# Inorder traversal of splay trees

Colm Ó Dúnlaing*
*Mathematics, Trinity College, Dublin 2, Ireland*

**Abstract**

Splay trees, a form of self-adjusting binary tree, were introduced by Sleator and Tarjan in the early 1980s. Their main use is to store ordered lists. The idea is to keep the trees reasonably well balanced through a 'splay heuristic.' Sleator and Tarjan showed that if amortised rather than worst-case times are considered, splay trees are optimal. Splay trees have the advantage of simplicity: they are much easier to implement than 2-3 trees, AVL trees, or red-black trees.

What if one uses splaying during an inorder traversal of the tree? Sleator and Tarjans' analysis guarantees $O(n \log(n))$ overall cost. On the other hand, the cost of ordinary inorder traversal is linear, whether or not the tree is balanced.

We present some data which suggests that the traversal time is $O(n)$, and demonstrate an $O(n \log\log(n))$ upper bound. This upper bound is reached through a rather unusual unbounded-history recurrence and by using weaker bounds, such as the $O(n \log(n))$ bound, along the way.

## 1 Splay trees

When a binary tree is used to store ordered lists, it is the inorder (symmetric order) of the nodes which matters. A rotation (Figure 1) applied to a subtree at $y$ preserves inorder, and therefore rotations may be used freely.

A *splay tree* is a binary tree whose height is adjusted according to a certain *splay heuristic.* The heuristic is as follows: in order to perform some operation on a tree, lookup, split, join, or whatever, locate some significant node $x$ and bring it to the root by a series of splay steps.

The three kinds of splay step are called 'zig,' 'zigzig,' and 'zigzag.' A 'zig' step makes a child of the root the root by a single rotation. A 'zigzig' or 'zigzag' step moves $x$ two places closer to the root by two rotations. In the zigzig case, both $x$ and its parent are left- or right children; in the zigzag case, $x$ is a left child and its parent a right child or vice-versa.

When a node is brought to the root by splaying, all nodes along the path are brought closer to the root — their depth is halved.

As a result of these heuristics, the *amortised* cost (i.e., the cost averaged over a sequence of operations) of lookup, split, join, and so on, is optimal ($O(\log(n))$ per operation). These

*e-mail: odunlain@maths.tcd.ie. Mathematics department website: http://www.maths.tcd.ie. Some of this work was presented at the second Irish Conference on the Mathematical Foundations of Computer Science and Information Technology (MFCSIT '02), Galway, Ireland, July 2002.

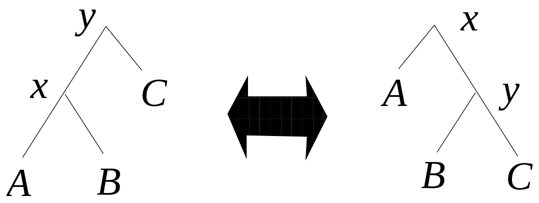


Figure 1: rotation, a reversible operation which preserves inorder.

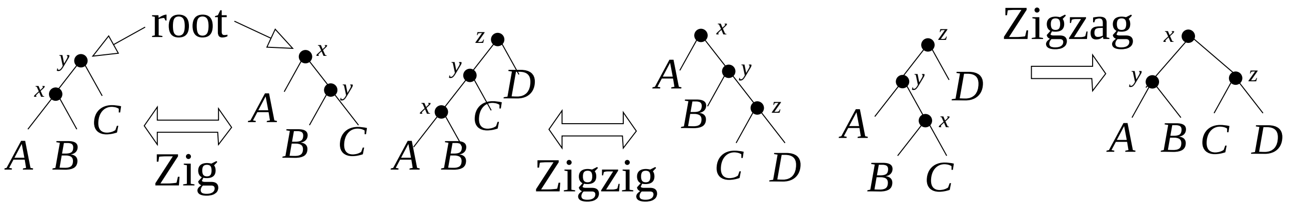


Figure 2: three splay steps.

operations are applied to an evolving forest of binary trees. If the operations begin with a forest of $n$ trivial trees then the total actual cost never exceeds the total amortised cost.

What if one uses the splay heuristic to traverse an entire tree in left-to-right order? Explicitly,

**(1.1) Definition** *A* traversal *of a splay tree involves accessing the nodes in inorder, through a sequence of* fetch *operations. In the first* fetch *operation the leftmost node is accessed and brought to the root by splaying. Subsequently, a* fetch *operation means accessing the inorder successor of the root, if it exists, and bringing it up to the root by splaying.*

Figure 4 illustrates inorder traversal of the complete tree of height 3 (15 nodes) by repeated splaying to root. The arrows are labelled with the number of rotations applied, which is proportional to the work done. The total number of rotations is 27 for this example.

**(1.2)** Computer implementations produced the table below, giving data about the cost of traversing complete trees of height $h$ and size $n = 2^{h+1} - 1$, for $h = 0$ to 23 (the limit of memory capacity), and 'leftmost trees' of the same size $n$ — trees in which right children do not exist. It is fairly clear that 'rightmost trees' in which left children don't exist are optimal, with overall traversal cost $n - 1$.

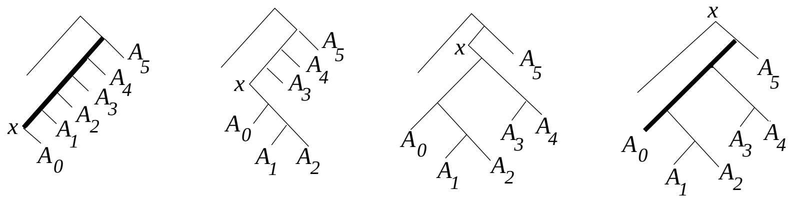


Figure 3: A fetch operation (1.1).

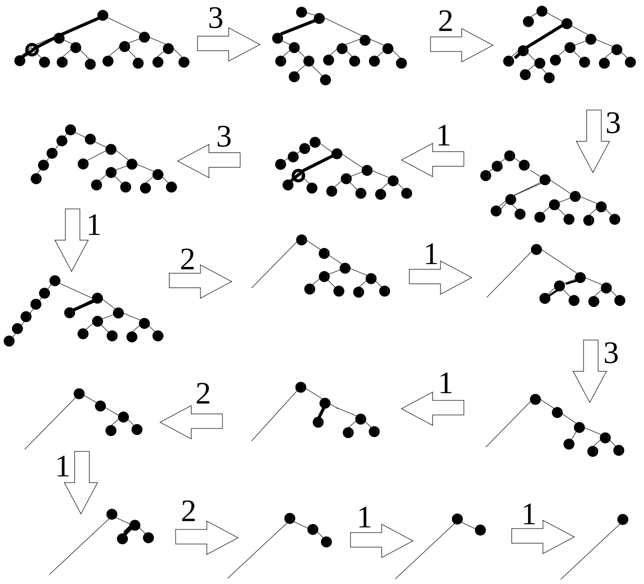


Figure 4: traversal of the complete tree of height 3. The spine, see (2.3) below, is shown with heavy lines.

