Algorithmica (1990) 5: 295-311

Algorithmica

Analysis of the Standard Deletion Algorithms in Exact Fit Domain Binary Search Trees[1]

Joseph Culberson[2] and J. Ian Munro[3]

Abstract. It is well known that the expected search time in an N node binary search tree generated by a random sequence of insertions is $O(\log N)$. Little has been published about the asymptotic cost when insertions and deletions are made following the usual algorithms with no attempt to retain balance. We show that after a sufficient number of updates, each consisting of choosing an element at random, removing it, and reinserting the same value, that the average search cost is $\Theta(N^{1/2})$.

Key Words. Binary search tree, Data structure, Average case analysis.

1. Introduction. Binary search trees are well-known data structures, often used when fast search, insertion, and deletion are required. They also support nearest-neighbor and range queries. When the search trees are well balanced, any of these operations can be done in $O(\log N)$ time on trees containing N items.

We follow the usual definitions of the field. A *binary search tree* is a finite set of nodes which is either empty, or consists of a root and two disjoint binary trees called the left and right subtrees. Each node contains a distinct member of a linear ordered set called a *key*. If v is a nonempty node of a tree, then $l(v)$ designates the left subtree, $r(v)$ the right subtree, and $k(v)$ the key contained in v. The *father*, $f(v)$ of the node or subtree rooted at v are equivalent and are defined by the relation $f(l(v)) = f(r(v)) = v$. The *search property* is defined by the rule that for each node $v \in T$, each key in the left subtree of v is less than $k(v)$, and each key in the right subtree of v is greater than $k(v)$. Henceforth, we refer to binary search trees simply as trees.

The idea of the next larger key in the subtree rooted at node v is crucial to the deletion algorithms we discuss. Hence, the *successor*, $s(v)$, of a node $v \in T$ is defined to be that node in the right subtree with the minimum key. If the subtree is empty, then the function is undefined. A natural definition for $s(v)$, when the subtree is empty, would be the ancestor of v which contains the next largest key; however, for purposes of the description of the algorithm and the subsequent analysis, the above definition is preferable.

---

[1] This work was done in part while the first author was at the University of Waterloo. This work was supported by an NSERC '67 Science Scholarship and Grant A-8237 and the Information Technology Research Centre of Ontario

[2] Department of Computing Science, University of Alberta, Edmonton, Alberta, Canada T6G 2H1

[3] Data Structuring Group, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

Received July 28, 1986; revised August 15, 1988. Communicated by Robert Sedgewick.

Our concern is the effect of deletions and insertions on the expected cost of accessing elements in a binary search tree using standard deletion and insertion schemes that do not explicitly rebalance the tree. The most natural insertion algorithm [3], [18], [11] is to add a new element in the position at which a search for it ends unsuccessfully. Our interest is in the combined effect of this insertion scheme and the deletion algorithms of Hibbard [11] and Knuth [16]. Hibbard proposed his deletion scheme in 1962. In this algorithm, if the right subtree of the node containing the key to be deleted is not empty, then the next largest key is removed from the node containing it and used to replace the deleted key. The empty node is then deleted as described in detail below. Otherwise, the node is deleted, and the left subtree, if it exists, becomes the son of the node's parent. Knuth suggested an improvement. If the left subtree is empty, the node can be deleted directly, attaching the right subtree to the node's parent. This algorithm often results in a reduced average cost for any given deletion, and never produces a worse cost.

Hibbard [11] observed that doing $N + 1$ random insertions, followed by one random deletion by his method, results in the same shape distribution of trees as is obtained by doing N random insertions. Thus, it was thought that deletions do not affect the expected cost of binary search trees. The Knuth algorithm does not have this property. Knott [14] observed that after making another insertion this property is no longer preserved for the Hibbard algorithm either. Knott performed simulations of these algorithms on small trees, and noted that the empirical evidence suggested that the expected cost was reduced.

An exact analysis appears to be very difficult, having been so far accomplished for trees of three or four nodes [13], [2]. In each of these cases, the analysis verifies that the expected cost is reduced over that of a tree built from random insertions.

Eppinger [8], and Culberson [4], and Culberson and Munro [7] ran more extensive simulations in which a large increase in the expected search cost was observed for larger trees. Eppinger conjectured that the expected cost is $\Theta(\log^3 N)$. In what follows, we show that if, after each deletion, we re-insert the same value that we deleted, then the steady-state expected cost is $\Theta(N^{1/2})$. The results of extensive simulations strongly supported the conjecture that this bound also applies to binary search trees wherein the insertion values are drawn at random from some domain.

The key ideas of our proof hinge on the asymmetry inherent in the standard deletion algorithms. In either algorithm, when both subtrees of the node to be deleted are nonempty, we consistently choose the next key to the right of the node as the replacement key. If we randomly decide between a left or right choice for the replacement, then the asymmetry disappears. Experimental evidence lends credence to the hypothesis that in this case the expected cost is reduced on average by a long sequence of updates, and thus remains $O(\log N)$.

Our analysis will give bounds on the *Internal Path Length* (IPL) of a tree, which is the sum over all the nodes in a tree of the length of the path to the node from the root node. The average length of a search path is the IPL of the tree divided by the number of nodes N. The average length of a search path represents the average cost of accessing a key.

To insert a key, a leaf is created containing the key and is attached in the unique position that maintains the search property of the tree. This algorithm was discovered independently by several researchers including Windley [18], Booth and Colin [3], and Hibbard [11]. As is well known (see, for example, Knuth [16, Chapter 6.2.2]) the expected IPL of a tree formed by this insertion process is approximately $1.386N \log_2 N$. We caution the reader that this is quite different from all trees being equally likely, which has $\Theta(N^{3/2})$ IPL [15].

