The Expected Performance of Traversal Algorithms in Binary Trees

KEITH BRINCK
Department of Computer Science, University of Iowa, Iowa City, Iowa 52242, U.S.A.

The paper compares expected performance measures for common traversal algorithms operating on threaded and unthreaded binary trees, under the assumption that the trees are selected from the distribution induced by random insertions. The results are shown to be similar to those derived in an earlier paper for binary trees selected from the uniform distribution.

1. INTRODUCTION

In an earlier paper Brinck and Foo investigated the performances of various algorithms for traversing threaded and unthreaded binary trees in each of the three standard traversal orders (preorder, inorder and postorder).[1] In particular, under the assumption that each binary tree is equally probable, the average times required to compute the successor and predecessor nodes of a random node were derived for each traversal order. Using these results it was then possible to compare directly the expected performances of tree traversal algorithms utilising stacks with those that use threads.

The results of this earlier paper are now extended to cover the case where the trees are selected, not uniformly, but from the distribution induced by random insertions.

We would like to stress that, over both distributions, the issue is not one of determining the execution order [which is obviously always linear], but is instead one of determining the constant of proportionality. While it may not be of great theoretical interest, a difference of a factor of two, for example, would be of some practical interest.

2. BACKGROUND

The definitions required conform largely to those used in standard sources such as Knuth or Reingold, Nievergelt and Deo.[2,3] However, for completeness we will briefly reiterate the main background required.

An n-node binary tree is defined to be a rooted tree where each of the n nodes has zero, one or two immediate descendants, and a distinction is made between the left and right subtrees. A left shell node is any node that can be reached from the root of a binary tree by traversing exclusively left branches, and a left shell branch is a branch connecting two left shell nodes. Right shell nodes and branches are defined analogously. Note that the root of a tree is by definition both a left and right shell node. The shell of a tree consists of all the left and right shell nodes and branches.

This paper is ultimately concerned with the analysis of traversal algorithms in threaded and unthreaded trees. By a threaded tree we mean a tree where null left and right links are set to point to the inorder predecessor and successor nodes respectively, an extra tagbit is included in each node to distinguish between threads and branches, and a special node called the header is created and set to point to the root of the tree in the standard fashion (Ref. 2, p. 322).

Over the uniform distribution binary trees are regarded as being unlabelled, with each different tree being equally probable. The number of such distinct n-node binary trees is given by the nth Catalan number, defined by

$$C_n = \frac{1}{n+1}\binom{2n}{n}.$$

A binary search tree is an n-node binary tree where each of the n nodes has a unique label from 1 to n attached, such that for any node its label is greater than the label of every node in its left subtree and less than the label of every node in its right subtree. If we regard the ith label as belonging to the node with the ith largest content then this definition is equivalent to that normally used for such trees in the literature. (Note that the single-node binary tree is by definition a binary search tree.)

In this paper we investigate the execution time required for several common tree traversal algorithms operating over a distribution generated by building binary search trees from permutations. This distribution arises when each n-node binary tree is considered to have grown as the result of a random series of n ordered insertions, each of which preserves the binary search property. Since there are $n!$ possible initial orderings of the n nodes to be inserted, and each input ordering (or permutation) gives rise to a binary search tree, there are $n!$ trees in this distribution. It should be noted, however, that the resulting trees are not all different and that the mapping between permutations and trees is a many-to-one mapping. Note also that given an arbitrary n-node binary tree there is only one way of assigning the labels $1 \ldots n$ to its nodes such that the binary search property holds. Thus the distinction between trees arises not from the labelling of the nodes, but from the order in which the nodes were inserted into the tree. To emphasise that the distribution is induced by permutations we refer to it as the permutation distribution, and refer to trees selected from it as permutation trees. Similarly, trees selected from the uniform distribution are referred to as uniform trees.