| $h$ | $n$ | $K_h$ | $K_h - 2K_{h-1}$ | ratio | $L_n$ | ratio | $K_h - L_n$ $-2n$ |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | n/a | 0.000000 | 0 | 0.000000 | $-2$ |
| 1 | 3 | 3 | 3 | 1.000000 | 4 | 1.333333 | $-5$ |
| 2 | 7 | 10 | 4 | 1.428571 | 16 | 2.285714 | $-8$ |
| 3 | 15 | 27 | 7 | 1.800000 | 48 | 3.200000 | $-9$ |
| 4 | 31 | 62 | 8 | 2.000000 | 114 | 3.677419 | $-10$ |
| 5 | 63 | 135 | 11 | 2.142857 | 250 | 3.968254 | $-11$ |
| 6 | 127 | 282 | 12 | 2.220472 | 528 | 4.157480 | $-8$ |
| 7 | 255 | 589 | 25 | 2.309804 | 1090 | 4.274510 | $-9$ |
| 8 | 511 | 1204 | 26 | 2.356164 | 2216 | 4.336595 | $-10$ |
| 9 | 1023 | 2437 | 29 | 2.382209 | 4472 | 4.371457 | $-11$ |
| 10 | 2047 | 4904 | 30 | 2.395701 | 9000 | 4.396678 | 2 |
| 11 | 4095 | 9847 | 39 | 2.404640 | 18038 | 4.404884 | 1 |
| 12 | 8191 | 19734 | 40 | 2.409230 | 36116 | 4.409230 | 0 |
| 13 | 16383 | 39511 | 43 | 2.411707 | 72276 | 4.411646 | $-1$ |
| 14 | 32767 | 79066 | 44 | 2.412976 | 144598 | 4.412915 | $-2$ |
| 15 | 65535 | 158187 | 55 | 2.413779 | 289254 | 4.413733 | $-3$ |
| 16 | 131071 | 316430 | 56 | 2.414188 | 578568 | 4.414157 | $-4$ |
| 17 | 262143 | 632919 | 59 | 2.414404 | 1157200 | 4.414385 | $-5$ |
| 18 | 524287 | 1265898 | 60 | 2.414513 | 2314468 | 4.414506 | $-4$ |
| 19 | 1048575 | 2531865 | 69 | 2.414577 | 4629010 | 4.414572 | $-5$ |
| 20 | 2097151 | 5063800 | 70 | 2.414609 | 9258112 | 4.414614 | 10 |
| 21 | 4194303 | 10127673 | 73 | 2.414626 | 18516288 | 4.414628 | 9 |
| 22 | 8388607 | 20255420 | 74 | 2.414635 | 37032642 | 4.414635 | 8 |
| 23 | 16777215 | 40510931 | 91 | 2.414640 | 74065368 | 4.414640 | 7 |

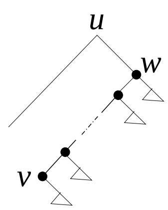


Figure 5: spine, all nodes from $v$ to $w$.

The table lists $h$, $n$, the cost $K_h$ of traversing the complete tree through splaying, a 'difference column' giving $K_h - 2K_{h-1}$, the ratio $K_h/n$, the cost $L_n$ of traversing the 'leftmost tree' through splaying, and the ratio $L_n/n$. The rightmost column in the table allows one to compare $L_n$ with $K_h$. The difference is very close to $2n$, which is plausible, since after about $\log_2(n)$ iterations traversing the 'leftmost tree,' overall cost about $2n$, the processed tree very closely resembles the complete tree of size $n$.

It is easy to explain the even-odd periodicity in $K_h - 2K_{h-1}$, though hardly worth the effort. This quantity shows regularities in $K_h$. However, we have not derived an explicit formula for $K_h$, and a query of the on-line version of Sloane's Handbook of Integer Sequences [2] did not produce any matches.

The ratio columns suggest $O(n)$ behaviour for these trees. Sleator and Tarjan's analysis [1] guarantees that the overall cost of traversal is $O(n\log(n))$ on a tree with $n$ nodes (Lemma 2.2). Considering that iteration on a non-self-adjusting tree takes overall time $O(n)$, this is disappointing, but a more specialised analysis leads to the following stronger result:

**(1.3) Theorem** *Suppose that a binary tree with $n$ nodes is traversed in inorder by repeated splaying as described above. Then the overall cost is $O(n\log\log(n))$.*

The remainder of this paper is devoted to a proof of this theorem.

# 2 The spine and spine blocks

**(2.1) The tree at 'time' $t$.** A binary tree $T$ of $n$ nodes is subjected to inorder traversal through a sequence of fetch operations which involve splaying to the root (see 1.1). Let $T(0) = T$, and in general for $0 \le t \le n$ let $T(t)$ be the tree as it is after $t$ fetch operations. $T(t)$ we call the *tree at time $t$*.

We begin with Sleator and Tarjans' analysis applied to the cost of traversing a binary tree.

**(2.2) Lemma** *The cost of traversing $T$ by repeated splaying is at most $5n\log_2(n)$.*

**Proof.** For $0 \le s \le n$ and any node $x$ in $T(s)$, define its *rank* $r_s(x)$ as

$$\log_2\left(\frac{\text{number of descendants of } x}{n}\right).$$

Thus $-\log_2(n) \le r_s(x) \le 0$ always. Define the potential $\Phi_s$ of $T(s)$ as the sum of node ranks in $T(s)$. Thus $-n\log_2(n) \le \Phi_s \le 0$.

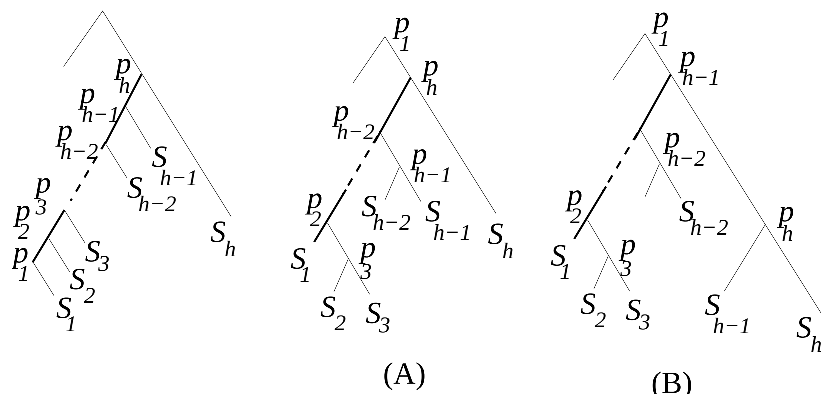


Figure 6: result of fetching a node $p_1$. Successive right subtrees $S_i$ and $S_{i+1}$ are combined. The effect is slightly different depending on whether the spine has (A) even or (B) odd cardinality.

The *amortised cost* of the $s$-th fetch operation, for $1 \leq s \leq n$, is

$$\text{cost of the fetch} \quad + \quad \Phi_s - \Phi_{s-1}$$

For $1 \leq s \leq n$ let $x_s$ be the $s$-th node in inorder, so it is the $s$-th node fetched, and let $x_0$ be the root of $T(0)$.

By an analysis due to Sleator and Tarjan [1], the amortised cost of the $s$-th fetch operation is at most $1 + 3r_s(x_s) - 3r_{s-1}(x_s) \leq 1 + 3\log(n)$. Hence the overall amortised cost is at most $n(1 + 3\log(n))$. Add $\Phi_0 - \Phi_n \leq n\log_2(n)$ to get the actual cost, i.e., at most $n + 4n\log_2(n) \leq 5n\log_2(n)$. Q.E.D.

**Left branches, spine, and spine block.** In any tree, a left branch is a nonempty sequence $v_1, v_2, \ldots v_\ell$ of nodes in the tree, where for $1 \leq p \leq \ell - 1$, $v_p$ is the left child of $v_{p+1}$. The branch is *maximal* if $v_1$ has no left child and $v_\ell$ is not a left child.

