GEORGE E. FORSYTHE AWARD

AN EMPIRICAL STUDY OF INSERTION AND DELETION IN BINARY SEARCH TREES

JEFFREY L. EPPINGER *Carnegie-Mellon University*

Jeff Eppinger's current interests include distributed systems, database design, and analysis of algorithms.

This research was sponsored in part by the Office of Naval Research under contract N00014-76-C-0370.

Author's Present Address: Jeffrey L. Eppinger, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania 15213.

0001-0782/83/0900-0663 75¢

***ABSTRACT:** This paper describes an experiment on the effect of insertions and deletions on the path length of unbalanced binary search trees. Repeatedly inserting and deleting nodes in a random binary tree yields a tree that is no longer random. The expected internal path length differs when different deletion algorithms are used. Previous empirical studies indicated that expected internal path length tends to decrease after repeated insertions and asymmetric deletions. This study shows that performing a larger number of insertions and asymmetric deletions actually increases the expected internal path length, and that for sufficiently large trees, the expected internal path length becomes worse than that of a random tree. With a symmetric deletion algorithm, however, the experiments indicate that performing a large number of insertions and deletions decreases the expected internal path length, and that the expected internal path length remains better than that of a random tree.*

1. INTRODUCTION

A binary tree created by inserting n randomly chosen keys into an empty tree has an expected internal path length of $I_n \approx 1.386n \lg n - 2.846n$.[1] Randomly deleting k nodes from such a tree yields a tree whose expected internal path length is I_{n-k}. Unfortunately, performing insertions after deletions does not produce binary trees whose internal path length is predicted by this function. A theoretical explanation of the effect of performing deletions and then insertions on binary trees is still lacking [6].

This paper presents an empirical study on the effect of applying random insertions and deletions to random binary search trees and analyzes results of experiments comparing asymmetric and symmetric deletion algorithms. In a previous empirical study, Knott [3] suggested that the expected internal path length tends to *decrease* after repeated insertions and asymmetric deletions. In this study, the large number of insertions and asymmetric deletions performed suggests that the expected internal path length first decreases but eventually begins to *increase*. For sufficiently large trees, expected internal path length becomes *worse* than that of a random tree. However, experiments using the symmetric deletion algorithm show that performing a large number of insertions and symmetric deletions *decreases* the expected internal path length (making the trees better than random).

Section 2 describes the insertion and deletion algorithms used in this study and provides an overview of some of the previous work in this area. The statistics used in this study are defined in Section 3 which also mentions specifics about how the data was gathered. The observations in Section 4 give an interpretation of the data, and the conclusions are summarized in Section 5.

[1] Throughout this paper, lg x denotes $\log_2 x$.

```
PROCEDURE Insert(VAR root : NodePtr; x : DataType);
BEGIN
    IF root = NIL
        THEN BEGIN
            NEW(root); root↑.data := x;
            root↑.lChild := NIL; root↑.rChild := NIL
        END
    ELSE IF x < root↑.data
        THEN Insert(root↑.lChild, x)
    ELSE Insert(root↑.rChild, x)
END;
```

FIGURE 1. The Insertion Procedure.

2. BACKGROUND

Insertion Algorithm. The structure of binary trees naturally leads to one insertion algorithm. To insert a node into a binary tree known not to contain that node, compare the new and current keys and insert the node into the left or right subtree, whichever maintains the invariant of the data structure. The Pascal code for this algorithm is provided in Figure 1. (For further explanation, see Algorithm T in [6].)

Unlike insertion, there are many reasonable deletion algorithms from which to choose. This paper describes experiments with Hibbard's asymmetric deletion algorithm [1] and a trivially modified version of this algorithm to make it symmetric.

Asymmetric Deletion Algorithm. A node's *successor* is defined to be the smallest node in its right subtree. Similarly a node's *predecessor* is defined to be the largest node in its left subtree. To delete a node from a binary tree, replace the node with its successor, i.e., the node that contains the next larger key. The Pascal code for this algorithm is given in Figure 2. Figure 3 shows examples of the insertion algorithm and this deletion algorithm applied to a particular binary tree. (For further explanation, see Algorithm D in [6]).

Symmetric Deletion Algorithm. To delete a node from a binary tree, replace the node with its successor or predecessor. Alternately choose the successor and predecessor (so that half the time the **RightDelete** routine is called and half the time a suitably modified version of this routine, **LeftDelete,** is called).