A common quantity that arises in analyses over the permutation distribution is the nth harmonic number, defined by

$$h_n = 1+\tfrac{1}{2}+\tfrac{1}{3}+\tfrac{1}{4}+\ldots+\tfrac{1}{n}.$$

For our purposes it is sufficient to know that

$$h_n = \log(n)+\gamma+0(\tfrac{1}{n}),$$

where $\gamma = 0.5772156649$ is Euler's constant, but interested readers should consult Ref. 3 for more details.

We are interested in comparing the performance of

algorithms that traverse permutation trees in the three standard orders using respectively stacks and threads, and also to compare these traversal times with the corresponding times over the uniform distribution. In doing this we wish to make the basis for comparison as machine-independent as possible, so that the results are minimally affected by technological advances in either hardware or software. Since the algorithms under study all run on binary trees, our ideal basic operation is one that moves a pointer from one node in the tree directly to another. Operations such as traversing a branch or thread, or pushing/popping a node from the stack, would then count as basic operations, but other operations performed may not be as easily categorised, and some doubt may often exist as to how they should be counted. In summary, we include any operations that can be safely construed to be equivalent to moving a pointer from one node to another within the tree, but admit that sometimes the decision may appear to be subjective.

3. ENUMERATION RESULTS OVER PERMUTATION TREES

In order to determine the expected performance of traversal algorithms over permutation trees we will first need to derive some counting results similar to those derived for uniform trees in Ref. 1. In that paper, enumerated properties of uniform trees were counted by setting up and solving recurrence relations similar to the one used to count the number of distinct n-node binary trees (Ref. 2, p. 388). For example, the number of nodes with exactly two immediate descendants was determined as follows.

Lemma[1]

The total number of nodes, over all n-node uniform binary trees, with exactly two immediate descendants is

$$\frac{(n-1)(n-2)}{2(2n-1)}C_n.$$

Proof.

The result is obtained by solving

$$k_0 = 0; \quad k_1 = 0,$$
$$k_n = \sum_{i=0}^{n-1} [2k_i C_{n-i-1}] + C_n - 2C_{n-1}, \quad n \geqslant 2,$$

where $C_n - 2C_{n-1}$ is the number of trees where the root has two immediate descendants, and the summation term counts the number of nodes that have exactly two immediate descendants over all possible combinations of left and right subtrees.

As there are C_n uniform binary trees in all, dividing the expression for k_n by C_n tells us that the expected number of nodes with exactly two immediate descendants in an n-node binary tree selected uniformly at random is

$$\frac{(n-1)(n-2)}{2(2n-1)}.$$

This, and other similarly derived properties, have been used to determine the expected performances of traversal algorithms in threaded and unthreaded binary trees over the uniform distribution.

In order to derive the corresponding results over the permutation distribution we will need to obtain similar counting results for permutation trees. As is the case with uniform trees, this is achieved via the solution of appropriate recurrence relations, and we now derive the precise mechanism for doing so.

As mentioned above, permutation trees differ from uniform trees only in their distribution: some permutation trees may be identical, whereas each uniform tree is unique. Consider those binary trees with a configuration of i nodes in the left subtree (and hence $n-i-1$ nodes in the right subtree). Define $u_{n,i}$ and $p_{n,i}$ to be the number of uniform and permutation trees respectively with this configuration. Then we know that

$$u_{n,i} = C_i C_{n-i-1},$$

i.e. the number of binary trees with i nodes in the left subtree is equal to the product of the total number of different trees in each subtree. And of course

$$C_n = \sum_{i=0}^{n-1} u_{n,i} = \sum_{i=0}^{n-1} C_i C_{n-i-1} = \frac{1}{n+1}\binom{2n}{n}.$$

