Decimal Equivalent of Binary Linked List

Given a singly linked list of length n. The link list represents a **binary number,** ie- it contains only 0s and 1s. Find its decimal equivalent.
The significance of the bits **decreases** with the increasing index in the linked list.
An empty linked list is considered to represent the decimal value **0**.

Since the answer can be very large, **answer modulo 10^9+7** should be printed.

Example 1:
Input:
n = 3
Linked List = {0, 1, 1}
Output:
3
Explanation:
$0*2^2 + 1*2^1 + 1*2^0 = 1 + 2 + 0 = 3$

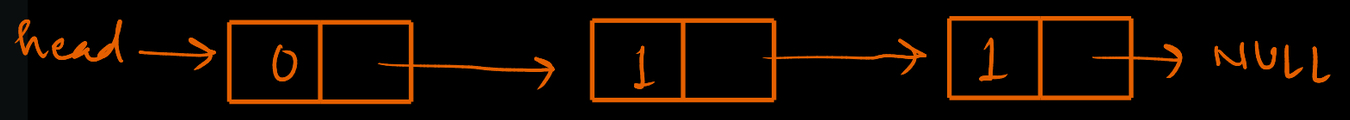
Example 2:
Input:
n = 4
Linked List = {1, 1, 1, 0}
Output:
14
Explanation:
$1*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = 8 + 4 + 2 + 0 = 14$

Your Task:
You do not have to take any input or print anything. Complete the function **decimalValue()** which takes a **head node** of a linked list as an input parameter and returns decimal representation of it.

Expected Time Complexity: O(n)
Expected Auxiliary Space: O(1)

Constraints:
0 <= n <= 100
Data of each node is either 0 or 1

Input:



Output = 3.

Brute force:

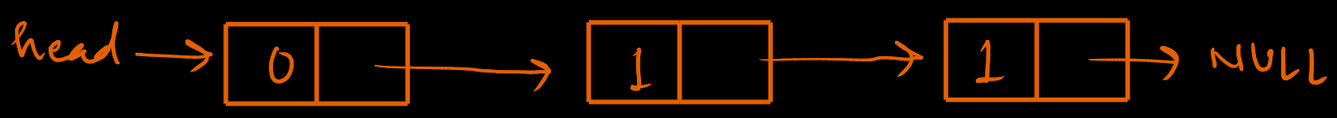
→ Just follow the explaination approach.
→ Find the size of LL.
→ Get the values of each node power with modulo.

Note: This will give TLE after running some test cases. So, we need to optimize the code.

```
class Solution
{
    public:
        long long unsigned int powerOf2(int n, int MOD) {
            long long unsigned int result = 1;
            for (int i = 0; i < n; ++i) {
                result = (result * 2) % MOD;
            }
            return result;
        }

        long long unsigned int decimalValue(Node *head)
        {
           int modulo = 1000000007;
           Node *temp = head;
           int size = -1;
           long long unsigned int ans = 0;

           while(temp != NULL) {
               size++;
               temp = temp->next;
           }

           temp = head;

           while(temp != NULL) {
                if(temp->data) {
                    long long unsigned int power = powerOf2(size, modulo);
                    ans = (ans + power) % modulo;
                }
               size--;
               temp = temp->next;
           }
           return ans;
        }
};
```

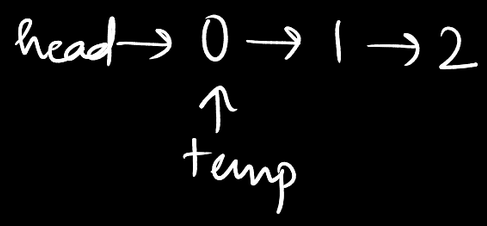
Optimal Approach :-

Input :



Output = 3.

ans = 0



→ Left shift previous values by multiplying with 2.

→ Add the current data.

→ Move to next node.

Why this works?

$0 \rightarrow 1 \rightarrow 1$

$3 = 0*2*2 + 1*2 + 1*2^0$

Now if we closely observe our while loop

1st Iteration :

ans = 0 * 2 ;

ans

ans = (0 + 0) ;

head → data

2nd Iteration :

ans = 0 * 2 ;

ans = (0 * 2 + 1) ;

0 → 1 → 1 → NULL

head

3rd Iteration :

ans = { (0 * 2 + 1) } * 2 ;

ans = ans + 1 ;

{ (0 * 2 + 1) } * 2 + 1

On further expansion

$0*2*2 + 1*2 + 1*2^0$

```
class Solution
{
    public:
    long long unsigned int decimalValue(Node *head) {
        const int MOD = 1000000007;
        long long unsigned int ans = 0;
        while (head != NULL) {
            ans = (ans * 2) % MOD;
            ans = (ans + head->data) % MOD;
            head = head->next;
        }
        return ans;
    }

};
```