In this paper, a random insertion consists of inserting a key which has been randomly selected from a uniform distribution. When performing a random deletion, each of the nodes in the tree has an equal chance of being selected for deletion. Knuth [7] describes this and several other ways to "randomly select" keys for insertion and deletion, and discusses how these schemes are related to one another.

Consider building a binary tree using n keys chosen randomly from a uniform distribution (*i.e.*, all $n!$ permutations of the keys are equally likely). There are $\binom{2n}{n}/(n+1)$ possible shapes for this tree [4], each with some probability of occurring; call the distribution D_n. By this definition, inserting a new node into this binary tree would yield a tree of size $n+1$ whose shape occurs with a probability defined by D_{n+1}. Binary trees whose distribution of shapes is D_n are called *random binary trees.*

```
PROCEDURE RightDelete(VAR root : NodePtr; x : DataType);
    VAR copy, successor, succPtr : NodePtr;
BEGIN
    IF x < root↑.data
        THEN RightDelete(root↑.lChild, x)
    ELSE IF x > root↑.data
        THEN RightDelete(root↑.rChild, x)
    ELSE BEGIN
        copy := root;
        IF root↑.rChild = NIL
    { Case I: There is no successor. }
            THEN root := root↑.lChild
        ELSE IF root↑.rChild↑.lChild = NIL
    { Case II: The successor is the right child. }
            THEN BEGIN
                root↑.rChild↑.lChild := root↑.lChild;
                root := root↑.rChild
            END
    { Case III: The successor is the leftmost child in the right subtree. }
        ELSE BEGIN
            succPtr := root↑.rChild;
            WHILE succPtr↑.lChild↑.lChild <> NIL DO
                succPtr := succPtr↑.lChild;
            successor := succPtr↑.lChild;
            succPtr↑.lChild := successor↑.rChild;
            successor↑.lChild := root↑.lChild;
            successor↑.rChild := root↑.rChild;
            root := successor
        END;
        DISPOSE(copy)
    END
END;
```

FIGURE 2. The Asymmetric Deletion Procedure.

Hibbard [1] proved that deleting a random node, (*i.e.*, where each node has an equal probability of being deleted), from a binary tree of size n with distribution of shapes D_n, yields a tree with a distribution of shapes D_{n-1}.

Strangely, performing random insertion and deletion operations on a random tree does not preserve this distribution of shapes. Consider building a binary tree of size n, as described above. Since the keys are chosen from a uniform distribution, the probability of inserting a new node in any particular interkey gap is $1/(n+1)$. After one random deletion, the distribution of shapes will be D_{n-1}, but the probability of inserting a new node where the deleted node used to be will be $2/(n+1)$ while all other places are still $1/(n+1)$. Knuth [6] describes this phenomenon as follows:

> The shape of the tree is random after deletions, but the relative distribution of values in a given tree shape may change, and it turns out that the first random insertion

```
FOR i := 1 TO tsize DO RndInsert;
... gather data ...
FOR i := 1 TO intervals DO BEGIN
    FOR j := 1 TO isize DO BEGIN RndInsert; RndDelete END;
    ... gather data ...
END;
FreeTree;
```

FIGURE 3. Examples of Insertion and Asymmetric Deletion.

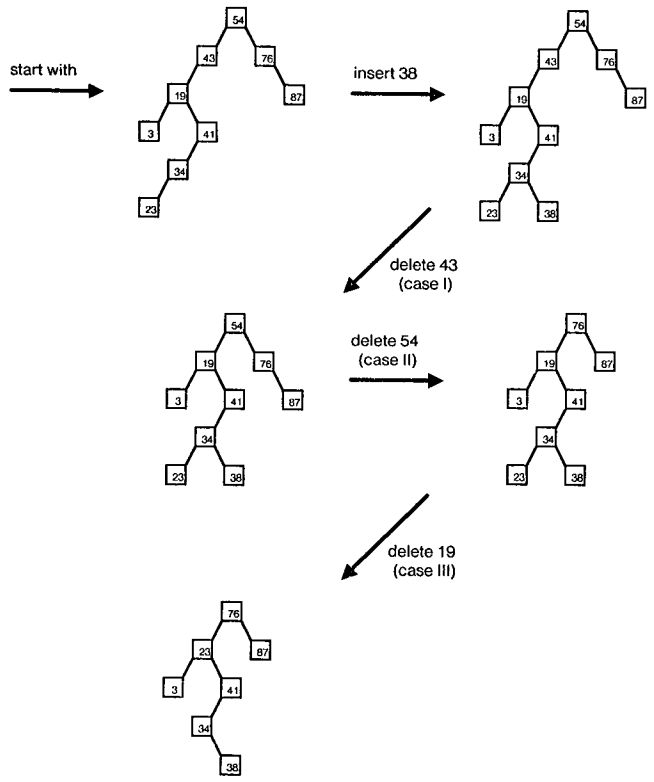


FIGURE 4. The Simulation Outer Loop.

