DATA STRUCTURE AND ALGORITHM

ASSIGNIMENT-3

Question-1

Climbing Stairs

You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

Input: n = 2

Output: 2

Explanation: There are two ways to climb to the top.

1. 1 step + 1 step

2. 2 steps

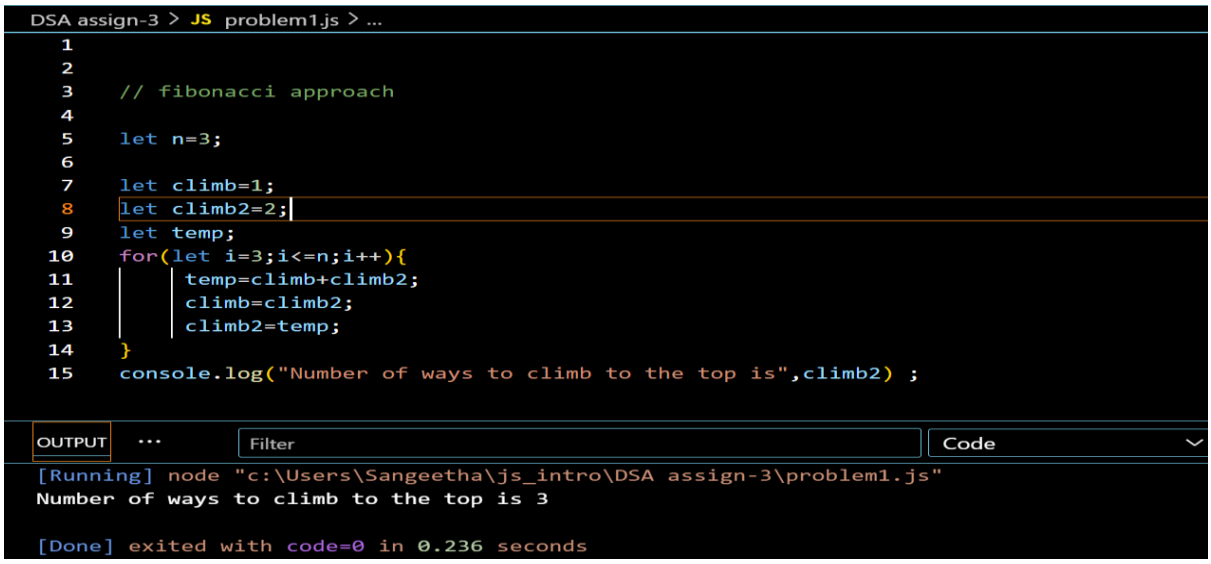
Question link

https://leetcode.com/problems/climbing-stairs/description/

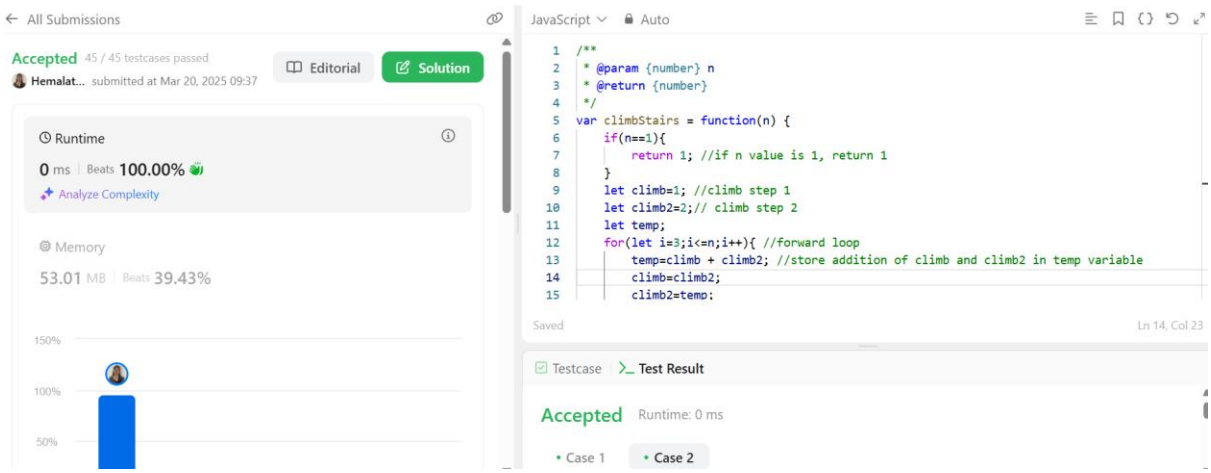
Code:

```
let n=3;
let climb=1;
let climb2=2;
let temp;
```

```
for(let i=3;i<=n;i++){
    temp=climb+climb2;
    climb=climb2;
    climb2=temp;
}
console.log("Number of ways to climb to the top is",climb2) ;
```



