Question No. 70 :-
https://leetcode.com/problems/climbing-stairs/description/
Solution Link :-
https://leetcode.com/problems/climbing-stairs/submissions/1390455348/
Description :-

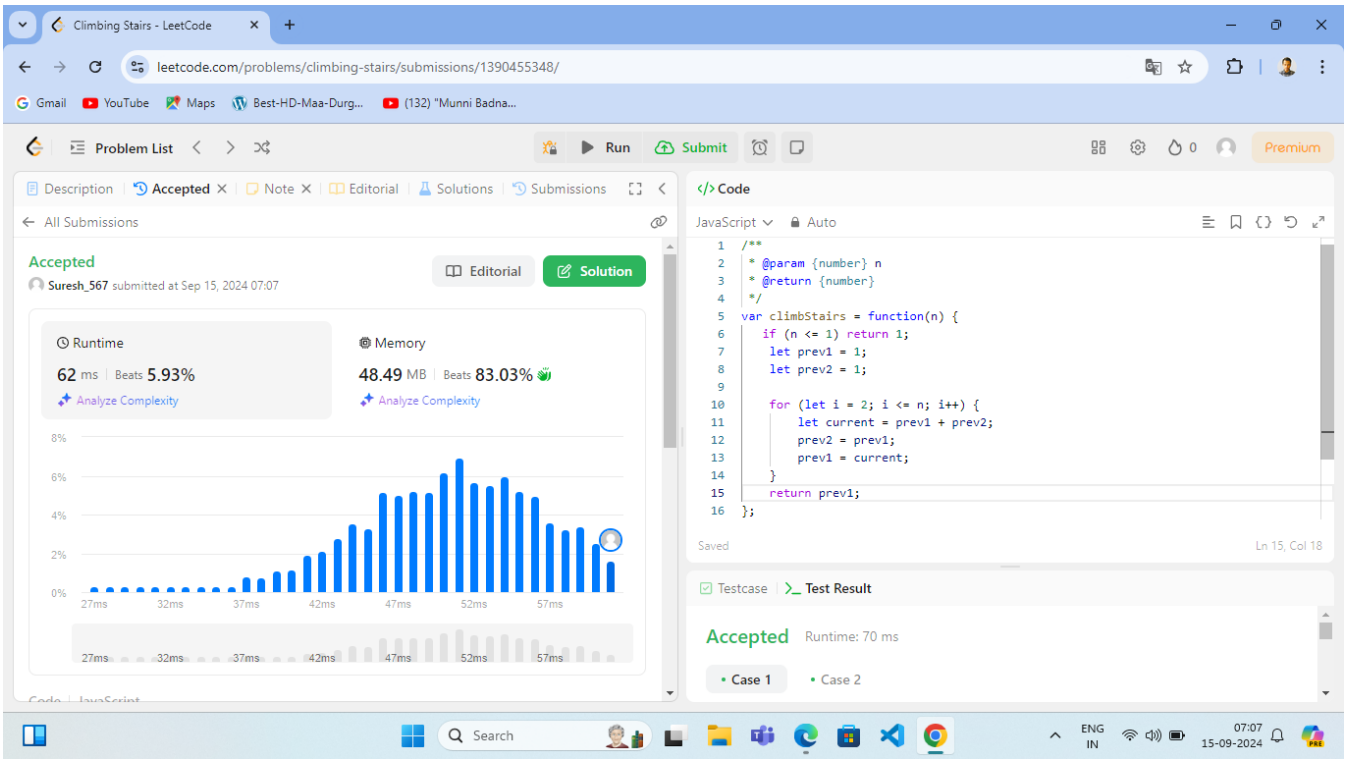
Time Complexity :- O(n)

This solution uses a loop that runs from 2 to n. Hence, the number of iterations is n - 1, making the time complexity **O(n)**.

Space Complexity :- O(1)

This implementation only uses a few variables (prev1, prev2, and current) regardless of the value of n. Therefore, the space required is constant.