The situation that arises over permutation trees is similar, the only difference being that each configuration does not arise uniquely. Let P_n equal the number of n-node permutation trees. Now we do not have that $b_{n,i}$ is simply equal to $P_i P_{n-i-1}$ because, although this does allow for the repeated trees in each subtree, it does not count the several times that the entire tree can be repeated. Therefore we also need to multiply $P_i P_{n-i-1}$ by the number of times that each entire tree can be repeated. For the configuration to be such that the tree contains i nodes in the left subtree, the root must be the node labelled $i+1$. Any of the remaining $n-1$ nodes may make up the i nodes in the left subtree, and hence there are $\binom{n-1}{i}$ ways of placing nodes into the left subtree (after which the content of the right subtree is uniquely determined). Thus,

$$p_{n,i} = P_n P_{n-i-1}\binom{n-1}{i},$$

and

$$P_n = \sum_{i=0}^{n-1} p_{n,i} = \sum_{i=0}^{n-1} P_n P_{n-i-1}\binom{n-1}{i}.$$

It is easy to verify that P_n evaluates to $n!$ as it should.

It now follows that once a recurrence relation has been derived for the uniform distribution it is straightforward to generate the recurrence that counts the same quantity over permutation trees: the term $i!$ is substituted for C_i throughout the original equation to allow for repeated subtrees, and the $\binom{n-1}{i}$ multiple configuration factor is inserted wherever summing over subtrees occurs. We will henceforth refer to this process as 'applying the standard transformation' to the original recurrence.

After transforming the relation from the lemma above we get:

$$k_0 = 0; \quad k_1 = 0,$$
$$k_n = \sum_{i=0}^{n-1} \left[2k_i(n-i-1)!\binom{n-1}{i}\right] + n! + 2(n-1)!,$$

which can be rewritten as

$$l_n = \frac{k_n}{n!} = \frac{1}{n}\sum_{i=0}^{n-1} [2l_i] + 1 - \frac{2}{n}.$$

Then

$$nl_n-(n-1)\,l_{n-1}=2l_{n-1}+1,\quad n\geqslant 2,$$
$$\text{i.e.}\quad nl_n=(n+1)\,l_{n-1}+1.$$

This can readily be solved to yield

$$l_n=\frac{n-2}{3},\quad \text{and hence}\quad k_n=\frac{n-2}{3}n!.$$

Table 1 gives the expected values of properties over both distributions that will be required later. The results for the uniform distribution are either obvious or will have been derived in Ref. 1. Some of the results for the permutation distribution are obvious and others may be found via standard means. However, all may be found by solving recurrences that are formed by standard transformation from the corresponding recurrences for the uniform distribution. But as this is a tedious and straightforward process, we will in all cases but one simply quote the final results. (Complete details may be found in Ref. 4.)

As mentioned, one property is considerably more difficult to evaluate over permutation trees than for uniform trees, and so before continuing we will investigate it in more detail. The property in question is the average number of nodes that lie on the path from the root of a permutation tree to the first node in the postorder traversal.

Let d_n denote the total number of such nodes over all n-node trees. The recurrence relation from which d_n can be computed for uniform trees is

$$d_0=0,$$
$$d_n=\sum_{i=0}^{n-1}[d_i\,C_{n-i-1}]+C_{n-1}+d_{n-1}.$$

After applying the standard transformation this may be written as

$$d_0=0,$$
$$d_n=\sum_{i=0}^{n-1}[d_i(n-i-1)!\,(n-1_i)]+n!+d_{n-1}.$$

After setting $a_n=\frac{d_n}{n!}$, this may be expressed as

$$na_n=(n+1)\,a_{n-1}+1-a_{n-2}.$$

Attempting to solve this via the usual means of a differential equation leads to a very complex integral and so we alternatively rewrite it as

$$a_n-a_{n-1}=\tfrac{1}{n}(a_{n-1}-a_{n-2})+\tfrac{1}{n},$$

and make the substitution $b_n=a_n-a_{n-1}$ to obtain the relation

$$b_n=\tfrac{1}{n}(b_{n-1}+1).$$

This relation can now easily be solved by unwinding to yield

$$b_n=\frac{1}{n!}\sum_{i=0}^{n-1}i!=\sum_{i=0}^{n-1}\frac{i!}{n!}.$$

Now, as $a_n=b_1+b_2+\ldots+b_n$, we have that

$$a_n=\sum_{j=1}^{n}\sum_{i=0}^{j-1}\frac{i!}{j!}.$$

Since $a_n=(d_n/n!)$, a_n actually represents the average value of d_n per tree. The double summation derived is the exact value of a_n, and we now estimate this value.