Submission screenshot



Link

https://leetcode.com/problems/climbing-stairs/submissions/1579918252/

Conclusion

Time complexity O(n)

In this code ,we used Fibonacci approach .The loop runs (n-2) times. So the final time complexity is O(n).

Space complexity O(1) –No extra space has taken .In this case we use extra variable climb,climb2 and temp .These takes only constant space.

Problem 2

Merge two sorted linked list

You are given the heads of two sorted linked lists list1 and list2.Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists. Return the head of the merged linked list.

Example

Input: list1 = [1,2,4], list2 = [1,3,4]

Output: [1,1,2,3,4,4].

Question link

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

Code

```
class Node{
    constructor(data){
        this.data=data;
        this.next=null;
    }
}

let head1=new Node(1);
let first1=new Node(2);
let second1=new Node(4);

let head2=new Node(1);
let first2=new Node(3);
```

```
let second2=new Node(4);

head1.next=first1;
first1.next=second1;
second1.next=null;

head2.next=first2;
first2.next=second2;
second2.next=null;

function printList(head) {
    let temp = head;
    let result = "";
    while (temp !== null) {
        result += temp.data + " -> ";
        temp = temp.next;
    }
    console.log(result + "null");
}


console.log("List 1:");
printList(head1);

console.log("List 2:");
printList(head2);
```

```
// Merge two linked list
function mergeTwoLinkedList(head1,head2){
    let temporary=new Node(0);
    let head=temporary;



    while(head1 && head2){
        if(head1.data<=head2.data){ //if list1 data is equal or less than list2
            temporary.next=head1;
            head1=head1.next; // move forward
        }else{
            temporary.next=head2;
            head2=head2.next; // move forward
        }
        temporary=temporary.next;
    }

    if(head1 != null){ // if list1 is not equal to null
        temporary.next=head1;
    }else{
        temporary.next=head2;
    }

    return head.next;
}

let mergedHead = mergeTwoLinkedList(head1, head2);
```

console.log("Merged List:");

printList(mergedHead);

DSA assign-3 > JS problem2.js > ...