Screenshot :-



Question No. 21 :-

https://leetcode.com/problems/merge-two-sorted-lists/description/

Solution Link :-

https://leetcode.com/problems/merge-two-sorted-lists/submissions/1390462118/

Description :-

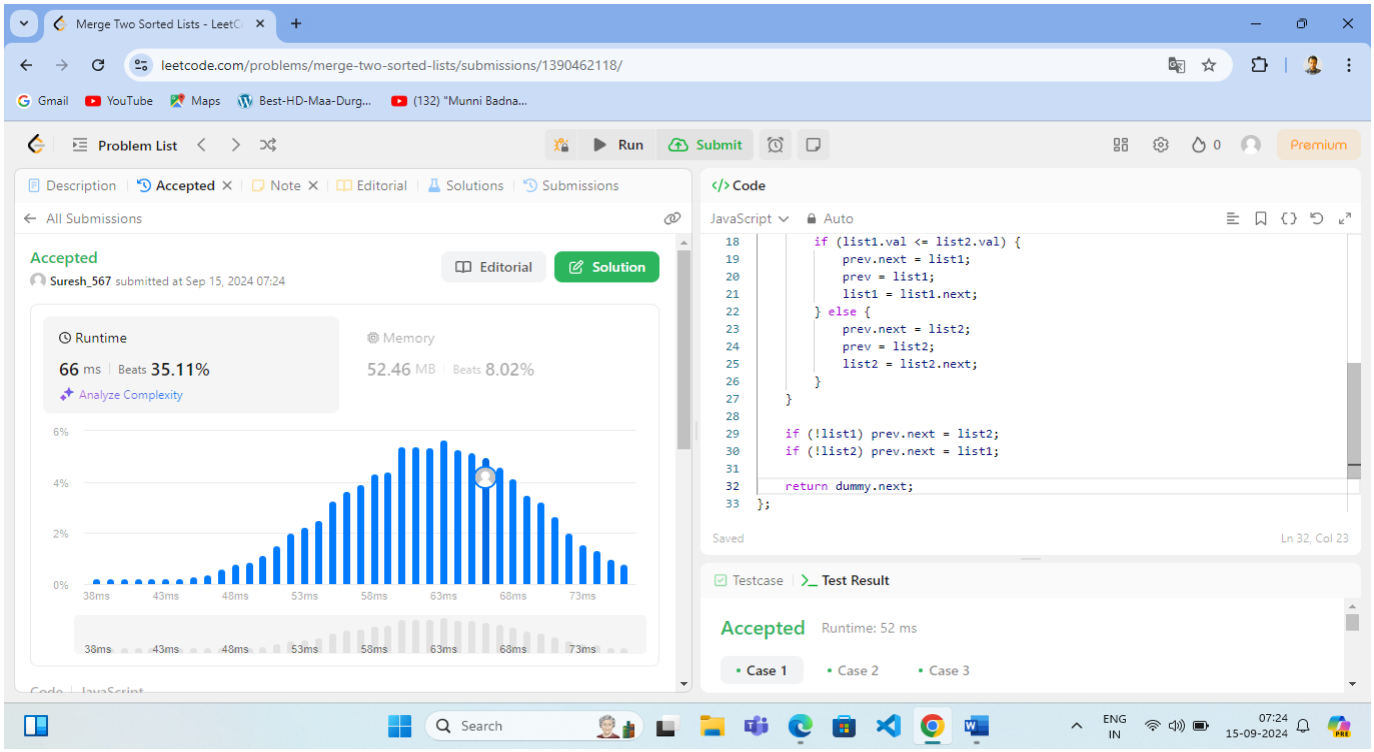
Time Complexity :- O(m+n)

Both lists is processed exactly once. If m is the length of list1 and n is the length of list2, the number of iterations is m + n.

Space Complexity :- O(1)

This implementation uses only a few pointers (dummy, prev, list1, and list2), all of which are constant space.

Screenshot :-



Question No. 234 :-

https://leetcode.com/problems/palindrome-linked-list/description/

Solution Link :-

https://leetcode.com/problems/palindrome-linked-list/submissions/1390469613/

Description :-

Time Complexity :- O(n)

Step 1: Reversing the first half of the list while finding the middle takes **O(n/2)**.