One of the earliest and best-known deletion algorithms is that of Hibbard [11] which we restate more formally below using the notation outlined above for navigating through the tree. The $[l/r]$ means either l or r as appropriate in the context.

HIBBARD ALGORITHM

To delete a key d from a tree T, find $v \in T$ such that $k(v) = d$.
if $r(v) = \varnothing$ **then do**
{ The right subtree is empty, so delete the node containing the key and reattach the left son as the appropriate left or right son of the parent }
if v is not the root $[l/r](f(v)) \leftarrow l(v)$;
{ If v is the root then $l(v)$ becomes the new root }
remove v from T.
else do
{Replace the key d with the key from the successor}
$k(v) \leftarrow k(s(v))$;
{ Delete the successor node, reattaching its right subtree. Note that the successor never has a left subtree }
if $s(v) = r(v)$ **then** $r(v) \leftarrow r(r(v))$
else $l(f(s(v))) \leftarrow r(s(v))$
remove $s(v)$ from T.

Knuth's modification is easily incorporated.

If we are interested in a precise mathematical analysis of long-term behavior, the obvious process to study is the one in which deletions and insertions alternate, thus maintaining a fixed number of nodes in the tree. We are concerned with the asymptotic growth of the average IPL with N, where N is the number of nodes.

Normally, we assume that the values that are inserted come from some fixed domain. In the analysis we present, however, we always reinsert the same value, and hence issues of domain are irrelevant.

2. The Skewing Factor. It is apparent that one of the features of the Hibbard algorithm is its decided asymmetry when the right subtree of the node is nonempty. In this case, the key is replaced by the next larger key from the tree, and we can think of the node as having moved to the right a short distance in the domain from which the keys are drawn. Keep in mind that the root node acts as a divider for the subtree for future insertions, with all keys larger than the root key falling to the

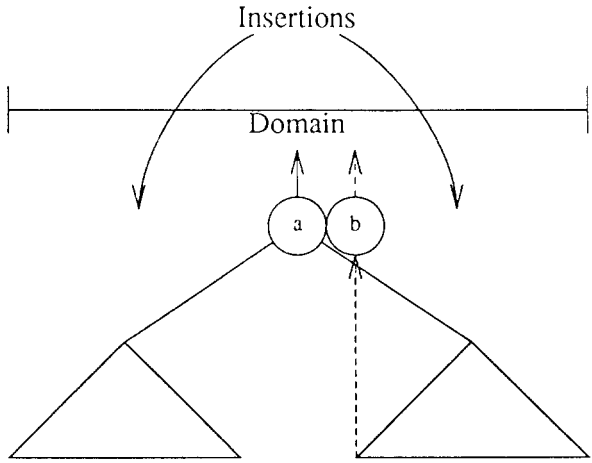

Fig. 1. Domain splitting by root.

right, and those smaller falling to the left. Having moved the root to the right, further insertions will be slightly more likely to fall to the left of the node than were insertions made prior to the move. Thus, the right subtree of the root will become smaller on average, with a corresponding increase in the left subtree. This is illustrated in Figure 1, where the larger key "b" replaces the deleted key "a."

We would like to think of the root as taking a step $1/N$th of the way across the domain each time it is updated. Thus, if there are k keys in the right subtree, then on average it would require approximately k moves of the root until the right subtree becomes empty. However, a complete analysis based on this idea has proved elusive [6]. (In an earlier analysis [5] it was claimed that the expected IPL for trees drawing new keys from a uniform distribution would be $\Theta(N^{3/2})$. Although we believe the result is correct, the proof assumes that the successive moves of a tag (defined in the next section) all have the same distribution. The distributions of such moves are subtly different, but probably close enough for purposes of the analysis.)

If we restrict the value inserted to be the value just deleted, then it will take exactly k moves of the root to empty a subtree of size k. We refer to this as a process on an *Exact Fit Domain* (EFD) tree, since one way of defining the situation is to restrict the domain so that the number of distinct values in the domain is exactly the number of nodes in the tree. We note that the root node of any subtree will slowly move to the right, until its right subtree becomes empty, when the next update will delete the node. Thus, the tree will become skewed to the left. One effect of this skewing is that the length of the path from the root of the tree to the leftmost node becomes greatly elongated. We call the nodes along this path the *backbone* of the tree. Our analysis focuses on the backbone of the EFD tree, and remarkably this restricted analysis is sufficient to show that the expected IPL of an EFD tree is $\Theta(N^{3/2})$.

3. The Analysis. One problem encountered in studying the backbone is that, when a key is scheduled to be deleted and the node containing it has no right subtree, the node disappears. We establish a system of *tags* for the updating process using the Hibbard algorithm.

The smallest key in the tree (and hence the key in the leftmost backbone node) receives a new tag whenever it is inserted. This is the only way a tag can be created. Whenever a key is deleted, all the tags currently attached to it are moved to the next larger key, unless the deleted key is the largest, in which case its tags are discarded. For completeness, we assume that initially the smallest key has a tag, and the keys in the remaining backbone nodes of the tree are tagged with *temporary tags*, which we subsequently ignore.

It is now an easy exercise to prove that, under the Hibbard algorithm, at any time during the updating process, precisely those keys in the backbone are tagged. Using this equivalence in what follows, we often refer to the backbone nodes as being tagged. Similarly, each contiguous set of untagged keys is precisely the set of keys in the right subtree of the node containing the tagged key to its left. We often use the term *interval* to refer to these sets of keys.

