

Leetcode Report

Rania Siddiqui 07494

Problem Statement

Basic Calculator: Given an arithmetic expression containing integers, addition (+), subtraction (-), and parentheses (()), evaluate the expression and return the result. You cannot use eval().

Example:

- Input: `s = "(1+(4+5+2)-3)+(6+8)"`
- Output: 23

Approach:

- Use a Stack: Store numbers and operators to manage operations efficiently.
- Handle Parentheses Using Recursion: Evaluate expressions within parentheses recursively.
- Use Sign Multipliers: Maintain a +1 or -1 multiplier to handle addition and subtraction correctly.

For this problem statement, I have included a Jupyter notebook file which contains all the code for each of the solutions.

Manual Solution

Approach:

The approach that I have used is not very different from what LLMs gave me as well. My approach evaluates the mathematical expression containing numbers, addition, subtraction, and parentheses using a stack-based approach. It first iterates through the string once, processing each character to handle numbers, signs, and parentheses. For numbers, it accumulates the value by reading consecutive digits. For signs (+ or -), it updates the result with the current number and resets the number for the next operation. When encountering an opening parenthesis (, it pushes the current result and sign onto the stack and resets them to handle the nested expression. Upon encountering a closing parenthesis), it pops the previous result and sign from the stack and combines them with the current result. With a time complexity of $O(N)$ and a space complexity of $O(N)$, this approach evaluates expressions.

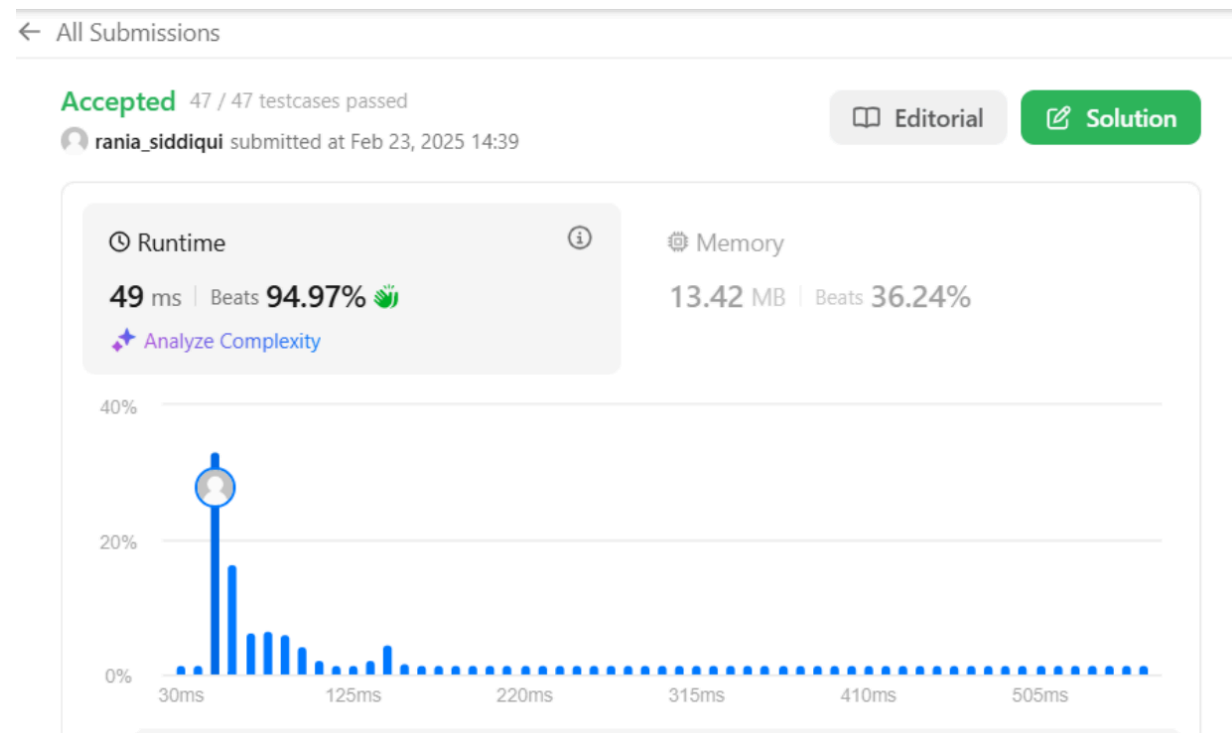
Complexity Analysis:

