Project Complexity

We assume that input expression has **n** characters.

earlyCheck()

First of all, earlyCheck function is called to check 3 invalid patterns of expression. There is a for loop that starts from 0 to N, so the complexity will be **O(n).**

getExpression()

then, a function named getExpression() is called to customize the input to make it easier to determine operators and operands.

There are some replace and replaceFirst functions that in worst case they count n characters.

Then there is a for loop that iterates through expression chars, so calculating the complexity including replaces + parenthesisCheck + For loop, all of them have the complexity of N so its complexity will be

O(a*n) ➔ O(n)

infixToPostfix()

then, we convert the customized infix expression to postfix. This function has a for loop to iterate N characters. At the end there is a while loop to iterate stack and create postfix expression (stack functions' size can't be much due to continues add, pop, peek calls so it won't have any effects). The complexity will be

O(a*n + b) ➔ O(n)

Project Complexity

evaluatePostfix()

after that, this function will be called to calculate the final answer. There is only one for loop that starts from 0 to N. but for showing step by step solution, replace method is called every time (worst case). so its complexity will be **O(n^2).**