The *lifetime* of a tag is the number of updates from the time of its creation until it has been discarded. We note that on an EFD tree, a tag will move exactly N times, the last move being the update which discards it. Since each key is chosen with equal probability on each update, the probability that a particular tag moves is $1/N$ per update, and thus the expected lifetime of a tag is N^2 updates.

Using this expected lifetime, we can state

LEMMA 1. *After an average of N^2 updates, an EFD tree will be in steady state.*

PROOF. We note that the updating process is a Markov chain, in which the different tree shapes are the states, and thus it must eventually reach a steady-state distribution. Consider a sequence of updates which starts with the first move of some tag and ends with the kth move. We specify this sequence by $1, s_1, 2, s_2, \ldots, s_{k-1}, k$, where each integer i, $1 \leq i \leq k$, represents the update on key i that moves the tag the ith time, and s_i, $1 \leq i < k$, are the sequences of updates between the successive moves of the tag. During the updates, in any s_i, the tag is on key $i + 1$ implying that key $i + 1$ is in the backbone, and all keys $j \leq i$ are in the left subtree of the node containing $i + 1$. Thus, in any s_i, only those updates on keys j, $1 \leq j \leq i$, can effect a change in the shape of the subtree to the left of the tag. By induction, the shape of the subtree to the left of the tag at the end of the sequence is entirely independent of the shape of the tree prior to the sequence. Since this is true for every tag, it follows that when the tag which was initially on the smallest key is removed from the tree, the tree shape has the same distribution as it will have after the removal of any subsequent tag. As stated previously, the expected lifetime of this tag is N^2 updates. Note that we do not mean by this that the distribution of tree shapes immediately after removing tags from the tree is the steady-state distribution, since clearly this is not the case. Rather it means that if we randomly sample the process after N^2 updates have occurred, then we will find the steady-state distribution. □

This result concurs with the simulation results of Eppinger [8] and Culberson [4], [6] on trees of random domain, wherein it was noted that after approximately N^2 updates, the IPL and other measures stabilized.

Note that when a tag is first created, the two smallest nodes in the tree are tagged. We can think of these two tags as defining an interval, which currently has zero keys in it. This interval also determines the size of the right subtree of the key under the leftmost tag. As long as the two tags are moved in such a way that they do not collide, that is, become attached to the same key, they will continue to define the right subtree of the leftmost tag of the pair in this way. The number of changes to a noncollapsing interval between two such tags during the lifetime of the upper tag must be less than or equal to $2N - 2$, since the upper tag can move no more than $N - 1$ times after the lower tag is created, and the lower tag must move no more times than the upper, or they will collide.

By computing bounds on the expected size of such noncollapsing intervals, we can bound the size of the subtrees of the backbone, as well as the number of nodes in the backbone. Note that only those updates which affect the size of the interval in question concern us here. Updates on keys other than the two at the extrema of the interval have no effect on the number of keys in the interval. Updates on the keys within the interval may change the shape of the subtree, but we do not analyze this shape. Since an update reinserts the deleted key, the new key must fall in the same interval, and so the number of keys in the interval does not change for such updates. Similar remarks apply to updates on keys which fall outside the interval. We refer to the number of keys in the interval as the size of the interval, or equivalently as the size of the subtree.

We now compute the expected size of the interval when the upper tag moves past the jth key as in Figure 2. To understand the motivation behind this, notice that as long as j is the largest (i.e., rightmost) key in the interval, then the size of the interval must be less than or equal to its size when j entered it. We see that the expected size of the right subtree of the root will be less than or equal to the value obtained by setting $j = N$ in the following lemma.

LEMMA 2. *In an EFD the expected size of the interval containing the* jth *smallest key at the time it enters the interval is*

$$E_j = \frac{2^{2j-2}}{\binom{2j-2}{j-1}} - 1 \approx \sqrt{\pi j}.$$

PROOF. On the first move of the upper tag, the second smallest key is added to the interval between the two tags. Similarly, the jth key is added to the interval on the $j-1$st move of the upper tag. To simplify the algebra, we let $k = j - 1$, and solve for the expected number of nodes in the interval between two tags after the upper tag has moved k times under the condition that after any ith move of the upper tag, $1 \le i \le k < N$, the lower tag has moved fewer than i times.

We can model the behavior of the number of keys in the interval by a simple coin-flipping process. Here a head corresponds to a move of the right tag, or an

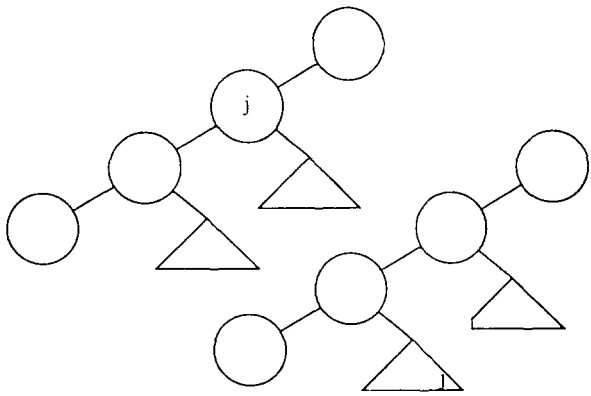

Fig. 2. Moving the jth node into the interval.

increase of one in the number of keys, and a tail to a move of the left tag, or a decrease of one in the number of keys. The number of keys in the interval corresponds to the number of heads minus the number of tails. We start with no heads or tails, and we wish to know what the expected difference is after some number of heads, given that the number of tails never exceeds the number of heads. This process is known in the literature as a simple random walk with an absorbing barrier at -1. See, for example, Feller [9].

