Time Complexity for Binary Search Tree Operations

Morgan Benavidez

July 29, 2022

1 Introduction

In the following sections, I will be discussing the Average, Worst and Best Time Complexities for the Search, Delete, and Insert functions operating on a Binary Search Tree. All of them share the same time complexities and 'O(1)' is referring to the constant time operations the algorithm takes after locating the node it wants to perform operations on.

2 Average

Let us assume we're operating on a balanced Binary Search Tree for the Average Case time complexity.

$$T(n) = T(\frac{n}{2}) + O(1) \tag{1}$$

$$O(1) = n^{\log_2 1} = n^0 = 1, Master Method - case 2 \tag{2}$$

$$T(n) = \Theta(n^{\log_2 1} * log(n)) \tag{3}$$

$$T(n) = \Theta(1 * log(n)) \tag{4}$$

$$T(n) = \Theta(log(n)) \tag{5}$$

3 Worst

Let us assume we're operating on a perfectly unbalanced Binary Search Tree for the Worst Case time complexity. This effectively makes the BST a Linked List.

Which we intuitively know has a worst case time complexity of O(n) because the node we're looking for could be the last item in the "Linked List".

$$T(n) = T(n-1) + O(1) \tag{6}$$

$$T(n) = T(n-2) + 2 * O(1) \tag{7}$$

$$T(n) = T(n-3) + 3 * O(1) \tag{8}$$

$$\vdots \tag{9}$$

$$T(n) = T(n-n) + n * O(1) \tag{10}$$

$$T(n) = n * O(1) \tag{11}$$

$$T(n) = O(n) \tag{12}$$

4 Best

The best case scenario is when we check the root of the tree and it is the node we need to perform the operation on. This means we will not have to traverse the tree at all and we only have the constant time operations to perform.

$$T(n) = T(n-n) + O(1) \tag{13}$$

$$T(n) = O(1) \tag{14}$$

5 Chart

Algorithm/Case	Average	Worst	Best
Search	$\Theta(log(n))$	O(n)	O(1)
Insert	$\Theta(log(n))$	O(n)	O(1)
Delete	$\Theta(log(n))$	O(n)	O(1)