Explaining the Behaviour of Binary Search Trees Under Prolonged Updates: A Model and Simulations

Article *in* The Computer Journal · February 1989

DOI: 10.1093/comjnl/32.1.68 · Source: DBLP

2 authors, including:

J. Ian Munro
University of Waterloo
297 PUBLICATIONS **7,835** CITATIONS

SEE PROFILE

Explaining the Behaviour of Binary Search Trees Under Prolonged Updates: A Model and Simulations*

J. CULBERSON[1] AND J. I. MUNRO[2]

[1] *Department of Computing Science, Univerity of Alberta, Edmonton, Alberta, Canada T6G 2E7*

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

In this paper we present an extensive study into the long-term behaviour of binary search trees subjected to updates using the usual deletion algorithms taught in introductory textbooks. We develop a model of the behaviour of such trees which leads us to conjecture that the asymptotic average search path length is $\Theta(N^{\frac{1}{2}})$. We present results of large simulations which strongly support this conjecture. However, introducing a simple modification to ensure symmetry in the algorithms, the model predicts no such long-term deterioration. Simulations in fact indicate that asymptotically the average path length of such trees is less than the $1.386\ldots\log_2 N$ average path length of trees generated from random insertion sequences.

Received December 1987, revised May 1988

1. INTRODUCTION AND BACKGROUND

In 1983 Eppinger[8] published simulation results which overturned a long-standing conjecture due to Knott[11] that performing random updates in a binary search tree did not adversely affect the performance of the structure. Knott based his conjecture on smaller simulations which were insufficient to show the long-term deterioration observed by Eppinger. This conjecture was supported by a theoretical treatment of three node trees by Jonassen and Knuth[10] and more recently of four node trees by Baeza-Yates.[2]

In the studies noted above and in this paper an update to a binary search tree consists of one random deletion followed by a random insertion. That is, an element in the tree is chosen at random for deletion and a new element is generated from a fixed and bounded distribution and is inserted. The insert/delete model of an update gives us a firmer basis for comparisons than one in which the structure itself changes sizes quite drastically. The key issue regarding the distribution from which elements are chosen is that it does not change over time; i.e. it is fixed. For simplicity we will deal with elements coming from a uniform distribution. This is of no technical importance as only the relative value of the elements matters. The updates are performed using the well-known deletion algorithm of Hibbard[9] and the usual insertion at a leaf algorithm. We define the *internal path length* (IPL) of a tree to be the sum over all nodes in the tree of the length of the path from the root to the node. The *Average Path Length* to any node in the tree is then just the IPL divided by the number of nodes. Knott's conjecture claimed that for a binary search tree subjected to a long sequence of such random updates, the average IPL would be less than that in a tree generated from a sequence of random insertions. It is well known[12] that the average IPL in a tree grown from a random sequence of N insertions is approximately $1.386..N\log N-2.846\,N$.† Since a fully balanced tree has an IPL of $\Omega(N\log N)$, Knott's conjecture was equivalent to claiming that the IPL of a binary search tree remained $\Theta(N\log N)$, but with a reduced coefficient.

Eppinger's results[8] indicated that this was false, and that furthermore the increase did not appear to be bounded by a constant factor. He conjectured, using regression analysis to support his claim, that the IPL tended to $\Theta(N\log^3 N)$. The formulation he gave was $0.0280\,N\log^3 N-0.392\,N\log^2 N+3.03\,N\log N-4.81\,N$. We claim that Eppinger's conjecture is itself too optimistic. In the following sections we develop a model for the behaviour of binary search trees under a long sequence of updates based on the analysis of Exact Fit Domain trees given in Ref. 6. The analysis we present here, although not mathematically complete, is intuitively appealing. Moreover, using this hypothetical model we can compute asymptotic expected values for various parameters measuring the shape of the trees. We have performed extensive simulations, and measured these values over a large number of trees. The results are in close agreement with the values predicted by the model. Assuming the correctness of these results, it follows that the IPL of such trees is in fact $\Theta(N^{\frac{3}{2}})$. The approximate formula we obtain from our model is $0.266\,n^{\frac{3}{2}}+0.693\,N\log N-2.008\,n$.

In Fig. 1 we display the average path length as measured by various simulations together with curves from ours and Eppinger's conjectures. The $\times$'s mark the data points from Eppinger's simulations, while the $\bigcirc$'s represent the average path lengths computed from the unadjusted IPLs of Table 5 (see Section 4).

We follow the usual definitions. A *binary search tree* is a finite set of nodes which is either empty, or consists of a root (containing a key) and two disjoint binary trees called the left and right subtrees. If v is a non-empty node of a tree, then $l(v)$ designates the left subtree, $r(v)$ the right subtree, and $k(v)$ the key of 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 will refer to binary search trees simply as trees.

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 dis-

* 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, Grant A-8053 and grant A-8237, and by the Information Technology Research Centre of Ontario.

† All logarithms are to the base 2 unless otherwise indicated.

