Hairy Search Trees

C. H. A. KOSTER AND TH. P. VAN DER WEIDE

Department of Information Systems, University of Nijmegen, Toernooiveld, NL-6525, ED Nijmegen, The Netherlands
Email: {kees,tvdw}@cs.kun.nl

Random search trees have the property that their depth depends on the order in which they are built. They have to be *balanced* in order to obtain a more efficient storage-and-retrieval data structure. Balancing a search tree is time consuming. This explains the popularity of data structures which approximate a balanced tree but have lower amortized balancing costs, such as AVL trees, Fibonacci trees and 2–3 trees. The algorithms for maintaining these data structures efficiently are complex and hard to derive. This observation has led to insertion algorithms that perform *local balancing* around the newly inserted node, without backtracking on the search path. This is also called a fringe heuristic. The resulting class of trees is referred to as 1-locally balanced trees, in this note referred to as *hairy trees*. In this note a simple analysis of their behaviour is provided.

Received February 24 1995, revised August 16 1995

1. HAIRY TREES

Locally balanced search trees have been invented and analysed a long time ago [1, 2], but they have not become as popular as unbalanced search trees or AVL-trees. In this note, we show how to obtain a simple form of locally balanced trees and to analyse their behaviour. These *hairy trees* are a class of search trees, characterized by:

$$\texttt{is hairy}\ (t) \equiv$$
$$\forall_{\text{node}\, v \in T}[v \text{ has single son } s \Rightarrow s \text{ is leaf}]$$

The intuition behind this condition is that it prevents trees from having list-like substructures longer than two nodes ('bare twigs'). Some examples of hairy trees are presented in Figure 1.

The class of hairy trees can be described by the following recursive definition:

1. is hairy (ϵ)
2. If t is a singleton tree and x some key value, then:
   - is hairy (t)
   - is hairy (Tree (x, t, ϵ))
   - is hairy (Tree (x, ϵ, t)).
3. If t_1 and t_2 are both non-empty hairy trees and x is some key value, then is hairy(Tree(x, t_1, t_2)).

where ϵ is the empty tree, and Tree (x, t_1, t_2) the constructor of trees. We will also overload this constructor for singleton trees: Tree (x) = Tree (x, ϵ, ϵ). The above inductive definitions give us the opportunity to use structural induction in reasoning about hairy trees.

Definition 1. *The function* single *counts the number of single-son nodes in a tree:*

single $(\epsilon) = 0$
single (Tree (x, t_1, t_2)) =
  if $t_1 = \epsilon \wedge t_2 = \epsilon$ *then* 0
  elif $t_1 = \epsilon \vee t_2 = \epsilon$ *then* 1
  else single (t_1) + single (t_2)

Definition 2. *The number of leaves of a tree is defined by:*

leaves $(\epsilon) = 0$
leaves (Tree (x)) = 1
leaves (Tree (x, t_1, t_2)) =
  leaves (t_1) + leaves (t_2) *if* $t_1 \neq \epsilon \vee t_2 \neq \epsilon$

The following property is easily proved by structural induction:

LEMMA 1.

$$\texttt{is hairy}\ (t) \Rightarrow 0 \leqslant \texttt{single}\ (t) \leqslant \texttt{leaves}\ (t)$$

Definition 3. *The number of keys in a tree is defined by:*

nkeys $(\epsilon) = 0$
nkeys (Tree (x, t_1, t_2)) = 1 + nkeys (t_1) + nkeys (t_2)

Definition 4. *The number of external nodes of a tree is defined by:*

ext $(\epsilon) = 1$
ext (Tree (x, t_1, t_2)) = ext (t_1) + ext (t_2)

LEMMA 2.

$$\texttt{nkeys}\ (t) + 1 = \texttt{ext}\ (t) = \texttt{single}\ (t) + 2 \times \texttt{leaves}\ (t)$$

Our goal in introducing hairy trees is to reduce the ratio between the number of single-son nodes and the number of external nodes in a search tree. This ratio will be denoted as $\Delta(t)$.

LEMMA 3.

$$\texttt{is hairy}\ (t) \Rightarrow 0 \leqslant \Delta(t) \leqslant \tfrac{1}{3}$$

Both bounds are sharp. This lemma is easily proved using the two previous lemmas.