Step 2: If the list has an odd number of elements, the fast pointer helps skip the middle node in constant time **O(1)**.

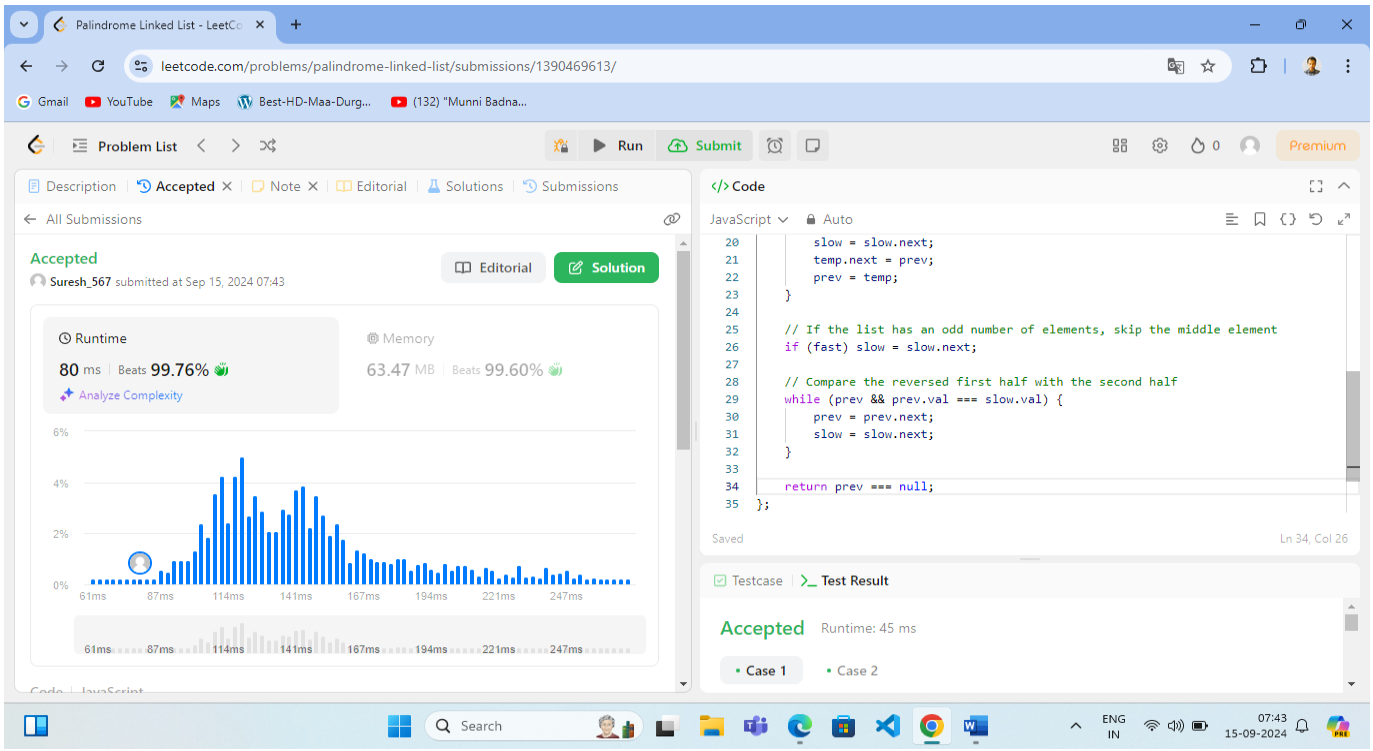
Step 3: Comparing the reversed first half and the second half takes **O(n/2)**.

So, The overall time complexity is **O(n)**.

Space Complexity :- O(1)

The space complexity is **O(1)** because this implementation reverses the first half of the list in place, using constant extra space for variables (slow, fast, prev, temp, etc.).

