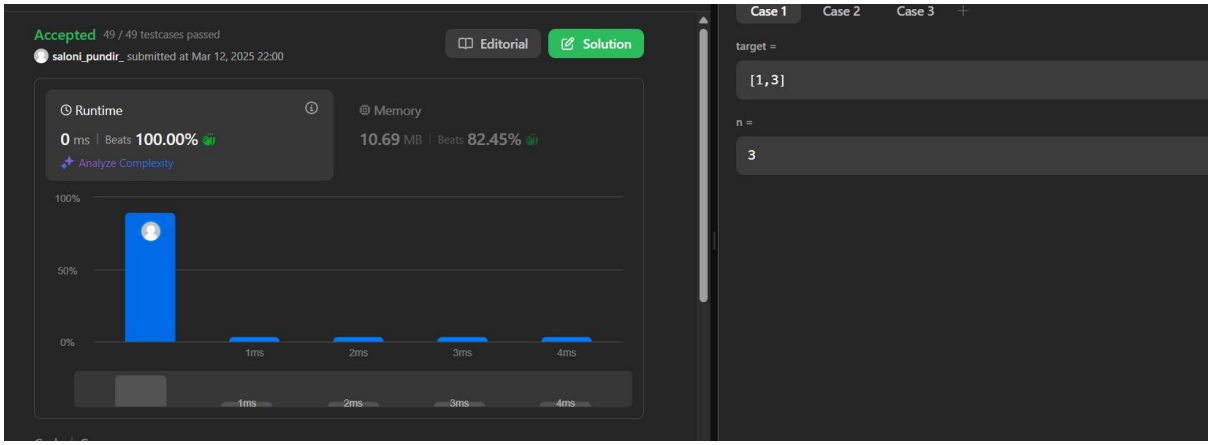
Output:



Q7. Implement Two Stacks in One ArrayCode:

```cpp
class Solution {
public:
    vector<string> buildArray(vector<int>& target, int n) {
        vector<string> ans;
        int j = 0;
        for (int i = 1; i <= n; i++) {
            if (j >= target.size()) break;
            // push
            if (target[j] == i) {
                ans.push_back("Push");
                j++;
            }
            // not req
            else {
                ans.push_back("Push");
                ans.push_back("Pop");
            }
        }

        return ans;
    }
};
```

Output:



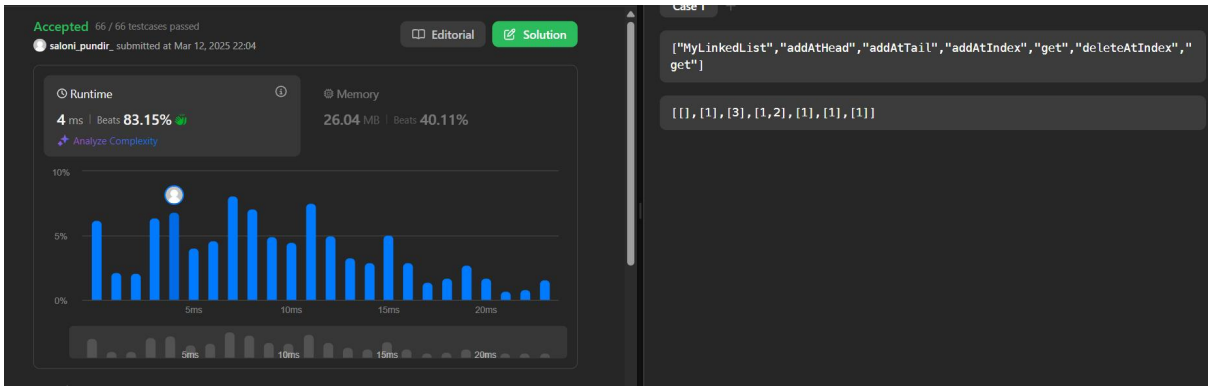
Linked List-Based Implementations

Q8. implement deque using doubly linked list leetcode

Code:

```
int get(int index) {
  Node *t = this->head;
  while (t) {
    if (!index) {
      return t->val;
    }
    index--;
    t = t->next;
  }
  return -1;
}

void addAtHead(int val) {
  Node *node = new Node(val, this->head);
  this->head = node;

  this->size++;
}

void addAtTail(int val) {
  if (this->head == nullptr) {
    this->head = new Node(val);
    this->size++;
    return;
  }

  Node *t = this->head;
  while (t->next) {
    t = t->next;
```