To aid in understanding the analysis, we represent the possible walks for $k \leq 3$ in Figure 3. Initially, we start at the origin, which is the top circle of the diagram. We use k to represent the number of steps to the right, and i the net distance moved to the right. Thus, i under the coin model is the difference between the number of heads and the number of tails. In the figure all moves are downward, and either to the right, corresponding to a head (move of the upper tag), or to the left corresponding to a tail (move of the lower tag).

Using our oblique coordinate system, we let (i, k) be the position in the diagram representing a distance of i from the origin after k heads. We let $P_{i,k}$ represent the number of ways of reaching (i, k) with no point on the path having a negative i component, and also the last move being be a head. In the diagram this corresponds to the number of paths by which we could reach the ith position from the origin along the upward diagonal ending at k. For example, the positions reachable with $k = 3$ are indicated by the double circles in the diagram. Here

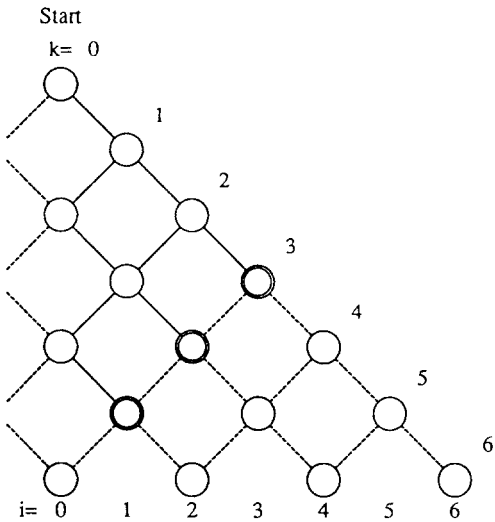

Fig. 3. Exact fit domain analysis.

$P_{1,3} = 2$, $P_{2,3} = 2$, and $P_{3,3} = 1$. Note that $P_{0,3} = 0$, since we are looking at the state immediately after the kth right move. Thus, in the diagram we do not include the dashed edges along the third diagonal in computing $P_{i,3}$. Note the implication that there is always at least one node in the initial right subtree of the root (for $N > 1$).

Since left and right moves are equally probable, in the unbounded case the probability of any given path of length j is $(\frac{1}{2})^j$. The depth of a point (i, k) is $2k - i$, and so the probability, $\mathrm{P}\{(i, k)\}$, of reaching (i, k) on a random walk by one of the $P_{i,k}$ paths is $(\frac{1}{2})^{2k-i}P_{i,k}$. Let Y be the event that, until k right moves have occurred, $i \geq 0$ at all times. Let $\mathrm{P}\{Y\}$ be the probability of Y. The expected value of i for a given k and under the condition that $i \geq 0$ at all times is

$$E_k = \mathrm{E}[i \mid Y] = \frac{\sum_{i=1}^{k} i\mathrm{P}\{(i, k)\}}{\mathrm{P}\{Y\}}$$

$$= \frac{\sum_{i=1}^{k} i(\frac{1}{2})^{2k-i}P_{i,k}}{\sum_{i=1}^{k} (\frac{1}{2})^{2k-i}P_{i,k}}.$$

Using the ballot theorem of Feller [9, Chapter III.1] the number of paths is

$$P_{i,k} = \frac{i}{2k - i}\binom{2k - i}{k}$$

and thus

$$E_k = \frac{\sum_{i=1}^{k} (\tfrac{1}{2})^{2k-i} \dfrac{i^2}{2k-i}\binom{2k-i}{k}}{\sum_{i=1}^{k} (\tfrac{1}{2})^{2k-i} \dfrac{i}{2k-i}\binom{2k-i}{k}} = \frac{N_k}{D_k}.$$

We can now solve for N_k and D_k. Letting $j = k - i$ we get

$$\begin{aligned} N_k &= (\tfrac{1}{2})^k \sum_{j=0}^{k-1} (\tfrac{1}{2})^j \frac{(k-j)^2}{k+j}\binom{k+j}{k} \\ &= (\tfrac{1}{2})^k \sum_{j=0}^{k-1} (\tfrac{1}{2})^j \left[(k+1)\binom{k+j+1}{k+1} - 4k\binom{k+j-1}{k} - \binom{k+j}{k}\right] \end{aligned}$$

Using the identity

$$\binom{x}{r-1} = \binom{x+1}{r} - \binom{x}{r}$$

we treat each of the terms in the preceding sum individually, ignoring for the moment the factors independent of j. The first term of the sum becomes

$$\begin{aligned} &\sum_{j=0}^{k-1} (\tfrac{1}{2})^j \binom{k+j+1}{k+1} \\ &\quad = \sum_{j=0}^{k-1} \left[\binom{k+j+2}{k+2}(\tfrac{1}{2})^j - \binom{k+j+1}{k+2}(\tfrac{1}{2})^{j-1}\right] + \sum_{j=0}^{k-1} (\tfrac{1}{2})^j \left[\frac{k+j+1}{k+2}\right] \end{aligned}$$

and using the telescoping property and applying repeatedly this becomes

$$\begin{aligned} &= \binom{2k+1}{k+2}(\tfrac{1}{2})^{k-1} + \binom{2k+1}{k+3}(\tfrac{1}{2})^{k-1} + \cdots + \binom{2k+1}{2k+1}(\tfrac{1}{2})^{k-1} \\ &= (\tfrac{1}{2})^{k-1}\left[\frac{2^{2k+1}}{2} - \binom{2k+1}{k+1}\right]. \end{aligned}$$