Screenshot :-



Question No. 141 :- https://leetcode.com/problems/linked-list-cycle/description/

Solution Link :-

https://leetcode.com/problems/linked-list-cycle/submissions/1390474350/

Description :-

Time Complexity :- O(n)

The fast pointer moves two steps at a time, and the slow pointer moves one step at a time. In the worst case (when a cycle exists), both pointers will eventually meet at some point inside the

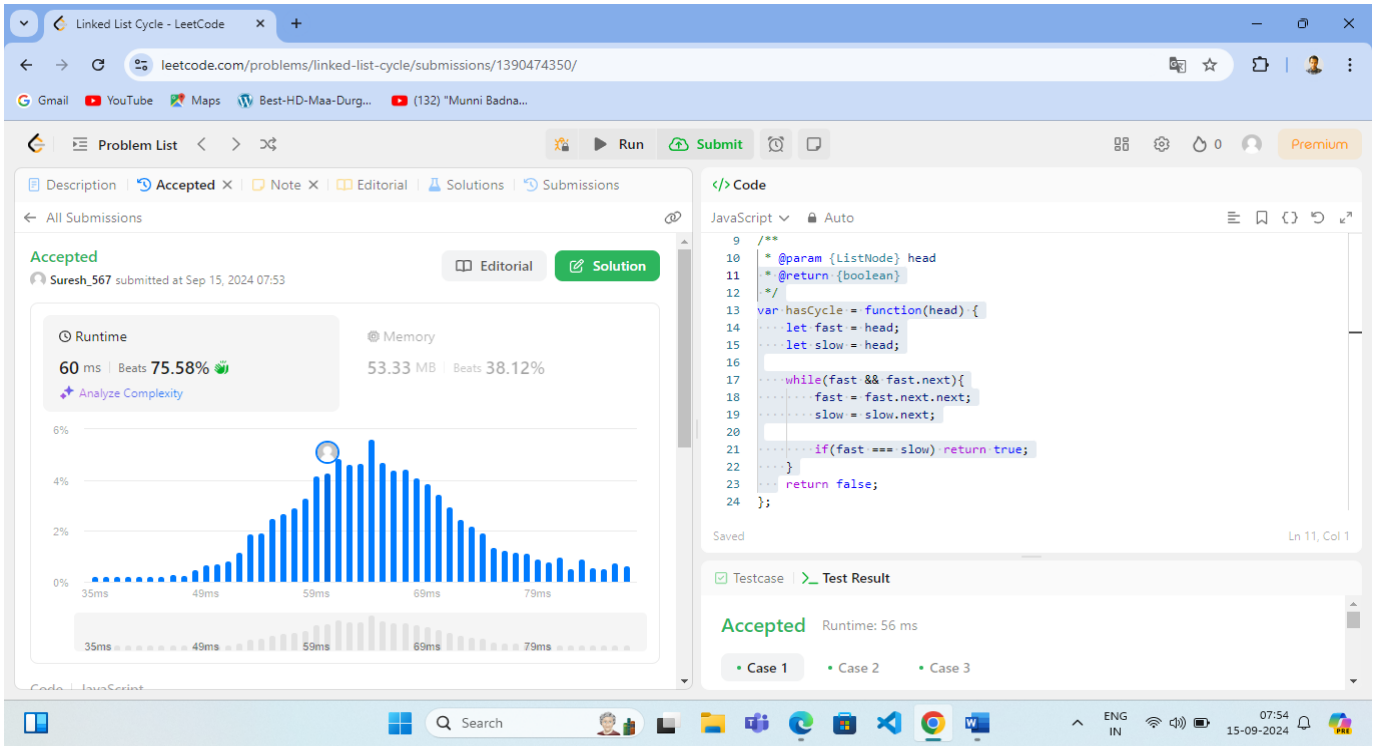
cycle. This process takes **O(n)** time, where n is the number of nodes in the linked list.

If no cycle exists, both pointers will traverse the entire list, which also takes **O(n)**.

Space Complexity :- O(1)

The space complexity is **O(1)** because only a few pointers (fast and slow) are used, and which are constant.

