COMBINATORICA
Akadémiai Kiadó – Springer-Verlag

COMBINATORICA 12 (1) (1992) 95–124

ON THE DEQUE CONJECTURE FOR THE SPLAY ALGORITHM*

RAJAMANI SUNDAR[†]

Received January 11, 1989
Revised January 28, 1991

Splay is a simple, efficient algorithm for searching binary search trees, devised by Sleator and Tarjan, that reorganizes the tree after each search by means of rotations. An open conjecture of Sleator and Tarjan states that Splay is, in essence, the fastest algorithm for processing any sequence of search operations on a binary search tree, using only rotations to reorganize the tree. Tarjan proved a basic special case of this conjecture, called the *Scanning Theorem,* and conjectured a more general special case, called the *Deque Conjecture.* The Deque Conjecture states that Splay requires linear time to process sequences of deque operations on a binary tree. We prove the following results:

1. Almost tight lower and upper bounds on the maximum numbers of occurrences of various types of right rotations in a sequence of right rotations performed on a binary tree. In particular, the lower bound for right 2-turns refutes Sleator's Right Turn Conjecture.
2. A linear times inverse Ackerman upper bound for the Deque Conjecture. This bound is derived using the above upper bounds.
3. Two new proofs of the Scanning Theorem, one, a simple potential-based proof that solves Tarjan's problem of finding a potential-based proof for the theorem, the other, an inductive proof that generalizes the theorem.

1. Introduction

We review the Splay Algorithm, its conjectures and previous works on them, and describe our results.

1.1. The Splay Algorithm and its conjectures

Splay is a simple, efficient algorithm for searching binary search trees, devised by Sleator and Tarjan [7]. A *splay* at an element x of a binary search tree first locates the element in the tree by traversing the path from the root of the tree to the element (called the *access path* of the element) and then transforms the tree by means of rotations in order to speed up future searches in the vicinity of the element. The splay transformation moves element x to the root of the tree along its access path by repeating the following step (See Figure 1):

Splay step.
Let p and g denote, respectively, the parent and the grandparent of x.
Case 1. p is the root: Make x the new root, by rotating the edge $[x,p]$.

* A preliminary version of this paper appeared in the *Proceedings of the Thirtieth Annual Symposium on Foundations of Computer Science*, 1989, 555–559.

† This work was supported in part by NSF grants CCR-8702271 and DMS-8703458 and ONR grant N00014-85-K-0046.

AMS subject classification code (1991): 68 P 05, 68 R 05

Case 2. $[x,p]$ is a left edge (*i.e.* an edge to a left child) and $[p,g]$ is a right edge, or vice versa: Rotate $[x,p]$; Rotate $[x,g]$.
Case 3. Either both $[x,p]$ and $[p,g]$ are left edges, or both are right edges: Rotate $[p,g]$; Rotate $[x,p]$.

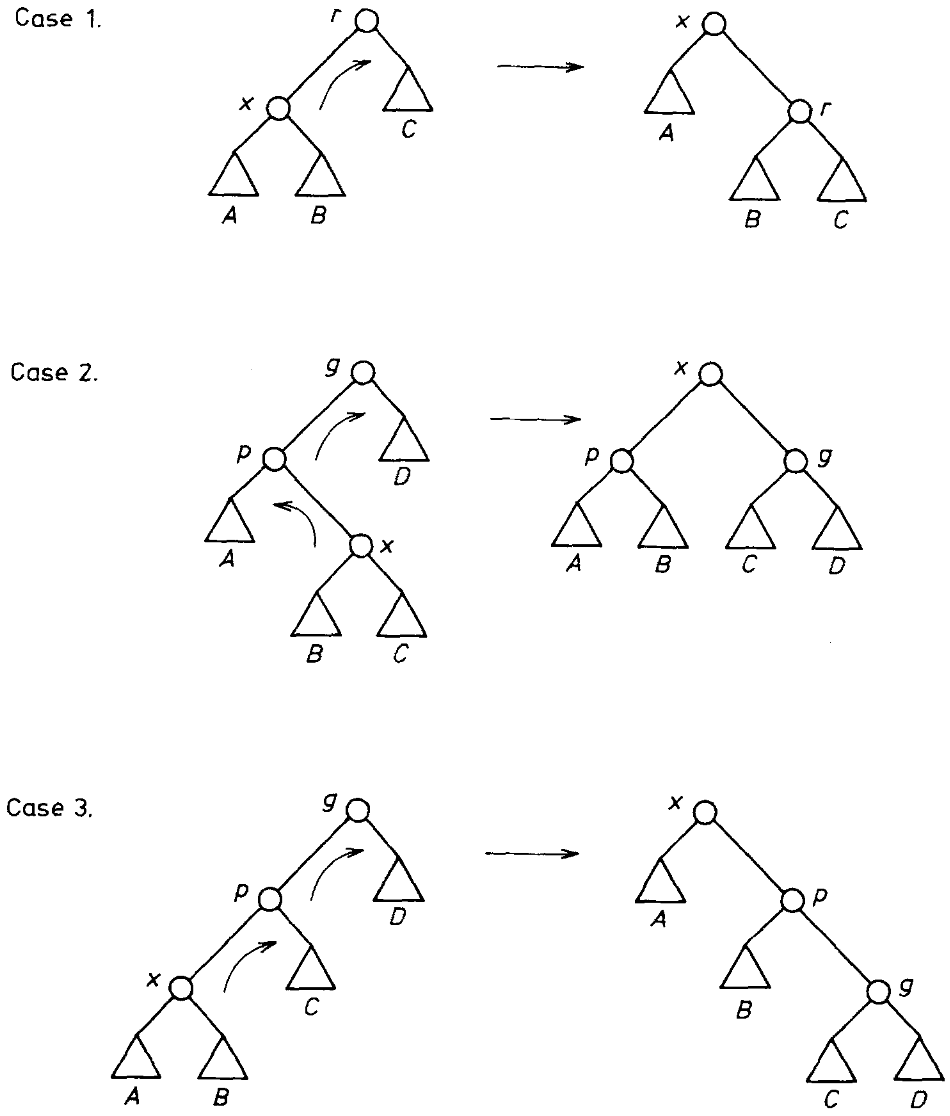

Fig. 1. A splay step

Sleator and Tarjan proved that Splay is, upto a constant factor, as efficient as the more complex traditional balanced tree algorithms for processing any sequence of binary search tree operations. They also showed that Splay actually behaves even faster on certain special kinds of sequences and conjectured that Splay is, upto a constant factor, the fastest rotation-based binary search tree algorithm for processing any sequence of searches on a binary search tree. We state this conjecture and some closely-related conjectures:

Conjecture 1 (Dynamic Optimality Conjecture [7]**).** *Let s denote an arbitrary sequence of searches of elements in a given n-node binary search tree. Define $\chi(s)$ equal to the minimum cost of executing sequence s on the tree using an algorithm that performs searches, incurring a cost equal to (1 + the distance of the element from the root) on each search, and transforms the tree by means of single rotations, incurring unit cost per single rotation. Splay takes $O(n + \chi(s))$ time to process s.*

Conjecture 2 (Deque Conjecture [9]**).** *Deque operations on a binary tree transform the tree by inserting or deleting nodes at the left or right end of the tree. We perform deque operations on a binary tree using Splay as follows (See Figure 2).* POP *splays at the leftmost node and removes it from the tree;* PUSH *inserts a new node to the left, making the old tree its right subtree;* EJECT *and* INJECT *are symmetric operations performed at the right end. Splay takes $O(m + n)$ time to process a sequence of m deque operations on an n-node binary tree.*

Conjecture 3 (Right Turn Conjecture [9]**).** *Define a right 2-turn on a binary tree to be a sequence of two right single rotations performed on the tree in which the bottom node of the first rotation coincides with the top node of the second rotation (See Figure 3). In a sequence of right 2-turns and right single rotations performed on an n-node binary tree, there are only $O(n)$ right 2-turns.*

The conjectures are related as follows. A stronger form of the Dynamic Optimality Conjecture that allows update operations as well as search operations implies the Deque Conjecture. The Right Turn Conjecture also implies the Deque Conjecture.

