# PA02 Report on Timing Analysis of Search in a BST

Danming Wang, 5833587, 06/05/2019

Section 1: Trends in the average time to search in a BST
This table reports statistics about the ***average time to search in a BST*** for different data sets.

| Dataset | Number of runs (W) | Minimum Time (micro seconds) | Maximum time (micro seconds) | Median (micro seconds) |
|---|---|---|---|---|
| 20 Ordered | 10 | 0.15 | 0.25 | 0.15 |
| 20 Random | 10 | 0.05 | 0.15 | 0.1 |
| 100 Ordered | 10 | 0.4 | 0.52 | 0.41 |
| 100 Random | 10 | 0.04 | 0.06 | 0.05 |
| 1000 Ordered | 10 | 5.517 | 5.997 | 5.6035 |
| 1000 Random | 10 | 0.059 | 0.073 | 0.061 |

(1) *Explain the trends that you observe in the above table.*

For data sets with same number of datas, the average time to search in a BST for ordered data sets is more than that for random data sets.
For ordered data sets, the average time to search in a BST increases as number of datas in the dataset increases.

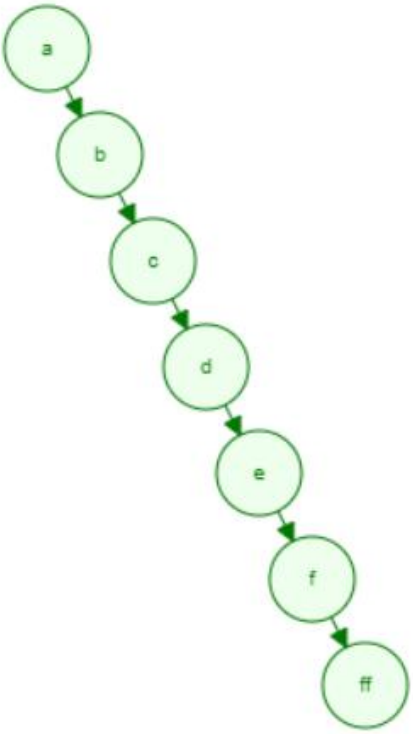
(2) *Are the trends as you expect? Why or why not?*

Yes.

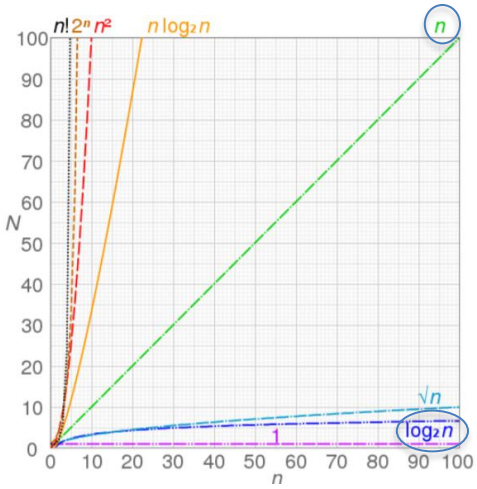
Because when inserting the ordered data sets into a BST, the root is the data with the minimum value and the resulting BST is a tree with only right children. So the worst case for search in such BST is to search through all the keys and the Big O running time of search is O(N). And O(N) increases as N increases.

When inserting a random data set into a BST, the resulting BST is relatively balanced. As we have analyzed before, the Big O running time of search in a balanced BST is O(logN), which is smaller than O(logN).