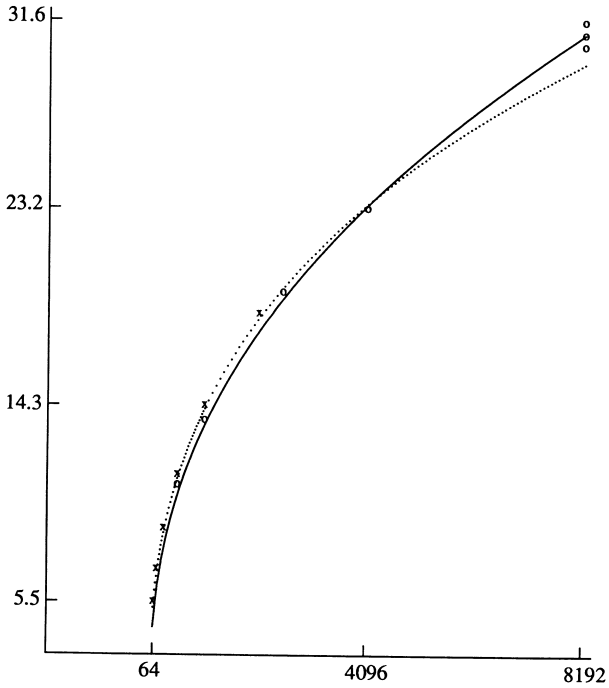

Figure 1. Comparison of conjectures to simulation data. Asymptotic average path length *versus* N. ——, New conjecture; · · ·, Eppinger estimate.

covered independently by several researchers including Windley,[15] Booth and Colin[3] and Hibbard,[9] and attributed to D. J. Wheeler and C. M. Berners-Lee.[7]

One of the earliest and best-known deletion algorithms is that of Hibbard.[9] We illustrate this algorithm in Fig. 2 for the relevant cases as discussed below. That is, suppose node v is to be deleted. There are three cases to consider.

(1) If v is a leaf, change the reference to v to null.

(2) If $r(v) = \varnothing$ but $l(v) \neq \varnothing$ change the reference of v's parent to refer to $l(v)$.

(3) If $r(v) \neq \varnothing$ delete the leftmost node in $r(v)$ from its current position (by one of the cases above) and copy its key into v. The copying is usually done by relinking.

2. WHAT WE EXPECT TO HAPPEN

In Eppinger's simulations, the IPL of large trees was first seen to decrease from its initial value. Then it slowly increased. It achieved a maximum at approximately N^2 updates, after which the value remained fairly steady. Any useful model should attempt some explanation of these features. We now proceed to develop and justify such a model.

The reason for the initial reduction would appear to be the rebalancing effect of Hibbard's algorithm when the right subtree is empty. In this case, the node is deleted and the *left* subtree is connected directly to the node's parent. This is illustrated in Fig. 2 by the deletion of node 4. If the left subtree has several nodes, then the length of the path from the root to each of these nodes is reduced by one, thus reducing the IPL. In our example, the paths to nodes 1, 2 and 3 have each been shortened by one. Knuth[12] noticed this, and suggested that the algorithm could be improved by also checking for an empty left subtree and using a rule which is symmetric to (2) above.

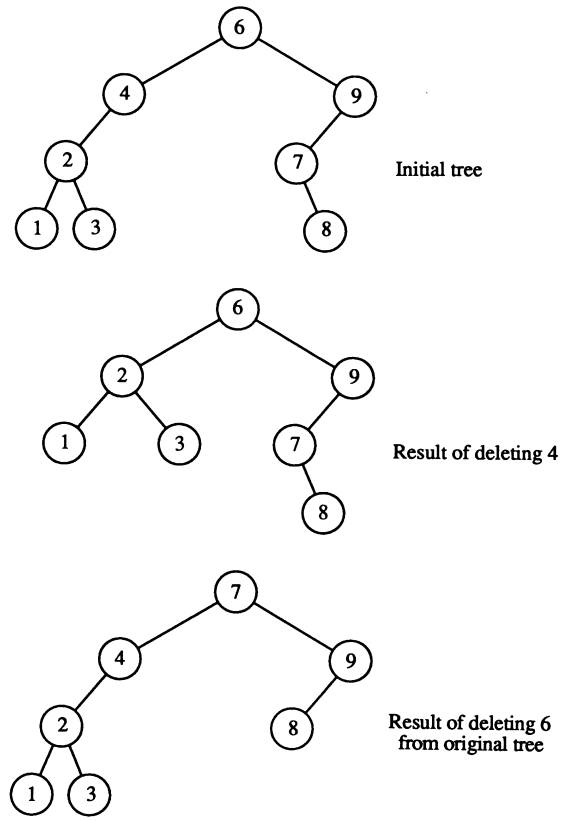

Figure 2. Example of deletion in a binary search tree.

In our example, using Hibbard's algorithm to delete 6 makes 7 the new root of the tree, even if the nodes 1, 2, 3 and 4 are not present, while Knuth's improved algorithm would simply move 9 to the root, without making further changes. In the tree actually used in the illustration, however, Knuth's algorithm would produce the same results for both deletions.

This improved algorithm is the one usually presented in textbooks.[1,16,14] Knott's simulations[11] indicated that the Knuth algorithm gave slightly better results than the Hibbard algorithm when used in a sequence of updates.

To verify that this rebalancing in the case of empty subtrees is the cause for the initial improvement, and to see if this algorithm was asymptotically better than Hibbard's, we simulated Knuth's algorithm on trees of up to $N = 1024$ nodes and far in excess of N^2 updates. As predicted, the initial improvement was slightly greater than Eppinger's results for the Hibbard algorithm. However, the long-term deterioration was just as pronounced, with the maximum IPL being only slightly less than for the Hibbard algorithm, as shown by Table 1. A more detailed study of this and other deletion algorithms is found in Ref. 4.

