Comparing Leaf and Root Insertion

Jaco Geldenhuys, Brink van der Merwe

Computer Science Division, Department of Mathematical Sciences, Stellenbosch University, Private Bag X1, 7602 Matieland, SOUTH AFRICA

ABSTRACT

We consider two ways of inserting a key into a binary search tree: leaf insertion which is the standard method, and root insertion which involves additional rotations. Although the respective cost of constructing leaf and root insertion binary search trees, in terms of comparisons, are the same in the average case, we show that in the worst case the construction of a root insertion binary search tree needs approximately 50% of the number of comparisons required by leaf insertion.

KEYWORDS: Binary search trees, leaf insertion, root insertion.

1 INTRODUCTION

Binary search trees have been used in computer science for about fifty years, but as Jonassen and Knuth noted [5], even a simple question about these data structures may require an unexpectedly non-trivial analysis to answer. In this paper we consider the relative merits of leaf insertion and root insertion, two ways of constructing (nonbalanced) binary search trees.

Leaf insertion is the “common” method of adding a key to a binary search tree. The result of inserting a key $a$ into an empty tree, is a tree with a root node with $a$ as its key and empty left and right subtrees. If $a$ is inserted into a non-empty tree, the result is the original tree, but with $a$ inserted recursively into the left (or right) subtrees, depending on whether it is smaller (or larger) than the root key.

Root insertion is similar to leaf insertion, except that after a key $a$ has been inserted, its node $n$ is moved to the root of the tree through a series of rotations. There are two kinds of rotations, as shown in Figure 1. If $n$ is the left child of its parent, the parent is right rotated. Similarly, if $n$ is the right child of its parent, the parent is left rotated. The construction of a four element tree is shown in Figure 2. For each root insertion, the number of comparisons required is equal to the number of rotations needed to move the new key to the root.

![Figure 1: Right and left rotation](image)

A comparison moves the key, to be inserted, down one level, while each rotation moves the key back up one level. Rotations are of course well-known from their use in AVL and splay trees. See for example [1] and [8].

It is important to note that rotations preserve the inorder numbering of a tree. In other words, rotation in a binary search tree produces another binary search tree.

To build an $n$-element tree, root insertion requires precisely $n - 1$ comparisons (compared to leaf insertion’s $\Theta(n \log n)$) in the best case, when the keys are arranged in ascending or descending order. This raises the question of whether it is possible that root insertion also has better average-case and worst-case behaviour (at least in terms of number of comparisons). Our main goal is to obtain the explicit value for $W_n^r$ in the following table.

<table>
<thead>
<tr>
<th></th>
<th>Leaf insertion</th>
<th>Root insertion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Best</td>
<td>$2 + \lfloor \log n \rfloor (n + 1) - 2^\lfloor \log n \rfloor + 1$</td>
<td>$n - 1$</td>
</tr>
<tr>
<td>Ave.</td>
<td>$2(n + 1)H_{n+1} - 4n - 2$</td>
<td>$A_n^r$</td>
</tr>
<tr>
<td>Worst</td>
<td>$n(n - 1)/2$</td>
<td>$W_n^r$</td>
</tr>
</tbody>
</table>

We denote by $H_n$ the $n$-th Harmonic number and $\log n$ is taken base 2. The average case for leaf
After introducing the necessary notation in Section 2, we prove in Section 3 that the tree built by leaf insertion from $a_1, a_2, \ldots, a_n$, is identical to the tree built by root insertion from $a_n, a_{n-1}, \ldots, a_1$. The worst case performance of root insertion is analysed in Section 4, and experimental results and conclusions are presented in Sections 5 and 6, respectively.

Our interest in the performance of root insertion stems from [7, Exercise 12.85], where the reader is asked to compute $W_n^r$ and from [9], where the result was verified by exhaustive search, for sequences of length 10 and smaller. The version of root insertion described above, may more precisely be referred to as bottom-up root insertion. In [9], top-down root insertion is considered and it is shown that:

1. The tree built by leaf insertion from $a_1, a_2, \ldots, a_n$ is identical to the tree built by top-down root insertion from $a_n, a_{n-1}, \ldots, a_1$.
2. $A_n^r = 2(n+1)H_{n+1} - 4n - 2$ for top-down root insertion.

In Section 3 we show that the trees constructed, and the number of comparisons required, for top-down and bottom-up root insertion are always equal.