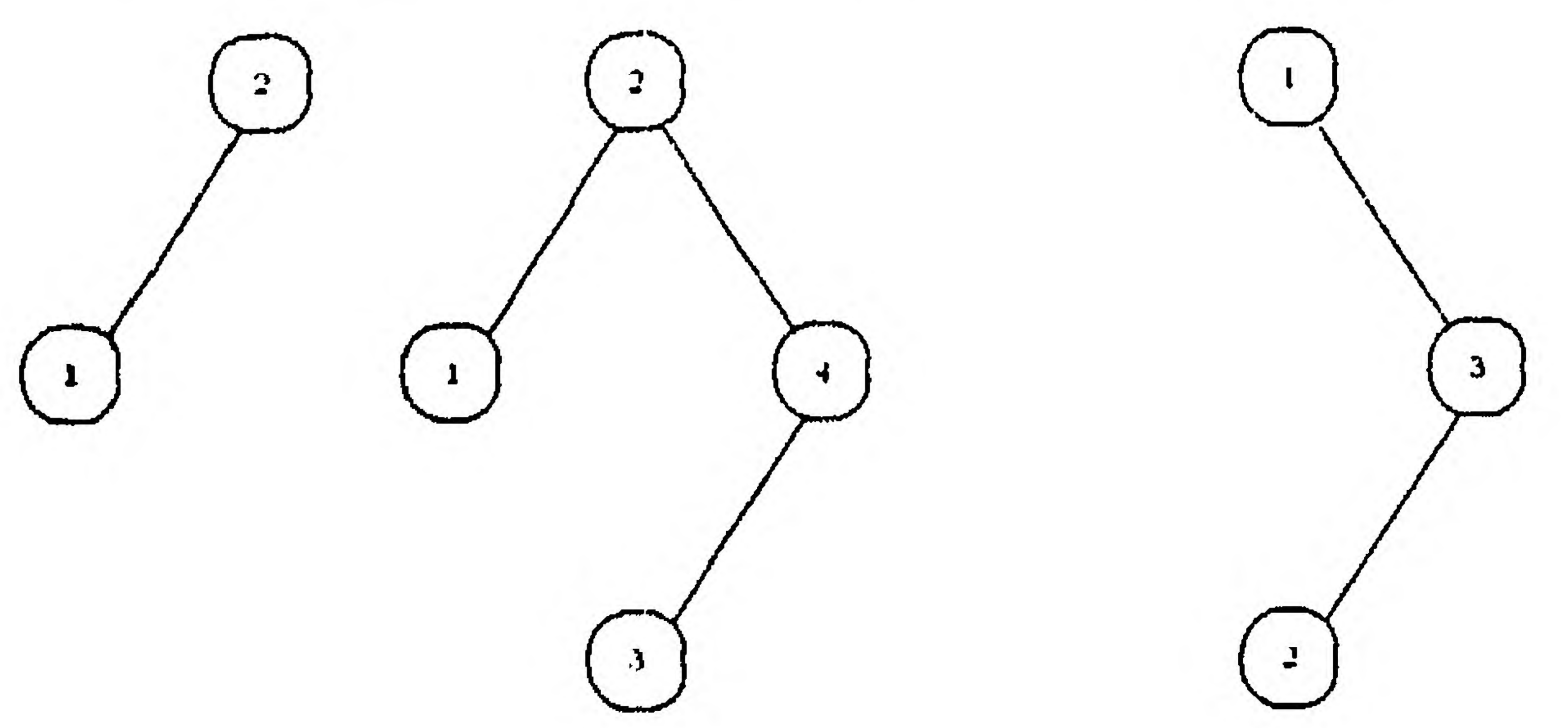

FIGURE 1. Two hairy trees and a non-hairy one.

In general, hairy trees are not balanced. In the worst case, a hairy tree of n elements has depth $\lceil (n+1)/2 \rceil$ (see Figure 2).