However, it is the long-term effect of the algorithms that interests us here. To understand the long-term increase in the IPL, we notice that in each of these algorithms the behaviour is decidedly asymmetric when the right subtree of the node containing the deleted key is not empty. Suppose that the key we are deleting is at the root of the tree. If the right subtree is non-empty, then the successor key becomes the new root key. Since new keys fall in the left or right subtree depending upon whether they are smaller or larger than the root, it follows that future insertions are more likely to fall in the

Table 1. Empirical mean internal path lengths from simulations (including those of Eppinger)

Number of nodes	Hibbard algorithm			Knuth algorithm	
	Mean IPL	Iterations	Trees	Mean IPL	Iterations
32	—	—	—	132.4	7500
64	349.9	10000	200	345.2	50000
128	889.8	50000	200	880.0	80000
256	2251.3	120000	100	2232.2	150000
512	5737.1	500000	50	5643.1	500000
1024	14687.2	2500000	25	14348.3	2000000
2048	37876.0	9000000	20	—	—

left subtree than previously. This is illustrated in Fig. 3. This effect will be repeated each time the root key is selected for deletion, until eventually the right subtree will become empty.

We define the *backbone* of a tree to be the set of nodes on the path from the root to the leftmost node in the tree. When the key in one of those backbone nodes is selected for deletion, there are two cases to consider. In the first case, the node has a non-empty right subtree. In this case the key in the successor replaces the deleted key, and that node is deleted. Some further adjustments may be required in the right subtree, but these do not affect the backbone. The average number of nodes to the right of the backbone will be reduced, and since this is happening to all the backbone nodes, the tree will eventually become skewed. These nodes are the ones indicated in Fig. 4.

In the case when the right subtree is empty, the node is simply deleted, and its left son becomes the left son of the node's parent (or it becomes the root if the deleted node was previously the root). These disappearing nodes make understanding the effects of the process more difficult. To aid in the presentation, we attach *tags* to some of the keys in the tree. These tags are illustrated in Fig. 4. Whenever a new smallest key is inserted into the tree, a new tag is created and attached to it. When a tagged key is deleted, all the attached tags are moved to the next larger key in the tree, unless the key happens to be the largest in the tree, in which case the tags are discarded. Note that several tags can accumulate on any key. If we initially tag all the keys in the backbone, then it is easily seen that at any time the keys in the backbone are exactly those tagged, if we are using the Hibbard

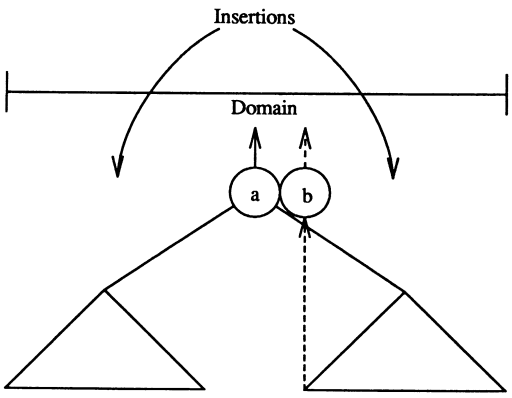

Figure 3. Domain splitting by root.

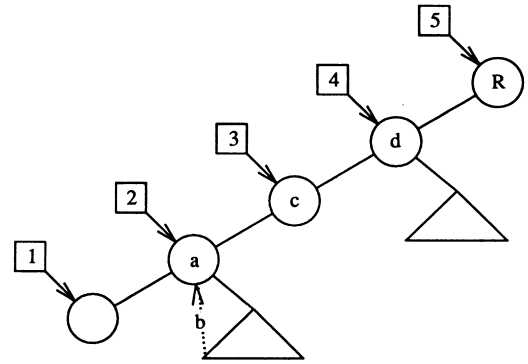

Figure 4. Tag model.

deletion algorithm. For the Knuth algorithm, the only differences occur in the subtrees where they are of no concern, or possibly when the leftmost node is deleted new nodes may be inserted into the backbone. In what follows we concentrate on the Hibbard algorithm, with suitable comments on the Knuth algorithm added where appropriate.

Eventually, with probability one, any given tag will be removed from the tree. During its *lifetime*, that is, while it remains on the tree, the tag performs a random walk across the domain of keys. It takes a step to the right of random size with probability $1/N$.

In Ref. 6 the domain is restricted so that when a key is deleted in an update, the following insertion re-inserts the same value. Trees so restricted are called *Exact Fit Domain* (EFD) trees. This restriction simplifies the analysis considerably, and there we show that in the steady state, EFD trees have an expected IPL of $\Theta(N^{\frac{3}{2}})$, although the upper and lower bounds differ by large constant factors. To derive this result, two asymptotic conditions were found to be sufficient. First, the expected size of the right subtree of the jth backbone node from the root is $O(\sqrt{N})$, and secondly, the expected number of nodes in the backbone is $\Theta(\sqrt{N})$. If we could confirm that these conditions hold for the current case, wherein new keys are selected at random for the insertion part of the update, we could then claim the same result.