According to Knuth [6], leaf insertion was discovered independently by several people during the 1950s. He cites an unpublished memorandum by A. I. Dumey dated August 1952, but the first published algorithms appeared in the early 1960s [2, 4]. The rotation operation was first proposed by Adelson-Velsky and Landis in their 1962 paper on balanced trees [1].

![Construction of a four element binary search tree by root inserting 1, 4, 3, and 2; the total number of comparisons (or rotations) is equal to 5.](image-url)

**Figure 2:** Construction of a four element binary search tree by root inserting 1, 4, 3, and 2; the total number of comparisons (or rotations) is equal to 5.
2 NOTATION

Let $K$ be an arbitrary set of keys with a corresponding total ordering $\prec$. A sequence $s = a_1a_2\ldots a_n$ is considered as a specific permutation of the $n$ distinct keys $a_1, \ldots, a_n$. The length $n$ of $s$ is denoted by $|s|$, and the reverse sequence $a_na_{n-1}\ldots a_1$ by rev($s$).

By $T_K$ we denote the set of binary trees over $K$, which are defined inductively as follows. We have that $t \in T_K$ if and only if

1. $t$ is the empty tree $\bot$, or
2. $t = a[u, v]$, where $u, v \in T_K$ and $a \in K$.

The following attributes will play an important role in the remainder of this paper.

<table>
<thead>
<tr>
<th>$t = \bot$</th>
<th>$t = a[u, v]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K(t)$</td>
<td>undefined</td>
</tr>
<tr>
<td>$L(t)$</td>
<td>$a$</td>
</tr>
<tr>
<td>$R(t)$</td>
<td>undefined</td>
</tr>
<tr>
<td>$H(t)$</td>
<td>0</td>
</tr>
<tr>
<td>${ \text{keys}(t) }$</td>
<td>${ a } \cup \text{keys}(u) \cup \text{keys}(v)$</td>
</tr>
<tr>
<td>${ \text{leaves}(t) }$</td>
<td>${ a }$ if $u = v = \bot$, else $\text{leaves}(u) \cup \text{leaves}(v)$</td>
</tr>
</tbody>
</table>

The set of binary search trees is a subset of $T_K$ denoted by $B_K$, and $t \in B_K$ if and only if $t \in T_K$ and

1. $t = \bot$, or
2. $t = a[u, v]$ where $u, v \in B_K$ and $b \prec a$ for all $b \in \text{keys}(u)$ and $b \prec c$ for all $c \in \text{keys}(v)$.

Since we deal exclusively with binary search trees from now on, we shall refer to them simply as trees. Note that we do not consider trees with duplicate keys.

We are now ready to formally define leaf insertion, bottom-up root insertion, and top-down root insertion.

**Definition 2.1** Let $t \in B_K$ and $a \in K$ with $a \notin \text{keys}(t)$. The tree that results from the leaf insertion of $a$ into $t$ is

$$LI(t, a) = \begin{cases} \{ \bot, \bot \} & \text{if } t = \bot, \\ K(t)[LI(L(t), a), R(t)] & \text{if } a \prec K(t), \\ K(t)[L(t), LI(R(t), a)] & \text{otherwise}. \end{cases}$$

Let $s = a_1a_2\ldots a_n$. The leaf insertion tree constructed from $s$ is given by

$$LI(s) = \begin{cases} \bot & \text{if } |s| = 0, \\ LI(LI(a_1a_2\ldots a_{n-1}), a_n) & \text{otherwise}. \end{cases}$$

**Definition 2.2** Let $t \in B_K$ and $a \in K$ with $a \notin \text{keys}(t)$. The tree that results from the bottom-up root insertion of $a$ into $t$ is

$$RT(t, a) = \begin{cases} a[\bot, \bot] & \text{if } t = \bot, \\ K(u)[L(u), K(t)[R(u), R(t)]] & \text{if } a \prec K(t), \\ K(u)[K(t)[L(u), R(u)], R(t)] & \text{otherwise}. \end{cases}$$

Let $s = a_1a_2\ldots a_n$. The bottom-up root insertion tree constructed from $s$ is given by

$$RT(s) = \begin{cases} \bot & \text{if } |s| = 0, \\ RT(RT(a_1a_2\ldots a_{n-1}), a_n) & \text{otherwise}. \end{cases}$$