**(2.3) Definition** *For* $0 \leq t < n$ *the* spine at time $t$ *or* spine$(t)$ *is the maximal left branch containing the next node to be fetched. The spine at time n is empty.*

*A* spine block *is a pair* $(B, s)$ *where* $B$ *is a nonempty contiguous sequence of nodes in* spine$(s)$. *By abuse of notation* $B$ *itself may be called a spine block, with* $s$ *left implicit.*

**(2.4)** Equivalently, the spine at time $t$ is the maximal left branch in $T(t)$ ending at the root if $t = 0$ and the root's right child if $t > 0$. At any time, a node either has already been fetched, or is a spine node, or is descended from the right child of a spine node.

**(2.5) Lemma** *The total traversal cost is* $\sum_{s=0}^{n-1} |\text{spine}(s)|$. *(Trivial.)* ∎

**(2.6) Definition** (i) *Given a spine block* $(B, s)$, *and any* $t \geq s$, *let*

$$c(t) = \begin{cases} 0 \quad \textit{if} \quad |B \cap \text{spine}(t)| \leq 1 \\ |(B \cap \text{spine}(t))| - 2 \quad \textit{if} \quad |B \cap \text{spine}(t)| \geq 2 \end{cases}$$

(ii) $S(B) = \sum_{t=s}^{n-1} c(t)$, *and* (iii) $S(d) = \max\{S(B) : |B| \leq d\}$.

**(2.7)** In defining $S(d)$, the maximum is calculated over all possible blocks of length $\leq d$ in all possible trees of whatever size. For this reason, it is obvious that $S(d)$ is monotonically non-decreasing with $d$, but it is not obvious that $S(d)$ is well-defined (finite). In fact it will be shown that $S(d)$ is $O(d \log\log(d))$ (Lemma 6.1) and this implies that $S(d)$ is well-defined.

If the term $-2$ were omitted it would be impossible to relate $S(B)$ to $|B|$: $S(d)$ would be undefined or infinite.

# 3 Fragments of a spine block

**(3.1) Lemma** *Let $(B, s)$ be a spine block where $0 \leq s \leq n-1$. The $(s+1)$st fetch operation affects the nodes in $B$, and the spine, as follows.*

(i) *If $|B|$ is even then exactly $|B|/2$ nodes in $B$ remain on the spine and, except for the node fetched if that belongs to $B$, $|B|/2$ are 'pushed down' off the spine to become right children of spine nodes. They will later rejoin the spine.*

(ii) *If $|B|$ is odd then at least $|B|/2-1$ and at most $|B|/2+1$ nodes in $B$ remain on the spine (respectively, are pushed off the spine).*

(iii) *The leftmost node $x$ in* spine$(s)$ *is the node fetched and it leaves the spine forever.*

(iv) *If the node $x$ had a right child $z$ in the tree $T(s)$ (2.1) then the maximal left branch containing $z$ gets 'pulled up' onto the spine, forming a leftmost interval in* spine$(s+1)$. *If $x$ had no right child then no new nodes are added to the spine. (Proof omitted. See Figures 3 and 6.)* ∎

**(3.2) Corollary** *Let $x$ be a descendant of the right child of a spine node at time $r$, and let $y$ be its leftmost descendant at the time.* (i) *Then $y$ will remain as the leftmost descendant of $x$ until after $x$ and $y$ have together been pulled up onto the spine.*
(ii) *When they are pulled up onto the spine, the node fetched is the inorder predecessor of $y$.*

**Proof.** (i) Until that time, $x$ can acquire new ancestors but cannot acquire new descendants. (ii) This follows from the above lemma, part (iv). Q.E.D.

**(3.3) Corollary** *If $(B, s)$ is a block, then for every $t \geq s$, $B \cap$* spine$(t)$ *is either empty or is a block of* spine$(t)$.

**Proof.** Otherwise, for some $t > s$, spine$(t)$ contains three nodes $u, v, w$ in left-to-right order (not necessarily consecutive) where $u, w \in B$ but $v \notin B$.

Both $u$ and $w$ are in spine$(s)$, and $u$ precedes $w$ in inorder, so $u$ is to the left of $w$ in spine$(s)$. Let $y$ be the leftmost descendant of $v$ in $T(s)$ (2.1).

The node $v$ comes between $u$ and $w$ in inorder, and is not on spine$(s)$, so there exists a node $v'$ on spine$(s)$ between $u$ and $w$ such that $v$ is in the right subtree of $v'$. Possibly $u = v'$ but $v' \neq w$. Since $y$ is in the same subtree, $v'$ and hence $u$ precedes $y$ in inorder. See Figure 7.

But $v$ doesn't rejoin the spine until $v'$ has been fetched, and $u = v$ or $u$ precedes $v$ in inorder. Therefore, when $v$ rejoins the spine, $u$ has already been fetched, a contradiction. Q.E.D.

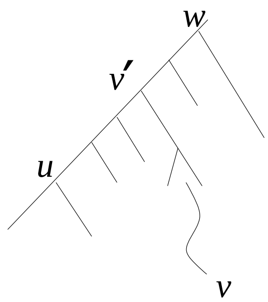


Figure 7: argument in Corollary 3.3.

**(3.4) Different generations.** Let $u$ be a node in a block $(B, s)$. It can be pushed off the spine and rejoin it many times. From the time $s$ until it is first pushed off the spine, it is 'first generation' (relative to the block $(B, s)$). Then it becomes second-generation, remaining so after it rejoins the spine, until it is again pushed off the spine, when it becomes third generation relative to $(B, s)$, and so on, until it is fetched.

**(3.5) Definition** *Given a block $(B, s)$, for any $t > s$, an* offline branch *is the intersection of $B$ with a maximal left branch in the right subtree of a spine node, providing that intersection is nonempty.*

**(3.6) Lemma** *An offline branch is a contiguous interval of nodes (in a left branch).*

**Proof.** A left branch $L$ will rejoin the spine when the inorder predecessor of its leftmost node is fetched at time $t$, say. If $L \cap B$ involved more than one contiguous interval, then $B \cap \text{spine}(t)$ would not be a block, contradicting Corollary 3.3. Q.E.D.

**(3.7) Definition** *A* child fragment *of $(B, s)$ is either an offline branch containing at least one second-generation node, or a spine block $(M, r)$ where $M$ was an (offline) child fragment at time $r - 1$, pulled up onto the spine at time $r$.*

*The* descendant fragments *of $(B, s)$ consist of $(B, s)$ itself, its child fragments $(M, r)$, and (recursively) their descendant fragments.*

**(3.8) Properties of child fragments.** A child fragment (offline) must coincide with an offline branch, but otherwise an offline branch (and $B \cap \text{spine}(t)$) can be the union of several descendant fragments.

Since a child fragment can contain higher-generation nodes, child fragments need not be disjoint.

By definition, if $(M, t)$ is a descendant of the fragment $(F, r)$, then $M \subseteq F$.