Screenshot :-



Question No. 19:-

https://leetcode.com/problems/remove-nth-node-from-end-of-list/description/

Solution Link :-

https://leetcode.com/problems/remove-nth-node-from-end-of-list/submissions/1390499610/

Description :-

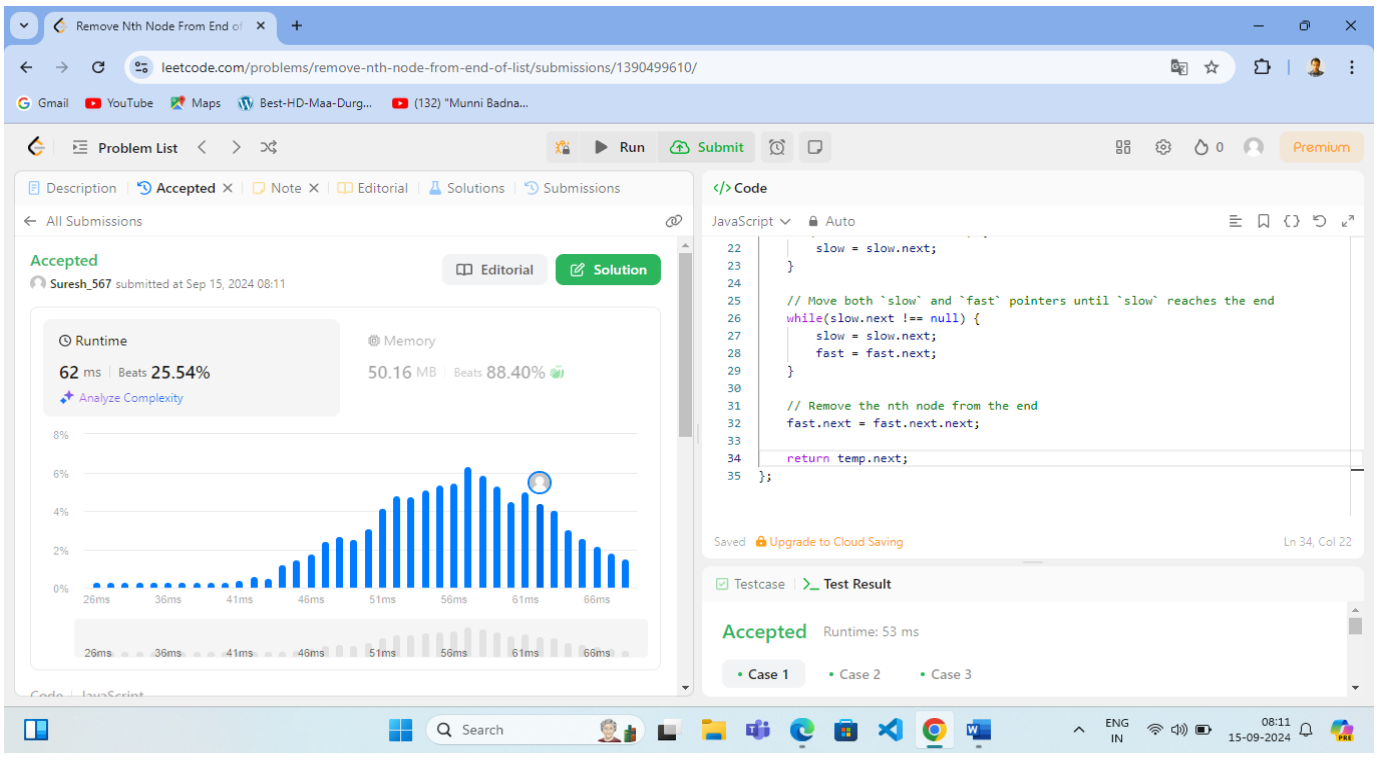
Time Complexity :- O(n)

This is a single pass over the list, and the time complexity is **O(n)**, where n is the length of the linked list.

Space Complexity :- O(1)

This implementation uses a few extra pointers (temp, slow, fast), all of which take constant space.