Let $l$ and $r$ be symbols that are not in $K$. Denote by $T_K[l, r]$ the trees in $T_{K[l, r]}$, with $K(t) \in K$, exactly one leaf node in $L(t)$ labeled by $l$, exactly one leaf node in $R(t)$ labeled by $r$, and all other nodes are labeled by keys in $K$. Let $t_1, t_2 \in T_{K[l, r]}$ and $t \in T_K[l, r]$. Then $[t_1, t_2]$ denotes the tree obtained by replacing the node labeled by $l$ with $t_1$, and the node labeled by $r$ with $t_2$. Using the same notation as for trees in $T_K$, we denote by $a[t_1, t_2]$, with $a \in K$, $t_1, t_2 \in T_{K[l, r]}$, the tree $t$ in $T_K[l, r]$ with $R(t) = a, L(t) = t_1$ and $R(t) = t_2$. We denote by $B_K[l, r]$ all trees $t \in T_K[l, r]$, such that $t[\bot, \bot] \in B_K$.

**Definition 2.3** Let $t \in B_K$ and $a \in K$ with $a \notin \text{keys}(t)$. The tree that results from the top-down root insertion of $a$ into $t$ is given by

$$RT^{\text{top}}(t, a) = RT^r(t, a[l, r]),$$

where $RT^r(t_1, t_2) \in B_K$, for $t_1 \in B_K$ and $t_2 \in B_K[l, r]$, is defined inductively on the height of $t_1$, as follows.

$$RT^r(t_1, t_2) = \begin{cases} t_2[\bot, \bot] & \text{if } t_1 = \bot, \\ RT^r(L(t_1), t_2[l, v]) & \text{if } K(t_2) \prec K(t_1), \\ RT^r(R(t_1), t_2[u, r]) & \text{otherwise}. \end{cases}$$

where $u = K(t_1)[L(t_1), l]$

Let $s = a_1a_2\ldots a_n$. The top-down root insertion tree constructed from $s$ is given by

$$RT^{\text{top}}(s) = \begin{cases} \bot & \text{if } |s| = 0, \\ RT^{\text{top}}(RT^{\text{top}}(a_1\ldots a_{n-1}), a_n) & \text{otherwise}. \end{cases}$$

To illustrate top-down root insertion, we consider $RT^{\text{top}}(3[1, \bot, 4[1, \bot], 2)$. We have that

$$RT^{\text{top}}(3[1, \bot, 4[1, \bot], 2) = RT^{\text{top}}(3[1, \bot, 4[1, \bot], 2, [2, 3[r, 4[1, \bot]]) = RT^{\text{top}}(3[2, 2[1, \bot, 3[3, 4[1, \bot]]) = 2[1, \bot, 3[3, 4[1, \bot]])$$
The definitions of top-down and bottom-up root insertion, are formal versions of the pseudocode for root insertion as described in [7] and [9], respectively.

The difference between leaf, and for example top-down root insertion, looks formidable when comparing Definitions 2.1 and 2.2, but as explained in the introduction, for \( t \in B_K \) and \( a \in K \), \( LI(t, a) \) and \( RI(t, a) \) require the same number of comparisons. From Definition 2.3, it can also be shown that \( LI(t, a) \) and \( RI^{top}(t, a) \) require the same number of comparisons. From now on we denote by \( C(t, a) \) the number of comparisons required for \( LI(t, a) \), \( RI(t, a) \) or \( RI^{top}(t, a) \). We can now define the cost required to build a binary tree with leaf insertion, bottom-up, and top-down root insertion respectively.

**Definition 2.4** For \( s = a_1a_2 \ldots a_n \), let \( \bar{s} = a_1a_2 \ldots a_{n-1} \). The cost to construct a tree for \( s \) with leaf insertion, is denoted by \( LC(s) \) and defined inductively as follows.

\[
LC(s) = \begin{cases} 
0 & \text{if } |s| = 1, \\
C(LT(\bar{s}), a_n) + LC(\bar{s}) & \text{if } |s| > 1.
\end{cases}
\]

Similarly, the cost to construct a tree for \( s \) with bottom-up root insertion, is denoted by \( RC(s) \) and defined inductively as follows.

\[
RC(s) = \begin{cases} 
0 & \text{if } |s| = 1, \\
C(RT(\bar{s}), a_n) + RC(\bar{s}) & \text{if } |s| > 1.
\end{cases}
\]