Similarly, the second term is

$$\sum_{j=0}^{k-1} (\tfrac{1}{2})^j \binom{k+j-1}{k} = (\tfrac{1}{2})^{k-1}\left[\frac{2^{2k-1}}{2} - \binom{2k-1}{k}\right].$$

And finally the third term is

$$\sum_{j=0}^{k-1} (\tfrac{1}{2})^j \binom{k+j}{k} = (\tfrac{1}{2})^{k-1}\left[2^{2k-1} - (\tfrac{1}{2})\binom{2k}{k}\right].$$

Thus, combining these terms, together with the coefficients, gives

$$N_k = (\tfrac{1}{2})^k\left[(k+1)(\tfrac{1}{2})^{k-1}\left[2^{2k} - \binom{2k+1}{k+1}\right] - 4k(\tfrac{1}{2})^{k-1}\left[2^{2k-2} - \binom{2k-1}{k}\right] - (\tfrac{1}{2})^{k-1}\left[2^{2k-1} - (\tfrac{1}{2})\binom{2k}{k}\right]\right]$$

and by straightforward expansion of the binomial coefficients this becomes

$$= (\tfrac{1}{2})^{2k-1}\left[2^{2k-1} - (\tfrac{1}{2})\binom{2k}{k}\right].$$

By an analogous set of manipulations we get

$$D_k = (\tfrac{1}{2})^{2k-1}(\tfrac{1}{2})\binom{2k}{k}.$$

Thus,

$$E_k = \frac{2^{2k-1} - (\tfrac{1}{2})\binom{2k}{k}}{(\tfrac{1}{2})\binom{2k}{k}} = \frac{2^{2k}(k!)^2}{(2k)!} - 1.$$

Using Stirling's approximation leads to

$$E_k \approx \sqrt{\pi k}.$$

Finally, recall that the upper tag starts on the second smallest key, and thus the kth move of the upper tag corresponding to adding the $k+1$st key to the subtree. Thus, by substituting $j-1$ for k we finish our proof. □

In the following lemma, the rth subtree refers to the right subtree of the rth backbone node from the right, where the root is the first backbone node. If there is no rth backbone node, because the backbone has fewer than r nodes, then we consider the rth subtree to be empty; that is, it has a size of zero.

LEMMA 3. *The expected size of the* rth *subtree on an EFD after sufficiently many updates is* $O(N^{1/2})$, *for all* r.

PROOF. This follows from the previous lemma on observing that the expected size of the subtree to the left of a tag on the kth node is $O(\sqrt{k}) = O(\sqrt{N})$. The topmost subtree is $O(\sqrt{N})$ when the Nth node is first introduced, and further updates can only reduce this size until the Nth node is eventually tagged again. □

In the next lemma we compute a bound on the length of the backbone.

LEMMA 4. *The expected number of nodes in the backbone of the EFD tree is* $\Theta(\sqrt{N})$ *after sufficiently many updates.*

PROOF. Let v be a node in the backbone of a tree, and let $r(v)$ be the right subtree of v. We define w_k, the *weight* of the backbone with respect to key k, where k is in one of the nodes in $r(v) \cup \{v\}$, as $1/(1 + |r(v)|)$. Thus, the number B of nodes in the backbone of an N node tree is

$$B = \sum_{k=1}^{N} w_k.$$

Intuitively, the density of the backbone nodes near the right of a tree is not increased by the fact that they are near the upper boundary. More formally, if we consider the first N nodes of an M node tree, where M may be arbitrarily large, we see that the number of nodes in the backbone over the first N nodes is identical to the number of backbone nodes in an N node tree in which the set of updates is the same as the set of updates over the first N nodes of the M node tree. This is true since an update on any key k does not affect the structure of the backbone to the left of the key. Thus, where we now consider the weights from the M node tree, we see that the number of backbone nodes over the first N nodes is

$$B \leq 1 + \sum_{k=1}^{N} w_k,$$

where the 1 comes from the observation that the rightmost backbone node may now be distributed over some of the keys to the right of the Nth node. If we consider the average weight taken over time on the kth key, we see that

$$\mathrm{E}[B] \leq 1 + \sum_{k=1}^{N} \mathrm{E}[w_k].$$

We now compute an upper bound on $\mathrm{E}[w_k]$, for $k \leq N$. We proceed in a manner similar to that in the proof of Lemma 2. That is, we compute a bound based upon the ways in which an initially adjacent pair of tags could have moved to define the interval containing k. (If k is tagged, we consider it to be part of the interval to the right.) First we compute the expected weight R_k on a key in the interval to the left of the kth key, immediately before the kth key first falls into the interval (actually, we compute the ratio for the $k + 1$st, but this will not make any difference in the asymptotic result that we derive):

$$R_k = \frac{\sum_{i=1}^{k} (1/i)(\frac{1}{2})^{2k-i} P_{i,k}}{\sum_{i=1}^{k} (\frac{1}{2})^{2k-i} P_{i,k}},$$

where the term $1/i$ is the assigned weight, given that there are $i - 1$ nodes in the subtree. Using the value of $P_{i,k}$ from Lemma 2, we find that the expected ratio is

$$R_k = \frac{\sum_{i=1}^{k} (\frac{1}{2})^{2k-i} \dfrac{1}{2k-i} \binom{2k-i}{k}}{D_k},$$

where D_k is defined as in Lemma 2. We now compute the value of the numerator by

$$\sum_{i=1}^{k} (\tfrac{1}{2})^{2k-i} \frac{1}{2k-i} \binom{2k-i}{k} = \frac{1}{k} \sum_{i=1}^{k} (\tfrac{1}{2})^{2k-i} \binom{2k-i-1}{k-1}$$

$$= \frac{1}{k} (\tfrac{1}{2})^{k} \sum_{j=0}^{k-1} (\tfrac{1}{2})^{j} \binom{k+j-1}{k-1}.$$