Table 1. Expected values of some properties over binary trees

	Uniform distribution	Permutation distribution
Branches	$n-1$	$n-1$
Threads	$n+1$	$n+1$
Leaves	$\frac{n(n+1)}{2(2n-1)}$	$\frac{n+1}{3}$
Nodes with exactly one descendant	$\frac{(n-1)(n+1)}{2n-1}$	$\frac{n+1}{3}$
Nodes with exactly two descendants	$\frac{(n-1)(n-2)}{2(2n-1)}$	$\frac{n-2}{3}$
Left branches	$\frac{n-1}{2}$	$\frac{n-1}{2}$
Left threads	$\frac{n+1}{2}$	$\frac{n+1}{2}$
Left branches and right threads	$\frac{n+1}{2(2n-1)(n-2)}$	$\frac{n+1}{6}$
Shell nodes	$\frac{5n-10}{n+2}$	$2h_n-1$
Left shell nodes	$\frac{3n-4}{n+2}$	h_n
Nodes on path from root to first node in postorder traversal	$\frac{4n(2n+1)}{(n+2)(n+3)}$	$\sim h_n+1.318$

where h_n is the nth harmonic number.

The value of $\Sigma_{j=1}^{n}i!/j!$ for i equal to an arbitrary value in the range 0 to $j-2$ has the form:

$$\frac{1}{k}\left[\frac{1}{k!}-\frac{1}{(n-k)(n-k+1)\ldots n}\right],$$

where $k=j-i+3$.[5] If the terms for a_n are arranged in tabular form indexed by i and j, then the diagonal terms sum to h_n and the sum of the entries on the kth off-diagonal is given by the term above. Thus the total sum of all off-diagonal terms may be given by

$$1+\frac{1}{2\times 2!}+\frac{1}{3\times 3!}+\frac{1}{4\times 4!}+\ldots+\frac{1}{(n-1)(n-1)!}$$
$$-0\left(\frac{1}{n}\right)-0\left(\frac{1}{n^2}\right)-0\left(\frac{1}{n^3}\right)-\ldots-0\left(\frac{1}{n^n}\right).$$

Disregarding $0(1/n)$ terms, etc., we have that

$$a_n\leqslant h_n+1+\frac{1}{2\times 2!}+\frac{1}{3\times 3!}+\ldots+\frac{1}{(n-1)(n-1)!}.$$

To evaluate the right-hand side of this expression, note that the exponential integral, $Ei(x)$, is defined by

$$Ei(x)=\int_{-\infty}^{x}\frac{e^{-t}}{t}dt$$
$$=\log(x)+\gamma+\sum_{i=1}^{\infty}\frac{1}{i\times i!},$$

where $\gamma=0.5772156649$ is Euler's constant. Thus

$$\lim_{n\to\infty}a_n=Ei(1)-\gamma+h_n$$
$$=h_n+\Phi,$$

where $\Phi = Ei(1) - \gamma$, and is approximately equal to 1.318 (i.e. 1.895117816 − 0.5772156649). Thus we get that

$$h_n \leqslant \frac{d_n}{n!} \leqslant h_n + 1.318.$$

4. TRAVERSALS OVER THE PERMUTATION DISTRIBUTION

The results of Brinck and Foo concerning traversals over uniform binary trees are now extended to cover the corresponding results over the permutation distribution.[1] The mechanism by which this is done is identical to that used to convert counting results to the new distribution: namely, the results are obtained by solving recurrences generated via standard transformation from recurrences over the uniform distribution. In general the reader is referred to the earlier paper for descriptions of how each original recurrence is derived, but we will perform the analysis of one algorithm in detail as an example of the technique involved.