after a deletion actually *destroys* the randomness property on shapes. This startling fact, first observed by Gary Knott in 1972, must be seen to be believed. Empirical evidence suggests strongly that the path length tends to *decrease* after repeated deletions and insertions, so the departure from randomness seems to be in the right direction; a theoretical explanation for this behavior is still lacking.

The expected number of comparisons used when searching for an element in a binary tree is proportional to the tree's path length. Thus, a binary tree is said to *improve* when its internal path length decreases. The internal path length of a tree is defined as the sum of the depths of the nodes in the tree

$$IPL = \sum_{i \in \{nodes\}} \text{distance}(root, i)$$

For a random tree containing n nodes, the expected IPL is denoted as I_n and the expected number of comparisons in a successful search is denoted as C_n. Knuth [6] gives the expected number of comparisons in a successful search, C_n, as $2(1 + 1/n)H_n - 3$ which is approximately equal to $1.386 \lg n - 1.846$.[2] The n^{th} harmonic number, $1 + \frac{1}{2} + \frac{1}{3} + \cdots + 1/n$, is denoted by H_n. Substituting into the relation $I_n = n(C_n - 1)$, one obtains the approximation $\approx 1.386n\, I_n \lg n - 2.846n$. A distribution of trees is said to be "better than random" when the expected IPL is less than I_n.

[2] The n^{th} harmonic number, $1 + \frac{1}{2} + \frac{1}{3} + \ldots + 1/n$, is denoted by H_n.

A theoretical explanation of what happens to the IPL of a binary tree after applying an arbitrary sequence of insertions and deletions has not been found. The analysis would be complicated. In his thesis, Knott introduces a lot of notation to describe binary trees to which sequences of insertions and deletions have been applied. In this paper, T empirically examine the effect on IPL of applying *pairs* of insertions and deletions to binary trees. Jonassen and Knuth [2] actually analyze the special case of binary trees with only three nodes; their solution involves the manipulation of Bessel functions.

3. METHODOLOGY

If a random sequence of insertions and deletions were applied to a random tree of size n, the resulting tree would probably not have the same number of nodes. Therefore, the original tree's IPL would not be directly comparable with the IPL of the new tree. In this study, sequences of *insertion/deletion pairs* (I/D pairs) are applied to random trees. Since the resulting tree always has the same size, it is easy to see whether any improvement has been made. (Knott's data was also obtained by using I/D pairs.) The first step of the simulation, then, is to insert n nodes into an empty tree, after which successive pairs of insertions followed by deletions are performed.

Let $\overline{IPL}_{n,i}$ denote the *measured* mean IPL of an n-node binary tree after applying i I/D pairs. Figures 5 through 10