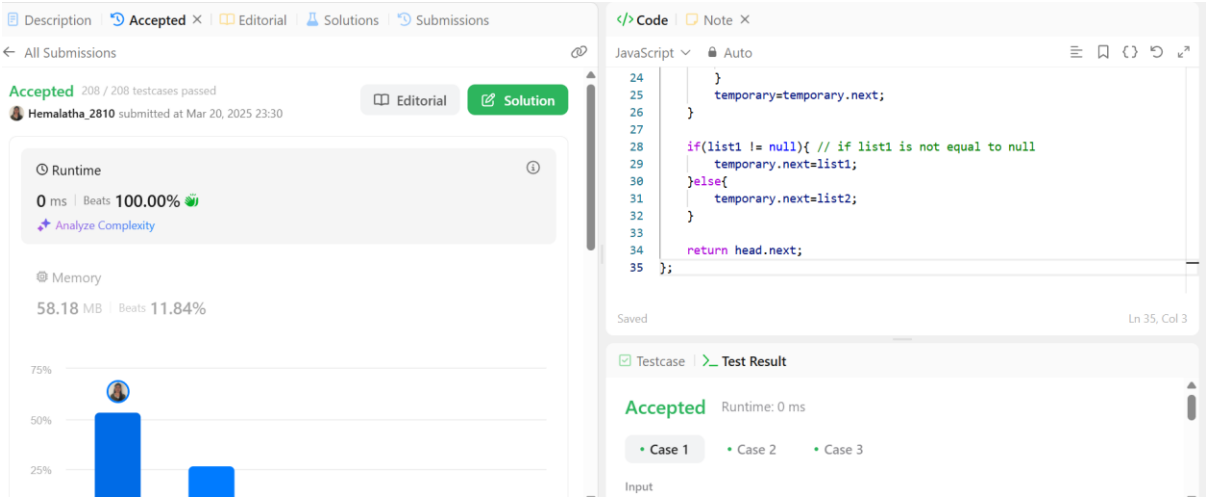
```javascript
class Node{
    constructor(data){
        this.next=null;
    }
}

let head1=new Node(1);
let first1=new Node(2);
let second1=new Node(4);

let head2=new Node(1);
let first2=new Node(3);
let second2=new Node(4);

```

OUTPUT ··· Filter Code

```
[Running] node "c:\Users\Sangeetha\js_intro\DSA assign-3\problem2.js"
List 1:
1 -> 2 -> 4 -> null
List 2:
1 -> 3 -> 4 -> null
Merged List:
1 -> 1 -> 2 -> 3 -> 4 -> 4 -> null
```

Submission screenshot

https://leetcode.com/problems/merge-two-sorted-lists/submissions/1580547088/



Conclusion

Time complexity O(m+n)

In this problem, where m and n are the lengths of list1 and list2 (since we iterate through both lists once).

Space complexity O(m+n)

when merging two lists with m and n nodes, there are **O(m + n)** recursive calls.

Problem 3

Palindrome linked list

Given the head of a singly linked list, return true if it is a palindrome or false otherwise.

Example

Input: head = [1, 2, 2, 1]

Output: true

Question link

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

Code:

```
class Node{
    constructor(data){
        this.data=data;
        this.next=null;
    }
}
let head=new Node(1);
let first=new Node(2);
let second=new Node(2);
let third=new Node(1);


head.next=first;
```

```
first.next=second;
second.next=third;
third.next=null;


function palindromeList(head){
if (!head || !head.next) {
    return true;
}
let slow = head;
let fast = head;
while (fast && fast.next) {
    slow = slow.next;
    fast = fast.next.next;
}
let previous = null;
let current = slow;
while (current) {
    let nextNode = current.next;
    current.next = previous;
    previous = current;
    current = nextNode;
}
```

