3062. Winner of the Linked List Game

outcome of the match (the higher value wins).

Linked List Easy

Problem Description

pattern is followed where even-indexed nodes (0, 2, 4, ...) contain even integers and odd-indexed nodes (1, 3, 5, ...) contain odd integers. We can imagine these nodes as being part of 'pairs' (two adjacent nodes, starting with the head of the list). The task is to keep a score for two conceptual teams, 'Even' and 'Odd', by comparing the integer values in each pair: if the integer in the even-indexed node is higher, team 'Even' scores a point; if the integer in the odd-indexed node is higher, team 'Odd' scores a point. If both teams have the same score at the end of the list, the result is a tie. We need to return the name of the team with the higher score, or "Tie" if it's a draw. Intuition

The problem provides us with a <u>linked list</u> where each node contains an integer. The list has an even number of nodes, and a

The essence of the solution is a straightforward simulation. We iterate through the linked list, two nodes at a time, comparing the

To implement this, we can maintain two variables, odd and even, that will serve as the score counters for the 'Odd' and 'Even' teams, respectively. Starting from the head of the list, we increment either odd or even depending on which node in the current pair has a larger value. Since we know the list has an even length, we are assured that we will always end on a node that is part

values of each node in a pair. This is akin to a match where each pair leads to a contest, and points are awarded according to the

of a complete pair. After traversing the entire list, we simply compare our two counters and return the team that accumulated the higher score or "Tie" if both scores are even. **Solution Approach** The given Python solution follows an iterative approach common in dealing with linked lists. It doesn't use any complex data

structures or algorithms but relies on simple comparison and arithmetic operations. The core idea is to simulate the process

mentioned in the problem statement using a loop. Below is a step-by-step explanation of the code: Initial Score Counters: Two variables, odd and even, are initialized to zero. These variables act as score counters for the

"Odd" and "Even" teams. Iteration Over the List: A while loop is used to traverse the linked list. In each iteration, the loop checks the existence of the current node and, implicitly due to the even length of the list, its next node.

Accessing Node Values: Within each iteration, node values are accessed using head.val for the even-indexed node and head.next.val for the odd-indexed node.

Comparing Values: For each pair, the values of the nodes (a and b) are compared:

- ∘ If the value of the even-indexed node (a) is greater than the odd-indexed node (b), the variable even is incremented. ∘ If the value of the odd-indexed node (b) is greater than the even-indexed node (a), the variable odd is incremented. Moving to the Next Pair: The head is moved two nodes forward to the next pair by setting head to head next next.
- **Determining the Winner:** Finally, the function compares the accumulated scores for "Odd" and "Even": If odd is greater than even, the function returns the string "0dd".

End of List: Once the algorithm has compared all pairs, and the end of the list is reached (when head is null), the loop ends.

- If even is greater than odd, it returns the string "Even". If odd and even are equal, it returns the string "Tie".
- This approach has a linear time complexity of O(n), where n is the length of the linked list, since it requires a single pass through the list. The algorithm's space complexity is O(1) - irrespective of the size of the input, it only uses a fixed amount of additional
- **Example Walkthrough**

Node 1 (value 4) -> Node 2 (value 3) -> Node 3 (value 6) -> Node 4 (value 5) Remember, the list is zero-indexed, so Node 1 is at index 0, Node 2 is at index 1, and so on.

Iteration Over the List: We begin from Node 1 (at index 0). Accessing Node Values: Node 1 has the value 4 (even-indexed), and its next Node 2 has the value 3 (odd-indexed). 3.

Following the solution approach:

Comparing Values: Since 4 is greater than 3, even now becomes 1, while odd remains at 0.

memory (for the score counters and temporary variables).

Let's consider a linked list with the following even-numbered nodes:

Initial Score Counters: We initialize two variables, odd and even to 0.

Moving to the Next Pair: We move on to the next pair, Node 3 and Node 4.

Moving to the Next Pair: Since there are no more pairs, we exit the loop.

Comparing Values: Node 3's value 6 is greater than Node 4's value 5, so we increment even again to 2. The odd score is still 0.

Determining the Winner: Comparing odd 0 with even 2, even is greater. Therefore, we return "Even" as the winner.

In this example, the Even team wins by a score of 2-0. If, at any point, the team odd had a greater value in their respective index,

Accessing Node Values: Now, Node 3 (even-indexed) has a value 6, and Node 4 (odd-indexed) has a value 5.

- we would have incremented the odd counter instead. In a case where both teams end up with the same score, we would return
- # Definition for singly-linked list.

Traverse the linked list two nodes at a time.

Retrieve the values of the two adjacent nodes.

Determine the result based on the players' points.

Increment odd points if the odd indexed player wins the round.

Python class ListNode: def init (self, val=0, next=None): self.val = val

def gameResult(self. head: Optional[ListNode]) -> str: # Initialize counters for the odd and even indexed players' points. odd points = 0

class Solution:

"Tie".