Let us specify that the domain for our trees is $(0, N+1)$, and that all keys are independently and uniformly distributed over this interval. As previously noted, the details of the distribution are not important as long as it is fixed. Thus, if the tree has N nodes, the expected interval size between nodes is one. The probability distribution of the size of an interval is

$$P\{X > x\} = \left(1 - \frac{x}{N+1}\right)^N.$$

This also applies to the intervals at the ends of the domain.

A tag is created on the smallest key, which has the above distribution. Its expected position is 1, making the expected remaining distance to the right end equal to N. After its creation, the key has probability $1/N$ of being deleted on the next update. If it is deleted on the next update, then the tag moves to the next larger key, a distance which has again the distribution given above. Thus the tag will move an expected distance of $1/N$ on the second update after its creation. If we assume that the tag moves $1/N$ on every update until it falls off the tree, it would

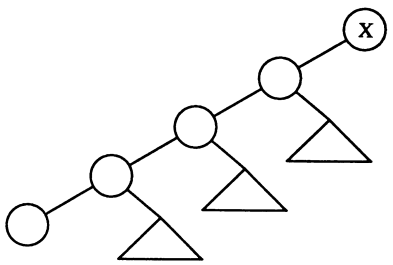

Figure 5. Collect data when X deleted.

require N^2 updates. However, on subsequent updates, the probability distribution of the distance moved is subtly different from the above, and has only been partially analysed (see Ref. 5). Nevertheless, it would appear that asymptotically the above distribution is a good approximation to the actual distributions.

Now consider the interval between two tags, such that initially they are on the two smallest keys in the tree. We attempt to derive the expected size of the interval to the right of the lower tag at the time when the upper tag is removed from the tree under the condition that they never collide, that is, that they never land on the same key. If they ever landed on the same key, then of course they would both be removed at same time. Since only tagged nodes are in the backbone, and no tag can be inserted between a pair using the Hibbard algorithm, this means that the remaining tag is in fact on the root. Thus, we are looking at the expected size of the interval to the right of the root just when a new root is created. This is illustrated in Fig. 5. Under the assumption of a random distribution of keys over the stated interval, this size is also the expected number of nodes in the right subtree of the root at these times.

We assume that it is equally probable that the lower or upper tag is moved each time the interval is changed, and that the distribution of the size of the move is given in each case by the above distribution. We let **C** be the random variable indicating the size of the change. The distribution $f(c)$ in terms of the random variable **X** is

$$f(c) = \tfrac{1}{2} f_x(|c|)$$

It is easily seen that, due to symmetry, all odd moments of this distribution are zero. Also, applying symmetry again, we find the second moment of **C** to be

$$E[\mathbf{C}^2] = \int_0^{N+1} x^2 \frac{N}{N+1}\left(1-\frac{x}{N+1}\right)^{N-1} dx$$

$$= 2\frac{N+1}{N+2}$$

$$\approx 2, \quad N \to \infty.$$

This is also the variance of **C**, since the expected value is zero.

Now consider how many steps have been made until the upper tag is removed. The number of steps may be greater than or less than N, but asymptotically at least it seems reasonable to assume that taking steps of size approximately 1 would mean that we require N steps on average for the tag to fall off the tree. Of course, this is not strictly true, but as with previous assumptions, it is a convincing approximation which simplifies the analysis.

Even so there remain difficulties. The major one relates to the conditional nature of the walk, in that we want only those walks in which the upper tag always remains ahead of the lower one. In Ref. 6 this conditional expectation also appeared, but the walk was discrete and the solution was easily found.

We approximate the preceding random walk with the following discrete random walk, which allows us to make use of the results in Ref. 6. We define a random walk by

$$S_N = \sum_{i=1}^{N} X_i,$$

where the distribution density function of the random variables X_i are given by

$$r(x) = \begin{cases} \frac{1}{2} & x = -\sqrt{2} \\ \frac{1}{2} & x = \sqrt{2}. \end{cases}$$

We have then a simple random walk, as defined for the EFD analysis,[6] except that the step sizes are $\sqrt{2}$. We choose this step size so that the second moment will be equivalent to that of the preceding random walk.

It can be readily seen that this distribution has all odd moments equal to zero, and that the second moment (and the variance) is equal to 2. We expect S_N to behave then somewhat like the previous random walk. It is easy to see that asymptotically these two unbounded walks should converge in distribution. Whether convergence also holds when we apply the condition that the walks never go below the origin is open, but it seems reasonable that first and second moments would be asymptotically equivalent, and thus that the expected interval should be equivalent.

In Ref. 6 it was shown that for the EFD the asymptotic expected size of the interval containing the kth key from the left of the tree would be $\sqrt{(\pi k)}$. Noting that the step size for the EFD is one, while in this walk it is $\sqrt{2}$, we see that the expected interval size for this walk is $\sqrt{(2\pi k)}$. The expected size of the right subtree of the root is then less than $\sqrt{(2\pi N)}$. We could guess that the upper tag moves $N - O(\sqrt{(N)})$ times when it and the lower tag are known to diverge before falling off the tree. This means that the estimate may be a bit high for small N.

