Q1) 70. Climbing Stairs

Solution 1:

```
/**
 * @param {number} n
 * @return {number}
 */
var climbStairs = function(n) {
    if (n === 0 || n === 1) {
        return 1;
    }
    return climbStairs(n-1) + climbStairs(n-2);
};
```

Time complexity: O(2^n)
Space Complexity: O(n)

Solution 2:

```
/**
 * @param {number} n
 * @return {number}
 */

function climbStairsMemo(n, memo) {
    if (n === 0 || n === 1) {
        return 1;
    }
    if (!memo.has(n)) {
        memo.set(n, climbStairsMemo(n - 1, memo) + climbStairsMemo(n - 2,
memo));
    }
    return memo.get(n);
}
var climbStairs = function(n) {
    let memo = new Map();
    return climbStairsMemo(n, memo);
};
```

Time complexity: O(n)
Space Complexity: O(n)

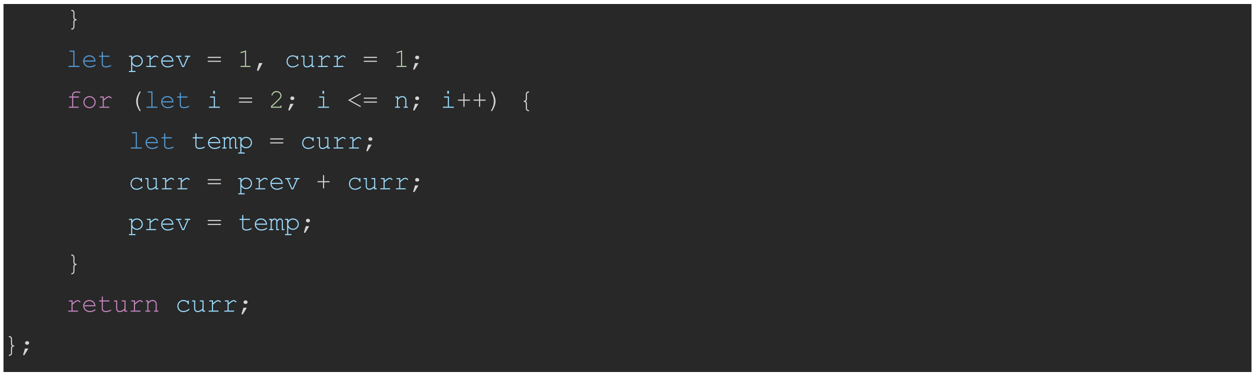
Solution 3:

```
/**
 * @param {number} n
 * @return {number}
 */
var climbStairs = function(n) {
    if (n === 0 || n === 1) {
        return 1;
    }

    let dp = new Array(n + 1).fill(0);
    dp[0] = dp[1] = 1;

    for (let i = 2; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 2];
    }

    return dp[n];
};
```

Time complexity: O(n)
Space Complexity: O(n)

Solution 4

```
/**
 * @param {number} n
 * @return {number}
 */
var climbStairs = function(n) {
    if (n === 0 || n === 1) {
        return 1;
```

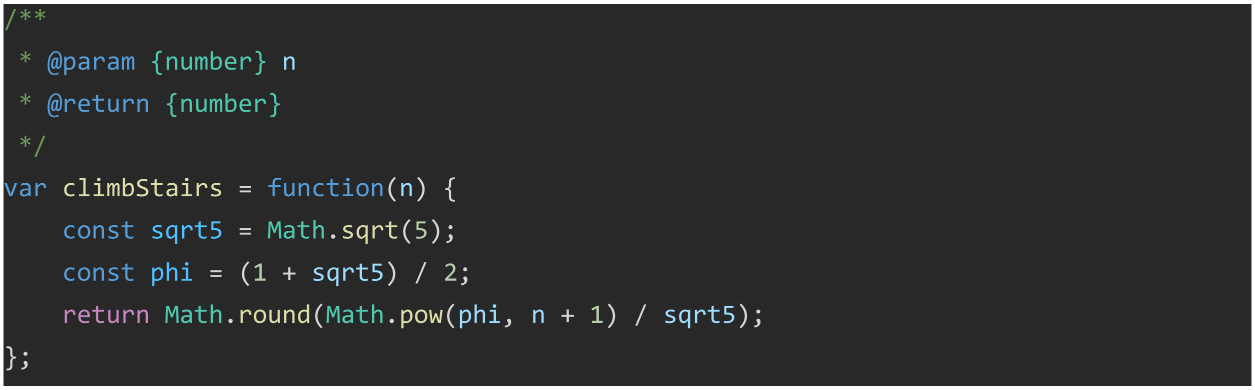


```
    }
    let prev = 1, curr = 1;
    for (let i = 2; i <= n; i++) {
        let temp = curr;
        curr = prev + curr;
        prev = temp;
    }
    return curr;
};
```