1.2. Terminology

We define the basic terminology used in the paper. A *binary search tree* over an ordered universe is a binary tree whose nodes are assigned elements from the universe in *symmetric order:* that is, for any node x assigned an element e, the elements in the left subtree of x are lesser than e and the elements in the right subtree of x are greater than e. The path between the root and the leftmost node in a binary tree is called the *left path.* A tree in which the left path is the entire tree is called a *left path tree.* The edge between a node and its left child in a binary tree is called a *left edge.* The paths in a binary tree that comprise only left edges are called *left subpaths.* The *left depth* of a node in a binary tree is defined to be the number of left edges on the path between the node and the root. The terms *right path tree, right path, right edge, right subpath,* and *right depth* are defined analogously. A *single rotation* of an edge $[x, p]$ in a binary tree is a transformation that makes x the parent of p by transferring one of the subtrees of x to p (See Figure 3). A single rotation is called *right* or *left,* respectively, according to whether $[x, p]$ was originally a left edge or a right edge. A *rotation* on a binary tree is a sequence of single rotations performed on the tree. A rotation is called *left* or *right,* respectively, if it consists solely of left single rotations or solely of right single rotations. A *double rotation* on a binary tree is a sequence of two single rotations performed on the tree that have a

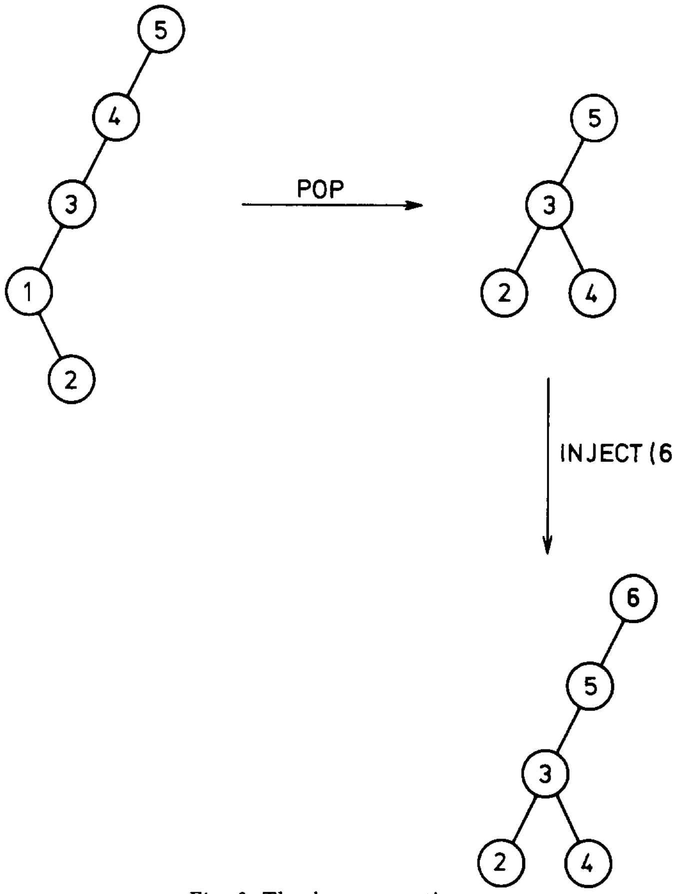

Fig. 2. The deque operations

node in common (as, for instance, by a splay step). A *left path rotation* on a binary tree is a right single rotation performed on the left path of the tree. A *right path rotation* is defined analogously.

We define the Ackerman hierarchy of functions $\{A_i | i \geq 0\}$, its inverse hierarchy $\{\hat{\alpha}_i | i \geq 0\}$, and inverse functions $\bar{\alpha}$ and α of the Ackerman function as follows:

$$\begin{aligned}
A_0(j) &= 2j \quad \text{for all } j \geq 1 \\
A_1(j) &= 2^j \quad \text{for all } j \geq 1 \\
A_i(j) &= \begin{cases} A_{i-1}(2) & \text{if } i \geq 2 \text{ and } j = 1 \\ A_{i-1}(A_i(j-1)) & \text{if } i \geq 2 \text{ and } j \geq 2 \end{cases} \\
\hat{\alpha}_i(n) &= \min\{k \geq 1 | A_i(k) \geq n\} \quad \text{for all } n \geq 1 \\
\bar{\alpha}(n) &= \min\{k \geq 1 | A_k(1) \geq n\} \quad \text{for all } n \geq 1 \\
\alpha(m,n) &= \min\{k \geq 1 | A_k(\lfloor m/n \rfloor) > \log n\} \quad \text{for all } m \geq n \geq 1
\end{aligned}$$

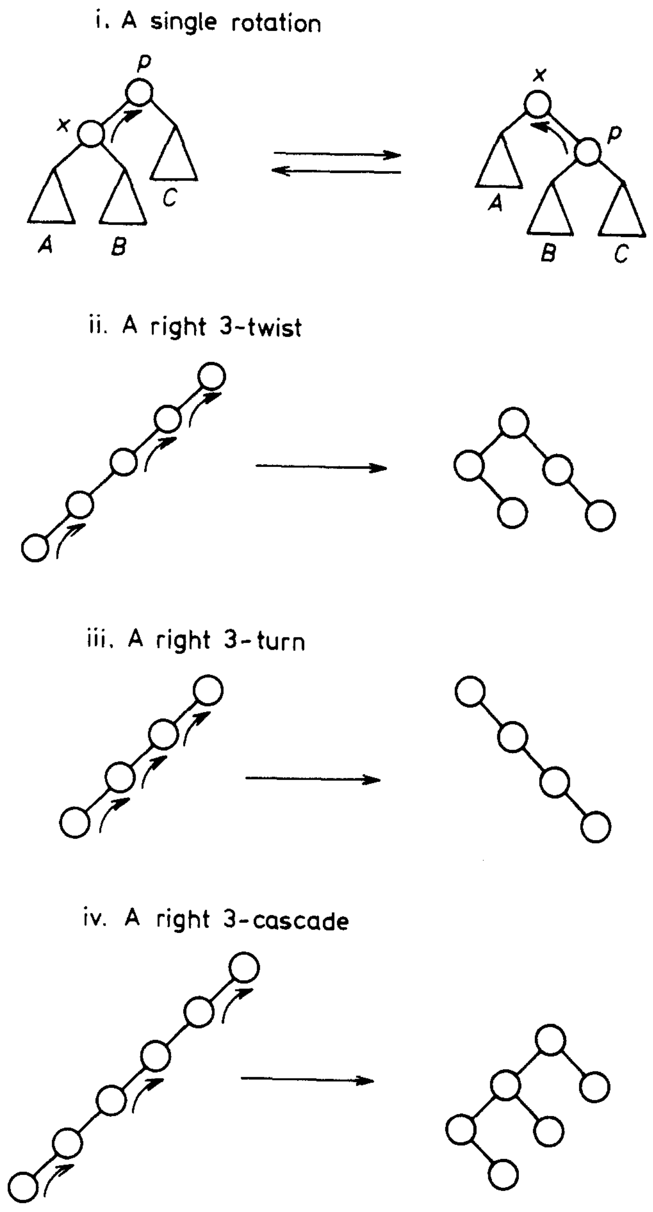

Fig. 3. The various types of rotations

The following table concretizes this definition:

i	0	1	2	3	4
$A_i(j)$	$2j$	2^j	$\left.2^{2^{\cdot^{\cdot^{2}}}}\right\}j$		
$\bar{\alpha}_i(n)$	$\lceil n/2 \rceil$	$\lceil \log n \rceil$	$\leq \log^* n$	$\leq \log^{**} n$	$\leq \log^{***} n$
$\{n \mid \bar{\alpha}(n) = i\}$		$[1,2]$	$[3,4]$	$[5,16]$	$[17, \left.2^{2^{\cdot^{\cdot^{2}}}}\right\}16]$

1.3. Previous works

Previous works on the Dynamic Optimality Conjecture have been mostly directed towards resolving its corollaries. Tarjan [9] proved that Splay requires linear time to sequentially scan the nodes of an n-node binary tree in symmetric order. This theorem, called the *Scanning Theorem,* is a corollary of all of the above conjectures. He also extended his proof to a proof of the Deque Conjecture when all the output operations are performed at one end of the tree. Lucas [6] obtained an $O(n\bar{\alpha}(n))$ upper bound for the Deque Conjecture when all the operations are output operations and the initial tree is a simple path between the leftmost and rightmost nodes. Building upon the work of Cole et al. [3], Cole [1], [2] recently proved Sleator and Tarjan's Dynamic Finger Conjecture [7] for the Splay Algorithm which is a corollary of the Dynamic Optimality Conjecture. Wilber [11] gave two elegant techniques for lower-bounding $\chi(s)$. The techniques yield optimal lower bounds for some special sequences (such as $\chi(s) = \Omega(n \log n)$ for the bit-reversal permutation), but it is not clear how tight these lower bounds are for general sequences.

A related combinatorial question that has been studied is, how many single rotations are needed, in the worst case, to transform one n-node binary tree into another n-node binary tree? Culik and Wood [4] noted that $2n - 2$ rotations suffice and, later, Sleator et al. [8] derived the optimal bound of $2n - 6$ rotations for all sufficiently large n.

1.4. Our Results

Our work is directed towards resolving the Deque Conjecture. A good understanding of the powers of various types of rotations on binary trees would equip us with the necessary tools to tackle the conjecture. We prove almost tight upper and lower bounds on the maximum numbers of occurrences of various types of right rotations in a sequence of right rotations performed on a binary tree. We study the following types of rotations (See Figure 3):

Right twist: For all $k \geq 1$, a right k-twist is a sequence of k right single rotations performed along a left subpath of a binary tree, traversing the subpath top-down.

Right turn: For all $k \geq 1$, a right k-turn is a right k-twist that converts a left subpath of k edges in a binary tree into a right subpath.

Right cascade: For all $k \geq 1$, a right k-cascade is a right k-twist that rotates every other edge lying on a left subpath of $2k - 1$ edges in a binary tree.

