Study Guide: Add Two Numbers (LeetCode #2)

Problem Summary

Add two numbers represented as linked lists in reverse order. Each node contains a single digit.

Example: `[2,4,3] + [5,6,4] = [7,0,8]` (342 + 465 = 807)

Solution Comparison

Your Solution Analysis

Approach: Convert → Calculate → Convert Back

```
// Your flow:
Linked List → Array → String → Number → Add → String → Array → Linked List
```

What you did:

1. Recursively extracted values into arrays
2. Joined arrays to form number strings: `"342"` and `"465"`
3. Converted to numbers and added: `342 + 465 = 807`
4. Converted result back to string, split, and reversed
5. Recursively built new linked list from the array

Critical Issues with Your Approach

🚨 **FATAL BUG: Number Overflow**

Your solution **WILL FAIL** for large numbers!

```
const sum = Number(reverseL1.join("")) + Number(reverseL2.join(""));
```

JavaScript's `Number` type is limited to `Number.MAX_SAFE_INTEGER` (9,007,199,254,740,991).

This breaks on Example 3:

```
l1 = [9,9,9,9,9,9,9]  // 9,999,999
l2 = [9,9,9,9]        // 9,999
Sum = 10,009,998      // Still works
```

But fails on larger inputs:

```
l1 = [9,9,9,9,9,9,9,9,9,9,9,9,9,9,9]  // Too large!
// JavaScript loses precision
```

⚠️ Space Complexity Issues

- Creating intermediate arrays: O(n + m) space
- Creating strings: O(n + m) space
- Total extra space: O(n + m) when you only need O(1)

⚠️ Time Complexity Issues

- Traverse list → array: O(n + m)
- Join to string: O(n + m)
- Number conversion: O(n + m)
- Addition: O(1)
- String split/reverse: O(result length)
- Rebuild list: O(result length)
- **Multiple passes through the data!**

The Right Mental Model

🧠 Think Like Elementary School Addition

Remember how you add numbers by hand?

```
   3 4 2
 + 4 6 5
 -------
```

You go **right to left**, adding digit by digit, carrying over when needed.

But wait! The problem gives you digits **already reversed** (right to left):

- `[2,4,3]` represents 342
- `[5,6,4]` represents 465

So you can **process the lists left to right** as if you're doing elementary addition!

🎯 The "Carry" Pattern

This is the key mental model: **Track the carry digit**

```
Step 1: 2 + 5 = 7, carry = 0
Step 2: 4 + 6 = 10, carry = 1 → store 0
Step 3: 3 + 4 + carry(1) = 8, carry = 0
```

🔑 The "Dummy Head" Technique

Instead of handling the first node as a special case, create a dummy:

```
let dummy = new ListNode(0);  // Throwaway node
let temp = dummy;              // Builder pointer

// Build: dummy → 7 → 0 → 8 → null
// Return: dummy.next (skip the dummy)
```

This pattern appears in MANY linked list problems!

Optimal Solution Breakdown

```
function addTwoNumbersOptimal(
  l1: ListNode | null,
  l2: ListNode | null,
): ListNode | null {
  let dummy = new ListNode(0);    // 1. Dummy head trick
  let temp = dummy;                // 2. Builder pointer
```

```
  let carry = 0;                    // 3. Carry tracker

  while (l1 !== null || l2 !== null || carry !== 0) {  // 4. Process until done
    let val1 = l1 ? l1.val : 0;     // 5. Handle different lengths
    let val2 = l2 ? l2.val : 0;

    let sum = val1 + val2 + carry; // 6. Elementary addition
    carry = Math.floor(sum / 10);  // 7. Extract carry (tens digit)
    temp.next = new ListNode(sum % 10);  // 8. Store ones digit
    temp = temp.next;              // 9. Move builder forward

    if (l1 !== null) l1 = l1.next; // 10. Advance pointers
    if (l2 !== null) l2 = l2.next;
  }

  return dummy.next;                // 11. Skip dummy head
}
```

Line-by-Line Mental Model

Lines 57-59: Set up infrastructure

- `dummy` : Anchor point (never moves)
- `temp` : Builder that moves forward
- `carry` : Tracks overflow to next digit

Line 61: Continue while there's ANYTHING left to process

- Either list has nodes, OR
- We have a carry to handle (important for cases like 99 + 1 = 100)

Lines 62-63: Handle mismatched lengths gracefully

- If a list ended, treat it as 0
- No special cases needed!

Line 65: The core addition

- Add both values + any carry from previous step

Line 66: Extract carry for next iteration

- `15 / 10 = 1.5 → floor = 1`
- `7 / 10 = 0.7 → floor = 0`

Line 67: Store the ones digit

- `15 % 10 = 5`
- `7 % 10 = 7`

Line 68: Move builder forward

- We're done with this node, move to next

Lines 70-71: Advance input pointers

- Only if they exist (null-safe)

Line 74: Return the real list (skip dummy)

Complexity Analysis

Metric	Your Solution	Optimal Solution
Time	O(n + m) × multiple passes	O(max(n, m)) × single pass
Space	O(n + m) extra arrays/strings	O(1) except result
Overflow Risk	❌ Fails on large numbers	✅ Handles any size
Code Clarity	Complex recursion	Clear iteration

Key Takeaways

❌ What Not to Do

1. **Never convert linked list → number** when the constraint says lists can be 100 nodes
    - That's potentially 100 digits (10^100), far beyond JavaScript's number limit!
2. **Avoid unnecessary conversions** (list → array → string → number)
3. **Don't overcomplicate with recursion** when iteration is clearer

✅ What to Remember

1. **"Carry" pattern** is fundamental for digit-by-digit operations
2. **Dummy head trick** simplifies linked list construction
3. **Process in-place** when possible (O(1) space)
4. **The problem pre-reverses the lists for you!** Just process left-to-right
5. **Think "elementary school addition"** - that's literally the algorithm

---

Pattern Recognition

This problem teaches you patterns that appear in:

- **Multiply Strings** (LeetCode #43) - Similar carry logic
- **Add Binary** (LeetCode #67) - Same pattern, base-2
- **Plus One** (LeetCode #66) - Simplified version
- **Merge Two Sorted Lists** (LeetCode #21) - Dummy head technique
- **Remove Nth Node** (LeetCode #19) - Two-pointer on linked list

---

Practice Exercise

Try solving this variant mentally using the optimal approach:

```
Input: l1 = [9,9], l2 = [1]
```

Mental walkthrough:

1. 9 + 1 = 10 → carry=1, store 0
2. 9 + 0 + carry(1) = 10 → carry=1, store 0
3. 0 + 0 + carry(1) = 1 → carry=0, store 1
4. Result: [0,0,1] (represents 100)

Notice how the carry extends the result length!

Final Verdict

Your solution: Shows algorithmic thinking but misses the fundamental insight that this is an **elementary addition problem**. The conversion approach is clever but introduces bugs and inefficiency.

Optimal solution: Recognizes the problem's structure matches manual addition. Processes digit-by-digit with carry tracking - exactly how you learned in elementary school.

The lesson: Sometimes the simplest mental model (grade school math) is the best algorithm. Don't over-engineer!