2. INSERTION IN HAIRY SEARCH TREES

We present[1] the operation `enterh` for inserting a key into a search tree, which maintains the search tree as a hairy tree by restructuring it whenever a node is inserted at the end of a twig. Its structure follows the case-distinction in the definition of `is hairy`.

```
PROC enterh (TREE VAR t, EL CONST e):
  { is hairy search tree (t) }
  IF   is empty (t) THEN t := tree (e)
  ELIFe  < t.key      THEN enter left
  ELIF t. key < e     THEN enter right
  FI
  {is hairy search tree (t), is in (e, t)}
ENDPROC enterh;
```

with the refinements:

```
enter left:
  IF is empty (t.left)
  THEN extend left
  ELIF is empty (t.right)
  THEN
    IF e < t.left.key
    THEN enter left left
    ELIF t.left.key < e
    THEN enter left right
    FI
  ELSE enterh (t.left, e)
  FI.

enter right:
  IF is empty (t.right)
  THEN extend right
  ELIF is empty (t.left)
  THEN
    IF e < t.right.key
    THEN enter right left
    ELIF t.right.key < e
    THEN enter right right
    FI
  ELSE enterh (t.right, e)
  FI.

extend left:
  t.left := tree (e).

enter left left:
  t : =tree (t.left.key, tree (e), tree (t.key)).

enter left right:
  t : =tree (e, t.left, tree (t.key)).

extend right:
  t.right := tree (e).

enter right right:
  t := tree (t.right.key, tree (t.key),
          tree(e)).

enter right left:
  t := tree (e, tree (t.key), t. right).
```

[1] The programming language used is Elan [3], an educational algorithmic language. (Obtainable from `ftp://ftp.cs.kun/nl/pub/elan`.)

