برای محاسبه عبارت داده شده

ابتدا String ورودی را داخل stringBuilder قرار دادیم ← O(n) (یک بیماین روی کارکترهای رشته اولیه انجام می‌دهد و n تعداد کارکترهای رشته است.

سپس از تابع replaceE استفاده کردیم که یک بیماین روی کارکترهای رشته انجام داده و به جای e مقدار 2.71 قرار می‌دهد ← O(n) (n تعداد کارکترهای رشته ورودی)

سپس تابع replacePI دقیقاً مانند تابع بالا عمل کرده و به جای PI ← 3.14 قرار می‌دهد ← O(n)

سپس تابع splitAndTrim که این هم مانند دو تابع بالا یک بیماین روی کارکترها انجام می‌دهد ← O(n)
(n تعداد کارکترهای رشته ورودی)

سپس تابع infixToPostfix ← ابتدا حداکثر رشته را در یک stringBuilder می‌ریزد ← O(n)

← در نتیجه که stringBuilder.insert استفاده می‌کنیم داریم ← O(n)
← اگر از stringBuilder استفاده نکنیم ← O(n) ولی اگر از این تابع استفاده کنیم پیچیدگی تابع infix برابر O(n²) می‌شود.

سپس تابع calculate ← ابتدا همه متغیرها را داخل stack ذخیره می‌کند. ← O(n)

یک دور داخل حلقه while متغیر stack را بررسی می‌کنیم ← O(n) ← و درصورت استفاده از تابع factorial ← O(n) ← n تعداد اجرای تابع factorial که بستگی به عدد ورودی آن دارد که برابر عدد ورودی است.

در نهایت پیچیدگی محاسبه عبارت از نوع O(n) و بدترین حالت stringbuilder ← O(n²)

☆ در تابع infixTo ← یک حلقه while داریم که تعداد تکرار آن به قدرتی است که پیچیدگی را تغییر دهد

و در تابع calculate تابع factorial دارای O(n) که این موضوع به تعداد تابع فاکتوریل

که مقدار ورودی تابع است.

بستگی دارد و می‌توان پیچیدگی سقفی برای آن تعیین کرد