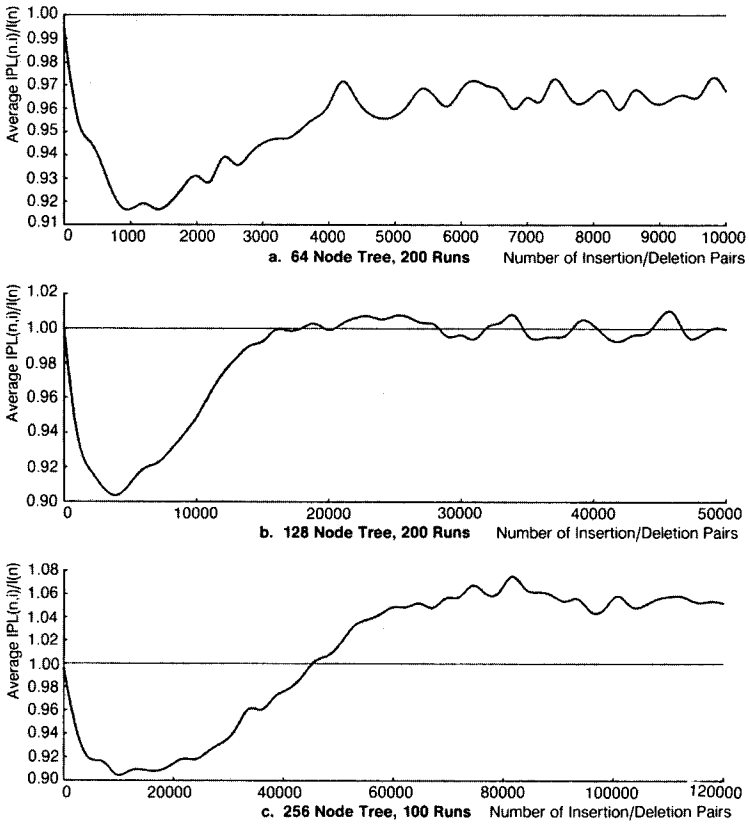
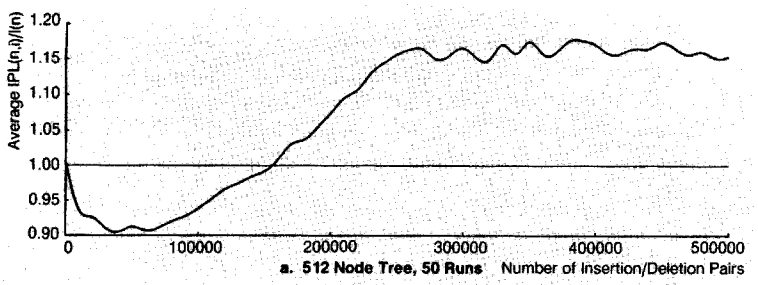
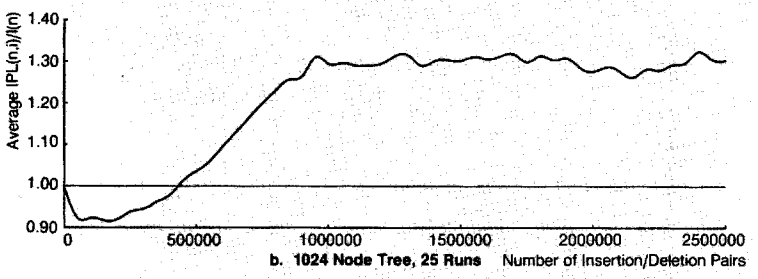
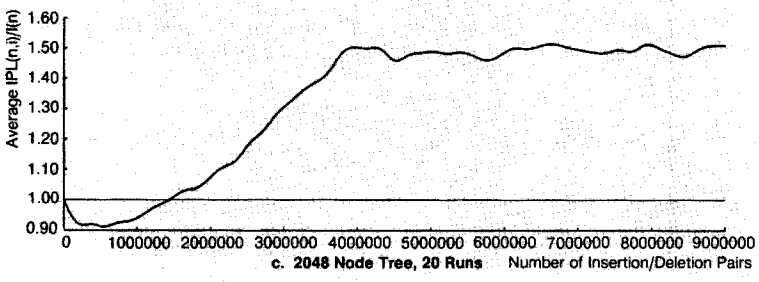


FIGURE 5. Asymmetric Deletions.