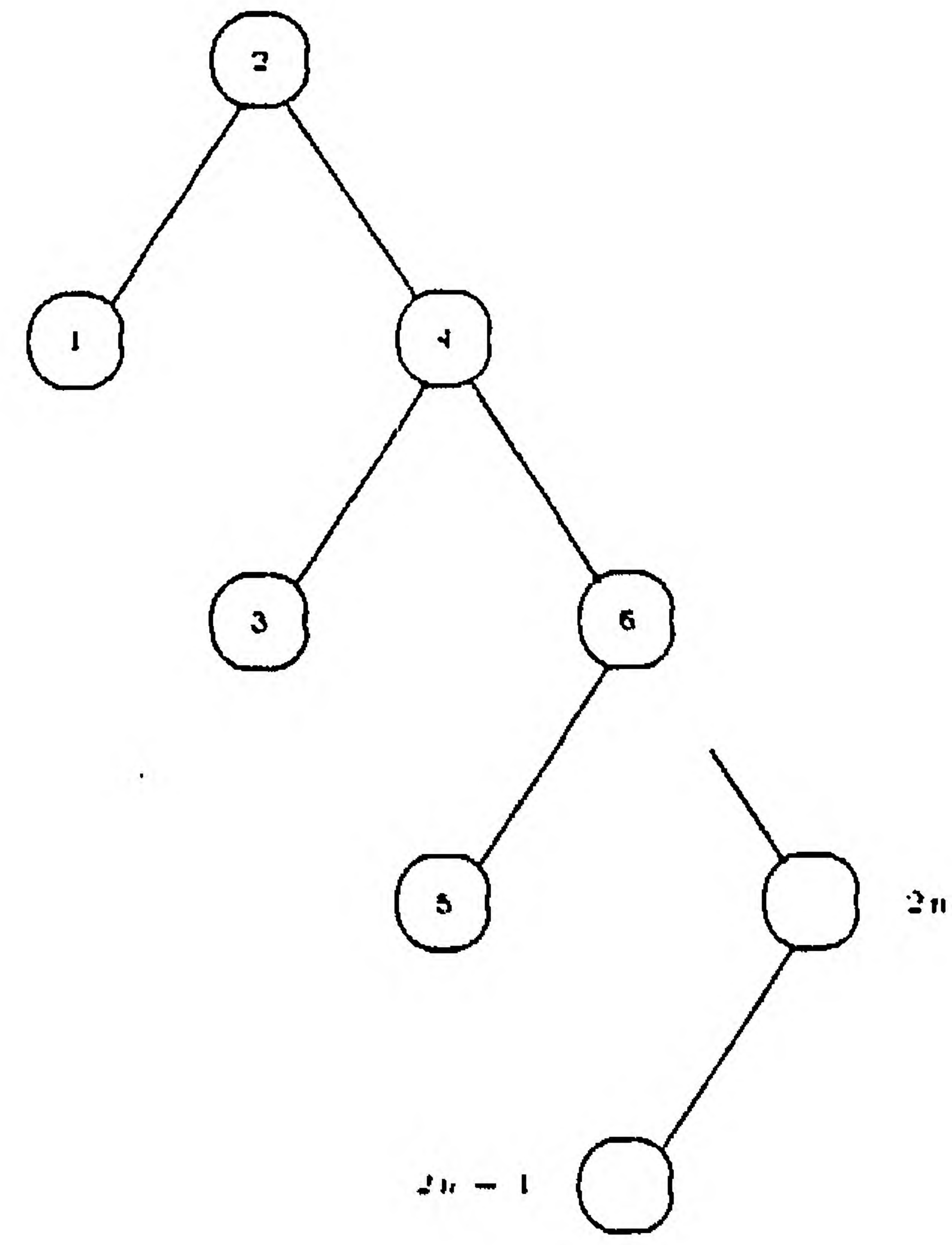

FIGURE 2. The worst hairy tree.

The correctness of this algorithm is easy to prove, since it closely follows the inductive structure of the definition of `is hairy`. The implementation may be further optimized by transformational techniques (unfolding, specialization and elimination of recursion).

3. EFFICIENCY OF HAIRY TREES

We analyse the efficiency of hairy trees in terms of the cost of a random successful search (S_n) in a tree with n keys, and the cost of a random unsuccessful search (U_n). Let I_n be the average internal path length of all hairy trees with n keys (see [4]), so $S_n = I_n/n$. Furthermore, let

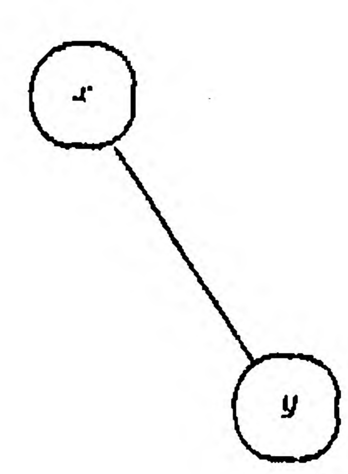

FIGURE 3. Addition via single-son node.