The function possesses the input string `s` in a single pass, while using a stack to handle the parenthesis. The loop runs in $O(n)$. The stack operations contribute to at most $O(n)$, thus the overall time complexity is $O(n)$, where `n` is the length of the input string. If we talk about the

space complexity, this will be determined by the stack usage, which is used to handle the parenthesis. The stack stores (result, sign) pairs whenever an opening parenthesis (is encountered. In the **worst case**, if the expression is deeply nested, We push **$O(N)$** elements onto the stack. Thus, in the worst case, the **stack space is $O(N)$** . The overall space complexity is also $O(n)$. The worst case time complexity is also $O(n)$ because each character in the input is processed **exactly once** in a single pass. Even with the worst nesting (deep parentheses), each character is pushed and popped **at most once** in **$O(1)$** time.

Results

All the testcases were passed.



LLM#1 Solution (ChatGPT 4o-mini)

Approach

The algorithm evaluates a mathematical expression using a stack-based approach while handling +, -, and parentheses. It processes the input string in a single pass ($O(n)$) time complexity by iterating through each character. Numbers are built digit-by-digit, and operators update the result with the appropriate sign. When encountering '(', the current result and sign are pushed onto a stack to handle nested expressions separately. When encountering ')', the inner result is computed and merged with the previous stored values. The stack ensures correct handling of parentheses, leading to a worst-case space complexity of $O(n)$ (fully nested expressions). Since each character is processed at most

twice (once during traversal and once via stack operations), the approach is efficient for evaluating arithmetic expressions while maintaining a linear complexity.

Complexity Analysis:

Again, this approach also takes $O(n)$ time complexity, since the function processes each character in s , exactly once using a single-for-loop. This loop runs in $O(n)$. The stack operations run in $O(n)$ and there are no nested loops or recursive calls that would increase complexity hence overall time complexity is $O(n)$. The space complexity is also $O(n)$ since the stack stores result, sign pairs whenever an opening parenthesis is encountered. Other result, num, sign variables are $O(1)$ constant space. Thus the overall complexity is $O(n)$, and also in the worst case the same.

Results:

Using this approach, all test cases passed.

Accepted 47 / 47 testcases passed


 rania_siddiqui submitted at Feb 23, 2025 13:12


 Editorial

 Solution

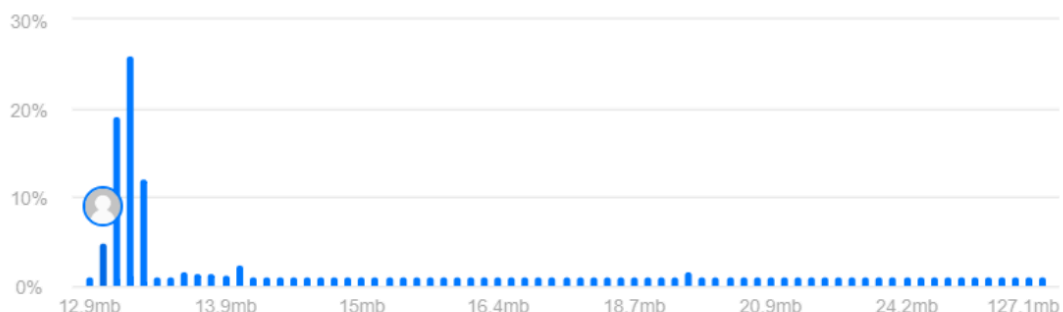
 Runtime

51 ms | Beats 91.39% 

 Memory

13.08 MB | Beats 94.97% 

 [Analyze Complexity](#)



