Time Complexity Problem

For this section, you are required to report on the space and time complexity of doBracketsMatch, the function defined in the JavaScript code below. Your report should include clear justifications for your solution.

I will attempt to answer this question by going through the code line by line. Let's first look at the function in question:

```
function doBracketsMatch(inputString, openingSymbol, closingSymbol) {
        let stack = new BracketStack()
        let isOpeningSymbol = isSymbol(openingSymbol)
        let isClosingSymbol = isSymbol(closingSymbol)
        for (let i = 0; i < inputString.length; i++) {
                let value = inputString[i]
                if (isOpeningSymbol(value)) stack.push()
                if (isClosingSymbol(value))
                        if (stack.isEmpty()) return false
                        else stack.pop()
        }
        return stack.isEmpty()
}
```

Now we can go through it line by line to see how much time each instruction would take:

let stack = new BracketStack()

- **This is an object declaration so it takes O(1)**

let isOpeningSymbol = isSymbol(openingSymbol)

- **Another declaration which takes O(1)**
- **O(1) + O(1) = O(2)**

let isClosingSymbol = isSymbol(closingSymbol)

- **Another declaration which takes O(1)**
- **O(2) + O(1) = O(3)**

for (let i = 0; i < inputString.length; i++) {

- **For loop affects the time of the algorithm in a linear fashion depending on n (the length of the string in this case)**
- **O(3) + n x O(1) = O(3) + O(n)**

let value = inputString[i]

- **Another declaration which takes O(1)**
- **O(3) + O(n) + O(1) = O(4) + O(n)**

if (isOpeningSymbol(value)) stack.push()

- **If statement instruction is a constant time action O(1)**
- **O(4) + O(n) + O(1) = O(5) + O(n)**

if (isClosingSymbol(value))

- **If statement instruction is a constant time action O(1)**
- **O(5) + O(n) +O(1) = O(6) + O(n)**

if (stack.isEmpty()) return false -> 0(1)

- **If statement instruction is a constant time action O(1)**
- **O(6) + O(n) + O(1) = O(7) + O(n)**

else stack.pop()

- **else statement instruction is a constant time action O(1)**
- **O(7) + O(n) + O(1) = O(8) + O(n)**

return stack.isEmpty()

- **return statement instruction is a constant time action O(1)**
- **O(8) + O(n) + O(1) = O(9) + O(n)**

So the total would come to T = O(9) + O(n), but since big-o does not care about constants we can regard the time complexity as *T = O(n)*