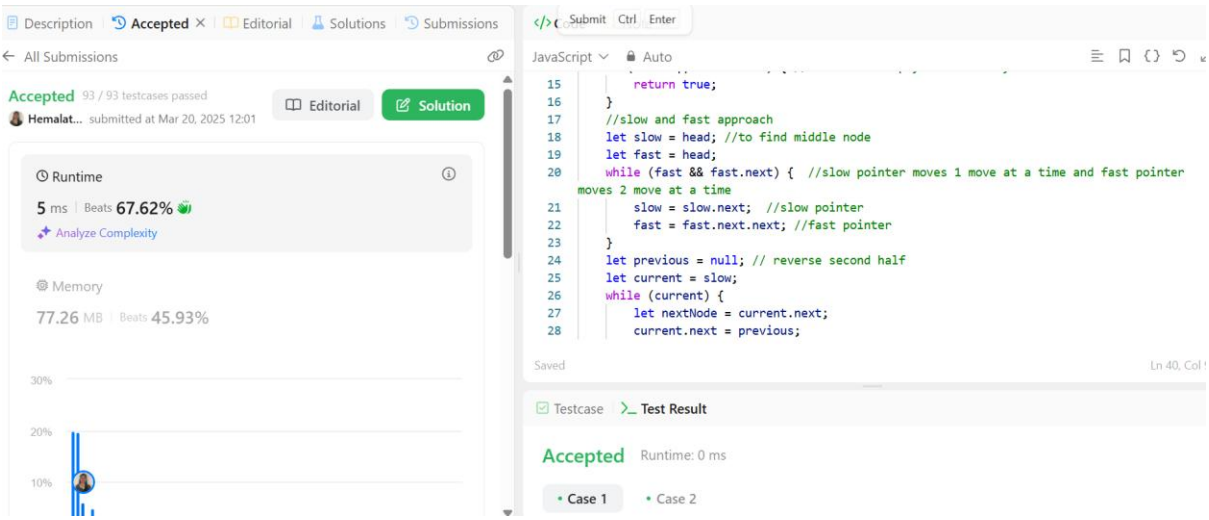
```
let firstHalf = head;

let secondHalf = previous;

    while (secondHalf) {

        if (firstHalf.data !== secondHalf.data) {

            return false;

        }

        firstHalf = firstHalf.next;

        secondHalf = secondHalf.next;

    }

return true;

}

const result=palindromeList(head);

console.log(result);
```

```
4
5   class Node{
6       constructor(data){
7           this.data=data;
8           this.next=null;
9       }
10  }
11  let head=new Node(1);
12  let first=new Node(2);
13  let second=new Node(2);
14  let third=new Node(1);
15
16  head.next=first;
```

OUTPUT ··· Filter Code

```
[Done] exited with code=0 in 0.124 seconds

[Running] node "c:\Users\Sangeetha\js_intro\DSA assign-3\problem3.js"
true
```

Submission screenshot



Link

https://leetcode.com/problems/palindrome-linked-list/submissions/1579911710/

Conclusion

Time complexity O(n)

- Finding middle takes O(n)
- Reversing process for second half takes O(n)
- O(n) + O(n) =O(n)

Space complexity O(1)

- No extra space required .

Problem 4

Cycle linked list

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

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).

Question link

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

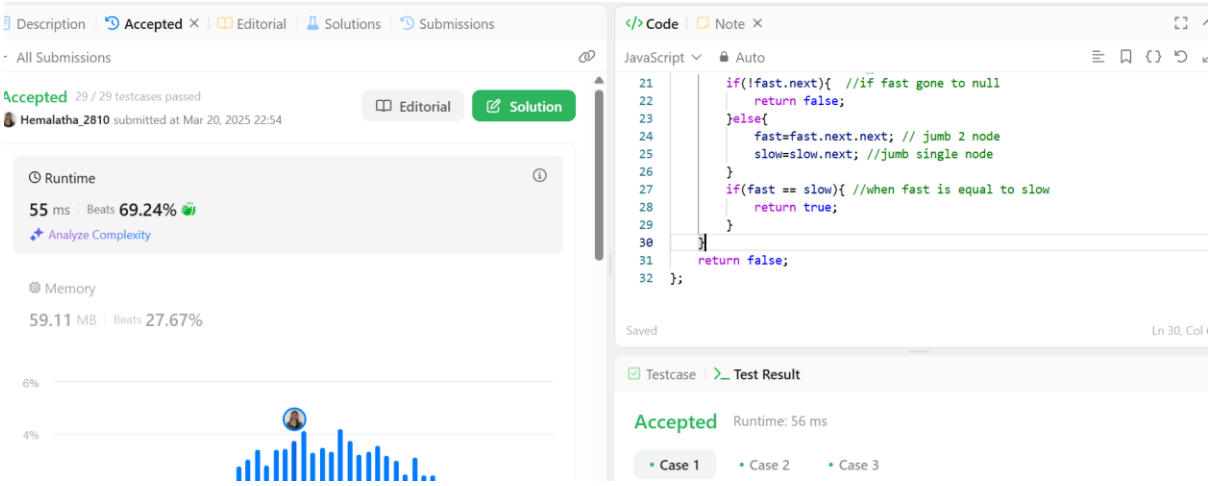
Code

```
if(!head){
        return false; //if list is empty
    }
    let fast=head; //pointer moves fast
    let slow=head; //pointer moves slow
```

```
while(fast && fast.next){
    if(!fast.next){  //if fast gone to null
        return false;
    }else{
        fast=fast.next.next; // jumb 2 node
        slow=slow.next; //jumb single node
    }
    if(fast == slow){ //when fast is equal to slow
        return true;
    }
}
return false;
```