Time complexity: O(n)
Space Complexity: O(1)

Better Solution

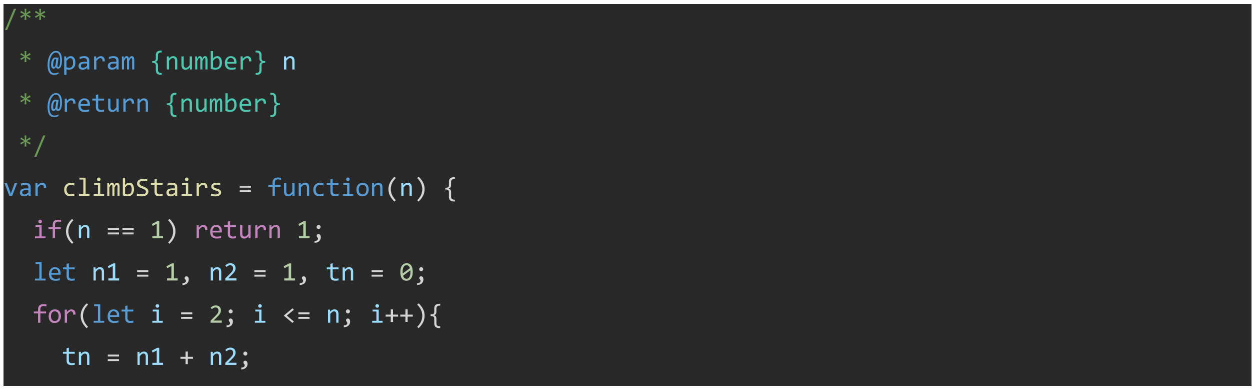
Closed-form expression for the nth Fibonacci number using Binet's formula:
F(n) = (φ^n - (1-φ)^n) / √5
φ = (1 + √5) / 2 is the golden ratio.



```
/**
 * @param {number} n
 * @return {number}
 */
var climbStairs = function(n) {
    const sqrt5 = Math.sqrt(5);
    const phi = (1 + sqrt5) / 2;
    return Math.round(Math.pow(phi, n + 1) / sqrt5);
};
```

Time complexity: O(1)
Space Complexity: O(1)

OR



```
/**
 * @param {number} n
 * @return {number}
 */
var climbStairs = function(n) {
  if(n == 1) return 1;
  let n1 = 1, n2 = 1, tn = 0;
  for(let i = 2; i <= n; i++){
    tn = n1 + n2;
    n1 = n2;
    n2 = tn;
  }
  return tn;
};
```

Time complexity: O(n)
Space Complexity: O(1)