A *right twist sequence* is a sequence of right twists performed on a binary tree. Define $Tw_k(n)$, $Tu_k(n)$ and $C_k(n)$, respectively, to be the maximum numbers of occurrences of k-twists, k-turns and k-cascades in a right twist sequence performed on an n-node binary tree. These numbers are well defined since a tree is transformed into a right path after $\binom{n}{2}$ right single rotations. We derive the following bounds for $Tw_k(n)$, $Tu_k(n)$ and $C_k(n)$:

	Upper bound	Lower bound
$Tw_k(n)$	$O(kn^{1+1/k})$	$\Omega(n^{1+1/k}) - O(n)$
$Tu_k(n)$ $C_k(n)$	$\begin{cases} O(n\hat{\alpha}_{\lfloor k/2 \rfloor}(n)) & \text{if } k \neq 3 \\ O(n \log\log n) & \text{if } k = 3 \end{cases}$	$\begin{cases} \Omega(n\hat{\alpha}_{\lfloor k/2 \rfloor}(n)) - O(n) & \text{if } k \neq 3 \\ \Omega(n \log\log n) & \text{if } k = 3 \end{cases}$

The bounds for $Tu_k(n)$ and $C_k(n)$ are tight if $k \leq 2\bar{\alpha}(n) - 5$ and the bounds for $Tw_k(n)$ are nearly tight. The Right Turn Conjecture is refuted by the lower bound of $\Omega(n \log n)$ for $Tu_2(n)$[1]. We apply the upper bound for cascades to derive an $O((m+n)\bar{\alpha}(m+n))$ upper bound for the Deque Conjecture.

Another approach to the Deque Conjecture is to find new proofs of the Scanning Theorem that might naturally extend to the Deque Conjecture setting. We obtain a simple potential-based proof that solves Tarjan's problem [9] of finding a potential-based proof of the theorem, and an inductive proof that generalizes the theorem. The new proofs enhance our understanding of the Scanning Theorem, but, so far, have not led to a proof of the Deque Conjecture.

The paper is organized as follows. In Section 2, we prove the bounds for $Tw_k(n)$, $Tu_k(n)$ and $C_k(n)$. In Section 3, we derive the upper bound for the Deque Conjecture. In Section 4, we describe the new proofs of the Scanning Theorem.

2. Counting twists, turns, and cascades

The two subsections of this section derive the upper and lower bounds for $Tw_k(n)$, $Tu_k(n)$ and $C_k(n)$.

2.1. Upper bounds

All our upper bound proofs are based on a recursive divide-and-conquer strategy that partitions the binary tree on which the right twist sequence is performed into a collection of vertex-disjoint subtrees, called *block trees.* The root and some other nodes within each block are labeled *global* and the global nodes of all of the block trees induce a new tree called the *global tree.* Each rotation on the original tree effects a similar rotation either on one of the block trees or on the global tree. This allows us to inductively count the number of rotations of each type in the sequence.

We need the notion of *blocks* in binary trees [9]. Consider an n-node binary tree B whose nodes are labeled from 1 to n in symmetric order. A *block* of B is an interval $[i,j] \subseteq [1,n]$ of nodes in B. Any block $[i,j]$ of B induces a binary tree $B|_{[i,j]}$, called the *block tree* of block $[i,j]$, which comprises exactly the nodes i to j. The root of $B|_{[i,j]}$ is the lowest common ancestor of nodes i and j in B. The left child

[1] S.R.Kosaraju has independently proved that $Tu_2(n) = \theta(n \log n)$. While his upper bound proof differs from ours, the lower bound constructions match.

of a node x in $B|_{[i,j]}$ is the highest node in the left subtree of x in B which lies in block $[i,j]$. The right child of a node in $B|_{[i,j]}$ is defined analogously. Notice that, for the subtree rooted at any node of B, the highest node of the subtree which lies in block $[i,j]$ is unique whenever it exists: if two equally highest nodes exist, then their lowest common ancestor in the subtree would be higher than the two nodes, resulting in a contradiction. How does a rotation on B affect a block tree $B|_{[i,j]}$? If both of the nodes involved in the rotation are in $B|_{[i,j]}$, then the rotation translates into a rotation on $B|_{[i,j]}$ involving the same pair of nodes. Otherwise, $B|_{[i,j]}$ is not affected.

The functions Tw_k, Tu_k and C_k are superadditive:

Lemma 1. *For all $k \geq 1$ and $m \geq n \geq 1$, we have:*
a. $\lfloor m/n \rfloor Tw_k(n) \leq Tw_k(m)$,
b. $\lfloor m/n \rfloor Tu_k(n) \leq Tu_k(m)$, and
c. $\lfloor m/n \rfloor C_k(n) \leq C_k(m)$.

Proof. We prove Part a.; Parts b. and c. are similar. Given a right twist sequence S for an n-node binary tree B that comprises $Tw_k(n)$ right k-twists, construct a new tree of size $\lfloor m/n \rfloor n \leq m$ by starting with a copy of B and successively inserting a new copy of B as the right subtree of the rightmost node in the current tree $\lfloor m/n \rfloor - 1$ times. Since S can be performed on each of the copies of B one after another, there exists a right twist sequence with $\lfloor m/n \rfloor Tw_k(n)$ k-turns for a tree of size m. Part a. follows immediately. ∎

The upper bound for twists is the simplest to derive. Define $L_i(j) = \binom{i+j-1}{i}$ for all $i \geq 1$ and $j \geq 1$. The upper bound for $Tw_k(n)$ for n of the form $L_k(j)$ is given by:

Lemma 2. $Tw_k(L_k(j)) \leq k\binom{k+j-1}{k+1}$ *for all $k \geq 1$ and $j \geq 1$.*

Proof. We use double induction on k and j.

Case 1. $k = 1$ or $j = 1$: Straightforward.

Case 2. $k \geq 2$ and $j \geq 2$: The tree is partitioned into a left block of $L_{k-1}(j)$ nodes and a right block of $L_k(j-1)$ nodes. A right twist sequence on the tree translates into corresponding right twist sequences on the left and right block trees. We classify the k-twists in the original sequence into three categories and count the number of k-twists of each type separately. In the first type of k-twist, the lowest $k-1$ single rotations involve only left block nodes. Such a k-twist translates into a $(k-1)$-twist on the left block tree. Applying induction to the induced right twist sequence performed on the left block tree, we see that there are at most $(k-1)\binom{k+j-2}{k}$ k-twists of the first type in the original right twist sequence. Similarly, the number of k-twists that involve only right block nodes is at most $k\binom{k+j-2}{k+1}$. Consider a k-twist that does not belong to these two categories. The highest single rotation of such a twist must involve only right block nodes; also, the lowest node involved in the twist must be a left block node. This implies that the highest node of the twist is a right block node that leaves the left path of its block as a result of the twist. Right rotations never add nodes to a block's left path, so the number of k-twists in the last category is at most the initial size of the left path of the right block $\leq L_k(j-1) = \binom{k+j-2}{k}$. It follows that the total number of right k-twists in the right twist sequence is bounded by

$$\begin{aligned}(k-1)\binom{k+j-2}{k} &+ k\binom{k+j-2}{k+1} + \binom{k+j-2}{k} \\ &= k\binom{k+j-2}{k} + k\binom{k+j-2}{k+1} \\ &= k\binom{k+j-1}{k+1}.\end{aligned}$$

∎

A simple calculation using the above lemma and Lemma 1a gives the upper bound for $Tw_k(n)$ for all n:

Theorem 1. *$Tw_k(n) \leq kn^{1+1/k}$ for all $k \geq 1$ and $n \geq 1$.*

Proof. Fix k and define $j = \min\left\{i \middle| n \leq \binom{k+i}{k}\right\}$. Then we have

$$\begin{aligned}T_k(n) &\leq T_k\left(\binom{k+j}{k}\right) \Big/ \left\lfloor \binom{k+j}{k} \Big/ n \right\rfloor && \text{(By Lemma 1a)} \\ &\leq 2k\binom{k+j}{k+1} n \Big/ \binom{k+j}{k} && \text{(By Lemma 2)} \\ &\leq kn^{1+1/k}.\end{aligned}$$

∎

We derive the upper bounds for turns and cascades. It is easy to see that $Tu_1(n) = C_1(n) = \binom{n}{2} \leq n\hat{\alpha}_0(n)$. Let us prove that $Tu_2(n) = O(n\hat{\alpha}_1(n))$. Consider any right twist sequence performed on a binary tree B. We divide B into a left block $[1, \lfloor n/2 \rfloor]$ and a right block $[\lfloor n/2 \rfloor + 1, n]$. Every 2-turn either involves nodes from only one block (*intrablock*) or involves nodes from both blocks (*interblock*). An intrablock 2-turn effects a 2-turn in the corresponding block tree and gets counted in the right twist sequence for the block tree. Every interblock 2-turn either adds a node to the right path of the left block tree or deletes a node from the left path of the right block tree (See Figure 4). Right rotations never remove nodes from a block's right path or add nodes to a block's left path, so the number of interblock 2-turns is at most $n-2$. This leads to the following recurrence for $Tu_2(n)$:

$$Tu_2(n) \leq \begin{cases} Tu_2(\lfloor n/2 \rfloor) + Tu_2(\lceil n/2 \rceil) + n - 2 & \text{if } n \geq 3 \\ 0 & \text{if } 1 \leq n \leq 2 \end{cases}$$

Solving the recurrence yields the desired bound for $Tu_2(n)$:

$$Tu_2(n) \leq n\lceil \log n \rceil - 2^{\lceil \log n \rceil} - n + 2 \leq n\hat{\alpha}_1(n).$$

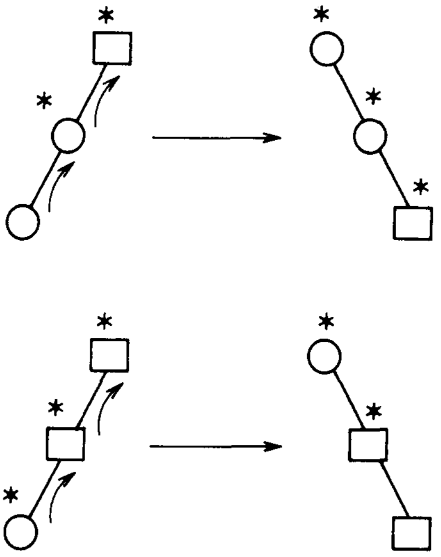

Fig. 4. The two types of interblock right 2-turns. Circles denote left block nodes and squares denote right block nodes. The stars identify the nodes that lie on the left/right path of a block

With a slight modification the same proof works for 2-cascades also. An interblock 2-cascade either decreases the size of the left path of the right block tree or increases the number of left block nodes whose left depth relative to the block is at most 1 (See Figure 5). Right rotations never increase the left depth of a node, so the number of interblock 2-cascades is at most $n-3$. The bound $C_2(n) \leq n\hat{\alpha}_1(n)$ follows.