Using the approximation $\sqrt{(2\pi i)}$ for the size of the interval following a tag at position i, we find that the proportion of nodes tagged in the vicinity of the ith node is approximately $1/\sqrt{(2\pi i)}$. Under assumptions of equal probability, the expected number of backbone nodes is

$$\mathrm{E}[B] = \sum_{i=1}^{N} \frac{1}{\sqrt{(2\pi i)}} \approx \frac{2\sqrt{N}}{\sqrt{(2\pi)}} \approx 0.798..\sqrt{N}.$$

However, this assigns a probability of $1/\sqrt{(2\pi)} = 0.3989$ to the first node being tagged when in fact we know that it is always tagged. Similarly, this formula slightly underestimates the other low-order nodes as well. Thus, for any N the number of nodes predicted by this formula should be a bit less than the actual expected number.

Assuming that the model outlind above is sufficiently accurate, we have then $\Theta(\sqrt{N})$ subtrees of the backbone with $O(\sqrt{N})$ nodes in each after sufficiently many updates. In Ref. 6 these two conditions were shown to be sufficient to guarantee that the expected IPL of the tree is $\Theta(N^{\frac{3}{2}})$. Briefly, the sum of the distances to the nodes

of the right subtrees of the backbone, even assuming that all the nodes could be clustered at the roots of the subtrees, is $\Omega(N^{\frac{3}{2}})$. On the other hand, with an average of $O(\sqrt{N})$ nodes in a subtree, the expected IPL of a subtree, even if it is linear, is $O(N)$. Since there are $O(\sqrt{N})$ subtrees, we have an expected IPL of $O(N^{\frac{3}{2}})$.

We can get a crude estimate of the coefficient of the leading term for the expected IPL by pushing our approximation a bit further, although we now depart somewhat further from full justification. In the idealized situation in which the ith backbone node is at its expected position, suppose that the size of the ith subtree is linear in i. Thus, the ith subtree from the left should contain

$$\sqrt{(2\pi N)}\frac{i}{\sqrt{(2N/\pi)}} - c \approx \pi i$$

nodes, where c is a constant and will now be ignored. Since there are $\mathbf{B}_K$ backbone nodes on average, the root of the ith subtree from the left of the tree is at a depth of approximately $\sqrt{(2N/\pi)} - i$ from the root. Ignoring any contribution the IPLs of the right subtrees might make to the leading coefficient, that is assuming the subtrees are roughly balanced, we can compute an approximation to the leading term of the IPL by

$$\begin{aligned}
&\int_{i=1}^{\sqrt{(2N/\pi)}} \pi i(\sqrt{[2N/\pi]} - i)\, di \\
&\quad = \int_{i=1}^{\sqrt{(2N/\pi)}} \sqrt{(2\pi N)} i di - \int_{i=1}^{\sqrt{(2N/\pi)}} \pi i^2 di \\
&\quad \approx \frac{\sqrt{(2\pi N)}\frac{2N}{\pi}}{2} - \frac{\pi\left(\frac{2N}{\pi}\right)^{\frac{3}{2}}}{3} \\
&\quad = \frac{1}{3}\left(\frac{2}{\pi}\right)^{\frac{1}{2}} N^{\frac{3}{2}} \\
&\quad \approx 0.266\, N^{\frac{3}{2}}.
\end{aligned}$$

This estimate ignores the contributions of the right subtrees, other than the distance from the root to the root of the subtrees. The subtrees themselves should make some contribution to the internal path length, since obviously the nodes cannot all be on the backbone of the tree.

For each subtree other than the one at the root, note that not only is the subtree root drifting to the right, but so are the roots of its neighbours. Thus the entire tree can be thought of as moving to the right. It is reasonable to expect that the subtrees are not greatly skewed, and the simulation results we present in subsequent sections confirm this.

Let us suppose without further justification that in fact the subtrees have an IPL close to that of the initial expected IPL of a tree of their respective size. That is, assuming πi nodes, we expect the IPL to be $2\pi i \ln(\pi i) - 2.846\, \pi i$, where ln is the natural log. Thus, over all the subtrees, we can approximate the total contribution by

$$\begin{aligned}
&\int_{i=1}^{\sqrt{(2N/\pi)}} 2\pi i \ln(\pi i) - 2.846\, \pi i di \\
&\qquad \approx 0.693\, N \log_2 N - 2.008\, N + 2.445.
\end{aligned}$$

Adding this result to the above gives our completed estimate of

$$0.266\, N^{\frac{3}{2}} + 0.693\, N \log N - 2.008\, N + 2.445.$$

Despite the simplified form of this model (or perhaps because of it) this value nevertheless agrees quite well with the simulation results presented in the following sections.

3. COMPARISON TO PREVIOUS SIMULATION RESULTS

In this section we evaluate the data from the extensive simulations of Eppinger[8] and our own,[4] using our model as a guide. Previously, Eppinger conjectured that the IPL was $\Omega(N \log^3 N)$ when the Hibbard or Knuth algorithms were used for many updates. In Ref. 6 it was shown that for Exact Fit Domains, Knuth's algorithm gave the same expected backbone distribution as Hibbard's. A study of Knuth's algorithm shows that it has very little effect either on the number of nodes in the backbone or on the size of the right subtrees. The intuitive reason is that it can only have an effect different from Hibbard's when the left subtree is empty. However, for the nodes of the backbone, this can only happen when the node being deleted is the smallest node in the tree, which is unlikely to have many nodes in its right subtree.