*(3) (Optional) Add any other related plots that you think are relevant to this question*



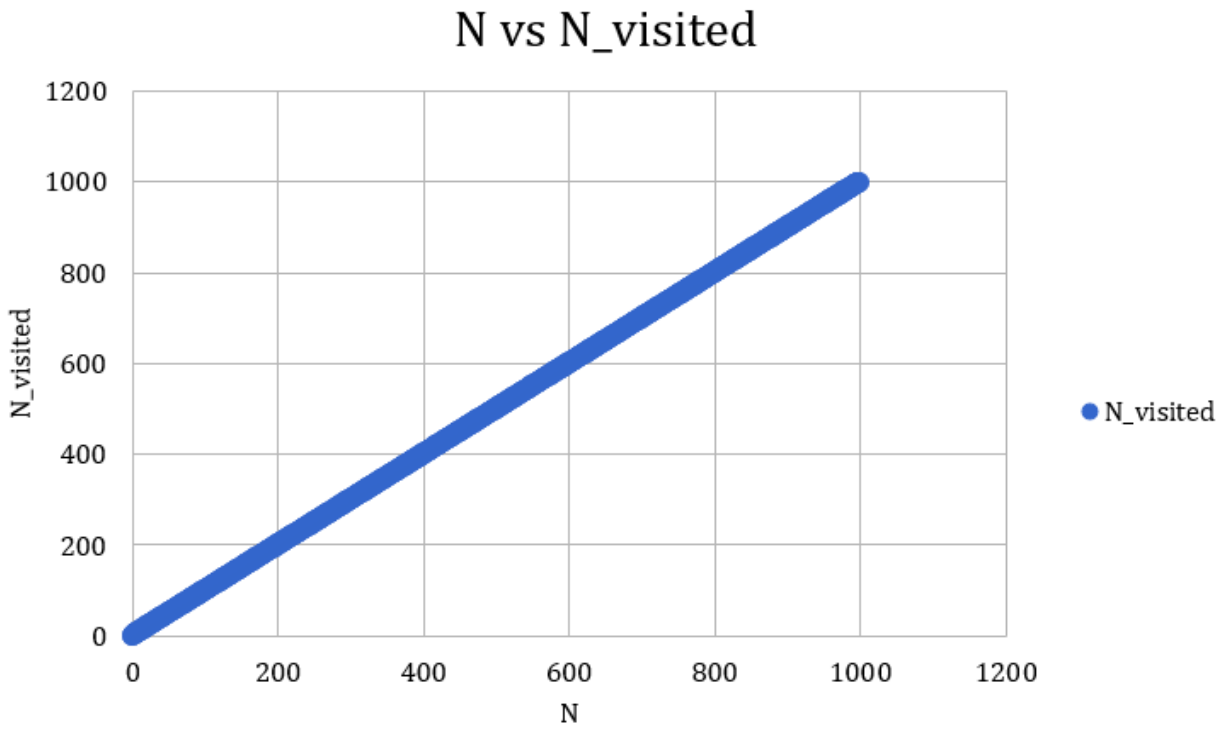

a simple BST example for ordered dataset



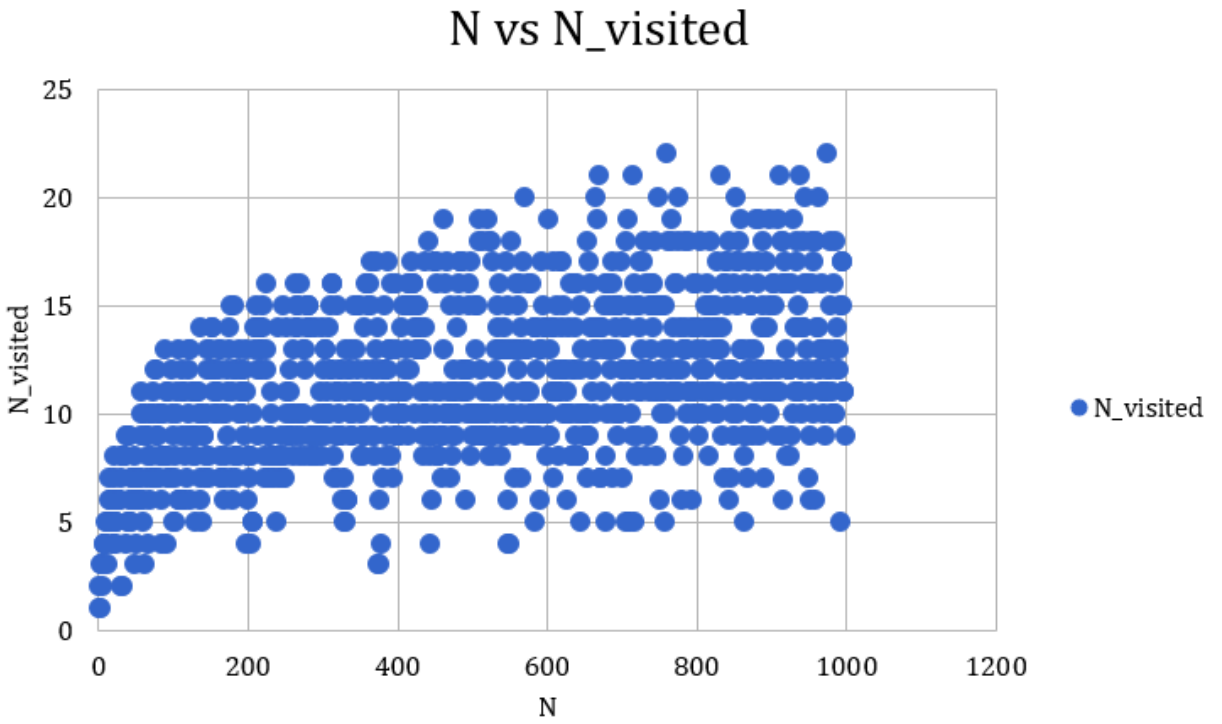

orders of growth

Graph (Import from google sheets)

input_1000_ordered.csv




input_1000_random.csv



(1) How does the number of operations for insert vary with the number of the current nodes present in the tree when (a) nodes are inserted into the BST in alphabetical order (b) nodes are inserted in random order?

When nodes are inserted into the BST in alphabetical order, number of operations for insert is the same as the number of the current nodes present in the tree (N). The number of operations for insert grows linearly with respect to the current nodes present in the tree. O(N)

When nodes are inserted into the BST in random order, number of operations for insert grows logarithmically with respect to the number of the current nodes present in the tree (N) increases. O(logN)

(2) Include the code for your insert function, do a Big O analysis and use to explain the trends you observed?

```
bool BST::insert(string n, double r) {
    // handle special case of empty tree first
    if (!root) {
            root = new Node(n,r);
            return true;
    }
    // otherwise use recursive helper
    return insert(n,r, root);
}
bool BST::insert(string m, double r,  Node *n) {
    if (m == n->name)
            return false;
    if (m < n->name) {
            if (n->left)
                return insert(m,r, n->left);
            else {
                n->left = new Node(m,r);
                n->left->parent = n;
                return true;
            }
    }
    else {
            if (n->right)
                return insert(m,r, n->right);
            else {
                n->right = new Node(m,r);
                n->right->parent = n;
                return true;
            }
    }
}
```

worst case: O(N)

Here running time is isomorphic to the number of nodes visited.

The worst case happens when the input data set is in alphabetical order. You need do go through all the current nodes present in the BST to insert a new Node. So the Big O running time of it is O(N), and N_visited = N. The graph of O(N) is similar to the graph of inserting 1000 ordered datas.

(3) Are the trends you observed as you expect? Why or Why not?

Yes.

Because we have done the Big O analysis for balanced BST and generic BST before. The Big O running time of insert for balanced BST is O(logN) and the Big O running time of insert for generic BST is O(N).

When the input data set is in alphabetical order, the BST for this ordered data set is a tree with only right children and the Big O running time of insert is O(N) as I have explained in (2).

When the input data set is in random order, the BST for this random data set tend to be balanced as the data size grows. The worst case of insert in a balanced BST happens when the N_visited equals to the height of the tree before insert operation, i.e. N_visited=LogN. So the Big O running time of insert in a balanced BST is O(logN), the graph of which is similar to the trends shown in the graph of inserting 1000 random datas.