In order to extend the above argument to k-turns and k-cascades for $k \geq 3$, we need an Ackerman-like hierarchy of functions $\{K_i | i \geq 1\}$:

$$K_1(j) = 8j \quad \text{for all } j \geq 1$$

$$K_2(j) = 2^{4j} \quad \text{for all } j \geq 1$$

$$K_i(j) = \begin{cases} iK_{i-2}(\lfloor i/2 \rfloor) & \text{if } i \geq 3 \text{ and } j = 1 \\ K_i(j-1)K_{i-2}(K_i(j-1)/4)/2 & \text{if } i \geq 3 \text{ and } j \geq 2 \end{cases}$$

The function K_i grows faster than the Ackerman function $A_{\lfloor i/2 \rfloor}$:

Lemma 3. *1.* $A_1^{(2)}(j) \leq K_3(j)$ *for all* $j \geq 1$.
2. $A_{\lfloor i/2 \rfloor}(j) \leq K_i(j)$ *for all* $i \neq 3$ *and* $j \geq 1$. ∎

The upper bound for $Tu_k(n)$ for n of the form $K_k(j)$ is given by:

Lemma 4. $Tu_k(K_k(j)) \leq 4jK_k(j)$, *for all* $k \geq 1$ *and* $j \geq 1$.

Proof. We use double induction on k and j.

Case 1. $1 \leq k \leq 2$: The lemma follows from the bounds $Tu_1(n) \leq n\hat{\alpha}_0(n)$ and $Tu_2(n) \leq n\hat{\alpha}_1(n)$.

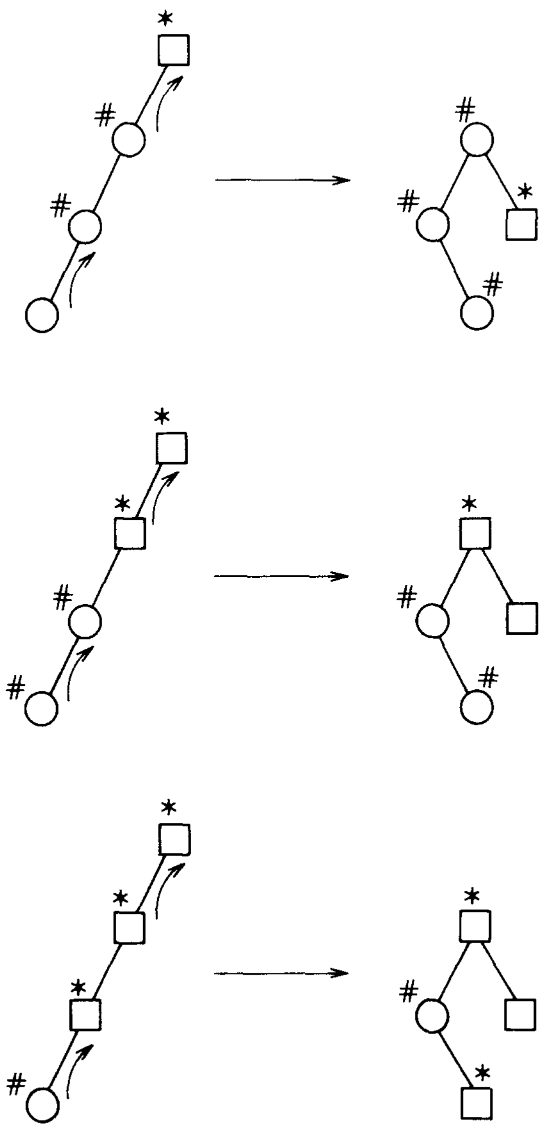

Fig. 5. The three types of interblock right 2-cascades. The sharps identify the left block nodes ;hat have a left depth of at most 1; the stars identify the right block nodes lying on the left path of their block

Case 2. $k \geq 3$ and $j = 1$: We need to show that $Tu_k(K_k(1)) \leq 4K_k(1)$. Consider a binary tree B having $K_k(1)$ nodes on which a right twist sequence is performed. Divide B into a sequence of $K_{k-2}(\lfloor k/2 \rfloor)/2$ blocks of size $2k$ each. Each k-turn is of one of the following types:

Type A. All of the nodes involved in the k-turn belong to a single block: Since a block has only $2k$ nodes, there can be at most one such k-turn per block.

Type B. Some two nodes of the k-turn belong to a single block, but not all of the nodes of the turn are in that block: Let C denote the block tree of this block. The k-turn causes either an increase in the size of the right path of C, or a decrease in the size of the left path of C, or both. Hence the number of Type-B k-turns is at most $2K_k(1)$.

Type C. Each node of the k-turn belongs to a different block: To handle this case, we label the root of each block *global*. The global nodes in B induce a binary tree G, called the *global tree*. The root of G is identical to the root of B. The left child of a node x in G is the highest global node in the left subtree of x in B. The right child of a node is defined similarly. It is easy to see that the left and right children of any node in G are unique. The effect on G of a rotation on B is analogous to the effect of such a rotation on a block tree of B: A rotation on B translates into a rotation on G if both of the nodes of the rotation are global; otherwise, G is unaffected. (If a rotation changes the root of a block then the global role passes from the old root to the new root but this does not affect the global tree.)

Suppose that the k-turn turns the left subpath $x_1 - x_2 - \cdots - x_{k+1}$ of B into a right subpath. Since all the x_is are from different blocks, the nodes $x_2, x_3, \ldots, x_k$ are all global. Therefore, the k-turn results in a $(k-2)$-turn on G (if x_1 or x_{k+1} is also global, then some right single rotations are also performed on G.) The number of $(k-2)$-turns that can be performed on G is at most

$$\begin{aligned} Tu_{k-2}(K_{k-2}(\lfloor k/2 \rfloor)/2) &\leq Tu_{k-2}(K_{k-2}(\lfloor k/2 \rfloor))/2 \text{ (By Lemma 1b)} \\ &\leq 2\lfloor k/2 \rfloor K_{k-2}(\lfloor k/2 \rfloor) \text{ (By the induction hypothesis)} \\ &< K_k(1). \end{aligned}$$

This gives an upper bound of $K_k(1)$ for the number of Type-C k-turns performed on B.

Summing together the above bounds for the three types of k-turns, we obtain a bound of

$$K_{k-2}(\lfloor k/2 \rfloor)/2 + 2K_k(1) + K_k(1) \leq 4K_k(1)$$

for the total number of k-turns in the right twist sequence. This completes Case 2.

Case 3. $k \geq 3$ and $j \geq 2$: We divide the binary tree on which the right twist sequence is executed into $K_k(j)/K_k(j-1)$ blocks of size $K_k(j-1)$ each. We split the k-turns into the three types defined in Case 2 and obtain the following tally for each type of turn:

Number of Type-A k-turns

$$\begin{aligned} &\leq \frac{K_k(j)}{K_k(j-1)}.4(j-1)K_k(j-1) \text{ (By the induction hypothesis)} \\ &\leq 4(j-1)K_k(j). \end{aligned}$$

Number of Type-B k-turns $\leq 2K_k(j)$.

Number of Type-C k-turns

$$\begin{aligned}
&\leq Tu_{k-2}(K_{k-2}(K_k(j-1)/4)/2)\\
&\leq Tu_{k-2}(K_{k-2}(K_k(j-1)/4))/2 \text{ (By Lemma 1b)}\\
&\leq K_k(j-1)K_{k-2}(K_k(j-1)/4)/2 \text{ (By the induction hypothesis)}\\
&= K_k(j).
\end{aligned}$$

Hence the total number of k-turns in the sequence is at most

$$4(j-1)K_k(j) + 2K_k(j) + K_k(j) < 4jK_k(j).$$

This finishes Case 3. ∎

Combining the above lemma with Lemmas 1b and 3, we obtain the upper bound for $Tu_k(n)$ for all k and n:

Theorem 2.

$$Tu_k(n) \leq \begin{cases} 8n\hat{\alpha}_{\lfloor k/2 \rfloor}(n) & \text{if } k \neq 3 \\ 8n \log\log n & \text{if } k = 3 \end{cases}$$ ∎

The upper bound for cascades is derived analogously:

Theorem 3.

$$C_k(n) \leq \begin{cases} 8n\hat{\alpha}_{\lfloor k/2 \rfloor}(n) & \text{if } k \neq 3 \\ 8n \log\log n & \text{if } k = 3 \end{cases}$$

Proof. It suffices to prove Lemma 4 for $C_k(n)$: $C_k(K_k(j)) \leq 4jK_k(j)$, for all $k \geq 1$ and $j \geq 1$. Referring to the proof of Lemma 4, only the handling of Cases 2 and 3 has to be modified. Consider Case 2. As before, the blocks have size $2k$ each. The k-cascades are categorized as follows:

Type A. All nodes involved in the cascade belong to a single block: There is at most one Type-A cascade per block.