Solution Implementation

self.next = next

even_points = 0

while head and head.next:

odd value = head.val

head = head.next.next

if odd points > even points:

} else if (oddScore < evenScore) {</pre>

ListNode(): val(0), next(nullptr) {}

int oddCounter = 0, evenCounter = 0;

// Traverse the linked list two steps at a time.

const secondValue = head.next.val;

// Retrieve the current two values.

const firstValue = head.val;

while (head && head.next) {

ListNode(int x) : val(x), next(nullptr) {}

ListNode(int x, ListNode *next) : val(x), next(next) {}

// Initialize counters for odd and even positioned nodes

// Iterate over the linked list two elements at a time

while (head != nullptr && head->next != nullptr) {

// Method to determine the result of the game based on odd or even indices

return "Even";

return "Tie";

* Definition for singly-linked list.

string gameResult(ListNode* head) {

// In case of a tie

} else {

* struct ListNode {

class Solution {

int val;

ListNode *next:

C++

/**

* };

public:

return "Odd"

even_value = head.next.val

Move to the next pair of nodes.

odd points += odd value < even value # Increment even points if the even indexed player wins the round. even points += odd value > even value

```
elif odd points < even_points:</pre>
            return "Even"
        else:
            return "Tie"
Java
/**
 * Definition for singly-linked list.
class ListNode {
    int val:
    ListNode next;
    ListNode() {}
    ListNode(int val) {
        this.val = val;
    ListNode(int val, ListNode next) {
        this.val = val;
        this.next = next;
class Solution {
    public String gameResult(ListNode head) {
        // Initialize counters for odd and even index matchups
        int oddScore = 0;
        int evenScore = 0;
        // Traverse the linked list two nodes at a time
        while (head != null && head.next != null) {
            int firstValue = head.val; // Value at current node (odd index)
            int secondValue = head.next.val; // Value at next node (even index)
            // Increment the odd score if the value at odd index is less than at even index
            oddScore += firstValue < secondValue ? 1 : 0;</pre>
            // Increment the even score if the value at odd index is greater than at even index
            evenScore += firstValue > secondValue ? 1 : 0;
            // Skip to the node after the next one for the next pair
            head = head.next.next;
        // Determine the result based on scores
        if (oddScore > evenScore) {
            return "Odd";
```

```
// Value at the current node (odd index)
            int currentValue = head->val;
            // Value at the next node (even index)
            int nextValue = head->next->val;
            // Increase the odd counter if the value at the odd index is less than even index value
            oddCounter += currentValue < nextValue;</pre>
            // Increase the even counter if the value at the odd index is greater than even index value
            evenCounter += currentValue > nextValue;
            // Move the pointer two nodes ahead in the list
            head = head->next->next;
        // Determine the game result based on odd and even counters
        if (oddCounter > evenCounter) {
            return "Odd";
        if (oddCounter < evenCounter) {</pre>
            return "Even";
        // If counters are equal, it's a tie
        return "Tie";
TypeScript
/**
 * Function to determine the game result from a linked list.
 * The list nodes represents numbers drawn by two players, odd and even, in turns.
 * Odd player draws on odd turns and Even player on even turns.
 * A player scores a point if their number is greater than the other player's.
 * Returns 'Odd' if the odd player wins, 'Even' if the even player wins, or 'Tie' if it's a draw.
 * @param {ListNode | null} head - the head of the linked list containing game numbers.
 * @return {string} - the result of the game: 'Odd', 'Even', or 'Tie'.
function gameResult(head: ListNode | null): string {
    // Initialize the scores for both players.
    let oddScore = 0;
    let evenScore = 0;
```

// Odd player's draw.

// Even player's draw.

```
// Compare the values and update scores accordingly.
        oddScore += firstValue > secondValue ? 1 : 0; // Odd player scores if their number is greater.
        evenScore += firstValue < secondValue ? 1 : 0; // Even player scores if their number is greater.
        // Move to the next pair of numbers.
        head = head.next.next;
    // Determine and return the result based on the scores.
    if (oddScore > evenScore) {
        return 'Odd';
    } else if (oddScore < evenScore) {</pre>
        return 'Even';
    } else {
        return 'Tie';
# Definition for singly-linked list.
class ListNode:
   def init (self, val=0, next=None):
        self.val = val
        self.next = next
class Solution:
    def gameResult(self, head: Optional[ListNode]) -> str:
        # Initialize counters for the odd and even indexed players' points.
        odd points = 0
        even_points = 0
        # Traverse the linked list two nodes at a time.
        while head and head.next:
            # Retrieve the values of the two adjacent nodes.
            odd value = head.val
            even_value = head.next.val
            # Increment odd points if the odd indexed player wins the round.
            odd points += odd value < even value
            # Increment even points if the even indexed player wins the round.
            even_points += odd_value > even_value
            # Move to the next pair of nodes.
            head = head.next.next
        # Determine the result based on the players' points.
        if odd points > even_points:
            return "Odd"
       elif odd points < even_points:</pre>
            return "Even"
        else:
            return "Tie"
```

Time and Space Complexity

The code performs a loop that iterates through the nodes of a linked list, where the step in each iteration moves two nodes forward (head to head next next). Despite this step, we still touch each node exactly once, therefore, the time complexity depends linearly on the number of nodes, resulting in O(n) time complexity, where n is the number of nodes in the list.

The space complexity of the algorithm is 0(1), as there are no data structures used that grow with the input size. Only a fixed number of variables (odd, even, a, b) are used regardless of the list size.