Algorithm 3.1. Compute inorder successor in a threaded tree

```
procedure insucc (p)
{
  q ← p.rhs
  if p.rtag = branch then
      while q.ltag = branch do q ← q.lhs
  return (q)
}
```

Algorithm 3.1 computes the inorder successor of its parameter node. The key recurrence describing the performance of this algorithm over the uniform distribution can be found to be given by

$$k_0 = 0,$$
$$k_n = \sum_{i=0}^{n-1} [2k_i C_{n-i-1} + C_i q_{n-i-1}], \quad n \geqslant 1,$$

where q_i denotes the number of left shell nodes over all i-node uniform trees. From Table 1 we see that q_{n-i-1} equals $(3i - 4/i + 2)\, C_i$, which may be written as $C_{i+1} - C_i$. The term k_n evaluates to $(n - 1/n + 2)\, nC_n$ and counts the number of nodes that are left shell nodes of a right subtree, thus giving the number of times that the test 'q.ltag = branch' is executed when considered over the input set of all nodes in all uniform trees. Since the while-clause is invoked once for each node with a right branch, and the test 'q.ltag = branch' fails exactly once in each invocation, we have that the statement 'q ← q.rhs' is executed exactly

$$k_n - \frac{n-1}{2} C_n = \frac{(n-1)(n-2)}{2(n+2)} C_n$$

times. The statement 'q ← p.rhs' is obviously executed exactly once for each call of *insucc*, and so the total number of link-traverses performed by Algorithm 3.1 is found to be

$$\left[n + \frac{(n-1)(n-2)}{2(n+2)} \right] C_n = \frac{3n^2 + n + 2}{2(n+2)} C_n.$$

Dividing by nC_n gives an average asymptotic to $\frac{3}{2}$ as n increases.

To obtain the corresponding result over permutation trees the standard transformation is applied to the relation above and yields

$$k_n = \sum_{i=0}^{n-1} [2k_i (n-i-1)! + i!\, \bar{q}_{n-i-1}] \binom{n-1}{i},$$

where the term $\bar{q}_{n-i-1}$ now denotes the total number of left shell nodes over the permutation distribution. From Table 1 we have that $\bar{q}_{n-i-1} = h_{n-i-1}(n-i-1)!$, and once it has been substituted for, the recurrence may be solved. Note that this is not equivalent to first substituting the value of $C_{n-i} - C_{n-i-1}$ for q_{n-i-1} into the equation and then applying the standard transformation – it is important to apply the standard transformation on the initial version of the recurrence, not on a version simplified under assumptions holding only over the uniform distribution.

Once a recurrence has been transformed it may be solved fairly easily – usually the generating function is expressed as a first-order linear homogeneous differential equation, which may be solved by standard means to yield the required coefficients. The relation above may be found to be equivalent to

$$nl_n = (n+1)\, l_{n-1} + h_{n-1}, \quad \text{where} \quad l_n = \frac{k_n}{n!}.$$

This results in the generating function for the coefficients l_n being given by the differential equation

$$\frac{dL(x)}{dx} - \frac{2}{1-x} L(x) = \frac{H(x)}{x},$$

where $H(x) = [-\log(1-x)/1-x]$ is the generating function for the harmonic numbers. This ultimately results in a value for k_n of $(n - h_n)\, n!$, and the total number of link-traverses over the permutation distribution is given by

$$\left(n + nh_n - \frac{n-1}{2} \right) n! = \left(\frac{3n+1}{2} - h_n \right) n!.$$

The resulting average of $(3n + 1/2n) - (h_n/n)$ is again asymptotic to $\frac{3}{2}$ as n increases.