Table 1 presents the average IPL from simulations of the Hibbard algorithm as measured by Eppinger,[8] and for the Knuth algorithm by us. These values were obtained from the published ratios by multiplying by the expected IPLs of random trees using the formula for random trees given in Knuth[12] (page 427). In these simulations, a random tree was first generated, then the prescribed number of iterations were performed. During this process the IPL was computed at frequent intervals. The process was repeated for the indicated number of trees. The average IPLs shown in Table 1 are obtained only from data collected after the first N^2 updates, since only then did the trees appear to be stable. The arguments of Section 2 show that this conjecture was justified. The extent of the simulations is shown by the large number of iterations and the number of trees simulated in Table 1. The Knuth algorithm was run on 50 trees in each case.

Results of regressions fitting the data from Table 1 to functions involving $N^{\frac{3}{2}}$, $N \log(N)$ and N are presented in Table 2. All regressions are unweighted. Note the close agreement (and low mean square) of the Hibbard regression with four terms $0.259\, N^{\frac{3}{2}} + 0.75\, N \log N - 1.2\, N + 18.4$ with the analytic prediction of $0.266\, N^{\frac{3}{2}}$.

Table 2. Regression coefficients for data of Table 1

	$N^{\frac{3}{2}}$	$N \log(N)$	N	Constant	Mean square
Hibbard	0.405	—	—	634.73	304850.0
	0.270	0.570	—	−14.38	79.1
	0.259	0.726	−1.228	18.44	5.5
Knuth	0.434	—	—	272.04	60566
	0.256	0.582	—	−9.51	5.9
	0.248	0.659	−0.527	−1.42	2.0

Table 3. Right subtree using Hibbard deletion

Number of nodes	Millions of updates	Number of subtrees	Subtree size		Backbone	
			Average	Variation	Average	Variation
512	7	236	51.9	801.8	18.4	4.2
1024	22	237	79.6	1765.0	25.4	4.4
2500	70	209	118.4	3652.2	40.8	6.4
4096	140	188	158.3	6529.5	51.5	8.4
8192	600	270	233.1	15466.2	71.1	14.7
8192	400	169	229.6	15250.6	72.3	13.7
8192	400	181	219.3	13532.8	73.2	10.0

4. CHECKING THE MEASURES FROM THE MODEL

We now consider simulations designed to measure the length of the backbone and the number of nodes in the right subtree as estimated in Section 2. These simulations were progressively extended to include extra data gathering as our understanding increased and we required more detailed information. Originally only the length of the backbone and the size of the right subtree were measured. Later, we extended the lengths of the runs measuring the IPL in addition to the backbone and right subtree size. The purpose was to allow comparison with the IPLs obtained by Eppinger. Since we gathered data at specific times as described below, we wished to test that this did not significantly affect our measures. Again, the lengths of the runs were extended even further, this time gathering information on the structure of the top five right subtrees of the backbone. Finally, three extra long simulations were performed for trees of size 8192 gathering all types of data throughout.

Figure 5 illustrates the times at which the data were gathered. A new right subtree for the root is created when the key in the current root is deleted, and the right subtree of the root is empty. Note that at these times the interval containing the new subtree has had the maximal time for growth, since any further change to this interval can only decrease the interval size by moving the root towards the right of the domain. This, of course, is not the same as saying that the right subtree is at its maximum size, or that it is the maximal subtree in the tree. However, it should give a good approximation to the subtree sizes as estimated in Section 2.

Unlike previous simulations, in this one we performed updates for a long time on a single tree. This makes the simulation more efficient, since the subtrees in the initial tree must all be discarded from the analysis, to ensure that all the measured subtrees are between tags created during the updating process. The data analysis routines checked the number of nodes in the backbone of the initial tree and discarded that many subtrees from the calculations. This means that approximately N^2 updates were wasted on each initial tree. If there are B nodes in the backbone of a tree at any point in the simulation, then after B more subtrees have been removed, the tree is made up entirely of new keys, and the shape of this new tree will be independent of the shape of the previous one.

Table 3 summarises the results of these simulations. These are the cumulative results from all three stages of the simulations. The column titled 'Number of subtrees' shows how many subtrees were used in computing the averages. The standard deviation of the subtree sizes is high, being approximately one-half of the mean in each case. The variance on the length of the backbone, on the other hand, is small. The number of updates is the number performed during the simulation. In general, the last new subtree occurred slightly prior to this time, and thus the number of updates required to generate the indicated subtrees was slightly less.

In Table 4, the averages for the sizes of the subtrees and the lengths of the backbone given in Table 3 are expressed as multiples of $N^{\frac{1}{2}}$. As expected, these values are close to, but generally less than, the value of $\sqrt{2\pi} \approx 2.5$ computed in the approximation model. The values range from a low of 2.29 for $N = 512$ to a high of 2.58 for $N = 8192$ and appear to increase with N. If we compute the average of the three runs of 8192, we find the coefficient for the subtree size is 2.512 and for the backbone it is 0.798, which agree fairly well with the conjected 2.506 and 0.798 respectively.