Now using the same techniques as in Lemma 2, this reduces to

$$= \frac{1}{2k}.$$

Recalling that

$$D_k = (\tfrac{1}{2})^{2k-1} (\tfrac{1}{2}) \binom{2k}{k}$$

from Lemma 2, and using Stirling's approximation we find that the expected weight is

$$R_k \approx \frac{\sqrt{\pi}}{2\sqrt{k}}.$$

Let Z_i be the event that the first tagged key to the right of k is key $k + i$, $i \geq 1$. Since we are considering the noncollapsing interval containing k, once a key $k + i$ enters the interval, the weight on k cannot decrease until $k + i + 1$ enters, but may increase as keys are deleted from the left of the interval. It follows that $\mathrm{E}[w_k | Z_i] \leq R_{k+i}$. (For $i > 1$, the inequality is strict, since Z_i is equivalent to stating that there are at least i keys in the interval to the left of a tag on $k + i$.) Since R_k is decreasing in k, $R_{k+i} \leq R_k$. We can extend our definition of Z_i using $i = M - k + 1$ for the event that no key to the right is tagged, in which case $w_k \leq 1/(M - N) < R_k$, for $k \leq N$ and large M. Then,

$$\mathrm{E}[w_k] = \sum_{i \geq 1} \mathrm{E}[w_k | Z_i] \mathrm{P}\{Z_i\} \leq R_k.$$

To complete the proof,

$$E[B] \leq 1 + \sum_{k=1}^{N} R_k$$

$$= \sum_{k=1}^{N} O\left(\frac{1}{\sqrt{k}}\right)$$

$$= O(\sqrt{N}),$$

where the last follows by comparing the summation of $1/\sqrt{k}$ with the integral to see that the result is $\approx 2\sqrt{N}$.

From Lemma 3 we see that the expected size of any subtree is $O(\sqrt{N})$, and thus by Jensen's inequality [10, p. 153] the expected number of nodes in the backbone is $\Omega(\sqrt{N})$. Together, these bounds complete the lemma. □

We note that in fact the upper bound is approximately $\sqrt{\pi N} \approx 1.77..\sqrt{N}$. Since this is based on the expected weight when the kth key first enters the interval, it is in a sense based on the minimal expected interval that is on a maximal expected weight. On the other hand, if we look at the expected interval size, we can see that when the kth key is first tagged the size of the interval to its right is still $O(\sqrt{k})$. This suggests that the expected size of the interval is $\sqrt{\pi k} \pm o(\sqrt{k})$ over all time. We conjecture that a better approximation to the coefficient can be obtained by ignoring the small order term and taking the inverse of this expectation to be the expected proportion of the time that the kth key is tagged. Thus,

$$\begin{aligned} \mathrm{E}[B] &\approx \sum_{k=1}^{N} \frac{1}{\sqrt{\pi k}} \\ &\approx \frac{2}{\sqrt{\pi}} \sqrt{N} \\ &\approx 1.128..\sqrt{N}. \end{aligned}$$

This conjecture has the virtue of being in excellent agreement with the results of the simulations performed upon EFD trees [6].

We now have two facts which together imply that the IPL of the EFD tree is $\Theta(N^{3/2})$ if more than N^2 updates have been performed. These are the upper bound on the subtree size, and the upper bound on the expected number of backbone nodes. We first prove the lower bound on the IPL.

LEMMA 5. *The asymptotic expected IPL of a tree is* $\Omega(N^{3/2})$.

PROOF. We let N_i be the number of nodes (including the backbone node) in the ith subtree from the right of the tree. Using the upper bound of Lemma 4, we choose K and c such that $\mathrm{E}[N_i] \leq K = cN^{1/2}$ for all i. Let C_i be the contribution of the i largest keys to the IPL. Then the expected contribution of the jK largest keys is

$$\mathrm{E}[C_{jK}] \geq \sum_{i=1}^{j} i\mathrm{E}[N_i] + (j+1)\left(jK - \sum_{i=1}^{j} \mathrm{E}[N_i]\right) - \sum_{i=1}^{j+1} 1$$

since the path to any node in the ith subtree (except the backbone node) must have length at least i, and the path to any node not in the first j subtrees (or the backbone) must have length of at least $j+1$. The last term compensates for the backbone nodes, where the ith backbone node is at distance $i-1$ from the root. An easy induction shows that

$$\sum_{i=1}^{j} i\mathrm{E}[N_i] + (j+1)\left(jK - \sum_{i=1}^{j} \mathrm{E}[N_i]\right) \geq \sum_{i=1}^{j} iK.$$

Thus $E[C_{jK}] \geq \sum_{i=1}^{j} iK - \sum_{i=1}^{j+1} 1$. Setting $j = \lfloor N/K \rfloor$, we find

$$E[C_N] \geq \sum_{i=1}^{\lfloor N/K \rfloor} iK - O(\sqrt{N}) = \Omega(N^{3/2}). \qquad \square$$

Next we prove the upper bound.

LEMMA 6. *The asymptotic IPL of a tree is* $O(N^{3/2})$.