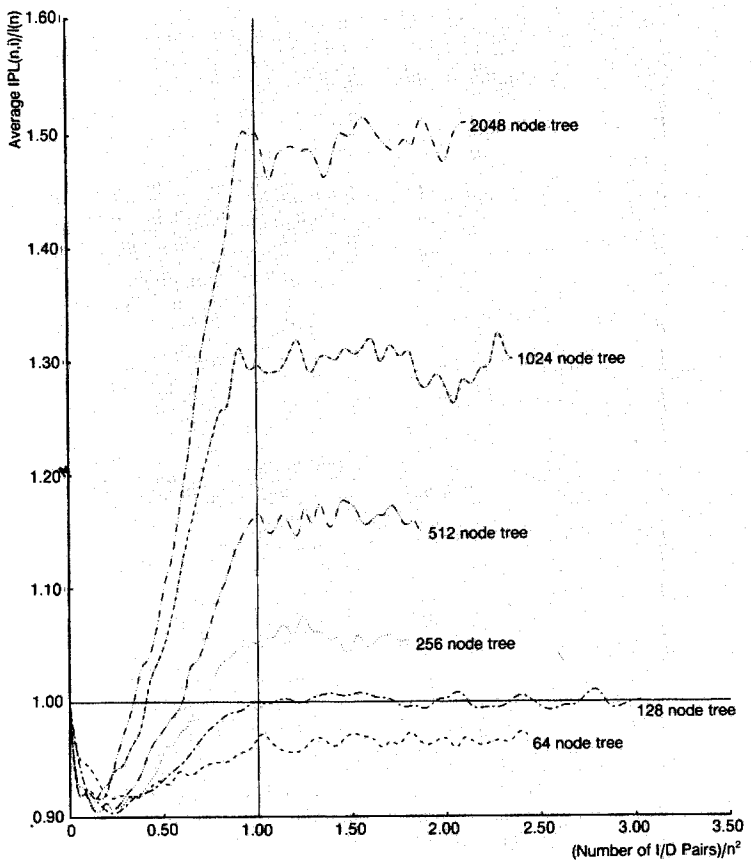


FIGURE 6. Asymmetric Deletions.

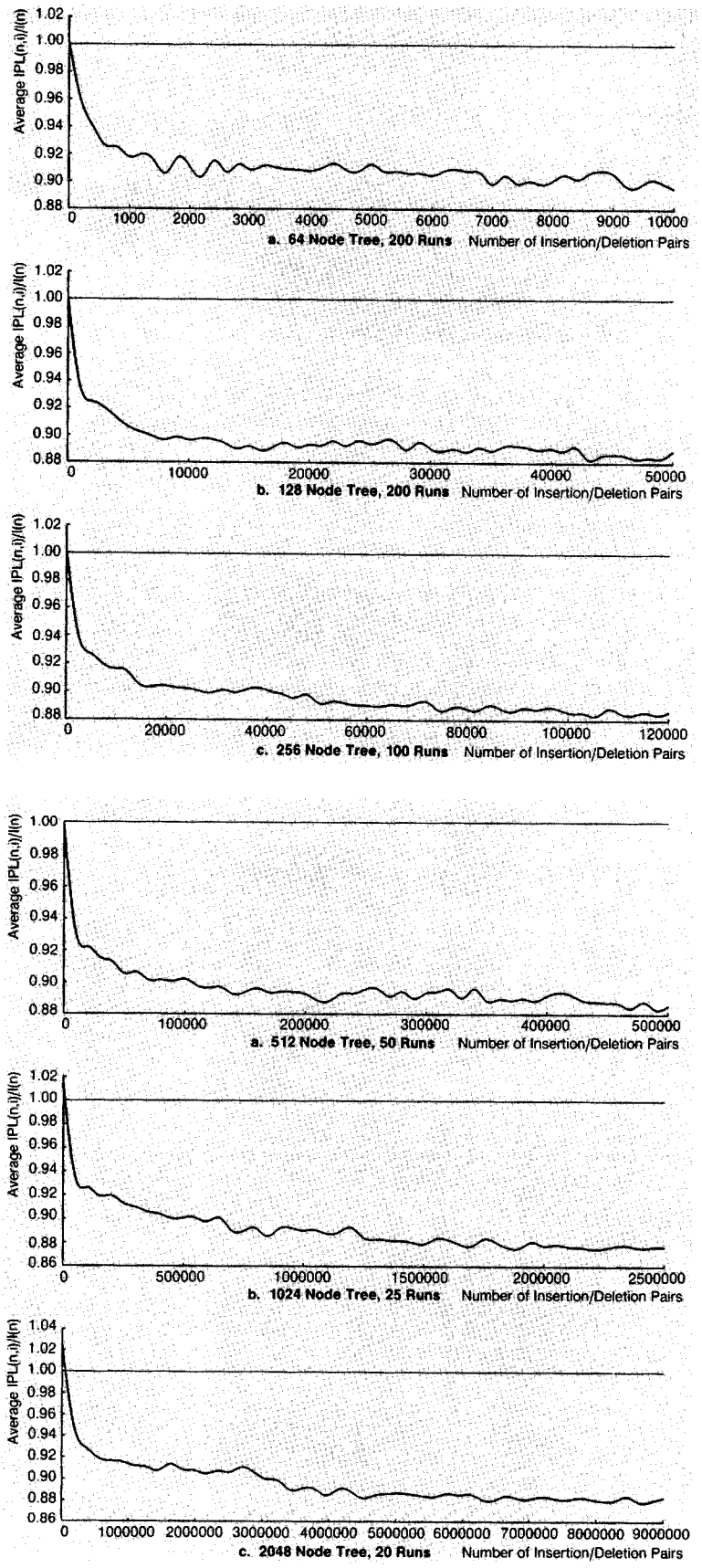
show $IPL_{n,i}/I_n$ plotted as a function of i. This ratio shows the improvement of the resulting tree's expected IPL as a fraction of the random tree's expected IPL.