Table 5 shows the average IPL computed during the extended simulations. We repeat that these values are computed using data collected at the special times when a new right subtree has been created at the root. Deleting the root when it has no right subtree moves every other node in the tree up one level. Thus we would expect the IPL to be reduced by about N from the IPL prior to that update. Averaging over these two cases would increase the result by $N/2$, which is the basis for the adjusted IPL in the table. In comparing the values obtained for trees of sizes 512 and 1024 in Table 5 with the values in Table 1, we see that the results are in reasonable agreement. The column indicating the number of subtrees indicates

Table 4. Average subtree and backbone values expressed as ratios to $\sqrt{N}$

Number of nodes in the tree (N)	Right subtree	Standard deviation	Backbone length	Standard deviation
512	2.29	1.25	0.813	0.091
1024	2.49	1.31	0.793	0.066
2500	2.37	1.21	0.816	0.051
4096	2.47	1.26	0.805	0.045
8192	2.58	1.37	0.786	0.042
8192	2.54	1.36	0.799	0.041
8192	2.42	1.29	0.809	0.035

Table 5. Average IPL from extended Hibbard simulations

Tree size	Number of subtrees	Average IPL	Adjusted IPL	Variance
512	107	5505	5761	112004
1024	137	13999	14511	645567
2500	112	48604	49854	4230873
4096	118	94987	97035	21450140
8192	270	250201	254297	214230784
8192	169	254515	258611	201874000
8192	181	259200	263296	77005696

Table 6. Regresson coefficients for combined data from simulations of the Hibbard algorithm

$N^{\frac{3}{2}}$	$N \lg(N)$	N	Constant
0.345	—	—	—
0.341	—	—	2707.50
0.269	0.517	—	191.09
0.266	0.537	—	—
0.317	−0.732	12.003	−567.80
0.292	−0.392	5.202	—

Table 7. Regression coefficients for combined adjusted data from simulations of the Hibbard algorithm

$N^{\frac{3}{2}}$	$N \lg(N)$	N	Constant
0.301	−0.268	7.833	−382.5
0.284	0.199	3.252	—
0.270	0.547	—	112.7
0.269	0.559	—	—
0.346	—	—	2777.2
0.351	—	—	—

the number of data points used in computing the means and variances.

If we combine this data with that from Eppinger[8] we increase the number of data points to 9, at the same time obtaining duplicate values for 512 and 1024 and of course 8192. Regression results for various combinations of $N^{\frac{3}{2}}$, $N \log_2 N$, and N are tabulated in Table 6. In Table 7 are similar regressions using the adjusted IPLs combined with previous results. These results are in good agreement with those of Table 2. Also, there seems to be good evidence for the model, even to reasonable approximations of the leading coefficient.

We now consider the measurements of the right subtrees themselves. We define the *Left Normal Measure* (LNM) of a tree to be the sum over all nodes of the tree of the number of nodes in the left subtree of the node. The *Right Normal Measure* (RNM) is defined analogously to be the sum over the right subtrees. If the tree is balanced, we expect LNM ≈ RNM.

These measures were taken for each of the topmost five right subtrees of the backbone each time data was gathered. We define the *Left Measure* (L) of a subtree to be the total over all the samples of the LNM of the subtree divided by the total over all samples of the number of nodes in the subtree. That is, it is an average measure of the LNM, although the term average is a bit fuzzy here since the size of the subtree also varies from sample to sample. The Right Measure R is similarly defined.

Referring to the typical results in Table 8, notice that the subtrees exhibit a tendency to be skewed to the right. This would seem to contradict the assumption that the subtrees are balanced. Also presented is an estimate of the average search-path length. This is just the sum of the Left and Right measures. (It is relatively easy to show that LNM + RNM = IPL. See Ref. 4 for details.) For comparison, the expected search cost of the random tree of the nearest integer size (see Knuth,[12] chapter 6.2.2), is also listed. In every case, the random cost exceeds our average cost. We thus seem to confirm the conjecture that the subtrees are reasonably well balanced. We conjecture that the difference $R-L$ is bounded by a constant, or at least that $(R-L/R+L)$ converges to zero. That is, we guess that the right subtrees remain well balanced for large trees.

Finally, we include a plot of the 270 subtrees from the first simulation of 8192 in Fig. 6. This gives at a glance an idea of the range and scatter of the subtree sizes when they first become right sons of the root.

5. CONCLUSION

We have presented extensive evidence that the use of Hibbard's deletion algorithm or Knuth's deletion algorithm when coupled with insertions causes the average internal path length of binary search trees to increase significantly if used for prolonged periods. The evidence indicates that the asymptotic IPL is in fact $\Omega(N^{\frac{3}{2}})$.

If we examine Fig. 1, we notice that the conjectured curve fits the unadjusted data points more closely than it fits the data points from Eppinger's simulations. Looking at the analysis that led to the formula, we see that we indeed computed this for the same selection of data points as that in the simulation. Although the analysis is

Table 8. Left/right balance of the top five subtrees

Tree size = 8192			Number of subtrees used = 169	
Average subtree size	Average left measure	Average right measure	Average path length	Expected path of a random tree
229.621	3.387	3.970	7.356	8.087
218.663	3.341	3.956	7.297	7.983
226.746	3.334	3.993	7.327	8.062
217.970	3.291	3.970	7.261	7.983
218.089	3.267	3.945	7.213	7.983