Submission link

https://leetcode.com/problems/linked-list-cycle/submissions/1580505470/



Conclusion

Time complexity O(n)

Each node is visited twice time and run until length of circle linked list

Space complexity O(1)

No extra space is required

Problem 5

Remove nth Node from end of list

Given the head of a linked list, remove the nth node from the end of the list and return its head.

Example

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

Output: [1,2,3,5]

Question link

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

Code

```
// remove nth node from end of list
class Node{
    constructor(data){
        this.data=data;
        this.next=null;
    }
}
let head=new Node(1);
let first=new Node(2);
let second=new Node(3);
```

```
let third=new Node(4);
let fourth=new Node(5);

head.next=first;
first.next=second;
second.next=third;
third.next=fourth;
fourth.next=null;

 function removeNode(head, n) {
    if(head==null || head.next== null ){
        return null;
    }
    let first=new Node(-1);
    first.next=head;
    let second=first;

    for(let i=0;i<n;i++){
        if(!second.next){
            return head;
        }
        second=second.next;
```

```
    }
    while(second.next){
        first=first.next;
        second=second.next;
    }
    if(first.next==head){
        head=head.next;
        return head;
    }
    first.next=first.next.next;
    return head;

}
const list=removeNode(head,2);
function printList(list){
    let current=list;
    while(current !=null){
        console.log(current.data);
        current=current.next;
    }
}
printList(list);
```

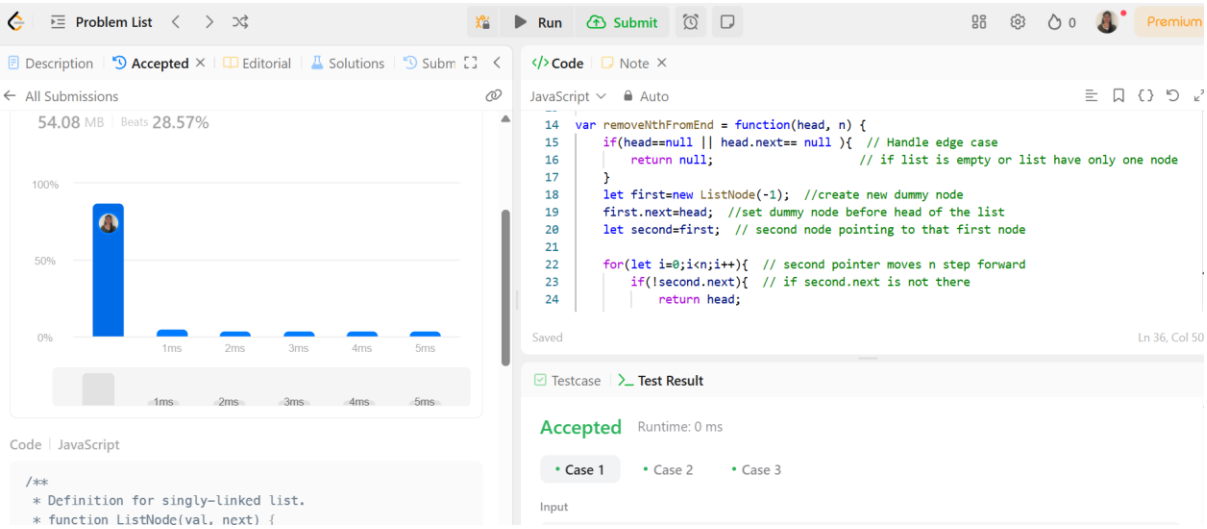
```javascript

const list=removeNode(head,2);

function printList(list){
    let current=list;
    while(current !=null){
        console.log(current.data);
        current=current.next;
    }
}
printList(list);

```

OUTPUT ··· Filter Code

```
[Running] node "c:\Users\Sangeetha\js_intro\DSA assign-3\problem5.js"
1
2
3
5
```

Leetcode submission link

https://leetcode.com/problems/remove-nth-node-from-end-of-list/submissions/1580121999/



Conclusion

I used two pointer technique in this problem.