PROOF. We build a pessimal tree, in which we put $\Theta(N^{1/2})$ nodes in the backbone with a leading coefficient larger than that of the expected number of nodes. We then add nodes until each subtree contains $\Theta(N^{1/2})$ nodes, again with a sufficiently large coefficient. Note that this tree will have more than N nodes. For our upper bound we assume that each subtree is linear, which is the worst case. We note that the IPL can be computed as the sum over each subtree of the IPL of that subtree, plus the sum of the distances to the roots of the subtrees times the number of nodes in the corresponding subtree. The IPL of each subtree is

$$\begin{aligned} \mathrm{IPL}_s &= \sum_{i=1}^{\Theta(N^{1/2})} i \\ &= \Theta(N). \end{aligned}$$

Since there are $\Theta(N^{1/2})$ such subtrees, this implies that the sum of all the IPLs of the subtrees is $\Theta(N^{3/2})$. The sum of the distances to the roots of the subtrees is

$$\sum_{i=1}^{\Theta(N^{1/2})} i\Theta(N^{1/2}) = \Theta(N^{3/2}).$$

Thus, the total of these two is also $\Theta(N^{3/2})$. We see that for each subtree of the expected asymptotic tree, the contribution to the IPL is less than the contribution of the corresponding subtree in the pessimal tree, and there are more subtrees in the pessimal tree than in the average tree. Hence, the expected asymptotic IPL is less than or equal to that of the pessimal tree, which completes the lemma. $\square$

Finally, we combine these two lemmas to form

THEOREM 1. *The IPL of the EFD tree is* $\Theta(N^{3/2})$.

Note that the above analysis does not depend upon the shape of the right subtrees of the backbone nodes. Thus this analysis applies to any algorithm on an EFD which is similar to the Hibbard algorithm in that the successor is used to replace the deleted node, or if no successor exists, then the node is deleted. Such an algorithm may restructure the right subtree in anyway it pleases without changing the asymptotic result.

We can also use similar techniques to analyze the following. Knuth [16] presents an improved algorithm that differs from Hibbard's only in checking for an empty left subtree before choosing the successor to replace the key. If the left subtree is

empty, then the right son replaces the node containing the deletion key, instead of the successor. This algorithm is one of the more popular textbook algorithms for deletion in binary search trees [1], [19], [17].

Knuth [16] shows that for one application, his rule always results in a tree that is at least as well balanced as the one produced by the Hibbard algorithm when applied to the same tree, and is often better. However, we have the following theorem.

THEOREM 2. *The Knuth algorithm used on an EFD tree results in* $\Theta(N^{3/2})$ *expected IPL.*

PROOF. In the previous analysis only the backbone deletions are considered. Thus the only deletions that could change the results of the analysis for the Knuth algorithm are those involving the leftmost node in the tree, since it is the only backbone node with an empty left subtree. Even then, the result differs only if the right subtree of that node has at least two nodes in its backbone. In that case, the leftmost tag moves to the right as before, but one or more additional tags must now be inserted between the two smallest tags. We call these tags *k-tags*. For example, if we delete "a" from the left tree in Figure 4 using the Knuth algorithm, then both "b" and "c" are added to the backbone, forcing us to introduce the k-tag labeled "0," in addition to the new tag "3" on "a." It is easily seen that both the Knuth and Hibbard algorithms produce the same set of tags, and that the tags move identically under each. In addition, the k-tags introduced by the Knuth algorithm also move as they would under the Hibbard algorithm, once they are created. Thus, the Knuth algorithm can only reduce the size of the resulting subtrees, and increase the length of the backbone, and so the average IPL is still $\Omega(N^{3/2})$.

To prove the upper bound, we divide the tree on the fth key from the left of the tree, with $f = 3\lceil \log_2 N \rceil$. For any sequence of updates, there are just two ways that a key to the right of the fth key can be k-tagged under the Knuth algorithm. Either the k-tag was created directly on a key to the right of (or on) the fth key, or the k-tag was created to the left of the fth key.

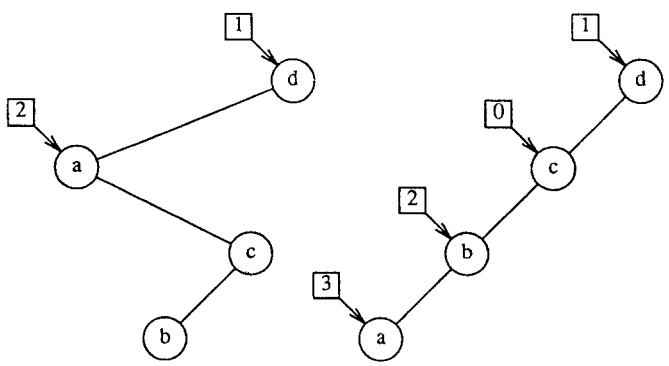

Fig. 4. Adding new tags by Knuth deletion.

We consider first the case of k-tags created to the right of the fth key. For any pair of adjacent tags, the probability that the upper tag passes the fth key before the first move of the lower tag is $(1/2)^f$, and thus the steady-state probability that the fth key is in the leftmost interval of the backbone is $O((1/2)^f)$. Only in this case can a k-tag be created directly on or to the right of the fth key by the Knuth algorithm. The probability that the smallest key is updated is $1/N$ per update, and even if the fth key is in the leftmost interval (and thus there may be nodes to its right also in the interval), less than N k-tags can be created on the next update. Thus the expected number of k-tags created to the right of the fth key per update is bounded by $O((1/N)/(N/2^f)) = O((\frac{1}{2})^f)$. We know that the expected time any tag remains on the tree is $O(N^2)$ updates, and thus using Little's Law (see, for example, [12]) we see that, in the steady state, the expected number of k-tags on the tree which were created to the right of the fth key is $O(N^2/2^f) = O(1/N)$ for $f = 3\lceil \log_2 N \rceil$.