Type B. One of the cascade rotations involves a pair of nodes belonging to a single block, but not all of the nodes of the cascade are in that block: If the cascade rotates an edge that lies on the left path of some block, then the length of the left path of the block decreases by at least 1. Alternately, if the lowest three nodes involved in the cascade are from the same block, then the number of nodes in that block whose left depth is at most 1 increases. We conclude that the number of Type-B cascades falling under the above categories is at most $2K_k(1) - K_{k-2}(\lfloor k/2 \rfloor)$. In every remaining Type-B k-cascade, only the lowest cascade rotation is intrablock and the lowest three nodes do not belong to the same block. Each such cascade behaves like a Type-C cascade in that it causes a $(k-2)$-cascade on the global tree (defined below) which accounts for it.

Type C. Each cascade rotation involves a pair of nodes belonging to different blocks: In this case for each block, in addition to the root of the block, we also label the left child of the root within the block global, if it exists; if the root has no left child, then the right child of the root is labeled global. Right rotations are propagated from the original tree to the global tree as described in Lemma 4 except in the following situation: When the edge joining the root and its left child, say l, in a block is rotated, the left child of l, say ll, now becomes global, and if ll is not adjacent to l in B, this results in a series of right single rotations on the global tree

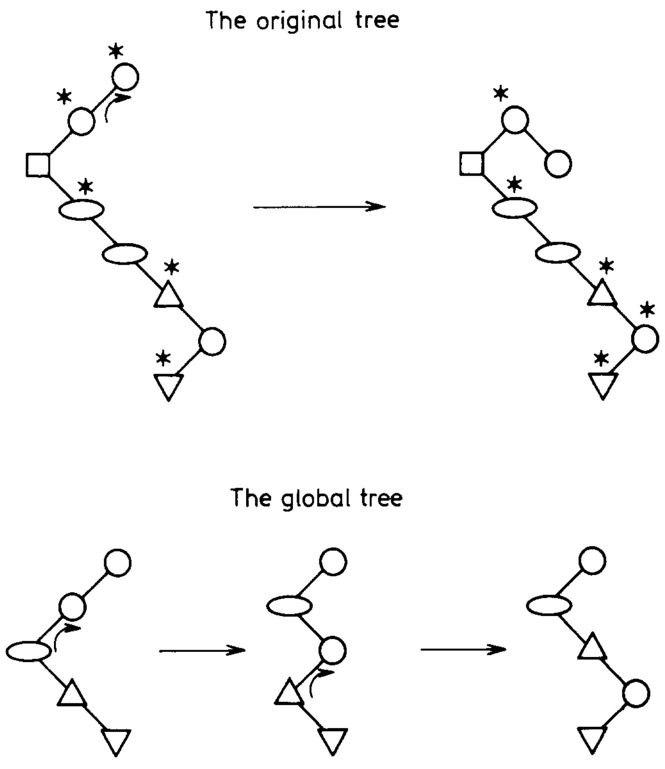

Fig. 6. The effect of a right single rotation involving the root of a block and its left child within the block on the global tree. Circles denote the nodes of the block; other symbols denote the nodes from other blocks. The starred nodes in the original tree are the global nodes

(See Figure 6). Under this definition of global tree, the $(k-2)$ interior rotations performed by any Type-C k-cascade are all global. Hence, each Type-C k-cascade translates into a right $(k-2)$-cascade and a sequence of right single rotations on the global tree. Therefore the total number of k-cascades in the sequence is at most

$$K_{k-2}(\lfloor k/2 \rfloor)/2 + 2K_k(1) - K_{k-2}(\lfloor k/2 \rfloor) + C_{k-2}(K_{k-2}(\lfloor k/2 \rfloor)) < 4K_k(1).$$

This completes Case 2 in the proof of Lemma 4 for cascades. Case 3 is handled similarly. ∎

2.2. Lower bounds

The lower bound right twist sequences are inductively constructed by mimicking the divide-and-conquer strategy used to derive the upper bounds. The lower bound sequences always transform a left path tree into a right path tree. The tree is partitioned into a collection of vertex-disjoint block trees and a global tree is formed by selecting nodes from each block tree. The lower bound sequence for a tree is constructed by inductively constructing similar lower bound sequences for the block trees and for the global tree and weaving these sequences together. Actually, we first inductively construct a sequence of right twists as well as deletions having sufficiently many rotations of the given type and then remove the deletions to obtain the lower bound sequence.

We need some definitions. For all positive integers k, a *right k-twist-deletion sequence* is defined to be an intermixed sequence of right single rotations, right k-twists and deletions of the leftmost node performed on a binary tree. *Right k-turn-deletion sequences* and *right k-cascade-deletion sequences* are defined analogously. Consider a right twist that is performed on some left subpath $x_0 - x_1 - \cdots - x_l$ of a binary tree, where x_0 is the lowest node on the subpath. x_0 is called the *base* of the twist. If x_k is the left child of a node y (say) in the tree, then the twist is called an *apex twist* and y is the *apex* of the twist. Otherwise, the twist is called *apexless.*

The lower bound for $Tw_k(n)$ for n of the form $L_k(j) = \binom{k+j-1}{k}$ is given by:

Lemma 5. $Tw_k(L_k(j)) \geq \binom{k+j-1}{k+1}$ *for all $k \geq 1$ and $j \geq 1$.*

Proof. For any pair of positive integers k and j, we inductively construct a right k-twist-deletion sequence for a left path tree of $L_k(j)$ nodes having the following properties:

1. The sequence deletes all the nodes from the tree.
2. A right k-twist always involves the leftmost node of the tree.
3. A deletion always deletes the root of the tree.
4. The sequence has exactly $\binom{k+j-1}{k+1}$ k-twists.

The removal of the deletions from the sequence would yield a right twist sequence having the desired number of k-twists.

Case 1. $k = 1$ or $j = 1$: Easy.

Case 2. $k \geq 2$ and $j \geq 2$: Divide the left path tree into a lower block of size $L_{k-1}(j)$ and an upper block of size $L_k(j-1)$. Recursively perform a right $(k-1)$-twist-deletion sequence, say $\hat{S}$, on the lower block tree. For each $(k-1)$-twist in $\hat{S}$, first rotate the edge joining the root of the lower block with its parent and then perform the $(k-1)$-twist on the block. This is equivalent to a k-twist involving the leftmost node of the tree. Each deletion in $\hat{S}$ is modified by first making the deleted node the root of the tree using right rotations and then deleting the node. By property 4 of $\hat{S}$, the number of $(k-1)$-twists in $\hat{S}$ is exactly $\binom{k+j-2}{k}$. The initial depth of the root of the lower block equals $L_k(j-1) = \binom{k+j-2}{k}$. Since each $(k-1)$-twist in $\hat{S}$ reduces the depth of the root of the lower block by 1 and no other operation in $\hat{S}$ affects the depth, it is always possible to rotate the root of the lower block just before the execution of any $(k-1)$-twist in $\hat{S}$. The construction is completed by recursively performing a right k-twist-deletion sequence, say $\bar{S}$, on the upper block.

The sequence obviously satisfies properties 1–3. The total number of k-twists performed by the sequence equals

$$\begin{aligned}
&(\text{the number of } (k-1)\text{-twists in } \hat{S}) + (\text{the number of } k\text{-twists in } \bar{S}) \\
&= \binom{k+j-2}{k} + \binom{k+j-2}{k+1} \text{ (By the induction hypothesis)} \\
&= \binom{k+j-1}{k+1}.
\end{aligned}$$

This proves property 4. ∎

Combining Lemma 1a with the above lemma yields:

Theorem 4. *$Tw_k(n) \geq n^{1+1/k}/2e - O(n)$ for all $k \geq 1$ and $n \geq 1$.* ∎

We construct the lower bound sequences for turns. As in the upper bound proof, we need a new Ackerman-like hierarchy of functions. Define:

$B_1(j) = j$ for all $j \geq 1$

$B_2(j) = 2^j - 1$ for all $j \geq 1$

$$B_i(j) = \begin{cases} 1 & \text{if } i \geq 3 \text{ and } j = 1 \\ ((i+1)jB_i(j-1)+1)B_{i-2}((i+1)jB_i(j-1)) & \text{if } i \geq 3 \text{ and } j \geq 2 \end{cases}$$

The function B_i grows essentially at the same rate as the Ackerman function $A_{\lfloor i/2 \rfloor}$:

Lemma 6. *1. $B_3(j) \leq A_1^{(2)}(2j)$, for all $j \geq 1$.*
2. $B_i(j) \leq A_{\lfloor i/2 \rfloor}(3j)$ for all $i \neq 3$ and $j \geq 1$. ∎

The lower bound for $Tu_k(n)$ for n of the form $B_k(j)$ is given by:

Lemma 7. *$Tu_k(B_k(j)) \geq (1/2)(j-3)B_k(j)$ for all $k \geq 1$ and $j \geq 1$.*

Proof. For any pair of positive integers k and j, we inductively construct a right k-turn-deletion sequence for the left path tree of $B_k(j)$ nodes having the following properties:

1. The sequence deletes all the nodes from the tree.
2. A right k-turn always involves the leftmost node of the tree.
3. A deletion always deletes the root of the tree.
4. The sequence comprises at least $(1/2)(j-3)B_k(j)$ apex k-turns. Further, if $k \geq 3$, there are no apexless k-turns in the sequence.
5. For any node x, the number of apex k-turns with base x is at most j.
6. For any node x, the number of apex k-turns with apex x is at most j.