Algorithm 3.2. Compute postorder successor in threaded tree

```
procedure postsucc (p)
{
  q ← p
  while p.rtag = branch do p ← p.rhs
  if p.lhs = q
    then // q and successor are in opposite shells
    {
      while p.rtag = branch do
      {
        p ← p.rhs
        while p.ltag = branch do p ← p.lhs
      }
    }
    else // q and successor are in the same shell
    {
      p ← p.lhs
      while p.rhs ≠ q do p ← p.rhs
    }
  return (q)
}
```

The transformed recurrence relation giving the total number of operations performed by Algorithm 3.2 (postorder successor) is

$$k_n = \sum_{i=0}^{n-1} [(2k_i + i! + 2h_i\, i! + d_i)(n-i-1)!\,(\tbinom{n-1}{i})] - d_{n-1},$$

where d_i is the total distance from the root to the first node in the postorder traversal. However, as given above, we do not have an exact formula for d_n, but the following bounds:

$$h_n \leqslant \frac{d_n}{n!} \leqslant h_n + 1.318.$$

We therefore set $d_i = (h_i + \delta)\, i!$ in the recurrence relation for k_n, and denote the modified relation by $\bar{k}_n$. Setting $\delta = 0$ in the solution for $\bar{k}_n$ will thus yield a lower bound on k_n, and setting $\delta = 1.318$ will give the asymptotic upper bound. (It is obviously true that the value of $\bar{k}_n$ is monotone in the value of δ.) Thus we have that

$$\bar{k}_n = \sum_{i=0}^{n-1} [(2\bar{k}_i + i! + 3h_i\, i! + \delta i!)(n-i-1)!\,(\tbinom{n-1}{i})] - (h_{n-1} + \delta)(n-1)!.$$

Let

$$l_n = \frac{k_n}{n!} \quad \text{and} \quad \bar{l}_n = \frac{\bar{k}_n}{n!}.$$

Then we have that

$$\bar{l}_n = \frac{1}{n} \sum_{i=0}^{n-1} [2\bar{l}_i + 3h_i + 1 + \delta] - \frac{1}{n}(h_{n-1} + \delta).$$

The generating function, $\bar{L}(x)$, of the coefficients $\bar{l}_n$ can be found to be given by the following differential equation:

$$\frac{d\bar{L}(x)}{dx} + \frac{2}{x-1}\bar{L}(x) = [1 + \delta x^2 - (2+x)\log(1-x)]\frac{1}{(1-x)^2},$$

which implies that

$$\bar{L}(x) = \left(\frac{3}{1-x} - \frac{1}{2}\right)\log(1-x) + \left(\frac{15}{4} + \frac{\delta}{3}\right)\frac{1}{(1-x)^2} - \frac{4+\delta}{1-x} + \frac{\delta}{3}(x+2) + \frac{1}{4}.$$

The coefficient of x^n in $\bar{L}(x)$ is thus given by

$$\bar{l}_n = \left(\frac{15}{4} + \frac{\delta}{3}\right)(n+1) + \frac{1}{2n} - 3h_n - (4+\delta) = \left(\frac{15}{4} + \frac{\delta}{3}\right)n + \frac{1}{2n} - 3h_n - \frac{1}{4} - \frac{2\delta}{3}.$$

By setting $\delta = 0$ and 1.318 respectively we obtain the following bounds on l_n:

$$3\tfrac{3}{4}n + \frac{1}{2n} - 3h_n - \tfrac{1}{4} \leqslant l_n \leqslant 4.19n + \frac{1}{2n} - 3h_n - 0.69.$$

Since $l_n = (k_n/n!)$, (l_n/n) represents the average number of operations performed per node in computing the postorder successor. However, asymptotically $\delta \to 1.318$, and so

$$\lim_{n\to\infty} \frac{l_n}{n} \approx 4.19.$$

Determining the expected performance of Algorithm 3.3 (preorder successor) requires only the expected number of branches and the expected number of right threads. The number of branches is always $n-1$, and Table 1 shows that the expected number of right threads is equal to $(n+1/2)$, independent of the two distributions in question. Hence, the expected number of operations required to compute the preorder successor is the same for permutation trees as for uniform trees, namely asymptotic to 3/2.[1]