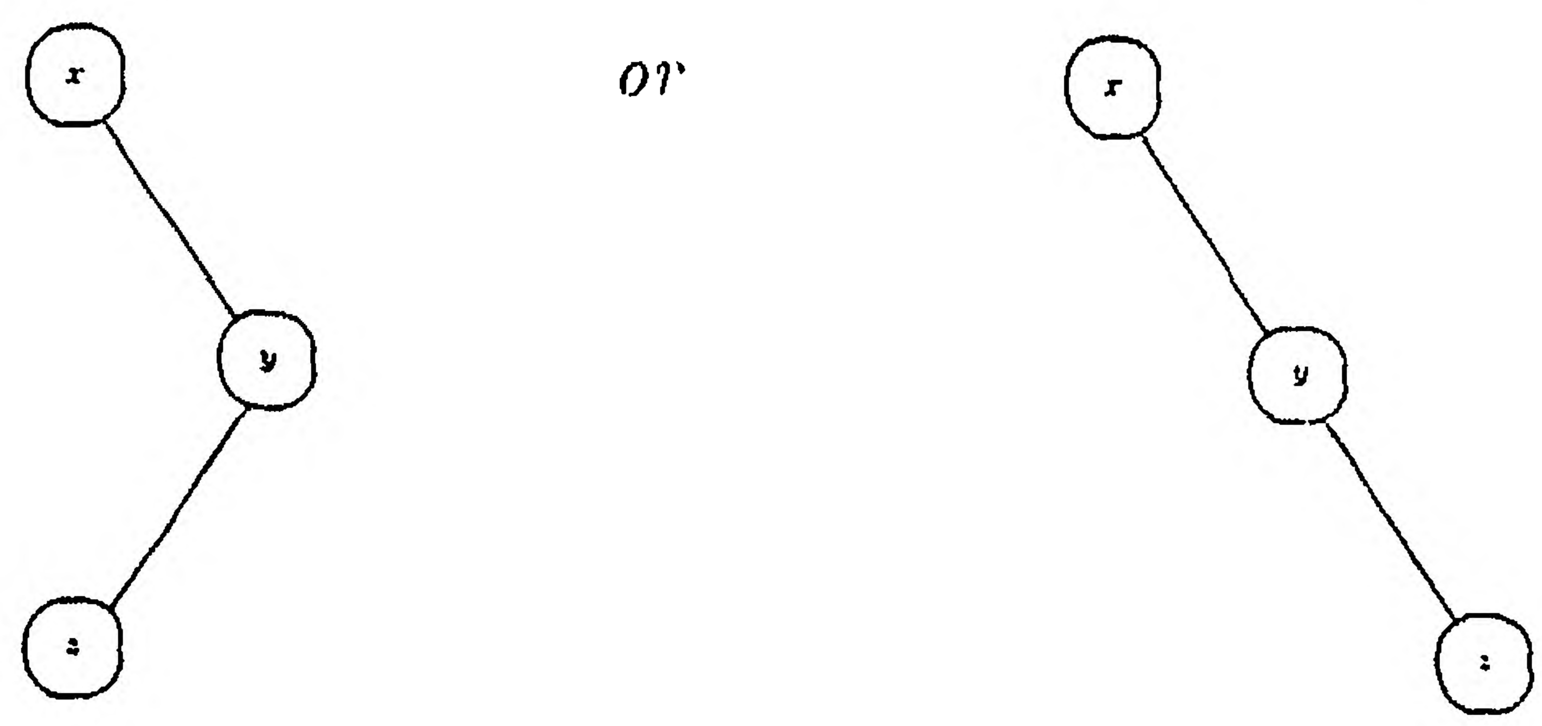

FIGURE 4. After addition.

Δ_n be the average ratio between the number of single-son nodes and the number of external nodes in hairy trees with n keys. Then we have:

$$I_{n+1} = I_n + (U_n + 1) - 2\Delta_n \qquad (1)$$

as obviously the internal path length is augmented with $U_n + 1$ by the insertion of a new key, and occasionally diminished by a restructing. A restructuring is performed if and only if we start from (up to symmetry) the situation of Figure 3, which is transformed by insertion of a node at its end into the one of the cases in Figure 4. After restructuring we have Figure 5.

In both cases the internal path length decreases by 1 as a result of restructuring. The probability of this situation to occur in tree t is:

$$\frac{2 \times \texttt{single}\ (t)}{\texttt{ext}\ (t)} = 2\Delta(t)$$

From equation (1) we derive:

$$I_n = \sum_{k=0}^{n-1} (U_k + 1 - 2\Delta_k) \qquad (2)$$

The following relation is well known:

$$S_n = \left(1 + \frac{1}{n} U_n\right) - 1$$

and can be rewritten as

$$I_n = (n+1)U_n - n \qquad (3)$$

Combining (2) and (3) yields

$$(n+1)U_n = \sum_{k=0}^{n-1} (U_k + 2 - 2\Delta_k)$$

This is transformed into a recurrence relation by computing $(n+1)U_n - nU_{n-1} = U_{n-1} + 2 - 2\Delta_{n-1}$, leading to:

$$U_n - U_{n-1} = \frac{2}{n+1}(1 - \Delta_{n-1})$$

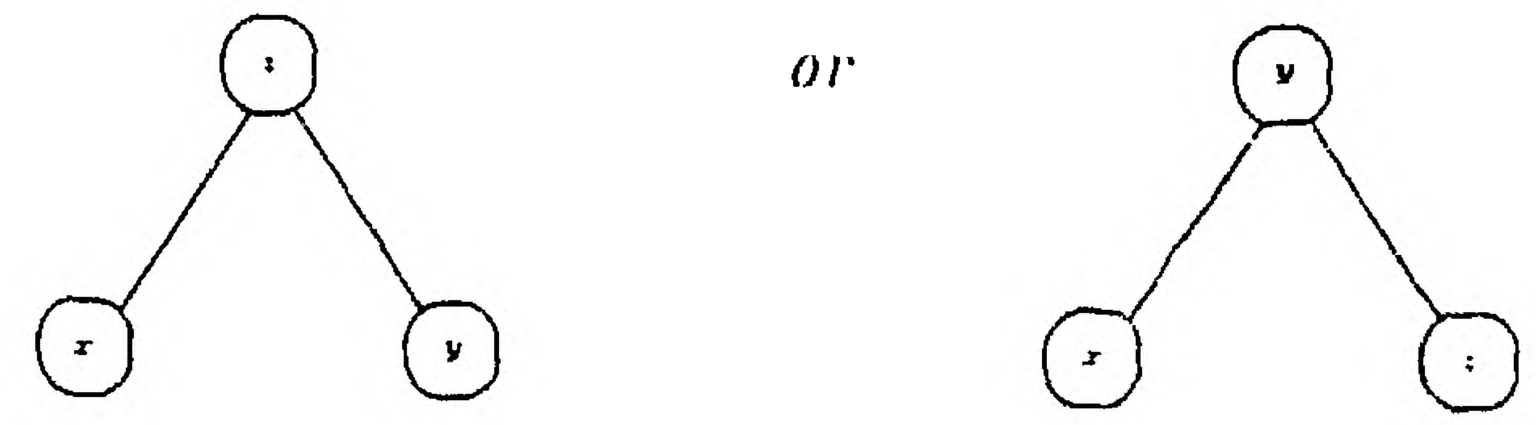

FIGURE 5. After restructuring.

and thus:

$$U_n = 2 \sum_{k=0}^{n-1} \frac{1}{k+2}(1 - \Delta_k)$$

Next we consider Δ_n. Let σ_n be the average number of single-son nodes in a hairy tree. When a new node is inserted via a search path through a single-son node, then the number of single-son nodes will be decremented by 1. As such a search path contains three external nodes, the probability of this event to occur equals $3\sigma_n/(n+1)$. If the search path does not contain a single-son node, then the number of single-son nodes will be incremented by 1. This event has a probability $1 - 3\sigma_n/(n+1)$. Combining these results leads to the following recurrence relation:

$$\sigma_{n+1} = \sigma_n - \frac{3\sigma_n}{n+1} + \frac{(n+1) - 3\sigma_n}{n+1}$$
$$= \frac{n-5}{n+1}\sigma_n + 1$$

From this recurrence relation we derive $\sigma_6 = 1$, and therefore $\sigma_n = (n+1)/7$ for $n > 6$. As $\Delta_n = \sigma_n/(n+1)$, we conclude:

$$\Delta_n = \tfrac{1}{7} \qquad \text{for } n > 6$$

LEMMA 4.

$$U_n = \tfrac{6}{7} U_n^{\sim} \approx 1.1883\ldots {}^2\log n$$

where $U_n^{\sim} = 2\sum_{k=0}^{n-1} 1/(k+2) \approx 1.3863\ldots {}^2\log n$ $-0.8456\ldots$ is the average cost of an unsuccessful search in a random binary tree. The result of this analysis is summarized in Table 1 (see [2, 4]).

4. CONCLUSIONS

The analysis of the complexity of hairy trees turns out to be particularly simple. Their efficiency lies about halfway between random search trees and AVL trees. Considering the simplicity of their implementation, it is surprising that this class of partially balance trees is not used widely in practice.

TABLE 1. Comparing methods

	Random search tree	*Hairy tree*	*AVL tree*	*Balanced tree*
Expected search time	$1.386\ldots{}^2\log(n)$	$1.188\ldots{}^2\log(n)$	$1.012\ldots{}^2\log(n)$	${}^2\log(n)$
Worst case depth	n	$\frac{n+1}{2}$	$1.440\ldots{}^2\log(n)$	${}^2\log(n)$

ACKNOWLEDGEMENTS

We wish to thank Jan van Leeuwen for his interest and advice. We also wish to thank the anonymous referees for their valuable comments which led to an improvement of the paper.

REFERENCES

[1] Poblete, P. V. and Munro, J. I. (1985) The analysis of a fringe heuristic for binary search trees. *J. Algorithms*, **6**, 335–350.

[2] Gonnet, G. H. (1983) *Handbook of Algorithms and Data Structures*. International Computer Science Services.

[3] Koster, C. H. A. (1987) *Top-Down Programming with Elan*. Ellis Horwood.

[4] Knuth, D. E. (1973) *The Art of Computer Programming. Volume 3: Fundamental Algorithms*. Addison-Wesley.

[5] Walker, A. and Wood, D. (1976) Locally balanced binary trees, *Comp. J.*, **19**, 322–325.