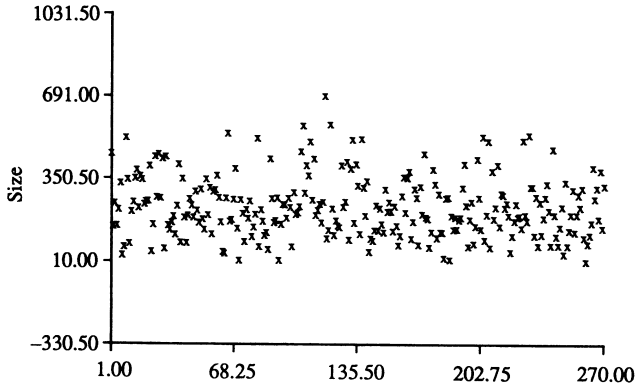

Figure 6. The 270 subtrees of 8192, from 600 million updates.

far too slack to engender much faith in the lower-order terms, it is interesting to speculate that adjusting the curve in the same way as the data would lead to an even closer fit of Eppinger's data. Using the adjusted data points causes the projections of Eppinger's curve to fit even less well for $n > 2500$.

Eppinger[8] simulated a modified version of Hibbard's algorithm which randomly decided between mirror-image versions of Hibbard's replacement criteria. That is, he modified Hibbard's algorithm so that it randomly decided to choose either the predecessor or the successor as the replacement node. This symmetric version of Hibbard's algorithm caused the IPL to be reduced on average as a large number of updates were performed, with no evidence of a long-term deterioration. Given our arguments in Section 2, we would expect that a similar symmetric version of Knuth's algorithm might lead to even greater improvement. We have confirmed these results by simulations (reported in greater detail in Refs 4 and 5). These results are summarized in Table 9, where the measured mean steady-state IPL is expressed as a ratio to the theoretical expected initial IPL.[12] The data for the Hibbard algorithms comes from Eppinger.[8] These data clearly exhibit the superiority of these algorithms over the asymmetric versions. Note that the ratios for the symmetric algorithms improve as the trees become larger. Observe that for the symmetric version of the Knuth algorithm the ratio is 0.871, or the IPL is about $1.207\, N \log N$, suggesting a number of intriguing hypotheses. For example, is the IPL converging to $12/7\, n \ln N$ ($\approx 1.19 \ldots N \log N$) plus lower-order terms? Such a distribution occurs in generating trees with a sequence of N random insertions and employing the 'fringe heuristic'[13] that any subtree of size 3 must be balanced. Clearly the classes are different as the symmetric deletion scheme can generate the prohibited dangling chains of length 3, but it is possible that this effects only the lower-order terms and that the 'global effect' is the same.

Table 9. Ratio of final path length to expected initial length

	Hibbard algorithm (from Eppinger)		Knuth algorithm	
Nodes	Asymmetric	Symmetric	Asymmetric	Symmetric
32	—	—	0.947	0.903
64	0.97	0.905	0.957	0.891
128	1.00	0.890	0.989	0.878
256	1.06	0.888	1.051	0.878
512	1.16	0.890	1.141	0.876
1024	1.30	0.881	1.277	0.871

REFERENCES

1. A. V. Aho, J. E. Hopcroft and J. D. Ullman, *Data Structures and Algorithms*. Addison-Wesley, New York, (1983).
2. R. A. Baeza-Yates, *A Trivial Algorithm Whose Analysis Isn't: A Continuation*. Research Report CS-86-67, University of Waterloo, Canada (1986).
3. A. D. Booth and A. J. T. Colin, On the efficiency of a new method of dictionary construction. *Information and Control* **3**, 327–334 (1960).
4. J. C. Culberson, *Updating Binary Trees*. Technical Report CS-84-08, University of Waterloo, Canada (1984).
5. J. Culberson, *The Effect of Asymmetric Updates in Binary Search Trees*. Technical Report CS-86-15, PhD Thesis, University of Waterloo, Canada (1986).
6. J. Culberson and J. I. Munro, Analysis of the standard deletion algorithms in exact fit domain binary search trees. *Algorithmica* (In the Press).
7. A. S. Douglas, Techniques for the recording of, and reference to data in a computer. *The Computer Journal* **2** (1), 1–9 (1959).
8. J. L. Eppinger, An empirical study of insertion and deletion in binary trees. *Comm. ACM 26* (1983).
9. T. N. Hibbard, Some combinatorial properties of certain trees with applications to searching and sorting. *Journal of Association of Computing Machinery* **9** (1), 13–28 (1962).
10. A. T. Jonassen and D. E. Knuth, A trivial algorithm whose analysis isn't. *Journal of Computer and System Sciences* **16**, 301–322 (1978).
11. G. D. Knott, Deletion in Binary Storage Trees. *Ph.D. Thesis STAN-CS-75-491*, Stanford University (1975).
12. D. E. Knuth, Searching and sorting. In *The Art of Computer Programming*, III (1973).
13. P. V. Poblete and J. I. Munro, The analysis of a fringe heuristic for binary search trees. *Journal of Algorithms* **6**, 336–350 (1985).
14. A. M. Tenenbaum and M. J. Augenstein, *Data Structures Using Pascal*. Prentice-Hall, Englewood Cliffs, New Jersey (1986).
15. P. F. Windley, Trees, forests and rearranging. *The Computer Journal* **3**, 84–88 (1960).
16. N. Wirth, *Algorithms & Data Structures*. Prentice-Hall, Englewood Cliffs, New Jersey (1986).