Q2)21. Merge Two Sorted Lists

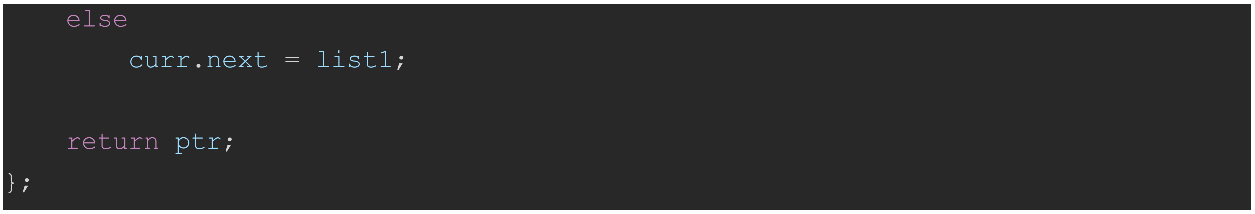
Solution 1:

```
/**
 * Definition for singly-linked list.
 * function ListNode(val, next) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.next = (next===undefined ? null : next)
 * }
 */
/**
 * @param {ListNode} list1
 * @param {ListNode} list2
 * @return {ListNode}
 */
var mergeTwoLists = function(l1, l2) {
    if (l1 === null)
        return l2;
    if (l2 === null)
        return l1;
    if (l1.val <= l2.val) {
        l1.next = mergeTwoLists(l1.next, l2);
        return l1;
    } else {
        l2.next = mergeTwoLists(l1, l2.next);
        return l2;
    }
};
```

Time complexity: O(m+n)
Space Complexity: O(m+n)

Better Solution:

```
/**
 * Definition for singly-linked list.
 * function ListNode(val, next) {
 *     this.val = (val === undefined ? 0 : val)
 *     this.next = (next === undefined ? null : next)
 * }
 */

var mergeTwoLists = function(list1, list2) {
    if (list1 === null)
        return list2;
    if (list2 === null)
        return list1;

    let ptr = list1;
    if (list1.val > list2.val) {
        ptr = list2;
        list2 = list2.next;
    } else {
        list1 = list1.next;
    }
    let curr = ptr;

    while (list1 && list2) {
        if (list1.val < list2.val) {
            curr.next = list1;
            list1 = list1.next;
        } else {
            curr.next = list2;
            list2 = list2.next;
        }
        curr = curr.next;
    }

    if (!list1)
        curr.next = list2;
```

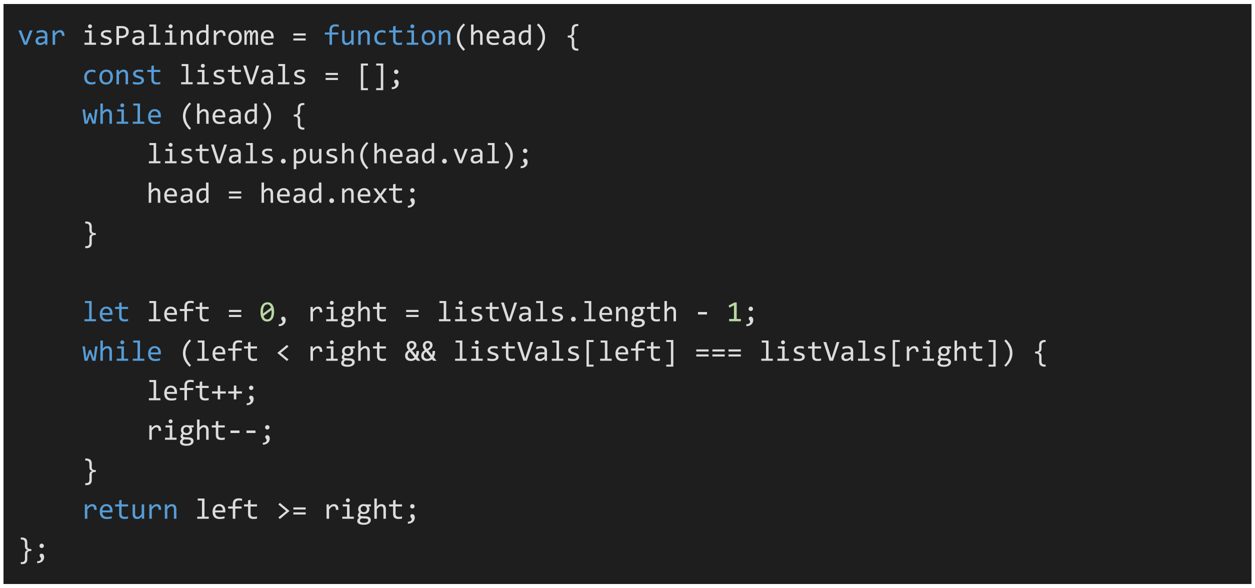


```
    else
        curr.next = list1;

    return ptr;
};
```