Case 1. $k = 1$: The sequence repeatedly rotates the leftmost node to the root and deletes it.

Case 2. $k = 2$: Divide the left path tree into a lower left subpath comprising $2^{j-1} - 1$ nodes, a middle node, and an upper left subpath comprising $2^{j-1} - 1$ nodes. Recursively perform a right 2-turn-deletion sequence on the lower subpath. Modify each deletion in this sequence as follows: Perform a 2-turn on the subpath defined by the deleted node, say x, its parent (the middle node), and its grand parent; make

x the root of the tree by successively rotating the edge joining it and its parent; delete x from the tree (this also deletes x from the lower subpath.) Next, delete the middle node which is currently the root of the tree. Finally, recursively perform a right 2-turn-deletion sequence on the upper subpath (See Figure 7).

This sequence performs $(j-2)2^{j-1}+1$ 2-turns of which exactly $j-1$ are apexless. Therefore the number of apex 2-turns is at least $(j-3)2^{j-1} \geq (1/2)(j-3)B_2(j)$. This proves property 4. The remaining properties are easy to check.

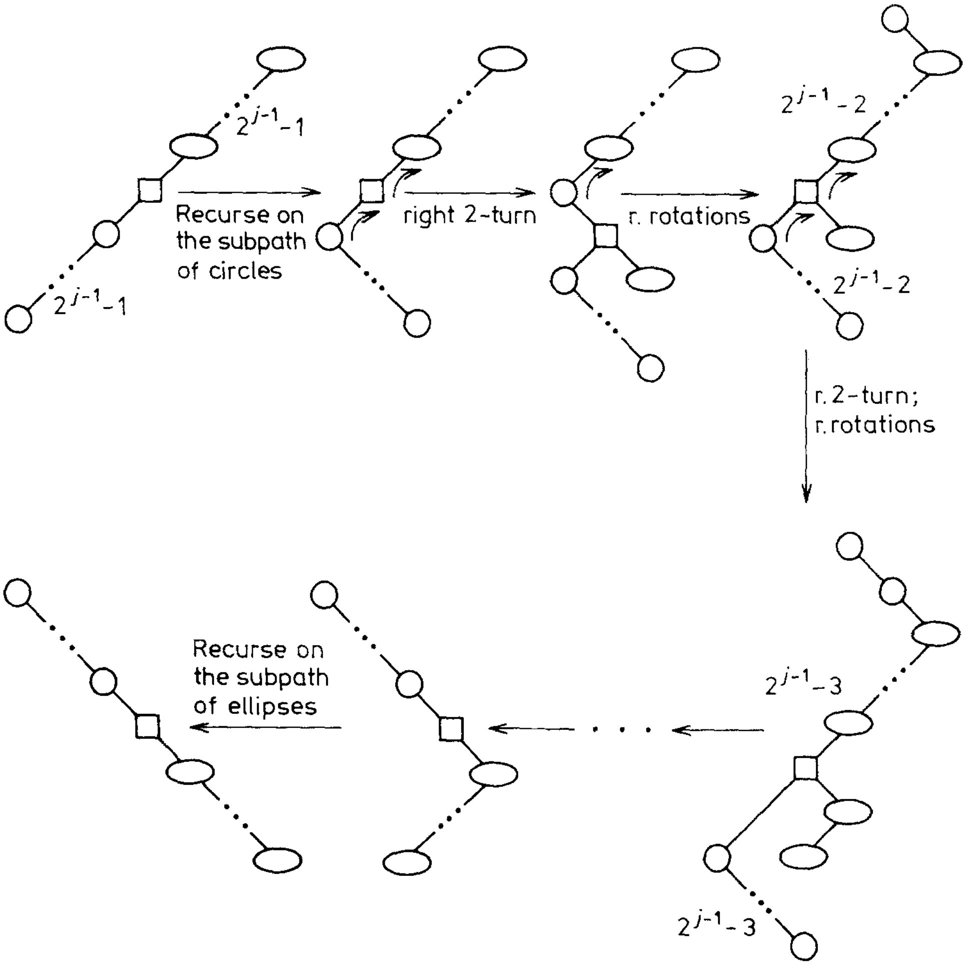

Fig. 7. The lower bound construction for right 2-turns. The construction recursively transforms a left path tree of size 2^j-1 into a right path tree.

Case 3. $k \geq 3$ and $j = 1$: Just delete the only node in the tree.

Case 4. $k \geq 3$ and $j \geq 2$: Let $s = (k+1)jB_k(j-1)$. We inductively construct the sequences of operations performed on the block trees of the tree:

Lemma 8. *There exists a right k-turn-deletion sequence for a left path tree of size $s+1$ satisfying properties 1 and 2 and the following properties:*

$\bar{3}$. *A deletion that is not the last operation in the sequence always deletes the left child of the root.*

$\bar{4}$. *The sequence comprises at least $(1/2)(j-4)s+j$ right k-turns all of which are apex turns.*

$\bar{5}$. *For any node x, the number of apex k-turns with base x is at most $j-1$.*

$\bar{6}$. *For any node x, the number of apex k-turns with apex x is at most $j-1$.*

$\bar{7}$. *The root of the tree is always the rightmost node.*

Proof. Divide the nodes of the tree excluding the root into a sequence of $(k+1)j$ blocks of size $B_k(j-1)$ each. Perform a right k-turn-deletion sequence obeying properties 1–6 on the lowest block. (The inductive hypothesis implies the existence of such a sequence.) Denote this sequence by S. Each deletion in S except the last is modified by rotating the deleted node up the tree until it is adjacent to the root and then deleting it. The deletion of the last node in the block, say x_1, is implemented differently. x_1 is rotated up the tree until it is in contact with the root of the next higher block, say x_2. x_2 is rotated upwards in a similar fashion in order to make it adjacent to the root of the next higher block, say x_3. In this manner we create a left subpath $x_1 - x_2 - \cdots - x_{k+1}$ containing the roots of the lowest $k+1$ blocks. Next, a right k-turn is performed on this subpath and then x_1 is rotated up the tree and deleted. Following this, S is executed on the blocks of nodes $x_2, x_3 \ldots, x_{k+1}$ in succession. Each deletion is modified by first making the deleted node adjacent to the root and then deleting it. At the conclusion of this sequence of operations, all the nodes in the lowest $k+1$ blocks of the tree have been deleted and at least $(1/2)(j-4)(k+1)B_k(j-1)+1$ apex k-turns have been performed. The above sequence of operations is repeated on each group of $k+1$ consecutive blocks, choosing the lowest group of blocks currently in the tree each time. The final operation of the sequence deletes the root.

It is obvious that the right k-turn-deletion sequence constructed above satisfies properties 1, 2, $\bar{3}$ and $\bar{7}$. Since there are j groups of $k+1$ blocks each, the total number of apex k-turns executed by the sequence is at least $(1/2)(j-4)s+j$. Further, by property 4 of S, the sequence performs only apex k-turns. This proves property $\bar{4}$. Properties $\bar{5}$ and $\bar{6}$ are easy to show using properties 5 and 6 of sequence S. ∎

We construct a right k-turn-deletion sequence for the left path tree of size $B_k(j)$ satisfying the six properties. The tree is partitioned into $B_{k-2}(s)$ blocks of size $s+1$ each. The root of each block is labeled global. The global nodes form a global tree as described in the proof of Lemma 4. By the induction hypothesis, there exists a right $(k-2)$-turn-deletion sequence, say $\bar{S}$, for the global tree, satisfying properties 1-6. We construct the right k-turn-deletion sequence, denoted S, for the original tree by mapping each global tree operation in $\bar{S}$ onto a sequence of original tree operations, preserving the correspondence between the two trees. The following invariants define the relationships between the two trees:

A. Let B denote the block containing the leftmost node of the tree and let x denote the root of B. Suppose that d nodes have been deleted from B so far. Then,
   i. The number of apex $(k-2)$-turns performed so far on the global tree that had x as their base is exactly d.

ii. Denote by $\hat{S}$ the right k-turn-deletion sequence constructed by Lemma 8 that deletes all the nodes from the left path tree of size $s+1$. Let T denote the tree that results when the prefix of $\hat{S}$ up to the d^{th} deletion is executed on the left path tree. The block tree of B equals T.

B. Consider any block B that does not contain the leftmost node of the tree. Let x denote the root of B. The block tree of B is a left path tree which is divided into the root and two subpaths. The nodes in the lower subpath, called *black nodes,* are the nodes in B that have participated in a k-turn. The nodes in the upper subpath are called *white nodes.*

i. If b denotes the number of black nodes currently in B, then exactly b of the apex $(k-2)$-turns performed so far on the global tree had x as their apex.