LLM#2 Solution (Gemini)

Approach

This approach is similar to the one that ChatGPT gave, except this function also first removes all spaces from the input string s to simplify processing. It then follows the same stack method, processes the string character by character, accumulating digits into operands and applying signs when operators are encountered. With a single pass through the string,

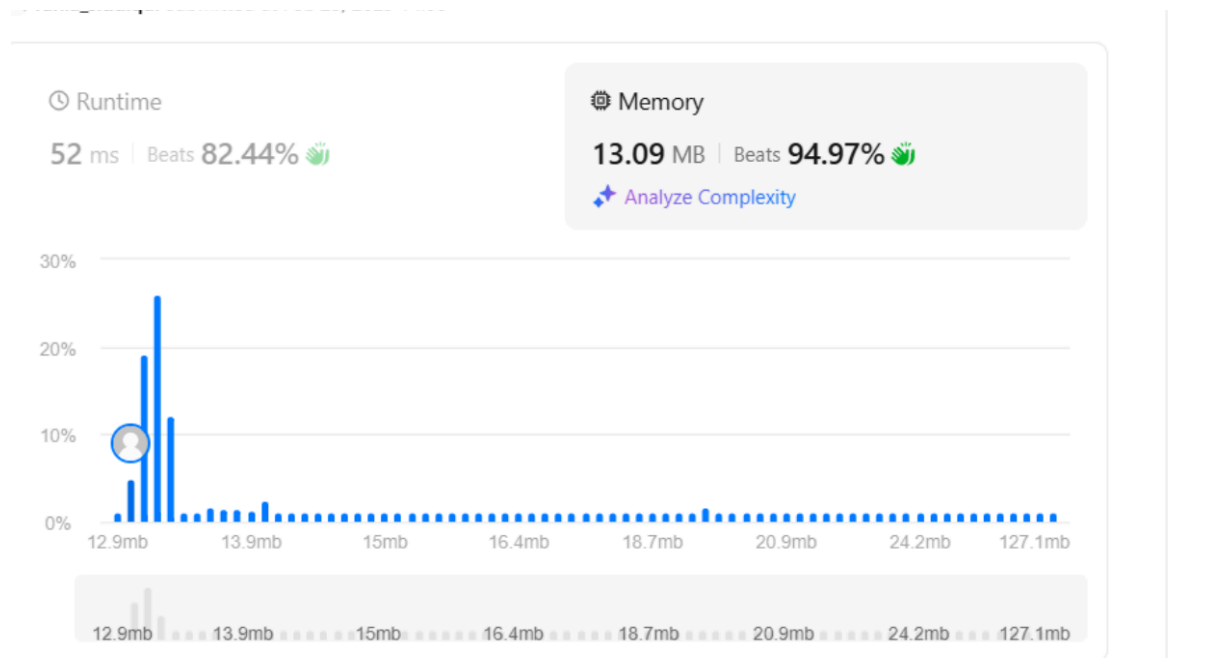
the code maintains a running result, efficiently handling addition, subtraction, and nested parentheses. The final result is returned after processing all characters.

Complexity Analysis:

Just like the first LLM approach, this has a space and time complexity of $O(n)$.

Results:

This approach passes all the testcases too.



Comparison and Analysis:

Correctness: With all the three solutions, they all correctly solve the problem with the given test cases.

Efficiency: All three solutions have a time and a space complexity of $O(n)$. However if we were to compare the solutions given by both the LLMs, the solution given by GPT is slightly more efficient compared to Gemini since it takes a total of 51ms compared to the gemini solution which takes 1ms more.

Readability and structure: All the three solutions follow a similar stack based easy to understand and follow through approach. The LLM solutions are both well-structured and efficient, offering concise yet powerful implementations. Their use of data structures like stack enhances performance without sacrificing clarity