Finally, the cost to construct a tree for \( s \) with top-down root insertion, is denoted by \( RC^{top}(s) \) and defined inductively as follows.

\[
RC^{top}(s) = \begin{cases} 
0 & \text{if } |s| = 1, \\
C(RT^{top}(\bar{s}), a_n) + RC^{top}(\bar{s}) & \text{if } |s| > 1.
\end{cases}
\]

### 3 PROPERTIES OF ROOT INSERTION

The main results in this section state that the leaf insertion tree of a sequence \( s \) is identical to the bottom-up root insertion tree of \( rev(s) \), and that top-down and bottom-up root insertion are equivalent in terms of trees constructed and number of comparisons required. In the first result, we show that if we use Definition 2.2, from the previous section, for top-down root insertion, then the inserted key do indeed end up at the root of the newly constructed tree.

**Lemma 3.1** Let \( t \in B_K \) and \( a \in K \) with \( a \not\in \text{keys}(t) \). Then \( K(RI(t, a)) = a \).

**Proof** (By strong induction over tree heights.)

**Base case:** Let \( t = \perp \) so that \( H(t) = 0 \). Then \( K(RI(t, a)) = K(RI(\perp, a)) = K(a[\perp, \perp]) = a \).

**Induction step:** Assume that the claim holds for all trees of height less than \( n \). In other words, \( K(RI(t, a)) = a \) for all \( t \in B_K \) such that \( H(t) < n \). Now consider \( t = [b[u, v] \in B_K \) where \( H(t) = n \). This means that \( H(u) < n \) and \( H(v) < n \). If \( a < b \), then

\[
K(RI(t, a)) = K(RI([b[u, v], a]) = K(K(w)[L(w), b[R(w), v]]) = w = RI(u, a); a < b = K(a[L(w), b[R(w), v]]) \text{ induc. } H(u) < n = a
\]

and similarly if \( b < a \).

The result stated in the next lemma will be used in an inductive way in order to obtain Theorem 3.3.

**Lemma 3.2** Let \( t \in B_K \) and \( a, b \in K \) with \( a, b \not\in \text{keys}(t) \), such that \( a \neq b \). Then \( RI(LI(t, b), a) = LI(RI(t, a), b) \).

**Proof** (By strong induction over tree heights.)

**Base case:** Let \( t = \perp \) and therefore \( H(t) = 0 \). If \( a < b \), then

\[
RI(LI(t, b), a) = RI(b[\perp, \perp], a) = a[\perp, b[\perp, \perp]]
\]

and

\[
LI(RI(t, a), b) = LI(a[\perp, \perp], b) = a[\perp, b[\perp, \perp]],
\]

and similarly, if \( b < a \).

**Induction step:** Assume that the claim holds for trees of height less than \( n \). In other words, \( LI(RI(t, a), b) = RI(LI(t, b), a) \) for all \( t \in B_K \) such that \( H(t) < n \). Now consider \( t = c[u, v] \in B_K \) where \( c \in K \) and \( H(t) = n \). This means that \( H(u) < n \) and \( H(v) < n \). There are six orderings of \( a, b, \) and \( c \) to consider. We assume that \( a < b < c \). The induction step for the other cases can be obtained by similar arguments.

\[
RI(LI(t, b), a) = RI(LI(c[u, v], b), a) \quad b < c
\]

\[
RI(c[L(u, b), v], a) \quad b < c
\]

\[
a[L(w'), c[R(w'), v]] = w' = RI(LI(u, b), a); a < c
\]

and

\[
LI(RI(t, a), b) = LI(RI(c[u, v], a), b) \quad b < c
\]

\[
LI(a[L(w), c[R(w), v]], b) \quad w = RI(u, a); a < c
\]

\[
a[L(w), LI(c[R(w), v], b)] \quad a < b
\]

\[
a[L(w), c[LI(R(w), b), v]] \quad b < c
\]
Suppose that \( w = RI(u, a) = d[x, y] \). By Lemma 3.1, \( K(RI(u, a)) = a \), hence \( d = a \). Thus
\[
L(w') = L(RI(LI(u, b), a)) \quad \text{def. of } w' \\
= L(LI(RI(u, a), b)) \quad \text{induc., } H(u) < n \\
= L(LI(a[x, y, b])) \\
= L(a[x, LI(y, b)]) \\
= x \\
= L(w) \\
\]
and
\[
R(w') = R(RI(LI(u, b), a)) \quad \text{def. of } w' \\
= R(LI(RI(u, a), b)) \quad \text{induc., } H(u) < n \\
= R(LI(a[x, y, b])) \\
= R(a[x, LI(y, b)]) \\
= LI(y, b) \\
= LI(R(w), b) \\
= w = d[x, y] \\
\]
So
\[
a[L(w), c[LI(R(w), b), v]] = a[L(w'), c[R(w'), v]], 
\]
and therefore \( LI(RI(t, a), b) = RI(LI(t, b), a) \).