It is possible for all the nodes in a fragment $(M, r)$ to recur in a descendant fragment $(M, t)$:

**(3.9) Definition** *A fragment $(M, t)$ of $(B, s)$ is* exceptional *if it is a proper descendant of another fragment $(M, r)$ containing exactly the same nodes. If $|M| = 1$ (respectively, 2) then it is called an exceptional* single *(respectively,* double*) fragment.*

**(3.10) Lemma** *Let $(M, t)$ be exceptional. Then it is either a single or a double fragment, and its parent is another fragment $(M, p)$.*

**Proof.** Suppose $M$ contains three consecutive nodes $u, v, w$, and $M \subseteq \text{spine}(r)$ where $r < t$. The $(r+1)$-st fetch operation either makes $u$ the parent of $v$, and $v$ cannot rejoin the spine before $u$ is fetched, or makes $v$ the parent of $w$, and $w$ cannot rejoin the spine before $v$ is fetched. In any case $u, v, w$ cannot again occur together on the spine, and therefore there could be no descendant fragment $(M, t)$, and $M$ would not be exceptional.

Let $(F, p)$ be the parent of $(M, t)$. Since it is a descendant of $(M, r)$, $F \subseteq M$, and since $(M, t)$ is its child, $M \subseteq F$. Hence $M = F$ as asserted. Q.E.D.

# 4 There are $O(|B|)$ non-exceptional descendant fragments

**(4.1) Definition** *Let $T$ be a tree possibly containing some nodes from a set $B$. The $B$-*prefix *of $T$ is the tree formed from those nodes of $T$ which are in $B$, together with all the ancestors in $T$ of such nodes. It may be empty.*

*The $B$-*depth *or* prefix depth *of $T$ is the depth of its $B$-prefix.*

*$T$ is $B$-*balanced *if in its $B$-prefix, the left and right subtrees of any node have the same height. It is* right-favoured *(for $B$) if, in its $B$-prefix, the left subtree of any node is no higher than the right subtree.*

*Similarly, one can define 'left-favoured.'*

**(4.2)** When during a fetch operation two successive spine nodes $p$ and $q$ have right subtrees $A$ and $A'$, and after the operation these two subtrees are the left and right subtrees of $q$ which is the right child of $p$, we speak of the subtrees $A$ and $A'$ as being *combined.*

**(4.3) Lemma** *Let $(B, s)$ be a spine block, and suppose that after $r$ further fetch operations* $\text{spine}(s + r)$ *still contains at least one first-generation node from $B$. Let $p_1, \ldots, p_{\ell-1}, p_\ell, \ldots p_h$ be the nodes in $B \cap \text{spine}(s+r)$, where $p_\ell$ is the lowest first-generation node and $p_h$ the highest (possibly $\ell = 1$ or $\ell = h$). Finally, if $p_1$ is not the lowest spine node, let $p_0$ be the node below it, and let $b = 0$, otherwise let $b = 1$. Then*

(i) *The right subtree at $p_h$ is left-favoured of prefix depth $\leq r - 1$.* (ii) *The right subtrees at $p_\ell, \ldots, p_{h-1}$ are $B$-balanced of prefix depth $r - 1$.* (iii) *The left subtree at $p_\ell$ has prefix depth at most $r - 1$.* (iv) *The right subtrees of all nodes from $p_b$ to $p_{\ell-1}$ are right-favoured of strictly increasing prefix depth up to a maximum of $r - 2$.* (v) *All nodes in $B$ which are not yet fetched are among $p_b, \ldots, p_h$ and their right subtrees.*

**Proof.** (v) is proved by induction. Initially, $r = 0$, and $B$ consists of all the nodes $p_j$. Going from $r$ to $r + 1$, right subtrees of $p_i$ and $p_{i+1}$ are combined to form a right subtree of $p_i$, containing the same nodes from $B$. If $p_h$ a node above $p_h$ on the spine acquires a new right subtree, the extra nodes come from a higher right subtree and do not come from $B$. If $p_b$ remains on the spine, then no node below $p_b$ on the spine acquires descendants from $B$. If $p_b$ is pushed off the spine, then $p_{b+1}$ is not, and the node below $p_b$ may acquire descendants from $B$, but it will take on the role of $p_b$ at the next stage.

Note also that if $p_b$ is pushed off the spine, having been the lowest node either from $B$ or whose right subtree intersects $B$, then $p_b$ becomes part of a left branch, and either (a) $p_b$ remains a right child, (b) it becomes a left child of $p_{b+1}$, or (c) $h = b$. (c) is excluded since then no first-generation node remains on the spine. In any case, it is impossible that a node pushed off the spine, and outside $B$, but whose right subtree intersects $B$, become the left child of another node from outside $B$.

Finally suppose that $p_b$ is fetched. If $p_b$ had no right child then no new nodes are added to the spine. Otherwise, let $q$ be the right child of $p_b$. If the subtree at $q$ does not intersect $B$ then we are finished. Otherwise let $q'$ be the highest node on the left branch ending at $q$ such that $q' \in B$ or its right subtree intersects $B$. By the above remarks, if $q' \notin B$ then its new parent on the spine belongs to $B$, and we are done.

(i)–(ii) follow by induction on $r$, since when combining two trees $A$ and $A'$, (i) if $A$ is $B$-balanced of prefix depth $r-1$ and $A'$ is left-favoured of prefix depth $\leq r-1$ then the combined tree is left-favoured of prefix depth $r$. (ii) If $A$ and $A'$ are $B$-balanced of the same prefix depth $r-1$, then the combined tree is $B$-balanced of prefix depth $r$.

(iv) The left subtree at $p_\ell$ is altered in two ways, first by combining pairs of successive right subtrees from $p_b \ldots p_{\ell-1}$, and $p_\ell$ if it is pushed off the spine. The property that they are right-favoured with strictly increasing prefix depth is preserved by combining.

Apart from the effects of combining, the right subtree $D$ of the node fetched may contain nodes from $B$. In this case the node fetched was $p_b$ and $D$ is right-favoured, and its prefix depth $d$ is less than that of the right subtree of $p_{b+1}$. Suppose $q_k$ is the root of $D$ and $q_1, \ldots, q_k$ are the nodes along the left branch ending at $q_k$ whose right subtrees contain nodes from $B$. Then these right subtrees are right-favoured with strictly increasing prefix depth, and the prefix depth of $q_k$ is less than $d$. These are the nodes in $B$ which rejoin the spine. Thus (iv) is preserved. Since (iv) holds, and the highest right subtree below the lowest first-generation node (after the fetch) has prefix depth at most $r-1$, (iii) holds also. Q.E.D.

**(4.4) Remark.** In the above lemma, part (iv), after the fetch, the right subtree of $q_k$ has prefix depth $d-1$ and the right subtree of $p_{b+1}$, which is next to $q_k$, has prefix depth at least $d+1$. The right subtree of prefix depth $d$ is 'missing.'

**(4.5) The bottom tree.** The *bottom tree* is the $B$-prefix of the subtree rooted at the lowest first-generation node from $B$ on the spine.

Intuitively the child offline fragments should fit into a forest of complete trees of suitable height, but the difficulty is that some child fragments may have already been consumed, no longer in the bottom tree.

We view the complete tree of height $r$ from the left branch ending at its root, a sequence of nodes and right subtrees. Each of the $r+1$ nodes on this branch can be viewed as a slot in which one can fit a right subtree: so long as the right subtree fitted into place at the $j$-th slot has height at most $j$, the overall arrangement groups the subtrees into a prefix of the complete tree.

**(4.6) Definition** *An* $r$-arrangement of subtrees *is a sequence* $T_0, T_1, \ldots, T_{r-1}$ *of trees where the* $j$*-th has prefix depth at most* $j$.