C. If x is a global node with a right child y in the global tree, y is also the right child of x in the original tree.

D. If x is a global node with a left child y in the global tree, there is a left subpath $x = x_0 - x_1 - \dots - x_{k+1} = y$ in the original tree such that $x_1, x_2, \dots, x_k$ are the set of white nodes in the block of x.

Each global tree operation in $\bar{S}$ is simulated as follows:

Right rotation: Suppose that a global tree edge $[x, y]$, such that y is a left child of x, is rotated. In the original tree we repeatedly rotate the edge connecting y and its parent until x becomes the right child of y. Only invariants C and D are affected by the rotations. It is not hard to see that both these invariants are true after the last rotation.

Deletion: Suppose that a global node x is deleted. Since x is the root of the global tree, it is also the root of the original tree. Let d denote the number of nodes deleted so far from the block of x. We perform $\hat{S}$ (the sequence constructed by Lemma 8) on the block tree of x starting immediately after the d^{th} deletion. Each deletion is modified so that the deleted node is first made the root of the tree and then deleted. Invariant A.ii ensures that this is valid and that this will result in the deletion of all the nodes in the block of x from the tree. Therefore this sequence of operations reestablishes the correspondence between the global tree and the original tree.

Apexless $(k-2)$-turn: Break up the turn into a sequence of $k-2$ global rotations and simulate each global rotation as specified above.

Apex $(k-2)$-turn: Suppose that a global tree subpath $x_1 - x_2 - \dots - x_{k-1}$ is turned and that x_1 is the base of the turn. Let x_0 denote the leftmost node in the block of x_1 and let x_k denote the parent of x_{k-1} in the original tree. We create the subpath $x_0 - x_1 - \dots - x_k$ in the original tree and perform a k-turn on this subpath. This is implemented as follows:

1. Let d denote the number of nodes deleted so far from the block of x_1. Execute the segment of sequence $\hat{S}$ between the d^{th} and the $(d+1)^{\text{st}}$ deletions (excluding the deletions) on the block tree of x_1. By Lemma 8, property $\bar{3}$, this makes node x_0 the left child of x_1.
2. Rotate x_1 up the tree until its parent is x_2. Continuing in this fashion, rotate the nodes $x_2, x_3, \dots, x_{k-1}$ upwards, creating a left subpath $x_0 - x_1 - \dots - x_k$.
3. Perform a k-turn on the subpath $x_0 - x_1 - \dots - x_k$.
4. Rotate x_0 up the tree, making it the root, and delete it.

5. Since x_k has become black due to the k-turn, the edge joining x_k and its left child is repeatedly rotated until the left child of x_k is not a global node.

Invariant B.i and property 6 of $\bar{S}$ guarantee that x_k is white at the beginning of this sequence of operations. Similarly, invariant A.i and property 5 of $\bar{S}$ guarantee that x_0 is well defined. Observe that all invariants are true at the end of the simulation.

The sequence of operations performed on the original tree during the simulation of $\bar{S}$ constitutes sequence S.

S deletes all the nodes in the original tree since $\bar{S}$ deletes all the nodes in the global tree. This proves that S satisfies property 1.

Properties 2 and 3 of S are apparent from the simulation procedure.

By Lemma 8, at least $(1/2)(j-4)s+j$ apex k-turns (*local turns*) are performed during the execution of $\hat{S}$ on any particular block. Hence the total number of local turns summed over all blocks is at least $(1/2)(j-4)sB_{k-2}(s)+jB_{k-2}(s)$. The number of turns involving global nodes (*global turns*) equals the number of $(k-2)$-turns in $\bar{S}$ which, by the induction hypothesis, is at least $(1/2)(s-3)B_{k-2}(s)$. Therefore the total number of k-turns in S is at least

$$\begin{aligned}&(1/2)(j-4)sB_{k-2}(s)+jB_{k-2}(s)+(1/2)(s-3)B_{k-2}(s)\\&\quad=((1/2)(j-3)s+j-(3/2))B_{k-2}(s)\\&\quad>(1/2)(j-3)B_k(j).\end{aligned}$$

Evidently, every k-turn in S has an apex. This proves property 4.

For any node x, there is at most one global turn with x as the base since x is deleted from the tree immediately after the turn. By Lemma 8, there are at most $j-1$ local turns with x as the base. We conclude property 5. Property 6 is proved analogously. ∎

Combining the above lemma with Lemma 1b yields:

Theorem 5.

$$Tu_k(n)\geq\begin{cases}(1/12)n\hat{\alpha}_{\lfloor k/2\rfloor}(n)-O(n) & \text{if } k\neq 3\\(1/8)n\log\log n-O(n) & \text{if } k=3\end{cases}$$ ∎

The lower bound for cascades is given by:

Theorem 6.

$$C_k(n)\geq\begin{cases}(1/12)n\hat{\alpha}_{\lfloor k/2\rfloor}(n)-O(n) & \text{if } k\neq 3\\(1/8)n\log\log n-O(n) & \text{if } k=3\end{cases}$$

Proof. We modify the lower bound proof for $Tu_k(n)$ given above. Define:

$$B'_1(j)=j \text{ for all } j\geq 1$$

$$B'_2(j)=3.2^j-2 \text{ for all } j\geq 1$$

$$B'_i(j)=\begin{cases}1 & \text{if } i\geq 3 \text{ and } j=1\\(4ijB'_i(j-1)+3)B'_{i-2}(4ijB'_i(j-1)) & \text{if } i\geq 3 \text{ and } j\geq 2\end{cases}$$

It is easy check that Lemma 6 holds for the new hierarchy $\{B'_i\}$. We prove the analogue of Lemma 7 for $C_k(n)$, which states that $C_k(B'_k(j))\geq(1/2)(j-3)B'_k(j)$ for all $k \geq 1$ and $j \geq 1$. We construct a right k-cascade-deletion sequence that converts a left path tree of size $B'_k(j)$ into a right path tree and satisfies the analogues of properties 1–6 for cascades.

Case 1. $k = 1$ or $j = 1$: Easy.

Case 2. $k = 2$: Divide the left path tree into a lower subpath and an upper subpath, each having $3.2^{j-1} - 2$ nodes, and two middle nodes. The right 2-cascade-deletion sequence is constructed by recursing on the lower and upper subpaths in turn and performing a 2-cascade involving the deleted node and the middle nodes for each deletion in the first recursive step. The sequence comprises $(3j-4)2^{j-1}-j+2 \geq (1/2)(3.2^j - 2)(j-3)$ apex 2-cascades and satisfies all the properties.

Case 3. $k \geq 3$ and $j \geq 2$: Let $s = 4kjB'_k(j-1)$. A *p,q-zigzag tree* is a tree that is constructed from a p-node left path tree by inserting a q-node left path tree as the right subtree of the leftmost node. We extend Lemma 8 to cascades and construct a right k-cascade-deletion sequence, say $\hat{S}$, for a $3,s$-zigzag tree that comprises $(1/2)(j-4)s+2j$ apex k-cascades. Each deletion in the sequence, except for the last two deletions, deletes the leftmost grandchild of the root. The proof divides the tree into $2j$ groups of $2k$ blocks each, each block, in turn, of size $B'_k(j-1)$, and recursively performs a right k-cascade-deletion sequence on each block, choosing the blocks in bottom-to-top order. A k-cascade is performed on the roots of the blocks within each group, yielding $2j$ extra cascades.

The tree is partitioned into $B'_{k-2}(s)$ blocks of size $s+3$ each. The global tree is constructed from the roots of the blocks and a right $(k-2)$-cascade-deletion sequence, say $\bar{S}$, satisfying properties 1–6 is recursively performed on it. The simulation of $\bar{S}$ on the original tree maintains the invariants A (with expression $s+3$ replacing $s+1$), C and D and the following modification of invariant B:

$\bar{\text{B}}$. Consider any block B that does not contain the leftmost node of the tree. Let x denote the root of B. Block B induces a connected subtree in the original tree. Further, if exactly b of the $(k-2)$-cascades performed so far on the global tree had x as their apex, then the block tree of B is a $(s-b+3), b$-zigzag tree.

The simulation of global tree operations other than apex $(k-2)$-cascades is as before. Consider an apex $(k-2)$-cascade in $\bar{S}$ involving a global tree subpath $x_1 - x_2 - \cdots - x_{2k-4}$, such that x_1 is the base of the cascade. Let y_1 and y_2 denote, respectively, the leftmost node in the block of x_1 and the left child of x_1. Let z_1 and z_2 denote, respectively, the parent and the grandparent of x_{2k-4} in the original tree. The global tree cascade is simulated on the original tree by creating the subpath $y_1 - y_2 - x_1 - x_2 - \cdots - x_{2k-4} - z_1 - z_2$ in the original tree, performing a k-cascade on this subpath, and finally deleting y_1 from the tree. We verify that the resulting sequence, say S, satisfies property 4:

$$\begin{aligned}
\#\ k\text{-cascades in } S &= \#\text{ local cascades} + \#\text{ global cascades} \\
&\geq ((1/2)(j-4)s + 2j)B'_{k-2}(s) + (1/2)(s-3)B'_{k-2}(s) \\
&> (1/2)(j-3)B'_k(j).
\end{aligned}$$

The rest of the properties of S are easy to check. This completes the proof of Lemma 7 for cascades. The theorem follows. ∎

3. An upper bound for the Deque Conjecture

In this section, we show that Splay takes $O((m+n)\bar{\alpha}(m+n))$ time to process a sequence of m deque operations on an n-node binary tree. We reduce a deque operation sequence to right cascade sequences on auxiliary trees and apply the upper bounds for cascades.