**Theorem 3.3** Let \( s \) be a sequence over \( K \). Then \( LT(s) = RT(\text{rev}(s)) \).

**Proof** (By strong induction over sequence lengths.) Base case: If \( s = a_1 \) and therefore \( |s| = 1 \), then \( LT(s) = a_1 = RI(LI(t, b), a) \).

Induction step: Assume that the claim holds for all sequences \( s \) such that \( |s| < n \). In other words, \( LT(s) = RT(\text{rev}(s)) \) for all sequences \( s \) such that \( |s| < n \). Consider \( s = a_1a_2\ldots a_n \).
\[
LT(s) \\
= LI(LT(a_1\ldots a_n), a_n) \\
= LI(RT(a_n\ldots a_1), a_n) \quad (\ast) \\
= LI(RI(RT(a_n\ldots a_2), a_1), a_n) \\
= RI(LI(RT(a_{n-1}\ldots a_2), a_n), a_1) \quad \text{Lemma 3.2} \\
= RI(LI(LT(a_2\ldots a_{n-1}), a_1), a_1) \quad (\ast\ast) \\
= RI(LT(a_2\ldots a_n), a_1) \\
= RI(RT(a_n\ldots a_2), a_1) \\
= RT(a_n\ldots a_1) \quad (\ast\ast\ast) \\
= RT(\text{rev}(s)) \\
\]
(The justification for steps (\ast), (\ast\ast), and (\ast\ast\ast) is based on induction: \( |a_1\ldots a_{n-1}| < n \), \( |a_{n-1}\ldots a_2| < n \), and \( |a_2\ldots a_n| < n \).)

The theorem just proven has important consequences: Any tree shape possible with leaf insertion is also possible with root insertion. Also, for a given tree \( t \), the number of sequences \( s \) and number of sequences \( s' \) of keys, such that \( RT(s) = t = LT(s') \), are equal.

In the final result we show the equivalence of top-down and bottom-up root insertion.

**Theorem 3.4** Let \( s = a_1a_2\ldots a_n \) be a sequence of distinct keys. Then \( RT(s) = RT^{\text{top}}(s) \) and \( RC(s) = RC^{\text{top}}(s) \).

**Proof** Theorem 3.3 states that \( RT(s) = LT(\text{rev}(s)) \), and we know from [9] that \( RT^{\text{top}}(s) = LT(\text{rev}(s)) \), and therefore \( RT(s) = RT^{\text{top}}(s) \). By Definitions 2.2 and 2.3, \( RC(s) = C(RT(a_1\ldots a_{n-1}), a_n) + RC(a_1\ldots a_{n-1}) \) and
\[
RC^{\text{top}}(s) = C(RT^{\text{top}}(a_1\ldots a_{n-1}), a_n) + RC^{\text{top}}(a_1\ldots a_{n-1})
\]
for \( n \geq 2 \). Since \( RT(a_1\ldots a_{n-1}) = RT^{\text{top}}(a_1\ldots a_{n-1}) \), it follows by induction that \( RC(s) = RC^{\text{top}}(s) \).

In the remainder of the paper we will only consider bottom-up root insertion, and will simply refer to it as root insertion.

### 4 WORST-CASE COST OF ROOT INSERTION

We define the worst-case cost of root insertion as \( W^*_n := \max\{RC(s)\}_{|s| = n} \). Our analysis of the worst-case cost of root insertion is based on expressing \( RC(s) \) in terms of \( RC(\tilde{s}) \), where we obtain \( \tilde{s} \) from \( s \) by removing two elements from \( s \). From this recurrence we derive an upper bound for \( W^*_n \), and finally we construct a sequence for which the upper bound is reached. The proof of the next lemma contains no deep insight, but it is technical in nature. We will in fact only provide a sketch of the proof.