$T_j$ *we call the subtree in the* $j$*-th* slot*, and if* $T_j$ *contains no nodes from* $B$ *we say that the* $j$*-th slot is* vacant. *(Slots are counted beginning at* 0.*)*

**(4.7) Lemma** *Under the condition of Lemma 4.3 all child fragments in the bottom tree, together with the child fragments which have been consumed up to time* $s+r$*, can be fitted into an* $r$*-arrangement.*

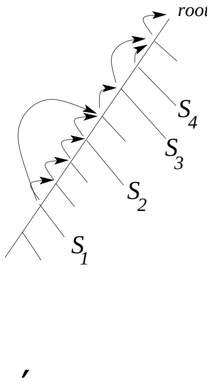


Figure 8: illustrating Lemma 4.7.

**Proof.** Induction on $r$, assuming that an $r$-arrangement exists for time $s+r$: it is converted into an $(r+1)$-arrangement at time $s+r+1$. The highest slot contains the right subtree of $q$, where $q$ is the lowest first-generation node in $B \cap \mathrm{spine}(s+r)$.

The bottom tree (at time $s+r$) is a prefix of the complete tree of height $r$. In the next fetch operation either (a) the right subtree at $q$ will be combined with a right subtree above it, and the resulting tree will be in the $r$-th slot, leaving the $(r-1)$-st vacant, (b) it will be combined with a right subtree below it, leaving the $(r-2)$-nd slot vacant, or (c) the node $q$ itself is fetched, its parent $q'$ becomes the root of the new bottom tree, and the right subtree of $q$, which is $B$-balanced of prefix depth $r-1$, becomes the left subtree of $q'$. The right subtree of $q'$ in $T(s+r+1)$ will occupy the $r$-th slot, and the $(r-1)$-st slot becomes vacant. Leave case (c) to the last.

Let $p$ be the node fetched at time $s+r+1$, where $p \neq q$. Let $A$ be the subtree at $p$, $C'$ and $D'$ its left and right subtrees, when $p$ last rejoined the spine — since $p \neq q$, $p$ is not first-generation. $A$ is right-favoured (4.3 (iv)), so the prefix depth of $C'$ is no greater than that of $D'$.

At time $s+r+1$ the right child $x$ of $p$ (if $x$ exists), and the left branch containing $x$, join the spine. Let $D$ be the right subtree of $p$, i.e., the subtree at $x$. $D$ contains $D'$ as a subtree, so its prefix depth is at least that of $D'$.

It follows from the remarks in (4.4) that the $d$-th slot is empty after the fetch, where $d$ is the prefix depth of $D$, and since $C'$ has prefix depth $\leq d$, it can be put in the $d$th slot. The subtree $C'$ is not part of the current tree, since all its nodes have been fetched. We consider it a 'virtual tree.'

We consider the slots, and right subtrees, in descending order down the spine. When two (real) subtrees $S$ and $S'$, in the $i$-th and $j$-th slots, $i < j$, are combined according to Lemma 4.3 (iv), the combined subtree fits into the $(j+1)$-st slot, and the $i$-th slot becomes vacant. It is necessary that the $(j+1)$-st slot have been made vacant.

We may assume by induction that there always exists a vacant slot above the slot being considered: the vacancy is left by the lower of two real subtrees which are combined, or exists initially in cases (a), (b), and (c).

If the $(j+1)$-st slot contained a real tree $S''$, then that tree has been combined with a higher and its slot became free. If it contained a virtual tree, then there exists a vacant slot above it. Let $v$ be the lowest vacant slot above it. For $j+1 \leq u \leq v-1$, move the tree in the $u$-th slot to the $(u+1)$-st. This leaves the $(j+1)$-st slot free, as desired. See Figure 8.

Finally we consider case (c). In this case, a new first-generation node $q'$ replaces $q$, and no right subtrees below $q$ are combined. All the virtual trees fitted into slots 0 to $r-2$ of the $r$-arrangement, so they (or rather, their $B$-prefixes) can be fitted into a complete tree of height $r-1$, which can occupy the vacant $(r-1)$-st slot in the $(r+1)$-arrangement. Q.E.D.

**(4.8) Definition** *For $r \geq 0$ an $r$-*group *is a sequence of numbers in descending order, bounded above by the corresponding terms in the sequence*

$$r+1, r, r-1, r-1, r-2, r-2, r-2, r-2, \ldots 1, 1$$

**(4.9) Explanation.** This sequence gives the sizes of all maximal left branches, in descending order, from the complete tree of height $r$. For $1 \leq j \leq r$, $j$ occurs $2^{r-j}$ times in the sequence, and $r+1$ occurs once. There are exactly as many terms as there are leaves in this tree — $2^r$ — and their sum is the number of nodes, $2^{r+1}-1$.

**(4.10) Corollary** *Assuming that after $r$ fetch operations the spine contains exactly one first-generation node $q$. Then $r \leq 1 + \lceil \log_2(|B|) \rceil$ (Lemma 5.1), and the child fragments of $B$ form an $r$-group.*

**Proof.** From Lemma 4.3 all the fragments are in the bottom tree, rooted at $q$. Lemma 4.7 shows that the bottom tree fragments and those child fragments which have already been processed can be fitted into a complete tree of depth $r$. Hence the lengths of all child fragments form an $r$-group. Q.E.D.

**(4.11) Corollary** *The total length of non-exceptional descendant fragments is $O(|B|)$.*

**Proof.** First, a weak bound. For any node $u$ in $B$, the non-exceptional fragments containing $u$ form a contracting sequence $B = S_1 \supseteq S_2 \ldots$ of distinct sets containing $u$. They are distinct because the fragments are non-exceptional (definition 3.9). Hence their sizes are decreasing and there are at most $|B|$ containing $u$, and. the total length of the non-exceptional fragments is at most $|B|^2$.

From the above Lemma, if $r = 1 + \lceil \log_2(|B|) \rceil$, then the lengths of the child fragments of $B$ are bounded above by the terms in an $r$-group. The total length of child fragments is bounded by the number of nodes in a complete tree of height $r$, that is, $2^{r+1}-1$. Ignoring the term $-1$, we study a sequence $f$ satisfying the recurrence

$$f(d) = 2^{r+1} + f(r+1) + 2f(r) + 4r(r-1) + 8f(r-2) \ldots$$

where $r = 1 + \lceil \log_2 d \rceil$, $f(0) = 0$ and $f(1) = 1$. Moreover, we can assume that $f(b) \leq b^2$ since $B$ has at most $|B|^2$ non-exceptional descendant fragments — the weak bound. Replace $r$ by $r+1$:

$$f(2d) = 2^{r+2} + f(r+2) + 2f(r+1) + 4r(r) + 8f(r-1) \ldots$$

Subtract twice the first from the second

$$f(2d) - 2f(d) = f(r+2) - f(r+1) \leq (r+2)^2.$$

Hence we consider $y_r = f(2^r)$ and study the recurrence

$$y_{r+1} - 2y_r = (r+2)^2$$

Substitute $z_r = y_r/2^r$ to get

$$2^{r+1}(z_{r+1} - z_r) = (r+2)^2, \quad \text{so} \quad z_{r+1} - z_r = \frac{(r+2)^2}{2^{r+1}}.$$

Hence $z_r \leq z_0 + \sum_{k=0}^{r-1}(k+2)^2/2^{k+1}$. This sum is bounded because the infinite series converges. Therefore $y_r$ is $O(2^r)$. Therefore $f(|B|)$ is $O(2^r)$, where $r \leq 1 + \lceil \log_2(|B|) \rceil$, so $B$ has $O(|B|)$ non-exceptional descendant fragments. Q.E.D.

