Insight: Listing the keys that result from an inorder traversal of a binary search tree will produce a sorted list.

Algorithm:
1. Insert keys into a binary search tree.
2. Perform an inorder traversal of the binary search tree, printing the keys as each node is visited.

Time Complexity Analysis:
1. The time complexity of insertion and an inorder traversal are dependent on the height of the binary search tree.
   1.1. Insertion:
      1.1.1. The time complexity of building a tree using the insertion algorithm is dependent on the height of the tree after the last key has been inserted.
      1.1.2. The height of the tree is dependent on the order in which keys are inserted. Consider the two examples below. Let *n* represent the number of keys. In both examples, $n = 7$.
         1.1.2.1. Suppose $n = 7$ keys are inserted in the following order: **nancy**, **gabrielle**, **ursula**, **daphne**, **julia**, **rosemary**, and **yvette**. The resulting tree is shown in Figure 1. Note that the height of the tree is $\lfloor \log_2 n \rfloor$. The number of operations required to build the tree is $O(n\lceil \log_2 n \rceil)$ Inserting key nancy requires $O(1)$ operations. Inserting **gabrielle** requires $O(\log_2 2)$ operations. Inserting **ursula** requires no more than $O(\log_2 3)$ operations. All insertions require no more than $O(\log_2 7) = O(\log_2 n)$ operations. There are 7 or *n* insertions. The time required to insert all keys is no more than the sum of the individual insertion or $O(n \log_2 n)$.

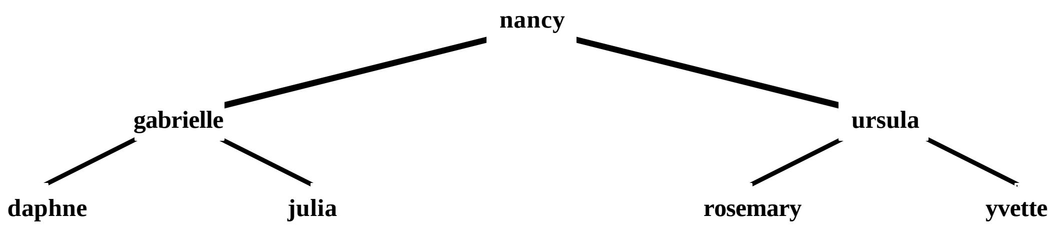

Figure 1. Binary tree for keys: **nancy**, **gabrielle**, **ursula**, **daphne**, **julia**, **rosemary**, and **yvette**.

         1.1.2.2. Suppose $n = 7$ keys are inserted in the following order: **daphne**, **gabrielle**, **julia**, **nancy**, **rosemary**, **ursula**, and **yvette**. The resulting tree is shown in Figure 2. Note that the height of the tree is $h(n) = n - 1 = 6$ and the number of operations required to build the tree is $O(n)$.

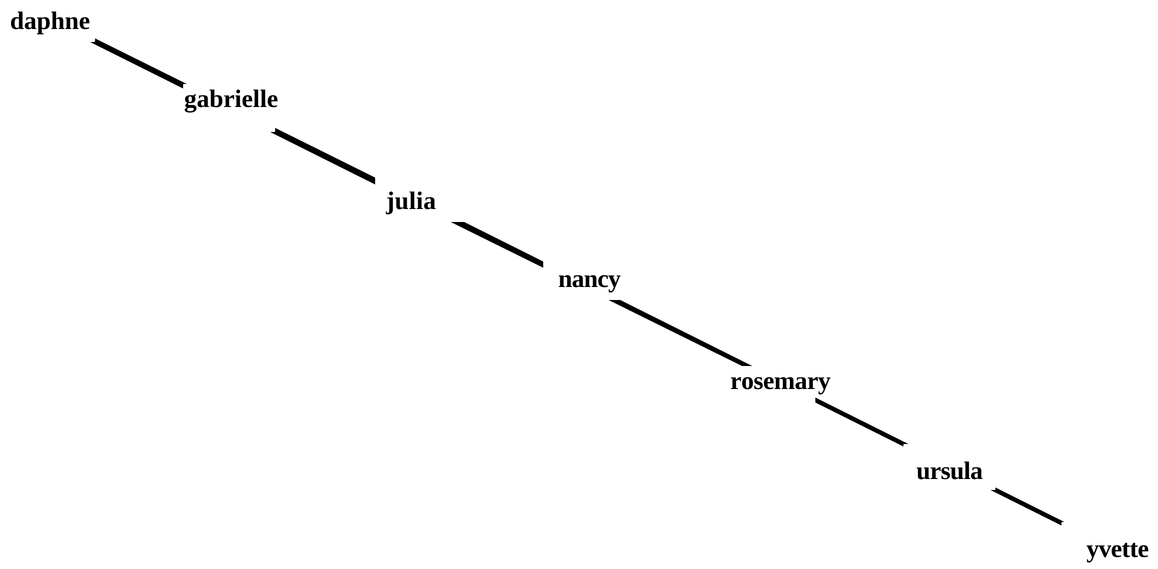

Figure 2. Binary tree for keys: **daphne**, **gabrielle**, **julia**, **nancy**, **rosemary**, **ursula**, and **yvette**

In the worst case, each insertion requires $O(n)$ operations. The sum of the time required for individual insertions is $O(n^2)$ .Neither the best-case time complexity $O(n \log_2 n)$ nor the worst-case time complexity $O(n^2)$will adequately characterize the time complexity of the tree sort algorithm.

What we want to know is the height of an average tree. For $n = 7$ keys we will have to find the average height of the number of trees that can be created from seven keys. We will have to find the average height of all the permutations of seven keys. $P(n,n) = n!$

In the average case, on a randomly ordered list of length n, treesort performs

$$2n \ln n + O(n) \approx 1.39 n \log_2 n + O(n)$$

comparisons of keys.[1]

1.2. Inorder Traversal

1.2.1. An inorder traversal is accomplished by visiting every node in the tree. Function inorder will be called $2n+1$ times where n is the number of nodes in the tree if *inorder* is implemented as a recursive function. The time complexity of an inorder traversal is $O(n)$.

[1] Kruse and Ryba *Data Structures and Program Design in C++*, Prentice-Hall, 1999 ISBN 0-13-768995-0, p 454