**Lemma 4.1** Suppose \( |s| \geq 3 \). Then there is a sequence \( \tilde{s} \), that is obtained from \( s \) by removing two of the keys, and keeping the other keys in \( s \) in their respective order, such that \( RC(s) \leq RC(\tilde{s}) + |s| + 1 \).

**Proof** Let \( s = a_1a_2\ldots a_n \). We consider seven cases that occur when considering the structure of \( RT(a_1\ldots a_n) \). In case 7 we consider the situation where \( RT(a_1\ldots a_n) \) contains a node, such that this node has a non-empty left subtree and a non-empty right subtree. All other scenarios are covered by case 1 through case 6. Also, case 1 and case 2 are symmetric cases, and similarly for cases 3 and 4, and cases 5 and 6. In cases 1–6, we define \( \tilde{s} \) to be \( a_1a_2\ldots a_{n-2} \).

**Case 1:** \( a_1 < a_{n-1} < a_n \) for \( i = 1,\ldots, n - 2 \). Let \( \tilde{s} = a_1a_2\ldots a_{n-2} \). From the definition of \( RC(s) \) we have that \( RC(s) = RC(\tilde{s}) + C(RT(\tilde{s}), a_{n-1}) + C(RT(a_1a_2\ldots a_{n-1}), a_n) \). But \( C(RT(\tilde{s}), a_{n-1}) \leq n - 2 \), since
Figure 3: The possibilities for the structure of \(RT(a_1 \ldots a_n)\) and \(RT(a_1 \ldots a_k)\), considered in the proof of Lemma 4.1.
it to the reader as an easy but tedious exercise to verify that \( \alpha \leq (k + 1) \). We show that each term \([C(LT(\text{rev}(s_i)), a_{i+1}) - C(LT(\text{rev}(s_i)), a_{i+1})]\) in \( \beta \) is at most 1, and therefore that \( \beta \leq n - k \). This follows from the following two observations on the trees \( LT(\text{rev}(s_i)) \) and \( LT(\text{rev}(s_i)) \) for \( k \leq \frac{n}{2} \): 

- \( a_l \) and \( a_r \) are leaf nodes in \( LT(\text{rev}(s_i)) \), and once we remove these leaf nodes from \( LT(\text{rev}(s_i)) \), the trees \( LT(\text{rev}(s_i)) \) and \( LT(\text{rev}(s_i)) \) are identical;
- any path from the root to a leaf in \( LT(\text{rev}(s_i)) \) contains at most one of \( a_l \) or \( a_r \).

From these two observations it follows that inserting \( a_{i+1} \) in \( RT(s) \) will be at most one comparison more expensive than inserting \( a_{i+1} \) in \( RT(s) \).

**Lemma 4.2** Let \( n > 1 \). Then \( W_n^\prime \leq n(n/4 + 1) - 2 - \alpha \) where \( \alpha = 0 \) if \( n \) is even and \( \alpha = 1/4 \) if \( n \) is odd.

**Proof** It is easy to verify that \( W_n^\prime = 0 \), \( W_2^\prime = 1 \), and \( W_3^\prime = 3 \). Using these values and the previous lemma, we have that 
\[
W_n^\prime \leq (n + 1) + (n - 1) + \cdots + 5 + W_3^\prime = n(n/4 + 1) - 2
\]
when \( n \) is even, and
\[
W_n^\prime \leq (n + 1) + (n - 1) + \cdots + 6 + W_3^\prime = n(n/4 + 1) - 5/4
\]
when \( n \) is odd.

**Theorem 4.3** Let \( n > 1 \). Then \( W_n^\prime = n(n/4 + 1) - 2 - \alpha \) where \( \alpha = 0 \) if \( n \) is even and \( \alpha = 1/4 \) if \( n \) is odd.