**The constants involved.** The infinite series mentioned above is quite easily summed, though we have omitted the calculations for simplicity. One can show that there are at most $44|B|$ exceptional fragments.

# 5 There are $O(n(\log\log(n))^2)$ exceptional fragments

Recall that an exceptional fragment is one whose parent fragment contains the same nodes (3.9), and an exceptional fragment has one or two nodes (3.10). We begin by considering the generation count (3.4) of nodes in a block $B$.

**(5.1) Lemma** *Suppose that $(B, s)$ is a block. Then it takes at most $1 + \lceil \log_2(|B|) \rceil$ fetch operations to reduce the number of first-generation nodes in $B \cap \text{spine}(t)$ to exactly one.*

**Proof.** If $|B| > 2$ then a single fetch operation will reduce the size of its intersection with the spine to at most $1 + |B|/2$ (Lemma 3.1). Fetch operations may return nodes in $B$ to the spine, but they are not first-generation.

After $k$ fetch operations the number of first-generation nodes on the spine is reduced to at most $2 - (1/2)^{k-1} + |B|/2^k$. After $\lceil \log_2(|B|) \rceil$ fetches there are at most 2, and after one more, at most 1. Hence by that time, or earlier, the number of first-generation nodes on the spine has been reduced to exactly one. Q.E.D.

**(5.2) Lemma** *Suppose that $q$ is the only first-generation node from $B$ remaining on the spine at time $s + r$. Then all unfetched nodes $u \in B$ are descendants of $q$ (Lemma 4.3 (v)), and for every descendant $u \in B$, let $A$ be the set of all nodes on the path from $u$ to $q$ at that time, except those whose right children are on the path.*

*Then whenever $u \in \text{spine}(t)$, all nodes above $u$ in $B \cap \text{spine}(t)$ are from $A \cup \{q\}$.*

**Proof.** When a node rejoins the spine through a fetch operation, its ancestors (in $B$) after the fetch were ancestors before the fetch. For a node to acquire a new ancestor from $B$, it must be in the right subtree of a spine node $p$, and either $p$ acquires a new parent, or a new right child. In the first case $p$ becomes a right child and the parent is not counted. In the second case the new right child was the parent of $p$, already an ancestor of $u$, and its left child $p$ was on the path.

Finally, whenever $u$ rejoins the spine all its ancestors are from $A \cup \{q\}$, since those whose left children were on the path had to be fetched before $u$ could join the spine. Q.E.D.

**(5.3) Lemma** *Let $(B, s)$ be a spine block. Suppose that at time $s+r$ there remains exactly one first-generation node on $B \cap \text{spine}(s+r)$. Thenceforth, for any node $u \in B$ not yet fetched, if $u$ achieves a generation count of $O(\log\log(|B|))$ it will be the highest, or second-highest, node in $B \cap \text{spine}(t)$ whenever it is on the spine.*

**Proof.** By the above Lemma, $r \leq 1 + \lceil \log_2(|B|) \rceil$. Let $q$ be the first-generation node remaining on the spine.

Suppose $u = q$. Then it will be the highest node in $B \cap \text{spine}(t)$, whenever it is in $\text{spine}(t)$, until it is fetched.

Let $u$ be any other node and let $A$ be the set of ancestors of $u$ in $B$, the earliest time $r' \geq s + r$ when $u$ is on the spine. By Lemma 5.2, $|A| \leq 1 + r$. Treating $(A, r')$ as a block, we can apply the above lemma: within $f \leq 1 + \lceil \log_2(1 + r) \rceil$ fetches, all but one node from $A$ have been pushed off the spine. But by that time $u$ can have acquired at most $f$ ancestors off the spine. Let $F$ be its ancestors from $B$ at the time; according to Lemma 5.2, thereafter all its ancestors in $B$ will come from $F$.

By the reasoning in Lemma 3.10, if $F_1$ and $F_2$ are the ancestors of $u$ in $B$ at two successive times when $u$ rejoins the spine, and $|F_1| > 2$, then $|F_2| < |F_1|$. Therefore within at most $|F|$ fetches, $u$ will be at the top of $B$ or second from top whenever it is on the spine. This gives a total of $2f$ fetches until that time, or $O(\log\log(|B|))$. Q.E.D.

**(5.4) Corollary** *Given a block $(B, s)$, there are $O(|B| \log\log(|B|))$ exceptional fragments which are not at the top of $B \cap \text{spine}(t)$ or next to the top.*

**Proof.** Let $u$ be the unique, or the higher, node in occurring in an exceptional fragment $S$ (Lemma 3.10).

Suppose as in the above Lemma that after $r$ further fetches exactly one first-generation node remains on the spine. Within this time, whenever an offline branch (Definition 3.5) rejoins the spine, its length is less than $r$, so the overall number of times a node $u$ can rejoin the spine is less than $r^2$, i.e., $O((\log(|B|))^2)$ (Lemma 5.1), which is $O(|B| \log\log(|B|))$..

The number of times $u$ rejoins the spine after time $s + r$, until it reaches the top or second from top, is $O(\log\log(|B|))$ from the above lemma. This gives $O(|B| \log\log(|B|))$ recurrences overall,

The same applies to the exceptional fragment containing $u$. Q.E.D.

**(5.5) Lemma** *Given $n$ variables $x_0, \ldots, x_{n-1}$, and constant $M$, consider the problem of maximising $\sum_i (\log_2(x))^2$ subject to $x_i \geq 1$ and $\sum x_i = M$. Then if $M > (4.9215536348)n$, the maximum is $n(\log_2(M/n))^2$.*

**Proof.** Make the $n$-th variable $x_{n-1}$ dependent and leave the others as independent, so $x_{n-1} = M - \sum_{i=0}^{n-2} x_i$. Let $f(x_0, \ldots, x_{n-2}) = \sum_{i=0}^{n-1} (\ln(x_i))^2$. Since this is directly proportional to $\sum (\log_2(x_i))^2$, it is enough to maximise $f$, subject to $x_i \geq 1$ $(i \leq n-2)$ and $\sum_{i=0}^{n-2} x_i \leq M-1$.

Since $x_{n-1}$ depends on the other variables, $\partial x_{n-1}/\partial x_i = -1$ for $i \leq n-2$. Looking for local maxima, we equate partial derivatives to zero.

$$\frac{\partial f}{\partial x_i} \quad = \quad 2\frac{\ln(x_i)}{x_i} - 2\frac{\ln(x_{n-1})}{x_{n-1}} \quad = \quad 0$$

One solution is that all the $x_i$ are equal, to $M/n$, and $f$ is $n(\ln(M/n))^2$ in this case.

However, the function $\ln(x)/x$ is not monotonic. It has a single stationary point at $x = e$ (the maximum), is zero at $x = 1$, and the $x$-axis is a horizontal asymptote. For any positive $h < 1/e$, the graph has two points at height $h$. Thus in general, if the partial derivatives are all zero, let $x = x_{n-1}$, and let $y$ be the other point such that $\ln(x)/x = \ln(y)/y$. Then some of the $x_i$ will equal $x$ and some others will equal $y$. There exist integers $r \geq 1$ and $s \geq 0$ with $r + s = n$ such that $f(x_0, \ldots, x_{n-2}) = r\ln(x)^2 + s\ln(y)^2$, and $rx + sy = M$. Let us consider the possibility that $r \neq n$, so $s \geq 1$ and $y = (M - rx)/s$

Thus we should consider the behaviour of the function

$$r(\ln(x))^2 + s(\ln(y))^2 \qquad (*)$$

where $r, s \geq 1$, $r + s = n$, and $y = (M - rx)/s$. Therefore $dy/dx = -r/s$. Differentiating with respect to $x$ we get

$$2r\left(\frac{\ln(x)}{x} - \frac{\ln(y)}{y}\right)$$

This derivative is negative at $x = 1$ and positive at $y = 1$.