The deletion algorithm given above generally replaces the node to be deleted with its *successor*, the "left-most node in the right subtree." The left and right subtrees are treated differently and, as observed below, this appears to have a profound affect on the behavior of binary trees. Such a deletion algorithm is called an *asymmetric* deletion algorithm. The *symmetric* deletion algorithm examined in this study is a trivially modified version of the asymmetric algorithm that *alternately* replaces the node to be deleted with its successor or its predecessor. Similar results have been obtained by *randomly* replacing the node to be deleted by its successor or predecessor.

To ensure that the results were not an artifact of the random number generator (or a bug), simulations were performed on both DEC-20s and Perqs. These machines have different architectures; the code for each implementation

FIGURE 7. Comparison Chart for Asymmetric Deletions.

FIGURE 8. Symmetric (Alternating) Deletions.



was separately written; and each implementation used a different random number generator. In the DEC-20 simulations, the random number generator used the linear congruential method to produce 36-bit pseudorandom numbers [5]. The random number generator for the Perqs is the feedback shift-register pseudorandom number generator described by Lewis and Payne [8]. The data presented in this paper was generated on the Perqs and took about one month of CPU time, but similar results were obtained for the smaller trees on the DEC-20s.

The outer loop of the simulation program (shown in Figure 4) is very simple. First, build a tree with **tsize** nodes, then gather data before and after each interval of **isize** I/D pairs.

4. OBSERVATIONS

The graphs in Figures 5 and 6 show the expected internal path length of n-node binary trees plotted against the number of insertion and asymmetric deletion pairs. Initially, $IPL_{n,i}$

FIGURE 9. Symmetric (Alternating) Deletions.

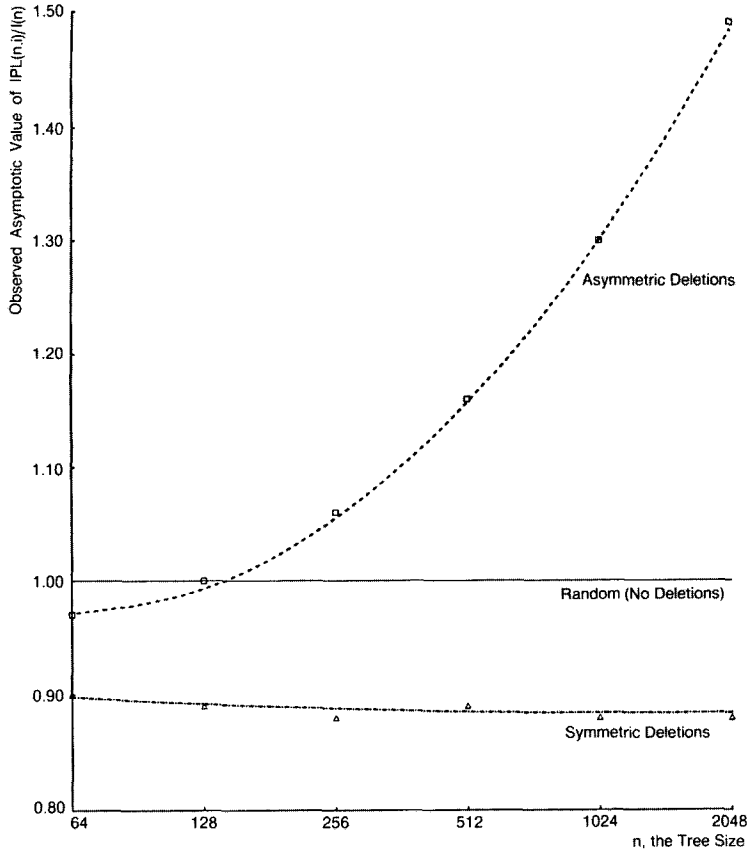


FIGURE 10. Comparison Chart for Symmetric (Alternating) Deletions.