Time complexity: O(m+n)
Space Complexity: O(1)

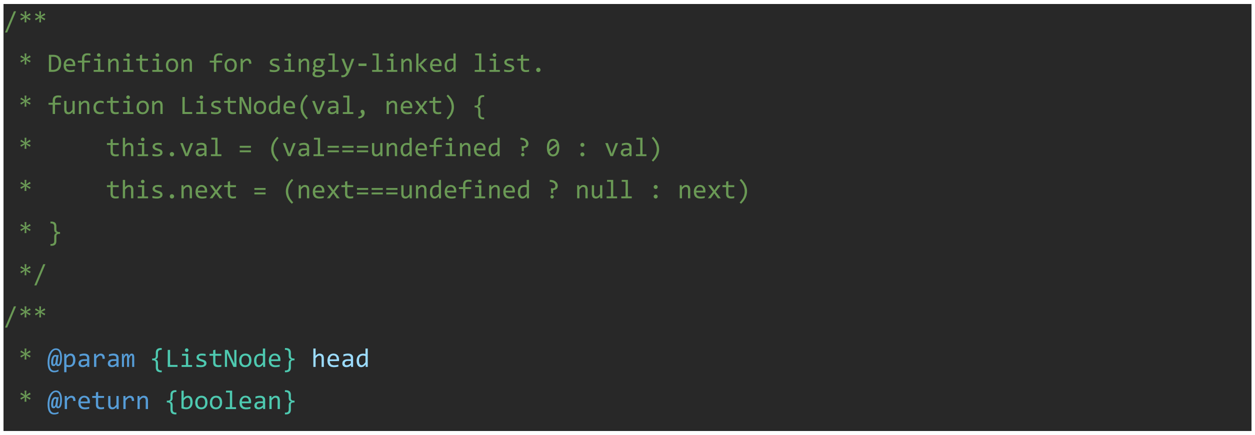
Q3) 234. Palindrome Linked List

Solution 1:



```
var isPalindrome = function(head) {
    const listVals = [];
    while (head) {
        listVals.push(head.val);
        head = head.next;
    }

    let left = 0, right = listVals.length - 1;
    while (left < right && listVals[left] === listVals[right]) {
        left++;
        right--;
    }
    return left >= right;
};
```

Or



```
/**
 * Definition for singly-linked list.
 * function ListNode(val, next) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.next = (next===undefined ? null : next)
 * }
 */
/**
 * @param {ListNode} head
 * @return {boolean}
 */
var isPalindrome = function(head) {
    let curr = { val: null, next: null };

    const solve = (node) => {
        if (node === null) return true;
        let ans = solve(node.next) && node.val === curr.val;
        curr = curr.next;
        return ans;
    };

    curr = head;
    return solve(head);
};
```

Time complexity: O(n)
Space Complexity: O(n)

Better Solution:

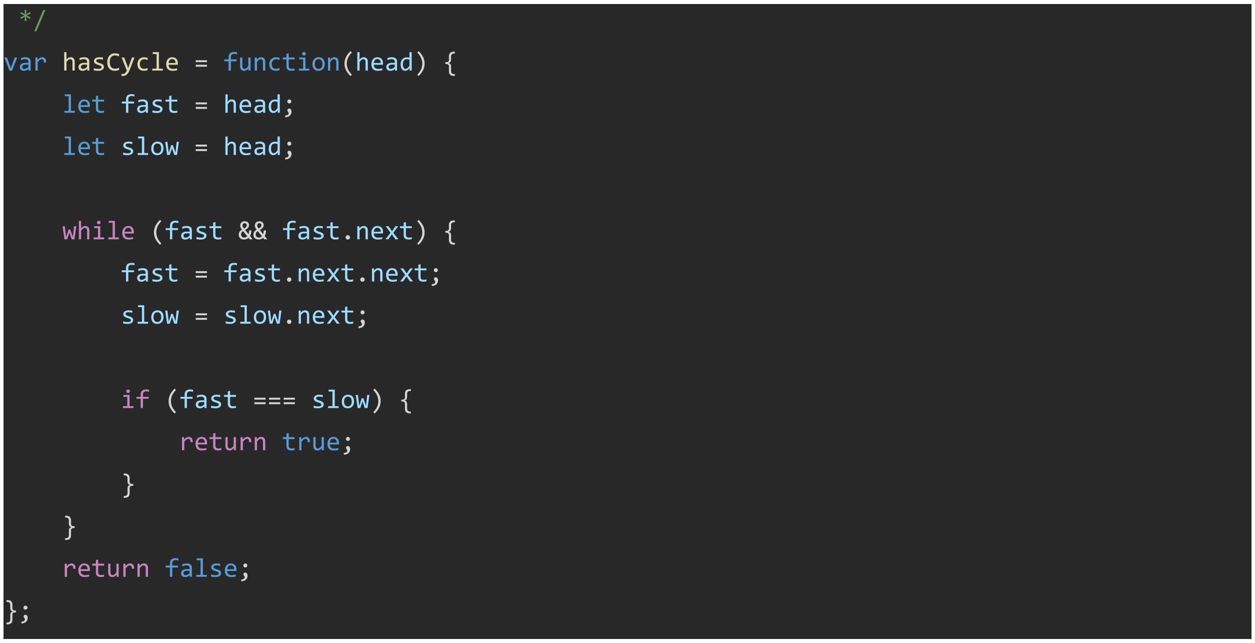
```
/**
 * Definition for singly-linked list.
 * function ListNode(val, next) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.next = (next===undefined ? null : next)
 * }
 */
/**
 * @param {ListNode} head
 * @return {boolean}
 */
function reverse(head) {
    let prev = null;
    let curr = head;
    while (curr) {
        let next = curr.next;
        curr.next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}

var isPalindrome = function(head) {
    let slow = head;
        let fast = head;
        while (fast && fast.next) {
            slow = slow.next;
            fast = fast.next.next;
        }
        let rev = reverse(slow);
        while (rev) {
            if (head.val !== rev.val) {
                return false;
            }
            head = head.next;
            rev = rev.next;
        }
        return true;
};
```

Time complexity: O(n)
Space Complexity: O(1)

Q4)141. Linked List Cycle

Solution 1

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */

/**
 * @param {ListNode} head
 * @return {boolean}
```

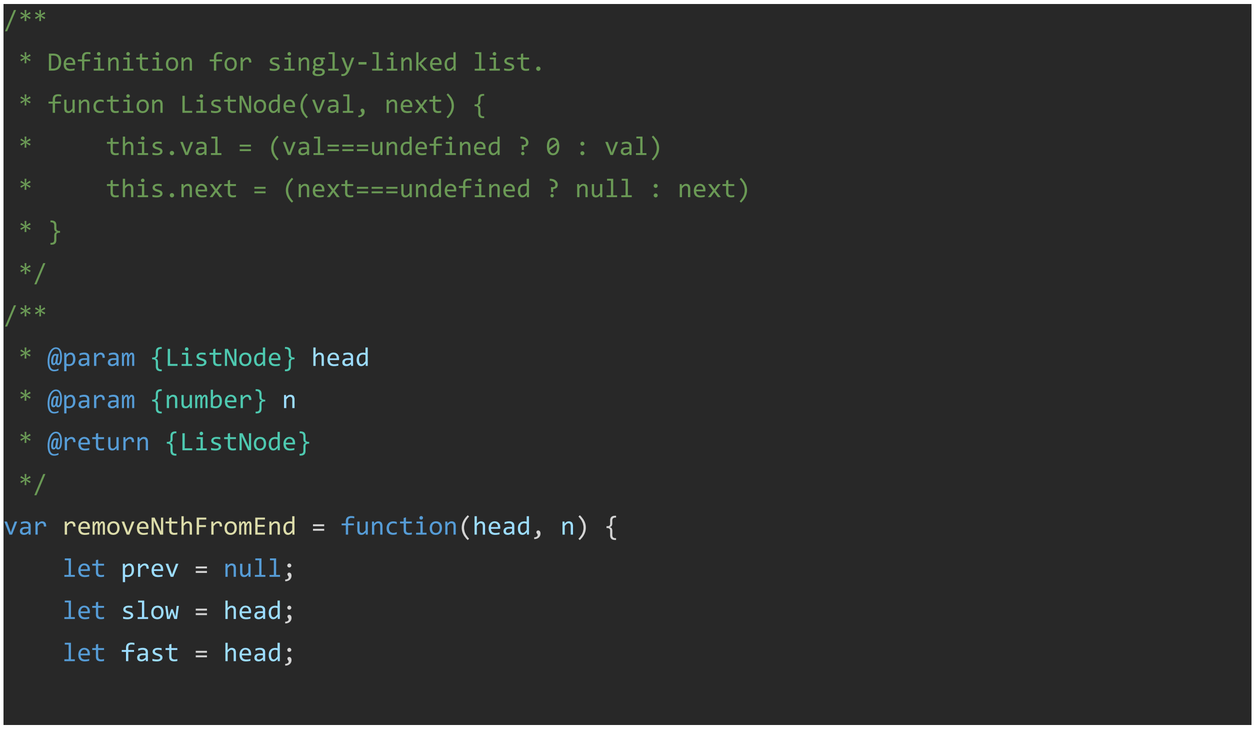