Differentiating again we get

$$2\frac{r}{s}\left(s\frac{1-\ln(x)}{x^2} + r\frac{1-\ln(y)}{y^2}\right)$$

The function $(1 - \ln(x))/x^2$ crosses the $x$-axis once at $x = e$, from positive to negative. The function $(1 - \ln(y)/y^2$, with $y = (M - rx)/s$, crosses the $x$-axis from above at $y = e$, $x = (M - se)/r$. As a function of $x$ it crosses the $x$-axis from below at this point. In order for them to add to zero, one must be positive and the other negative. This can happen at most twice. In other words: the function $(*)$ has at most two points of inflection, so it has at most three stationary points. One of them is given by $x = y = M/n$.

At this point the second derivative is $2n(r/s)(1 - \ln(M/n))/(M/n)^2$, which is negative if $M/n > e$. Since $M/n > 4.9215536348 > e$, this is a local maximum. Since at $x = 1$ the derivative is negative, the leftmost stationary point is a local minimum. Since at $y = 1$ (i.e., $x = (M - s)/r$) the derivative is positive, the rightmost stationary point is a local maximum. Therefore the only local maximum is where $x = M/n$.

This gives three candidates for the maximum: at $x = 1$, $s(\ln((M - r)/s))^2$, at $x = y$, $n(\ln(M/n))^2$, and at $y = 1$, $r(\ln((M - s)/r))^2$.

Returning to the original problem, either the maximum is at the interior point $x_i = M/n$, $0 \leq i \leq n - 1$, or it is on the boundary of the feasible region. Suppose that $r$ of the $x_i$'s are 1, but not the remaining $x_i$'s, at the true maximum.

By symmetry we can assume that the first $r$ variables and only those are 1 at the true maximum. We are therefore considering the function

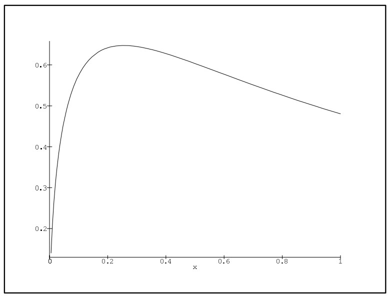


Figure 9: graph of $g(x) = x(\ln((1+x)/x))^2$.

$$\sum_{i=r}^{n-1} (\ln(x_i))^2$$

subject to $\sum_{i=r}^{n-1} x_i = M - r$. This is the original problem with fewer degrees of freedom, and the local-maximum solution for this problem is a true maximum. The true maximum must therefore be $(n-r)(\ln(M-r)/(n-r)))^2$ for some $r$, $0 \le r \le n-1$.

We claim that if $M > (4.9215536348)n$ then the maximum is given by $r = 0$. To see this, we consider the function $g(x) = x(\ln((1+x)/x))^2$. At $x = (n-r)/(M-n)$, $(M-n)g(x) = (n-r)(\ln((M-r)/(n-r)))^2$. See Figure 9.

$$\frac{dg}{dx} = \left(\ln\left(\frac{1+x}{x}\right)\right)^2 + 2x\ln\left(\frac{1+x}{x}\right)\left(\frac{x}{1+x}\right)\left(\frac{-1}{x^2}\right) = \ln\left(\frac{1+x}{x}\right)h(x),$$

where $h(x) = \ln((1+x)/x) - 2/(1+x)$. The zeroes of $h$ give the stationary points of $g$. Since

$$\frac{dh}{dx} = \left(\frac{x}{1+x}\right)\left(\frac{-1}{x^2}\right) + \frac{2}{(1+x)^2} = \frac{1}{1+x}\left(\frac{2}{1+x} - \frac{1}{x}\right)$$

$h$ has a stationary point only at $x = 1$, so it is monotonic for $0 < x < 1$ and can cross the $x$-axis at most once. Therefore $g$ has at most one stationary point. Also, $h(1/20) = (20/21)(\ln(21) - 40/21) > 0$, and $h(1) = (1/2)(\ln(2) - 1) < 0$. Therefore $h$, and $dg/dx$, crosses the $x$-axis from positive to negative at exactly one point, a local maximum for $g$. This maximum occurs at $x = 0.2550009749$.[1] The constraint on $M$ and $n$ ensures that the maximum is to the right of $n/(M-n)$, so $g(x)$ is monotonically increasing for $0 \le x \le n/(M-n)$, and hence $n(\log_2(M/n))^2$ is maximal. Q.E.D.

**(5.6) Lemma** *Suppose that the overall traversal time is bounded by $nf(n)$, where $f(n) > 4.9215536348$. Then there are $O(n(\log\log(n) + (\log(f(n)))^2)$ exceptional fragments overall.*

**Proof.** Much the same as in Lemma 5.3. A node $u$ is first introduced to the spine at time $s$, say. Let $b_s$ be the total number of nodes on spine($s$): $b_n = 0$. By the above arguments, either (i) $u$ is fetched before the nodes above it have been reduced to 1, or (ii) it rejoins the

[1] This was calculated using Maple.

spine at most $\log_2 \log_2(b_s)$ times until it reaches the top of the spine, or becomes second from top, and remains so every time it rejoins the spine, until it is fetched.

Thus overall, under case (ii), for each node $u$, there are $\log\log(n)$ events until it reaches the top or second from top. The top few nodes on the spine contribute $O(n)$ to the estimate.

It remains to account for case (i). The overall cost is $O(\sum_s(\log(b_s)^2))$. Since $\sum b_s \leq nf(n)$, by maximising the sum according to Lemma 5.5 we get $O(n(\log(f(n)))^2)$. Hence the overall count is $O(n(\log\log(n) + (\log(f(n)))^2))$. Q.E.D.

**(5.7) Corollary** *There are $O(n(\log\log(n))^2)$ exceptional fragments overall.*

**Proof.** From Lemma 2.2 the total spine length (i.e., the total traversal cost) is $\leq nf(n) = n(5\log_2(n))$. Q.E.D.

# 6 Proof of Theorem 1.3

Recall that $S(B)$ is defined as the sum of $|B \cap \text{spine}(t)| - 2$, where $t$ is any time $\geq s$ ($(B, s)$ is the spine block in question) such that the intersection contains more than one node. Also, $S(d)$ is the maximum of $S(B)$ for all spine blocks $B$ of size $\leq d$, in all possible trees. It has yet to be verified that $S(d)$ is well-defined (finite).

**(6.1) Lemma** *$S(d)$ is $O(d\log\log(d))$.*

**Proof.** Let $(B, s)$ be a block with $|B| \leq d$: we want to show that $S(B)$ is $O(|B|)$. We consider the $O(|B|)$ non-exceptional descendant fragments, together with those $O(|B|\log\log(|B|)$ exceptional fragments which do not contain the highest nor second-highest node in $B \cap \text{spine}(t)$ (Corollary 5.4).

If at time $t$, a fragment $F$ contains the highest node in $B \cap \text{spine}(t)$, or the second from highest, we say it is 'too high.'