Time complexity O(n) – we traverse the list once ,run until end node

Space complexity O(1) – No extra space required

Problem 6

Pow (x,n)

Implement pow(x, n), which calculates x raised to the power n (i.e., xn).

Example 1:

Input: x = 2.00000, n = 10

Output: 1024.00000

Example 2:

Input: x = 2.10000, n = 3

Output: 9.26100

Question link

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

Code

```
let x=2.000;

let n=10;

if(n==0){

    return 1;

}else if(n<0){

    x=1/x;

    n=-n;

}

    let result=1;
```

```
while (n > 0) {
    if (n % 2 !== 0) {
        result *= x;
    }
    x = x*x;
    n = Math.floor(n / 2);
}
console.log("Ans of x power n is ",result) ;
```

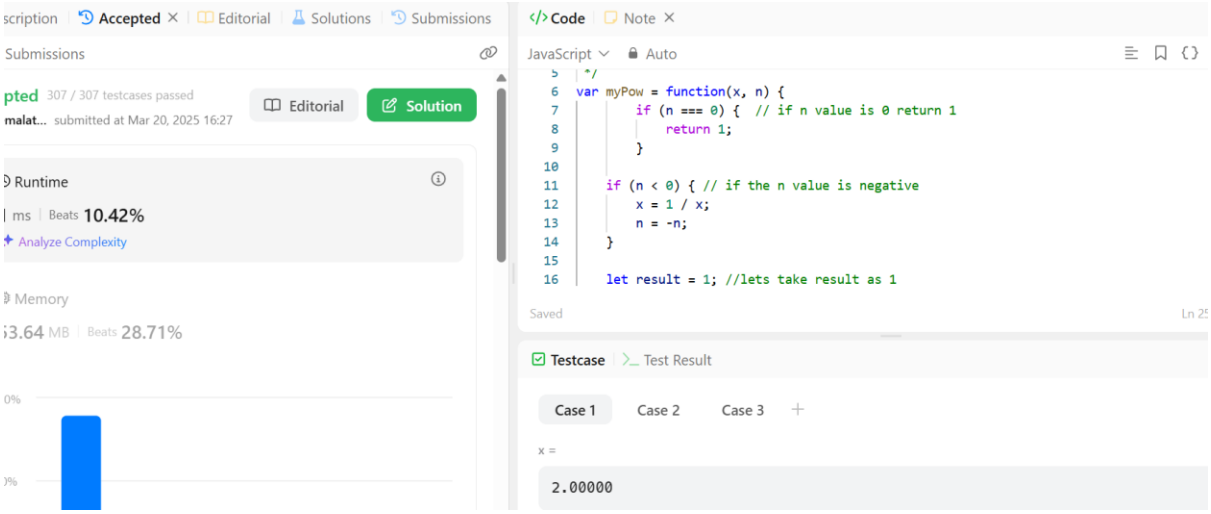
```
if(n==0){
    return 1;
}else if(n<0){
    x=1/x;
    n=-n;
}
    let result=1;
    while (n > 0) {
        if (n % 2 !== 0) {
            result *= x;
        }
        x = x*x;
        n = Math.floor(n / 2);
```

OUTPUT ··· Filter Code

```
[Running] node "c:\Users\Sangeetha\js_intro\DSA assign-3\problem6.js"
Ans of x power n is  1024

[Done] exited with code=0 in 0.144 seconds
```

Submission link

https://leetcode.com/problems/powx-n/submissions/1580142556/



Conclusion

Time complexity O(log n)

We applied exponential process.

