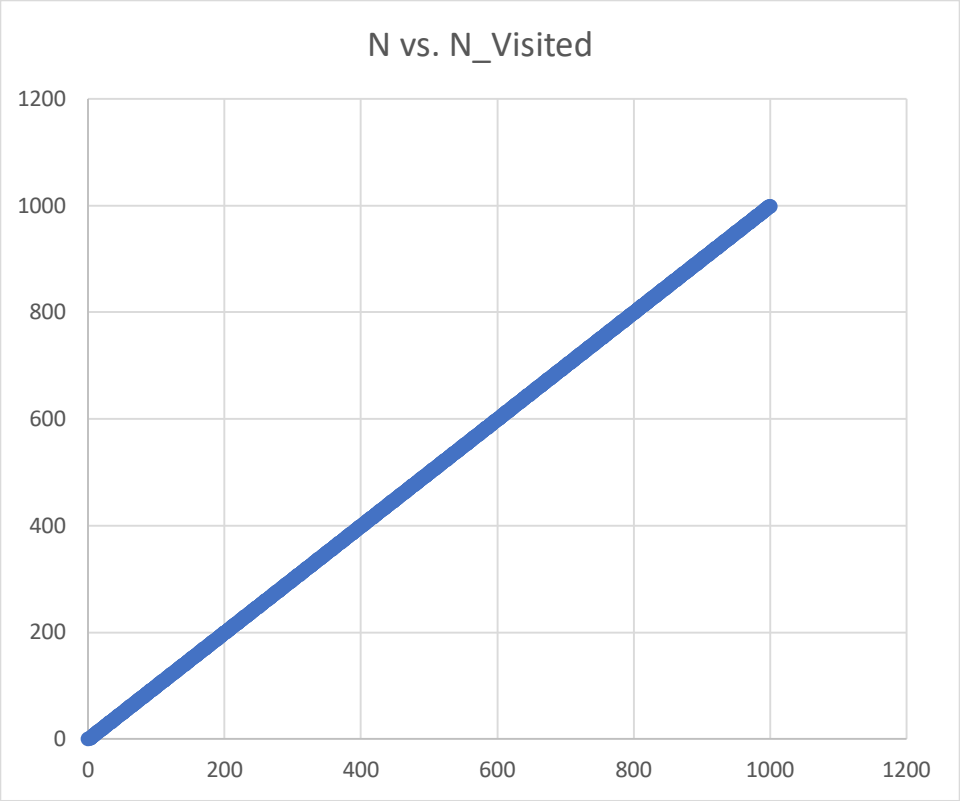
PA02 Report on Timing Analysis of Search in a BST

Trends in the average time to search for N codes in a BST (N is defined by the size of the Dataset)
This table reports statistics about the ***average time to search in a BST*** for different data sets.

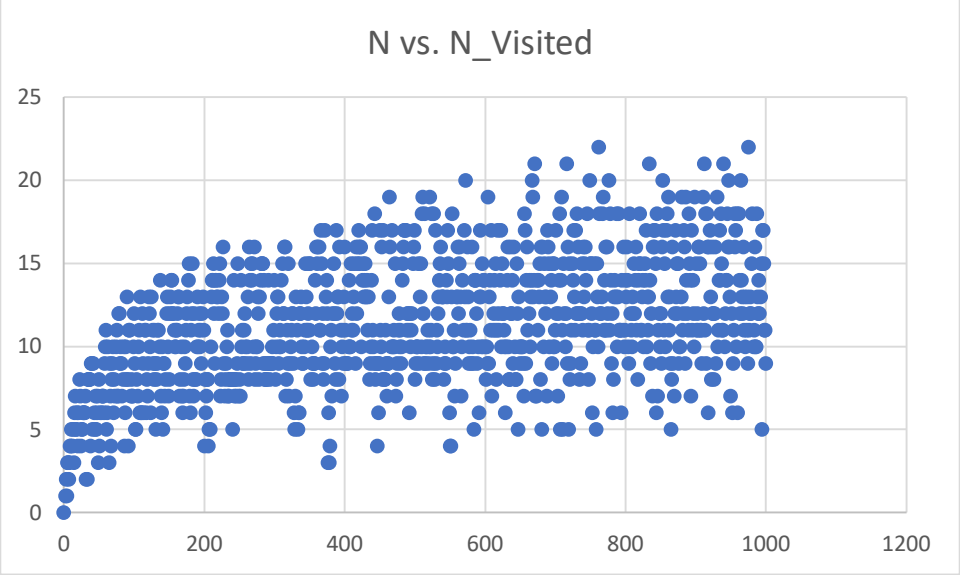
Dataset	Number of runs (W)	Minimum Time (micro seconds)	Maximum time (micro seconds)	Average (micro seconds)
20 Ordered	50	0.15	0.4	0.189
20 Random	50	0.1	0.35	0.194
100 Ordered	50	0.14	0.22	0.1592
100 Random	50	0.09	0.26	0.107
1000 Ordered	50	0.092	0.125	0.0971
1000 Random	50	0.009	0.153	0.09818

1. *Explain the trends that you observe in the above table.*
   Even though the size of datasets increased, the average time to access to node is still within a small range.
2. *Are the trends as you expect? Why or why not?*
   These trends are as I expected. The larger the dataset is, the longer it takes to search all elements, but the time to search a single node is still the same. Thus, even though the total time to search a tree increased, the time to access the node is still similar to previous tests.
3. *(Optional) Add any other related plots that you think are relevant to this question*

Graph of b)



This graph is for ordered 1000 inputs ↑



This graph is for random 1000 inputs

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?
    a. For inserting an ordered dataset, the number of operations is equal to the number of current nodes because all the new nodes are bigger than the current nodes.

Thus, the function will need to search and compare all the old nodes in order to insert at the end of the tree.

b. For inserting a random dataset, the number of operations are related to the depth of the parent of the new nodes. Thus, the nodes that the function visited is significantly less than inserting a sorted dataset.

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

```
void Movie::insert(string n, double r) {
    // create a new root
    if (!root) {
        root = new Movie::film(n,r,0);
        return;
    }

    // if the tree is not empty, use recursive helper to create nodes in the tree
    insert(n,r,root);
    return;
}

void Movie::insert(string n, double r, Movie::film *node) {
    // check if the movie is already in the tree
    if (n == node->name && r == node->rating) {
        return;
    }

    // if the n is smaller than the current name
    // if the left child DNE, create a new left child with name n
    // else insert under the left child

    // if the n is bigger than the current name
    // if the right child DNE, create a new right child with name n
    // else insert under the right child
    if (n < node->name) {
        if (node->left) {
            insert(n,r,node->left);
            return;
        } else {
            node->left = new Movie::film(n,r,node->depth+1);
            node->left->parent = node;
            return;
        }
    } else {
        if (node->right) {
            insert(n,r,node->right);
            return;
        } else {
            node->right = new Movie::film(n,r,node->depth+1);
            node->right->parent = node;
            return;
        }
    }

    return;
}
```

The worst case of the insert function is to insert a new node at the end of the tree, so the time complexity is O (log N) for insert random dataset and O (N) for sorted dataset. Thus, when insert a sorted dataset, the function takes longer and access more nodes, and the function access less nodes and takes less time when insert a random dataset.

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

This trend I observed is what I expected. When insert data from a ordered list, it is essentially creating a tree looks like a linked list, but when insert data from a random list, it is creating a balanced tree. Thus, since the balanced takes less time to both insert and search, I expected the inserting random 1000 inputs takes less time.