**Proof** From Lemma 4.2 we know that \( n(n/4 + 1) - 2 - \alpha \) is an upper bound for \( W_n^\prime \) when \( n > 1 \). All that remains is to show that the bound is reached for every \( n \). Consider the \( n \) keys \( a_1 < a_2 < \cdots < a_n \) and the sequence \( s = a_0, a_{m+1}, \ldots, a_n a_1 a_2 \ldots a_{m-1} \) where \( m = \lceil n/2 \rceil + 1 \). Let \( k = n - m + 1 \). The following table shows the cost of building the root insertion tree \( RT(s) \):

<table>
<thead>
<tr>
<th>Insert nr.</th>
<th>Key</th>
<th>Resulting tree</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i )</td>
<td>( a_t )</td>
<td>( t_i = RI(t_{i-1}, a) )</td>
<td>( C(t_{i-1}, a) )</td>
</tr>
<tr>
<td>1</td>
<td>( a_m )</td>
<td>( a_m[\bot, \bot] )</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>( a_{m+1} )</td>
<td>( a_m[\bot, \bot] )</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>( a_{m+2} )</td>
<td>( a_m[\bot, \bot] )</td>
<td>1</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( k )</td>
<td>( a_n )</td>
<td>( a_m[t_{k-1}, \bot] )</td>
<td>1</td>
</tr>
<tr>
<td>( k+1 )</td>
<td>( a_1 )</td>
<td>( a_1[\bot, t_k] )</td>
<td>( k )</td>
</tr>
<tr>
<td>( k+2 )</td>
<td>( a_2 )</td>
<td>( a_2[a_1, t_k] )</td>
<td>( k+1 )</td>
</tr>
<tr>
<td>( k+3 )</td>
<td>( a_3 )</td>
<td>( a_3[a_2, t_k] )</td>
<td>( k+1 )</td>
</tr>
<tr>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
<td>( \vdots )</td>
</tr>
<tr>
<td>( n )</td>
<td>( a_{m-1} )</td>
<td>( a_m[a_{m-1}, t_k] )</td>
<td>( k+1 )</td>
</tr>
</tbody>
</table>

Figure 4: Intermediary trees of the worst-case example in Theorem 4.3

where \( t_0 = \bot \) and \( u_r = a_r[a_{r-1}, \ldots, a_2[a_1[\bot, \bot], \bot], \bot] \). Figure 4 shows the resulting trees after the \( i \)-th insertion for \( 1 \leq i \leq k \) (on the left) and after the \( (k + i) \)-th insertion for \( 1 \leq i < m \) (on the right). Adding the numbers in the rightmost column of the table yields the desired result.

Although we shall not prove it, for \( n = 2 \) both possible sequences produce the worst-case result. For \( n = 3 \) and \( n = 4 \) there are four such sequences, and when \( n \geq 5 \) there are eight sequences when \( n \) is even, and sixteen when \( n \) is odd.

5 EXPERIMENTAL RESULTS

In this section we provide experimental results that will provide the impetus for future investigations. We will not state the various obvious but interesting questions that can be asked by considering these experimental results. The results for root insertion were obtained by a brute-force approach of considering all \( n! \) sequences of length \( n \), and counting for each sequence the number of comparisons required for root insertion.