Algorithm 3.3. Compute preorder successor in a threaded tree

```
procedure presucc (p)
{
  if p.ltag = branch then return (p.lhs)
  while p.rtag = thread do p ← p.rhs
  return (p.rhs)
}
```

As with threaded trees from the uniform distribution, computing the expected traversal time for each order over the permutation distribution is simply a matter of computing

$$E(\text{TRAV}) = E(\text{START}) + nE(\text{SUCC})$$

where E(TRAV) is the expected traversal time, E(START) is the expected time to compute the first node in the traversal, and E(SUCC) is the expected time to compute a successor. (A more formal justification of this technique may be found in Ref. 1.)

For preorder we have

$$E(\text{START}) = 1,$$

$$nE(\text{SUCC}) = \frac{3n+1}{2} \Rightarrow E(\text{TRAV}) = \frac{3n+3}{2}.$$

For inorder we have

$$E(\text{START}) = h_n,$$

$$nE(\text{SUCC}) = \frac{3n+1}{2} - h_n \Rightarrow E(\text{TRAV}) = \frac{3n+1}{2}.$$

For postorder we have

$$E(\text{START}) = d_n,$$

where d_n is as defined earlier. The quantity d_n is also involved in the computation of E(SUCC) for postorder and, since it was not know exactly, we obtained above the following bounds on the postorder value of E(SUCC):

$$3\tfrac{3}{4}n + \frac{1}{2n} - 3h_n - \tfrac{1}{4} \leqslant nE(\text{SUCC}) \leqslant 4.19n + \frac{1}{2n} - 3h_n - 0.69.$$

Since we have that $h_n \leqslant (d_n/n!) \leqslant h_n + 1.318$ the following bounds hold for postorder E(TRAV):

$$3\tfrac{3}{4}n + \frac{1}{2n} - 2h_n + 1.07 \leqslant E(\text{TRAV}) \leqslant 4.19n + \frac{1}{2n} - 2h_n + 0.63,$$

and so asymptotically

$$E(\text{TRAV}) \approx 4.19n - 2h_n.$$

The performances of Algorithms 3.4 and 3.5, which respectively perform an inorder and postorder traversal of a tree via a stack, can easily be seen to be distribution-independent. Thus from Brinck and Foo we get that their respective expected performance measures are $2n-2$ and $4n-3$ operations.[1]

The analysis of Algorithm 3.6, which performs a

Algorithm 3.4. Inorder (stack) traversal

```
procedure inscan (p)
{
  clear (S)
  while p ≠ nil do
  {
    while p.lhs ≠ nil do
    {
      push (S, p); p ← p.lhs
    }
    label: visit (p)
    if p.rhs ≠ nil then p ← p.rhs
                   else
                   {
                     p ← pop (S)
                     if p ≠ nil goto label
                   }
  }
}
```

Algorithm 3.5. Postorder (stack) traversal

```
procedure postscan (p)
{
  clear (S); q ← p
  while p ≠ nil do
  {
    while p.lhs ≠ nil do {push (S, p); p ← p.lhs}
    while (p ≠ nil) and ((p.rhs = nil) or (p.rhs = q)) do
    {
      visit (p)
      q ← p
      p ← pop (S)
    }
    if p ≠ nil then {push (S, p); p ← p.rhs}
  }
}
```

preorder traversal via a stack, requires the expected number of nodes with two immediate descendants which, as seen above, is not distribution-independent. It is not difficult to see that the total preorder traversal time over all trees is equal to twice the number of nodes with non-null left subtree plus twice the number of nodes with two immediate descendants. Taking the values of these quantities over the permutation distribution from Table 1 we have that the expected traversal time per tree is giver by

$$2\frac{n-1}{2}+2\frac{n-2}{3}=\tfrac{5}{3}n-\tfrac{7}{3}.$$

Algorithm 3.6. Preorder (stack) traversal

```
procedure prescan (p)
{
  clear (S)
  while p ≠ nil do
  {
    while p.lhs ≠ nil do
    {
      visit (p)
      if p.rhs ≠ nil then push (S, p)
      p ← p.lhs
    }
    visit (p)
    if p.rhs = nil then p ← pop (S)
    p ← p.rhs
  }
}
```