(a) Consider a fragment $(F, r)$ where $|F| \geq 2$. So long as $|F \cap \text{spine}(t)| \geq 3$, a single fetch operation will reduce its intersection with the spine to at most $1 + |F|/2 \leq 2|F|/3$ (Lemma 3.1) Thus the contribution to $S(B)$ is at most $|F|\sum(2/3)^i < 3|F|$. One more fetch will reduce the intersection to 1, so the overall contribution is at most $3|F| + 2$, until $|F \cap \text{spine}(t)| \leq 1$.

Since the total length of all non-exceptional descendant fragments is $O(|B|)$, the overall contribution of all non-exceptional fragments $F$, until their intersection with the spine is at most one node, is $O(|B|)$. This estimate counts the contribution of all such fragments $F$, without the $-2$ term specified for $S(B)$.

(b) When a double exceptional fragment, one containing two nodes, joins the spine, the next fetch operation moves one of these nodes off the spine, and one remains. This contribution from these fragments (unless they are too high) is $O(|B|\log\log(|B|)$.

(c) If $E$ is an exceptional fragment, or a fragment with just one node remaining on the spine, directly above or directly below a non-exceptional fragment $F$ at time $t$, with $|F \cap \text{spine}(t)| \geq 2$, then the contribution of $E$ to $S(B)$ (at most 2, Lemma 3.10) can be absorbed in that of $F$, by quadrupling the contribution of $F$ in (a). The contribution is still $O(|B|)$ overall.

(d) Likewise if $E$ is directly above or directly below a double exceptional fragment $F$, not too high, with both nodes on the spine, then the contribution of $E$ can be accounted for by quadrupling that of $F$ in (b), $O(|B|\log\log(|B|))$ overall.

(e) The contribution of the top two nodes in $B \cap \text{spine}(t)$ can be ignored because of the $-2$ term in the definition of $S(B)$.

(f) All that remains to be considered is when we have a fragment $F$ intersecting the spine in just one node, not too high, and not next to a fragment coming under case (a) or (b). If it is adjacent to another fragment $G$ which is not too high, then $G$ must also intersect the spine in just one node. The next fetch operation will move either $F$ or $G$ off the spine, so their joint contribution at time $t$ can be accounted for by allocating 2 units to every fragment of $B$.

Otherwise, $F$ is lowest in $B \cap \text{spine}(t)$, and not too high, so there are at least two other nodes in the intersection, and the fragment $G$ above $F$ must be too high. The next fetch operation will either move $F$ off the spine, an event which can be charged to $F$ ($O(|B| \log\log(|B|))$), or remove a node in $G$, and the node $u$ in $F$ will become too high, an event which can be charged to the node $u$ ($O(|B|)$). Q.E.D.

It follows that $S(d)$ is a well-defined function.

**(6.2)** From now on, $B_s$ will be the set of nodes on spine($s$) not ever previously on the spine.

**(6.3) Lemma** *$B_s$ is a contiguous sequence of nodes leftmost on the spine.*

**Proof.** True for $s = 0$, when $B_s$ consists of all nodes on the spine. Thereafter, $B_s$ must consist of nodes added to the spine, contained in the maximal left branch $L$ ending at the right child of the node fetched. If $v$ and $w$ are nodes in $L$ and $v$ is left of $w$ and $v$ was on the spine previously, then so was $w$, because it must have been pushed off the spine to become a right ancestor of $v$. Therefore if $L$ contains a node not previously on the spine, let $u$ he the rightmost: $B_s$ then consists of $u$ and all nodes left of $u$. Q.E.D.

In other words, either $B_s$ is empty or $(B_s, s)$ is a block in spine($s$). Clearly the sets $B_s$ are disjoint and their union is the entire set of nodes in $T$.

From now on, we have a division of the nodes of $T$ into blocks $(B_s, s)$ with disjoint sets $B_s$ (ignore those $s$ for which $B_s = \emptyset$, since the empty set is not regarded as a block). Having fixed the list of blocks $B_s$, we now mean by *exceptional fragment* an exceptional fragment of one of these blocks $(B_s, s)$.

**(6.4) Lemma** *Suppose that $nf(n)$ is an upper bound for the number of exceptional fragments. Then the overall cost of traversal is $O(n(\log\log(n) + f(n)))$.*

**Proof.** The overall cost of traversal is $\sum_{t=0}^{n-1} |\text{spine}(t)|$.

(a) The sum $\sum_{s=0}^{n-1} S(B_s)$, which is $O(n \log\log(n))$ (Lemma 6.1), accounts for the total cost except for the top two nodes in $B_s \cap \text{spine}(t)$ for each $t$, or where the intersection is just a single node.

(b) We can compensate very crudely for the two nodes discounted in $|B_s \cap \text{spine}(t)| - 2$, if the intersection contains more than two nodes, by adding $2S(|B_s|)$ to the estimate.

(c) Where $B \cap \text{spine}(t)$ contains two nodes, then the intersection consists of one or two fragments. If one, it is reduced; if two, one of them is removed from the spine. The contribution to the traversal cost can be charged to these fragments.

(d) Where $B \cap \text{spine}(t)$ contains exactly one node $u$, it comes from a single fragment $F$. If it is adjacent to another node which has been counted under (a–c), its contribution can be absorbed by trebling the estimate for (a–c). Otherwise either $u$ is the only node on the spine, an event occurring at most $n$ times, or there is an adjacent node $v$, the only node remaining on the spine from another fragment $G$. The next fetch operation removes $u$ or $v$ from the spine, so the cost can be charged to $F$ or to $G$.

Hence the overall traversal cost is $O(n(f(n) + \log\log(n)))$. Q.E.D.

**(6.5) Corollary** *The overall traversal cost is* $O(n(\log\log(n))^2)$.

**Proof.** According to Corollary 5.7 there are $O(n(\log\log(n))^2)$ exceptional fragments. Q.E.D.

The proof of our main theorem comes from the above, using a kind of bootstrapping argument.

**Proof of Theorem 1.3.**

From the above Corollary, the overall traversal time is $O(n(\log\log(n))^2)$.

From Lemma 5.6 it follows that there are $O(n(\log\log(n) + \log((\log\log(n))^2)$ exceptional fragments, which is $O(n\log\log(n))$. Therefore the overall traversal cost is $O(n\log\log(n))$. Q.E.D.

# 7 Conclusions

We have established that traversal costs $O(n\log\log(n))$ by a very laborious analysis. Our analysis shows that if exceptional fragments are ignored, the overall cost is $O(n)$, and that there are $O(n\log\log(n))$ exceptional fragments.

It seems unlikely that the $O(n\log\log(n))$ bound is tight. Despite the experimental data which suggests an $O(n)$ bound for some trees, or even for most trees, we cannot be confident that the bound is $O(n)$.

It remains to find a linear upper bound or a nonlinear lower bound, or at least to strengthen Lemma 5.3.

# 8 Acknowledgements

The author is grateful to Marc van Dongen for useful insights and suggestions.

# 9 References

1. Daniel Sleator and Robert E. Tarjan (1985). Self-adjusting binary search trees. *JACM* **32:3**, 652–686

2. N.J.A. Sloane (1973). *A Handbook of Integer Sequences.* Academic Press.

3. Robert E. Tarjan (1983). *Data structures and network algorithms.* CBMS-NSF regional conference series in applied mathematics no. 44. Society for industrial and applied mathematics.