Turning to the k-tags created to the left of the fth node, we note that once a k-tag is created it behaves identically under either algorithm. Thus, we can consider these k-tags as tags starting to the left on an $N - f + 1$ node tree under the Hibbard algorithm. As a worst case, let us assume that the fth key is tagged with a tag or k-tag at all times, and then treat the fth key as the smallest key of our $N - f + 1$ node tree. Therefore, by our previous lemmas, on average there are $O(\sqrt{N - f + 1})$ keys to the right of the fth key tagged with tags or k-tags which started to the left of the fth key.

Finally, even if the Knuth algorithm could somehow contrive to keep all the keys to the left of the fth key tagged or k-tagged, this adds at most a term of $O(\log N)$ to the expected number in the backbone. Thus, the expected number of nodes in the backbone for the Knuth algorithm is bounded by $O(\sqrt{N - f + 1}) + O(1/N) + O(\log N) = O(\sqrt{N})$, and we note that in fact this has the same leading coefficient as in the result for the Hibbard algorithm. We can now derive an upper bound of $O(N^{3/2})$ for the expected IPL using the proof of Theorem 2. □

We should mention that the Knuth algorithm is free to increase the rebalancing effects on the right subtrees, and, guessing that the size of the leftmost subtree is probably bounded by a small constant, we expect almost no change in the behavior of the backbone. Simulation results [4] suggest that on average the Knuth algorithm produces a tree with a slightly smaller cost than does the Hibbard algorithm.

4. Conclusion. We have shown that a restricted form of binary search trees develop an average IPL of $\Theta(N^{3/2})$ when subjected to a sufficiently long sequence of updates using the Hibbard deletion algorithm and standard leaf insertion. We have also shown that for these trees the improved algorithm given by Knuth has similar asymptotic behavior. We believe these results also apply to the more usual model wherein the keys to be inserted during the updating process are drawn at random.

When Binary Search Trees are to be used dynamically, we recommend a simple modification to the Knuth algorithm that appears to eliminate this deterioration in performance. When both subtrees are nonempty, choose at random either the successor or the predecessor as the replacement node for the deleted element, or for that matter, simply alternate. This removes the asymmetry of the algorithm, and it would appear that no skewing of the tree should take place. The experimental evidence [4], [8] tends to confirm this conjecture. Indeed, indications are that in trees subjected to updates using this algorithm the average search cost is less than that in trees grown from random sequences of insertions.

References

[1] Aho, A. V., Hopcroft, J. E., and Ullman, J. D., *Data Structures and Algorithms*, Addison-Wesley, Reading, MA, 1983.

[2] Baeza-Yates, R. A., A Trivial Algorithm Whose Analysis Isn't: A Continuation, *BIT*, **29** (1989), 88–113.

[3] Booth, A. D., and Colin, A. J. T., On the Efficiency of a New Method of Dictionary Construction, *Information and Control*, **3** (1960), 327–334.

[4] Culberson, J. C., Updating Binary Trees, Technical Report CS–84–08, University of Waterloo, Waterloo, Ontario, March 1984.

[5] Culberson, J., The Effect of Updates in Binary Search Trees, *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, Providence, RI, 1985, pp. 205–212.

[6] Culberson, J., *The Effect of Asymmetric Updates in Binary Search Trees*, Ph.D. Thesis, University of Waterloo, Waterloo, Ontario, 1986.

[7] Culberson, J., and Munro, J. I., Explaining the Behavior of Binary Search Trees Under Prolonged Updates: A Model and Simulations, *The Computer Journal*, **32** (1989), 68–75.

[8] Eppinger, J. L., An Empirical Study of Insertion and Deletion in Binary Trees, *Communications of the Association for Computing Machinery*, **26** (1983), 663–669.

[9] Feller, W., *An Introduction to Probability Theory and Its Applications*, Vol. I, Wiley, New York, 1968.

[10] Feller, W., *An Introduction to Probability Theory and Its Applications*, Vol. II, Wiley, New York, 1971.

[11] Hibbard, T. N., Some Combinatorial Properties of Certain Trees with Applications to Searching and Sorting, *Journal of the Association for Computing Machinery*, **9** (1962), 13–28

[12] Hillier, F. S., and Lieberman, G. J., *Introduction to Operations Research*, Holden-Day, San Francisco, 1980.

[13] Jonassen, A. T., and Knuth, D. E., A Trivial Algorithm Whose Analysis Isn't, *Journal of Computer and System Sciences*, **16** (1978), 301–322.

[14] Knott, G. D., Deletion in Binary Storage Trees, STAN–CS–75–491, Ph.D. Thesis, Stanford University, May 1975.

[15] Knuth, D. E., Fundamental Algorithms, *The Art of Computer Programming*, Vol. I, Addison-Wesley, Reading, MA, 1968.

[16] Knuth, D. E., Searching and Sorting. *The Art of Computer Programming*, Vol. III, Addison-Wesley, Reading, MA, 1973.

[17] Tenenbaum, A. M., and Augenstein, M. J., *Data Structures Using Pascal*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

[18] Windley, P. F., Trees, Forests, and Rearranging, *The Computer Journal*, **3** (1960), 84–88.

[19] Wirth, N., *Algorithms & Data Structures*, Prentice-Hall, Englewood Cliffs, NJ, 1986.