That this is slightly larger than the corresponding average over the uniform distribution of $\frac{3}{2}-\frac{9}{4}$ is to be expected owing to the relatively larger number of more balanced trees in the permutation distribution.

5. CONCLUDING REMARKS

Table 2 summarises the results of the previous section and compares them with those derived for the uniform distribution in Ref. 1. Generally, the results are virtually identical to those derived for the uniform distribution for inorder and preorder traversals, while for postorder the uniform figure also lies with the bounds derived for permutation trees. That these results are so similar for the two distributions is surprising in view of the fact that for inorder and postorder traversals over the permutation distribution the respective values of E(START) do not converge to a constant as for the uniform distribution.

The analyses in this paper have all been concerned with the time domain, but it would also be of great interest to examine the expected maximum stack depth reached by the various algorithms that traverse via stacks. In its simplest form this is related to determining the average height of the trees on which the algorithms operate, and some asymptotic work has been presented in this area in Refs 6, 7 and 8. While it is also possible to make use of the properties of the individual algorithms as we have done above, the recurrence relations that result are two-dimensional and quite complex. It is easy to reduce them to one-dimensional functional equations, but difficult to get much further.

Table 2. Asymptotic traversal algorithm performances

	Stacks			Threads		
	Best	Worst	Average	Best	Worst	Average
Uniform distribution						
Preorder	n	$2n-1$	$\frac{3}{2}n-\frac{9}{4}$	$n+1$	$2n$	$\frac{1}{2}(3n+3)$
Inorder	n	$3n-1$	$2n-2$	$n+1$	$2n$	$\frac{1}{2}(3n+1)$
Postorder	$4n-3$	$4n-3$	$4n-3$	$2n$	$n(n+1)$	$4n-6$
Permutation distribution						
Preorder	n	$2n-1$	$\frac{5}{3}n-\frac{7}{3}$	$n+1$	$2n$	$\frac{1}{2}(3n+3)$
Inorder	n	$3n-1$	$2n-2$	$n+1$	$2n$	$\frac{1}{2}(3n+1)$
Postorder	$4n-3$	$4n-3$	$4n-3$	$2n$	$n(n+1)$	$4.19n-2h_n$

Acknowledgements

I would like to thank Professor N. Y. Foo for help and encouragement during this work, and Professor E. M. Reingold for the suggestion that permutation trees might bear fruitful investigation in this manner.

REFERENCES

1. K. Brinck and N. Y. Foo, Analysis of algorithms on threaded trees. *The Computer Journal* **24** (2), 148–155 (1981).
2. D. E. Knuth, *Fundamental Algorithms*, 2nd ed. Addison Wesley, Reading, Mass. (1975).
3. E. M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithms: Theory and Practice*. Prentice Hall, Englewood Cliffs, NJ. (1977).
4. K. Brinck, Analysis of algorithms on threaded trees and related structures. *Ph.D. Thesis*, University of Sydney (1982).
5. M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*. Dover Publications, New York (1980).
6. J. M. Robson, The height of binary search trees. *The Australian Computer Journal* **11** (4), 151–153 (1979).
7. P. Flajolet and A. Odlyzko, The average height of binary trees and other simple trees. *INRIA* Report No. 56, Rocquencourt, France (1981).
8. R. Kemp, On the stack size of regularly distributed binary trees, 6th ICALP Conference, Udine (1979).