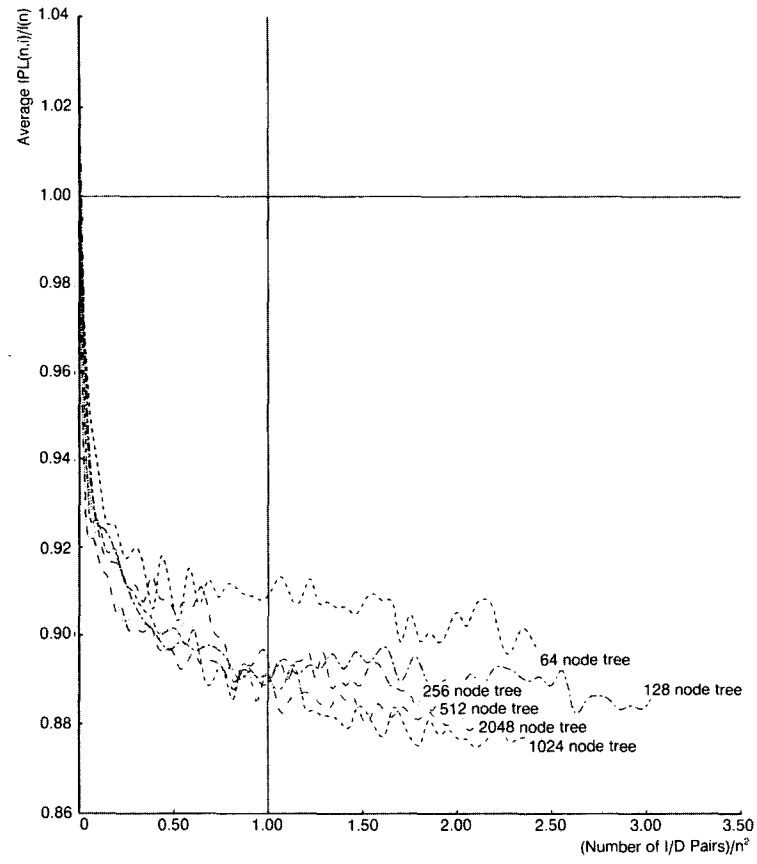


FIGURE 11. Comparison Chart of the Asymptotic Values of IPL(*n*, *i*).

decreases, as Knott observed. After some critical point, though, $\overline{IPL}_{n,i}$ starts to increase, eventually levelling off after approximately n^2 I/D pairs. Figure 7 is a comparison chart in which $\overline{IPL}_{n,i}/I_n$ is plotted as a function of i/n^2 for each of the values of n tested. (The latter ratio normalizes the x axis.)

Perhaps the most significant observation is that as n increases so does the asymptotic value for $\overline{IPL}_{n,i}/I_n$. Binary tree operations, such as insertion and deletion, can be modeled by Markov Chains (but the state space would be quite large). Since any binary tree may be obtained by applying some combination of I/D pairs to any other binary tree, the $\lim_{i\to\infty} \overline{IPL}_{n,i}$ exists [9]. Figure 7 suggests that

$$\lim_{i\to\infty} \overline{IPL}_{n,i} > I_n$$

for sufficiently large values of n (roughly greater than 128). Thus binary trees seem to become "worse than random" after many insertions and deletions.

The comparison chart in Figure 11 shows the asymptotic values of $\overline{IPL}_{n,i}/I_n$ for both deletion algorithms plotted against

Table I. Data for Asymmetric Deletions.

n	Samples	$\overline{IPL_{n,\,>n^2}}/I_n$	Variance
64	6000	0.97	0.01652
128	6800	1.00	0.01340
256	2300	1.06	0.00985
512	1200	1.16	0.00970
1024	750	1.30	0.01013
2048	5340	1.49	0.00771

Table II. Data for Symmetric Deletions.

n	Samples	$\overline{IPL_{n,\,>n^2}}/I_n$	Variance
64	6000	0.905	0.01654
128	6800	0.890	0.00916
256	2300	0.888	0.00615
512	1200	0.890	0.00347
1024	750	0.881	0.00235
2048	5340	0.883	0.00269

n (on a log scale). The data given in Table I summarizes the $\overline{IPL_{n,i}}$ and $\overline{IPL^2_{n,i}}$, for $i > n^2$. The asymmetric deletion curve appears to be quadratic. A least-squares multiple regression weighted by the inverse of the variance yields the following approximation:

$$\lim_{i\to\infty} \frac{\overline{IPL_{n,i}}}{I_n} \approx 0.0202 \lg^2 n - 0.241 \lg n + 1.69.$$

Substituting $I_n \approx 1.386n \lg n - 2.846n$ we obtain

$$\lim_{i\to\infty} \overline{IPL_{n,i}} \approx 0.0280n \lg^3 n - 0.392n \lg^2 n + 3.03n \lg n - 4.81n.$$