```
 */
var hasCycle = function(head) {
    let fast = head;
    let slow = head;

    while (fast && fast.next) {
        fast = fast.next.next;
        slow = slow.next;

        if (fast === slow) {
            return true;
        }
    }
    return false;
};
```

Time complexity: O(n)
Space Complexity: O(1)

Q5)19. Remove Nth Node From End of List

Better Solution:



```
/**
 * Definition for singly-linked list.
 * function ListNode(val, next) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.next = (next===undefined ? null : next)
 * }
 */
/**
 * @param {ListNode} head
 * @param {number} n
 * @return {ListNode}
 */
var removeNthFromEnd = function(head, n) {
    let prev = null;
    let slow = head;
    let fast = head;

    for (let i = 0; i < n; i++) {
        fast = fast.next;
    }

    while (fast) {
        prev = slow;
        slow = slow.next;
        fast = fast.next;
    }
    if(prev == null) head = head.next;
    else prev.next = slow.next;

    return head;
}
```

Time complexity: O(n)
Space Complexity: O(1)

Q6)50. Pow(x, n)

Solution 1

```
/**
 * @param {number} x
 * @param {number} n
 * @return {number}
 */
var myPow = function (x, n) {
    const mypow1 = (n) => {
        if (n <= 0)
            return 1;
        const res = mypow1 (n >>> 1);
        return res * res * (n % 2 ? x : 1);
    }
    let power = 1;
    power = mypow1(Math.abs(n));
    return n < 0 ? 1 / power : power;
};
```

Time complexity: O(log n)
Space Complexity: O(logn)

Better Solution

```
/**
 * @param {number} x
 * @param {number} n
 * @return {number}
 */
var myPow = function(x, n) {
    if (n < 0) {
        x = 1 / x;
        n = -n;
    }
    let result = 1;
    while (n) {
        if (n & 1) {
            result *= x;
        }
        x *= x;
        n >>= 1;
    }
    return result;
};
```

Time complexity: O(log n)
Space Complexity: O(1)

Q7)237. Delete Node in a Linked List

Solution 1

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
```

```
/**
 * @param {ListNode} node
 * @return {void} Do not return anything, modify node in-place instead.
 */
var deleteNode = function(node) {
    while(node && node.next) {
        node.val = node.next.val;
        if(!node.next.next) node.next = null;
        node = node.next;
    }
};
```

Time complexity: O(n)
Space Complexity: O(1)

Better Solution

```
/**
 * Definition for singly-linked list.
 * function ListNode(val) {
 *     this.val = val;
 *     this.next = null;
 * }
 */
/**
 * @param {ListNode} node
 * @return {void} Do not return anything, modify node in-place instead.
 */
var deleteNode = function(node) {
    node.val = node.next.val;
    node.next = node.next.next;
};
```

Time complexity: O(1)
Space Complexity: O(1)