Space complexity O(1)

No extra space required to run this code.

Problem 7

Delete node in linked list

There is a singly-linked list head and we want to delete a node node in it.You are given the node to be deleted node. You will not be given access to the first node of head.All the values of the linked list are unique, and it is guaranteed that the given node node is not the last node in the linked list.Delete the given node. Note that by deleting the node, we do not mean removing it from memory. We mean:The value of the given node should not exist in the linked list.

The number of nodes in the linked list should decrease by one.

All the values before node should be in the same order.

All the values after node should be in the same order.

Example

Input: head = [4,5,1,9], node = 5

Output: [4,1,9]

Explanation: You are given the second node with value 5, the linked list should become 4 -> 1 -> 9 after calling your function.

Question link

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

Code

```
class Node{
    constructor(data){
        this.data=data;
```

```
        this.next=null;
    }
}

let head=new Node(4);
let first=new Node(5);
let second=new Node(1);
let third =new Node(2);

head.next=first;
first.next=second;
second.next=third;
third.next=null;

function deleteNode(node){
    if (node == null || node.next == null) {
        console.log("Cannot delete the last node");
        return;
    }
    node.data=node.next.data;
    node.next=node.next.next;
}
```

```
deleteNode(first);


function printList(head){
    let current=head;
    while(current !=null){
        console.log(current.data);
        current=current.next;
    }
}
printList(head);
```

DSA assign-3 > JS problem7.js > printList

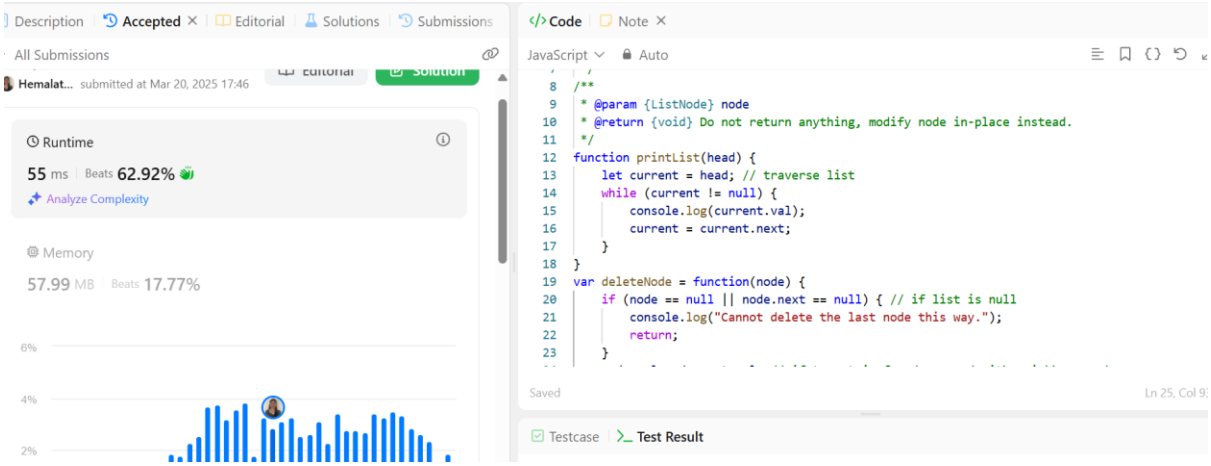
```
class Node{
    constructor(data){
        this.next=null;
    }
}

let head=new Node(4);
let first=new Node(5);
let second=new Node(1);
let third =new Node(2);

head.next=first;
first.next=second;
second.next=third;
```

OUTPUT ··· Filter Code

```
[Running] node "c:\Users\Sangeetha\js_intro\DSA assign-3\problem7.js"
4
1
2
```

Submission link

https://leetcode.com/problems/delete-node-in-a-linked-list/submissions/1580209032/



Conclusion

Time complexity O(1)

Best case :O(1)

Deleteing nodes with constant time.In this case,only few operation has performed in this problem.

Space complexity O(1)

No extra space required.