The graphs in Figures 8 and 9 show the corresponding plots of the data in Table II for the expected path length for symmetric deletions. The $\overline{IPL_{n,i}}$ decreases initially, as in the case of asymmetric deletions, but the asymptotic value of the expected internal path length seems to remain lower than that of a random tree. The comparison charts in Figures 10 and 11 indicate that

$$1 > \lim_{i\to\infty} \frac{\overline{IPL_{n,i}}}{I_n} \approx 0.88$$

or that

$$I_n > \lim_{i\to\infty} \overline{IPL_{n,i}} \approx 1.22n \lg n - 2.50n$$

The comparison chart in Figure 11 shows the asymptotic value of $\overline{IPL_{n,i}}$ slowly decreasing as n increases. Since a binary tree with n nodes cannot have an internal path length less than that of a perfect tree, we know that

$$\lim_{i\to\infty} \overline{IPL_{n,i}} = \Omega(n \log n).$$

5. CONCLUSIONS

The expected internal path length of a random binary tree is $I_n = \Theta(n, \log n)$. Empirical evidence suggests that performing many insertions and asymmetric deletions yields binary trees with an expected internal path length of $\overline{IPL_{n,i}} = \Theta(n \log^3 n)$. Thus, performing asymmetric deletions causes binary trees to become more unbalanced. Amazingly, the expected path length does not increase by a constant factor, but rather by a factor of $\log^2 n$. However, experiments show that the symmetric deletion algorithm improves the balance of binary trees leaving the expected internal path length $\Theta(n \log n)$; but with a *smaller* constant coefficient than the expected internal path length of a random binary tree.

Because this is an empirical study, the above conclusions can only be conjectures. No one has provided a theoretical explanation of the behavior of a binary tree's path length after applying deletions and then insertions. There is no proof that the asymptotic value of $\overline{IPL_{n,i}}$ is less than I_n when performing random insertions and symmetric deletions or that the asymptotic value of $\overline{IPL_{n,i}}$ is greater than I_n when applying insertions and asymmetric deletions. Furthermore, Jonassen and Knuth's intricate and subtle analysis of this problem for the special case of binary trees with three nodes gives an indication that a complete explanation may be quite difficult.

In closing, it should be noted that the results of this study will have little impact on the use of binary trees in practice. It takes approximately 1.5 million random insertions and asymmetric deletions to make a 2048-node binary tree worse than a random tree, and 4 million before its expected internal path length reaches the asymptotic value (which is just 50% worse). When so many operations are required, other data structures are probably more appropriate.

Acknowledgments: I would like to thank Jon Bentley, James Gosling, Diane Lambert, and Jim Saxe for their help and guidance.

REFERENCES

1. Hibbard, T.N. Some combinatorial properties of certain trees with applications to searching and sorting. *J. ACM 9*, 1 (January 1962) 13–28.
2. Jonassen, A.T. and Knuth, D.E. A trivial algorithm whose analysis isn't. *J. Comput. Syst. Sci. 16*, 3 (June 1978) 301–322.
3. Knott, G.D. *Deletion in binary storage trees*. Ph.D. Thesis. Stanford University (May, 1975), STAN-CS-75-491.
4. Knuth, D.E. *The Art of Computer Programming*. Volume I: *Fundamental Algorithms*. Addison-Wesley Publ. Co., Inc., Reading, Massachusetts, (1968) Section 2.3.4.4.
5. Knuth, D.E. *The Art of Computer Programming*. Volume II: *Seminumerical Algorithms*. Addison-Wesley Publ. Co., Inc., Reading Massachusetts (1969), Section 3.2.
6. Knuth, D.E. *The Art of Computer Programming*. Volume III: *Searching and Sorting* (Second Printing, March 1975). Addison-Wesley Publ. Co., Inc., Reading, Massachusetts, (1973), Section 6.2.2. Note: The Second Printing contains important changes in Section 6.2.2.
7. Knuth, D.E. Deletions that preserve randomness. *IEEE Trans. Softw. Eng. SE-3*, 5 (September 1977) 351–359.
8. Lewis, T.G. and Payne, W.H. Generalized feedback shift register pseudorandom number generator. *J. ACM 20*, 3 (July 1973) 456–468.
9. Ross, S.M. *Applied Probability Models with Optimization Applications*. Holden-Day, Inc., San Francisco (1970) Section 4.3.

CR Categories and Subject Descriptors: E.1 [**Data**]: Data Structures—*trees*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems—*sorting and searching*
General Terms: Algorithms, Theory
Additional Key Words and Phrases: binary trees

Received 9/82; revised 5/83; accepted 8/83