Even though a brute-force approach is sufficient to obtain our experimental results for leaf insertion, we briefly describe an inductive method that can be used to obtain the cost distribution, in terms of number of comparisons, for inserting \( n \) keys in a search tree by using leaf insertion. Although this result is most probably well-known, we could not find an appropriate reference. The reasoning required to obtain the result is more or less the same argument that is used to show that the average cost, \( A'_n \), to construct a leave insertion tree with \( n \) keys is given by the recurrence \( A'_n = n - 1 + 1/n \sum_{k=1}^{n/2}(A'_{k-1} + A'_{n-k}) \). See for example [3], section 5.7, for a discussion of this result. For each \( n \in \{1, 2, 3, \ldots\} \), let \( L_n(z) \) be the polynomial with the coefficient of \( z^n \) equal to the number of sequences of length \( n \) for which the cost of constructing the leaf insertion tree is equal to \( m \). For example, \( L_1(z) = 1 = z^0 \), since there is one sequence of length 1 and the cost of constructing
of the leaf insertion tree from this sequence is 0. As a notational convenience, we define \( L_0(z) \) to be 1. We have for example that \( L_2(z) = 2z \), since we have 2 sequences of length 2 and the cost of constructing a tree by leaf insertion from any of these two sequences is equal to 1. Also, \( L_3(z) = 2z^2 + 4z^3 \), since we have 2 sequences of length 3 for which the cost is 2, and 4 sequences for which the cost is 3. Note that the sum of the coefficients of \( L_n(z) \) is equal to \( n! \), since we have \( n! \) sequences of length \( n \). The polynomials \( L_n(z) \) can also be defined recursively as follows: Let \( n \geq 0 \), then \( L_{n+1}(z) = z^n \left( \sum_{i=0}^{n} \binom{n}{i} L_i(z) L_{n-i}(z) \right) \). Thus we have for example that \( L_4(z) = z^4(L_0(z)L_3(z) + 3L_1(z)L_2(z) + 3L_4(z)L_1(z) + L_3(z)L_0(z)) = 12z^4 + 4z^5 + 8z^6 \). Therefore, if we consider the 24 sequences of length 4, for 12 sequences the cost of constructing a leaf insertion tree is 4, for 4 sequences the cost is 5 and for 8 sequences the cost is 6. Similarly, \( L_4(z) = 16z^{10} + 8z^9 + 24z^8 + 32z^7 + 40z^6 \). The logic behind the formula for \( L_{n+1}(z) \) is simple. A tree with \((n+1)\) keys, consists of a root and a left subtree of size \( i \) and a right subtree of size \((n-i)\), for some \( i \) between 1 and \( n \). For any sequence \( a_1 \ldots a_{n+1} \) we select the \( i \) positions from 2, \ldots, \( n \) + 1 that will contain the keys of the left subtree. This can be done in \( \binom{n}{i} \) ways. The product \( L_{i}(z)L_{n-i}(z) \) has terms \( cz^m \), where \( c \) is the number of pairs of sequence \((s_1,s_2)\), where the length of \( s_1 \) is \( i \) and the cost of constructing a leaf insertion tree from \( s_1 \) is \( j \), and the length of \( s_2 \) is \( n-i \) and the cost of constructing a leaf insertion tree from \( s_2 \) is \( m-j \). The additional term \( z^n \), preceding \( \sum_{i=0}^{n} \binom{n}{i} L_i(z) L_{n-i}(z) \), is required since each key added to the left or right subtree will require one more comparison to be inserted in a tree with \((n+1)\) keys, than if it were simply inserted in the left or right subtree on its own.

In the table below we list for sequence lengths \( n = 2 \) to \( n = 13 \), the percentage of sequences for which we need fewer comparisons (in column “\( L < R \)”), the same number of comparisons (in column “\( L = R \)”), and more comparisons (in column “\( L > R \)”) for leaf insertion than for root insertion.

### Table

<table>
<thead>
<tr>
<th>( n )</th>
<th>( L &lt; R )</th>
<th>( L = R )</th>
<th>( L &gt; R )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.0000</td>
<td>1.0000</td>
<td>0.0000</td>
</tr>
<tr>
<td>3</td>
<td>0.3333</td>
<td>0.3333</td>
<td>0.3333</td>
</tr>
<tr>
<td>4</td>
<td>0.4167</td>
<td>0.2500</td>
<td>0.3333</td>
</tr>
<tr>
<td>5</td>
<td>0.4500</td>
<td>0.2000</td>
<td>0.3333</td>
</tr>
<tr>
<td>6</td>
<td>0.4750</td>
<td>0.1333</td>
<td>0.3917</td>
</tr>
<tr>
<td>7</td>
<td>0.5099</td>
<td>0.1040</td>
<td>0.3861</td>
</tr>
<tr>
<td>8</td>
<td>0.5160</td>
<td>0.0926</td>
<td>0.3915</td>
</tr>
<tr>
<td>9</td>
<td>0.5225</td>
<td>0.0819</td>
<td>0.3956</td>
</tr>
<tr>
<td>10</td>
<td>0.5312</td>
<td>0.0691</td>
<td>0.3997</td>
</tr>
<tr>
<td>11</td>
<td>0.5342</td>
<td>0.0627</td>
<td>0.4031</td>
</tr>
<tr>
<td>12</td>
<td>0.5366</td>
<td>0.0575</td>
<td>0.4059</td>
</tr>
<tr>
<td>13</td>
<td>0.5392</td>
<td>0.0525</td>
<td>0.4083</td>
</tr>
</tbody>
</table>

### 6 Conclusion

The main result in this paper states that if the worst case, \( n(n+4)/4 - 2 - \alpha (\alpha = 0 \) for \( n \) even, and \( \alpha = 1/4 \) for \( n \) odd) comparisons are required to build a binary search tree with \( n \) distinct keys, using root insertion. We were rather surprised by the fact that we could not find a proof of this result in the literature.

### 7 References


Figure 5: Cost distribution of leaf and root insertion for sequence lengths $n = 6$ to $n = 13$. The solid and dotted lines represent leaf and root insertion, respectively.