Output:



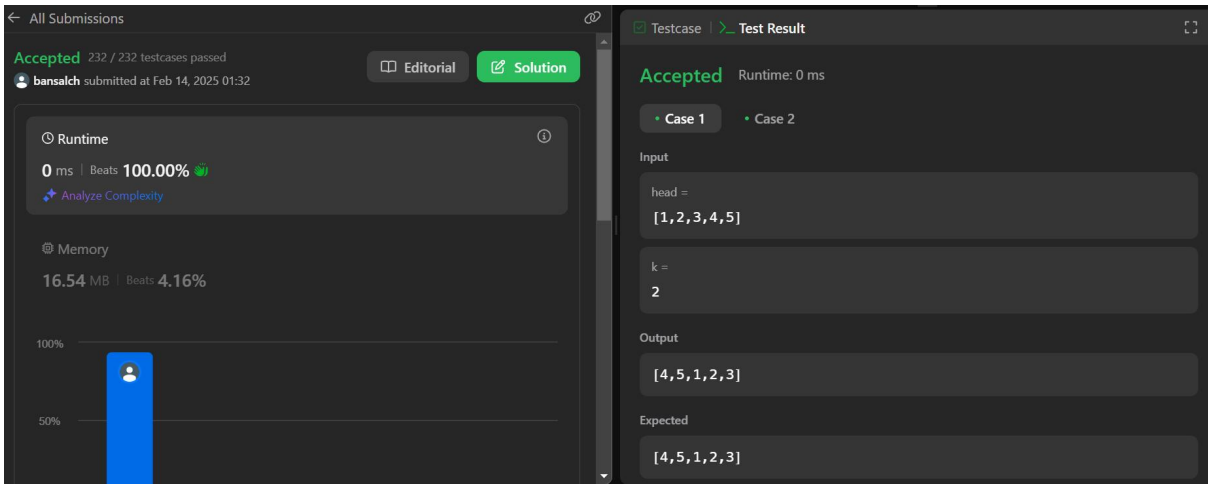
Heap-Based Implementations

Q9. Rotate a list.

Code:

```cpp
C++ Auto
13      ListNode* rotateRight(ListNode* head, int k) {
14          if (!head || !head->next || k == 0) return head;
15          ListNode* curr = head;
16          int length = 1;
17          while (curr->next) {
18              curr = curr->next;
19              length++;
20          }
21          curr->next = head;
22          k = k % length;
23          int stepsToNewHead = length - k;
24          ListNode* newTail = head;
25          for (int i = 1; i < stepsToNewHead; ++i) {
26              newTail = newTail->next;
27          }
28          ListNode* newHead = newTail->next;
29          newTail->next = nullptr;
30          return newHead;
31      }
32  };
```

Output:

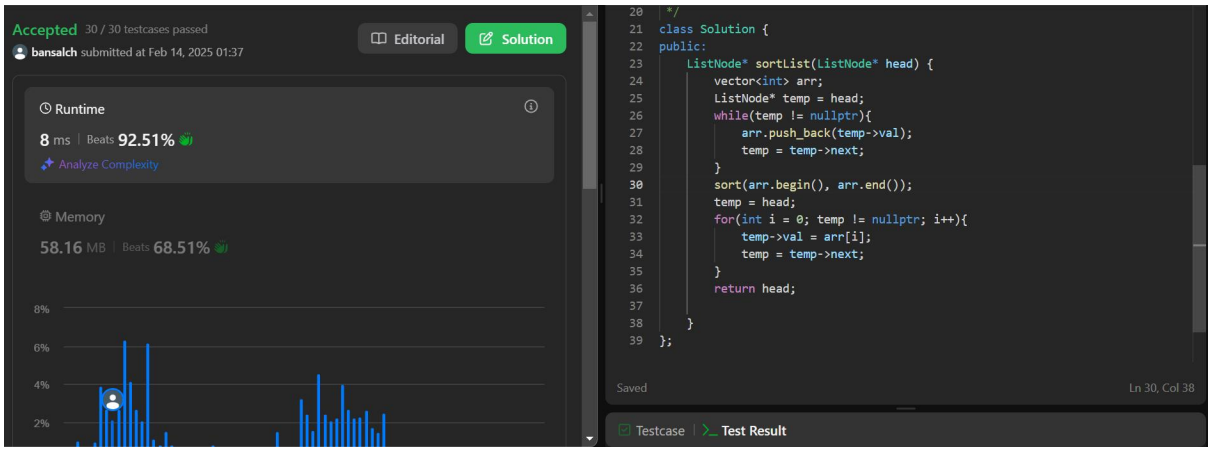


Q10. Sort list

Code:

```cpp
*/
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        vector<int> arr;
        ListNode* temp = head;
        while(temp != nullptr){
            arr.push_back(temp->val);
            temp = temp->next;
        }
        sort(arr.begin(), arr.end());
        temp = head;
        for(int i = 0; temp != nullptr; i++){
            temp->val = arr[i];
            temp = temp->next;
        }
        return head;

    }
};
```

Output:

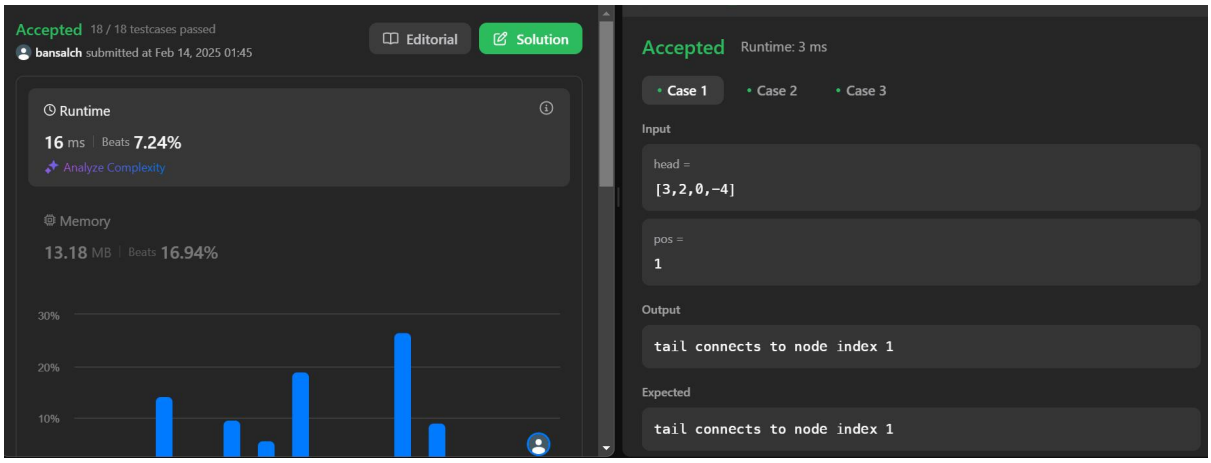


Q11. Detect a cycle in linked list 2.

Code:

```cpp
class Solution {
public:
    ListNode *detectCycle(ListNode *head) {
        unordered_set<ListNode*> visited;
        ListNode* current = head;
        while (current != nullptr) {
            if (visited.find(current) != visited.end()) {
                return current;
            }
            visited.insert(current);
            current = current->next;
        }
        return nullptr;
    }
};
```

Output:


Accepted 18 / 18 testcases passed
bansalch submitted at Feb 14, 2025 01:45
Editorial
Solution
Runtime
16 ms | Beats 7.24%
Analyze Complexity
Memory
13.18 MB | Beats 16.94%
30%
20%
10%
Accepted Runtime: 3 ms
Case 1
Case 2
Case 3
Input
head =
[3,2,0,-4]
pos =
1
Output
tail connects to node index 1
Expected
tail connects to node index 1