Assignment Four

Josh Seligman

joshua.seligman1@marist.edu

November 1, 2022

1 Binary Search Tree

1.1 The Data Structure

A binary search tree is a data structure that, for each node in the tree, all child nodes on its left are less than the value and all child nodes on the right are greater than or equal to the value in the given node. Therefore, when performing an in-order traversal by printing out all left-hand nodes, then the value of the given node, and lastly all of the right-hand nodes, all values will be printed out in order. As shown in Figures 1.1 and 1.2, the values are inserted into the tree in the order in which they are received. This can impact the time it takes to traverse the tree to find a given element, which will be examined in Section 1.2.

1.2 Asymptotic Analysis

Algorithm 1 provides the pseudocode for performing a lookup on a binary search tree. As shown on lines 6 and 8, the area of the tree gets cut in half for each level of the recursion tree. This causes the expected runtime for a binary search tree lookup to be the same as binary search at $O(log_2 n)$. However, as displayed in Figures 1.1 and 1.2, the order in which the data arrive makes a huge difference in the number of checks needed to find an element. For instance, when looking for the number 8 in the tree in Figure 1.1, it will start with the 5 and go to the right because $8 > 5$, then compare 8 with the 7 and also go to the right because 8

Figure 1.1: Sample binary search tree for the numbers 5, 3, 1, 2, 7, 6, 8.

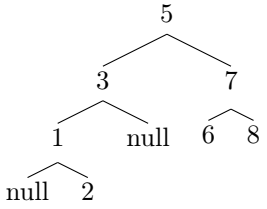
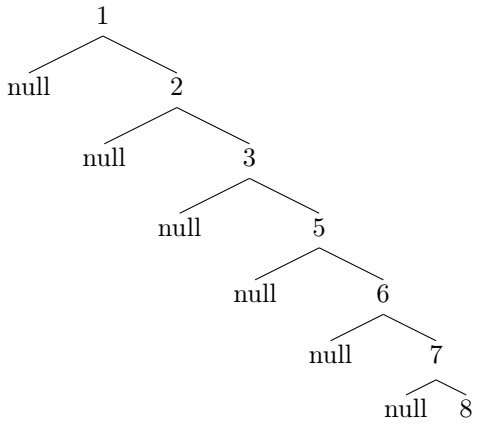


Figure 1.2: Sample binary search tree for the numbers 1, 2, 3, 5, 6, 7, 8.



Algorithm 1 Binary Search Tree Lookup. Assume *cur* starts off as the root of the tree.

```
1:  procedure BSTLookup(target, cur)
2:      out ← false    // Assume target is not found
3:      if target == cur.val then
4:          out ← true    // Found the target value
5:      else if target < cur.val then
6:          out ← BSTLookup(target, cur.left)    // Target is on the left
7:      else   // target ≥ cur.val
8:          out ← BSTLookup(target, cur.right)    // Target is on the right
9:      end if
10:     return out
11: end procedure
```

> 7, and then end when it finds the 8. Since there are 7 elements in the tree, it should take around $log_2 7$ comparisons to find an element, and 3 comparisons is very close to the expected outcome. However, when trying to find 8 in the tree in Figure 1.2, one will have to compare 8 with every single element in the tree, which degrades the binary search tree lookup to be $O(n)$ as it is just doing a linear search. Therefore, when working with a binary search tree, it is important to make sure the data are shuffled in a random order to ensure the tree is as close to being balanced as possible. Since shuffling is an $O(n)$ operation, taking the time to shuffle a sorted array before putting the data in a binary search tree will save a lot of time in the long run especially when doing a lot of lookups within the tree.