Screenshot :-



Question No. 50:-

https://leetcode.com/problems/powx-n/description/

Solution Link :- https://leetcode.com/problems/powx-n/submissions/1390518486/

Description :-

Time Complexity :- O(log n)

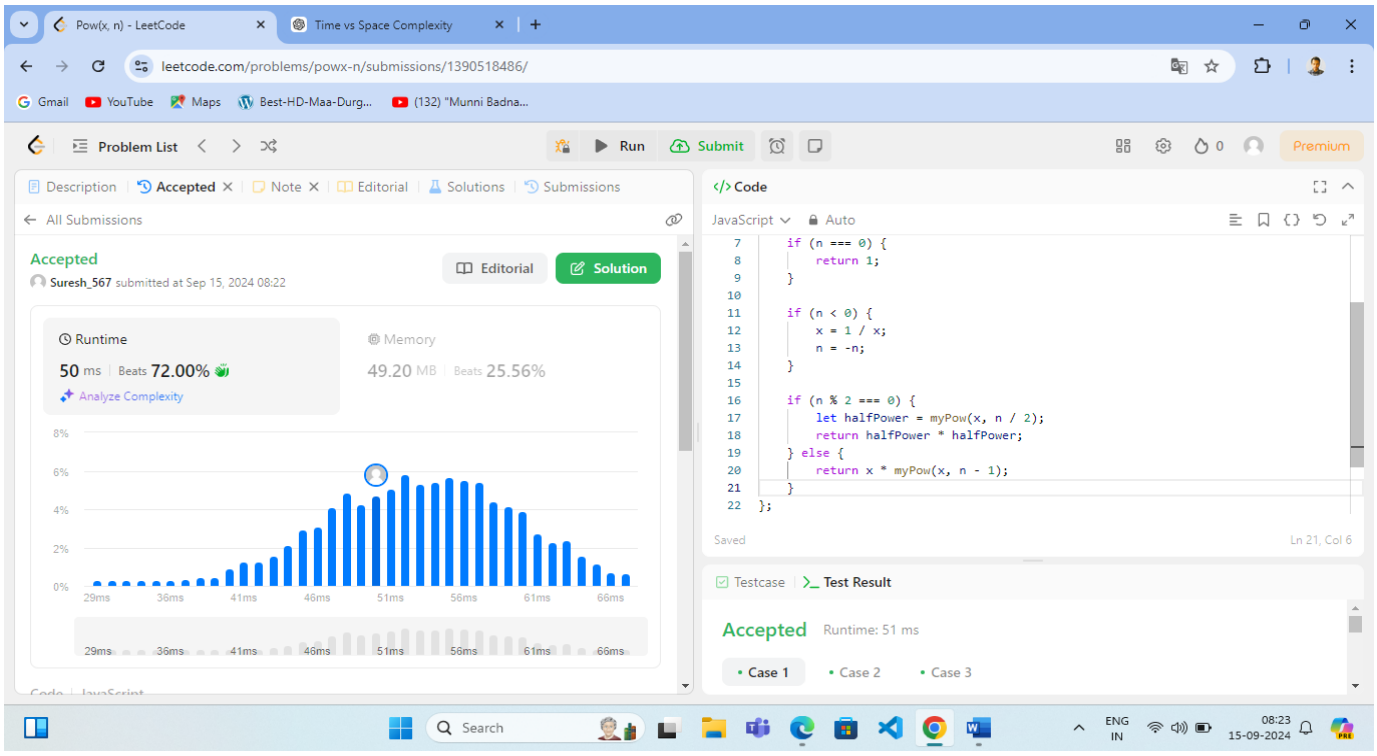
For each recursion, the exponent n is divided in two parts, leading to a logarithmic reduction in the number of calls. So, the time complexity is **O(log n)**.

Space Complexity :- O(log n)

The maximum depth of recursion is proportional to the number of times n is divided in two parts, which is **O(log n)**.

Thus, the space complexity is **O(log n)** due to the recursive call stack.

Screenshot :-



Question No. 237:-

https://leetcode.com/problems/delete-node-in-a-linked-list/description/

Solution Link :-

https://leetcode.com/problems/delete-node-in-a-linked-list/submissions/1390554473/

Description :-

Time Complexity :- O(1)

The time complexity is **O(1)** because the operations (copying the value and updating the pointer) take constant time.

Space Complexity :- O(1)

The space complexity is **O(1)** since no additional memory is used beyond the input node.

Screenshot :-