Define the *cost* of a deque operation or a right twist operation to be the number of single rotations performed by the operation.

The cost of a set of right cascades in a right twist sequence is given by:

Lemma 9. *Consider an arbitrary right twist sequence executed on an n-node binary tree. The total cost of any m right cascades in the sequence equals*

$$O((m+n)\alpha(m+n,n)).$$

Proof. Let $l = 2\alpha(m+n,n)+2$. Split each of the m right cascades into a sequence of right l-cascades followed by a sequence of at most $l-1$ rotations. By Theorem 3, the total number of right l-cascades is at most $8n\hat{\alpha}_{\lfloor l/2\rfloor}(n)$. This yields a bound of $(m+8n\hat{\alpha}_{\lfloor l/2\rfloor}(n))l$ for the number of rotations performed by the m right cascades. We bound $\hat{\alpha}_{\lfloor l/2\rfloor}(n)$ as follows:

$$\begin{aligned} A_{\alpha(m+n,n)+1}(\lfloor m/n\rfloor+2) &= A_{\alpha(m+n,n)}(A_{\alpha(m+n,n)+1}(\lfloor m/n\rfloor+1)) \\ &\geq A_1(A_{\alpha(m+n,n)}(\lfloor (m+n)/n\rfloor)) \\ &\geq n. \end{aligned}$$

Therefore $\hat{\alpha}_{\lfloor l/2\rfloor}(n) = \hat{\alpha}_{\alpha(m+n,n)+1}(n) \leq \lfloor m/n\rfloor + 2$. The lemma follows. ∎

Remark. Hart and Sharir [5] proved a result similar to Lemma 9 concerning sequences of certain path compression operations on rooted ordered trees. Their result can be derived from the analogue of Lemma 9 for turns by interpreting turns in a binary tree as path compressions on the rooted ordered tree representation of the binary tree. It is interesting that they also use ideas similar to blocks and global tree in their proof.

We estimate the cost of a sequence of deque operations performed at one end of a binary tree that also has left and right path rotations:

Lemma 10. *Consider an intermixed sequence of* Pops, Pushs, *left path rotations and right path rotations performed on an arbitrary n-node binary tree. The total cost of* Pop *operations equals $O((m+n)\bar{\alpha}(m+n))$, where m denotes the number of* Pops *and* Pushs *in the sequence.*

Proof. We simplify the sequence through a series of transformations without under-counting Pop rotations.

Simplification 1. *The first operation of the sequence is a* Pop.

Transformation. Delete the operations preceding the first Pop from the sequence and modify the initial tree by executing the deleted prefix of the sequence on it. ∎

Simplification 2. *The sequence does not contain* Push *operations.*

Transformation. For each Push operation, insert a node into the initial tree as the symmetric order successor of the last node that was popped before the Push.

The PUSH operation itself is implemented by just rotating its corresponding node to the root through right rotations. ∎

Define a PARTIALPOP to be a sequence of arbitrarily many right 2-turns performed on the leftmost node of a binary tree followed by deletion of the node.

Simplification 3. *The sequence consists of only* PARTIALPOPS *and left path rotations; the lemma is true if the total cost of* PARTIALPOP *operations equals* $O((m+n)\bar{\alpha}(m+n))$*, where* m *denotes the number of* PARTIALPOPS *in the sequence and* n *denotes the size of the initial tree.*

Transformation. Normalize the tree by rotating the nodes on the right path across the root into the left path and consider the resulting sequence. ∎

Simplification 4. *The sequence comprises only right cascades; the lemma is true if the total cost of any* m *cascades in the sequence equals* $O((m+n)\bar{\alpha}(m+n))$.

Transformation. Instead of deleting nodes at the end of PARTIALPOPS, rotate them upwards to the right path. ∎

The lemma follows from Simplification 4 and Lemma 9. ∎

The upper bound for the Deque Conjecture is given by:

Theorem 7. *The cost of performing an intermixed sequence of* m *deque operations on an arbitrary* n*-node binary tree using splay equals* $O((m+n)\bar{\alpha}(m+n))$.

Proof. Divide the sequence of operations into a series of epochs as follows: The first epoch comprises the first $\max\{\lfloor n/2 \rfloor, 1\}$ operations in the sequence. For all $i \geq 1$, if the tree contains k nodes at the end of epoch i, then epoch $i+1$ consists of the next $\max\{\lfloor k/2 \rfloor, 1\}$ operations in the sequence. The last epoch might consist of fewer operations than specified. It suffices to show that the cost of an epoch that starts with a k-node tree is $O(k\bar{\alpha}(k))$, since the sum of the sizes of the starting trees over all epochs is $O(m+n)$.

Consider an epoch whose initial tree, say T, has $k \geq 2$ nodes. Divide T into a left block of $\lfloor k/2 \rfloor$ nodes and a right block of $\lceil k/2 \rceil$ nodes. This partitioning ensures that neither block gets depleted before the epoch completes. The total cost of PUSHs and INJECTs is 0, since only rotations contribute to the operation cost. We show that the total cost of POPs is $O(k\bar{\alpha}(k))$. The same proof will apply to EJECTs.

A POP on T translates into a POP on the left block. The effect of a POP on the right block is a series of left path rotations. It is easy to see that the total number of single rotations performed by a POP is at most

$$\begin{aligned}&(\text{the number of single rotations performed by the POP on the left block}) +\\ &\qquad 2(\text{the number of left path rotations performed on the right block}) + 2.\end{aligned}$$

A PUSH operation on the tree propagates as a PUSH on the left block. An EJECT performs only right path rotations on the left block. An INJECT does not affect the left block. Hence by Lemma 10, the total number of single rotations performed by all the POPs on the left block equals $O(2\lfloor k/2 \rfloor \bar{\alpha}(2\lfloor k/2 \rfloor)) = O(k\bar{\alpha}(k))$. A left path rotation on the right block decreases the size of the left path of the block by 1. The initial size of this path is at most $\lceil k/2 \rceil$ and the size increases by at most 1 per deque operation. Therefore the total number of left path rotations performed on the right block due to POPs is at most $k+1$. This leads to an $O(k\bar{\alpha}(k))$ upper bound on the total cost of all the POPs performed during the epoch. ∎

4. New proofs of the Scanning Theorem

In the two subsections of this section, we describe a simple potential-based proof of the Scanning Theorem and an inductive proof that generalizes the theorem.

4.1. A simple potential-based proof

The proof rests on the observation that a certain subtree of the binary tree, called the *kernel tree,* which is mainly involved in the splay operations always has a very nice shape. As the nodes of the original tree are accessed using splays, the kernel tree evolves through insertions and deletions of nodes at the left end, and left path cascades caused by the splays. Each node of the kernel tree is assigned a *unimodal potential function,* that is, a potential function that initially steadily increases to a maximum value and then steadily decreases once the node has progressed sufficiently through the kernel tree. The nice shape of the kernel tree guarantees that most of the nodes involved in each splay are in their potential decrease phase, enabling their decrease in potentials to pay for all the rotations and the small increase in potentials of the nodes in their potential increase phase.

We need some definitions. A binary tree is called *rightist* if the depths of the leaves of the tree increase from left to right. The *left* and *right heights* of a binary tree are defined, respectively, to be the depths of the leftmost and rightmost nodes. The *right inner height* of a node x is defined to be the depth of the successor of x within x's subtree if x has a right subtree and 0 otherwise.

We introduce the *potential method* [10], used to bound the cost of a sequence of operations on a data structure. Each state of the data structure is assigned a real number, called its *potential.* The *amortized cost* of an operation is defined to be the actual cost of the operation *plus* the increase in potential caused by the operation. Then, the cost of a sequence of operations on the data structure is bounded by the total amortized cost of all the operations *plus* the total decrease in potential.

We are ready to describe the proof. At any time during the sequence of splays, the set of nodes in the current tree that have been involved in a splay rotation form a connected subtree of the tree, called the *kernel tree,* whose root coincides with the root of the right subtree of the tree. Initially, the kernel tree is empty. The sequence of splays on the the original tree propagates into an intermixed sequence of n PUSHs and n POPs on the kernel tree, where a PUSH inserts a new node at the bottom of the left path of the tree and a POP splays at and deletes the leftmost node of the tree. Our goal is to show that the cost of the sequence of operations on the kernel tree equals $O(n)$. The theorem would then follow immediately.

The argument focuses on the sequence of operations on the kernel tree. Since the kernel tree is created by a sequence of PUSHs and POPs, it satisfies the following two properties:

1. The subtrees hanging from the left and right paths of the tree are rightist.
2. If T_1 and T_2 are subtrees hanging from the left path with T_1 to the left of T_2, then $\text{rightheight}(T_1) \leq \text{leftheight}(T_2)$.

This can be easily shown using induction.

We use the following potential function. The potential of the kernel tree equals the sum of the potentials of all its nodes. The potential of a node consists of an *essential* component and a *nonessential* component. For any node x, let $ld(x)$ and $rih(x)$ denote, respectively, the left depth and the right inner height of x. The essential potential of x equals $\min\{\lceil \log ld(x) \rceil, rih(x)\}$ unless x is on the right path in which case its essential potential equals 0. The essential potential of a node is a unimodal function of time, since the potential first monotonely increases from 0 until the node's right inner height overtakes the logarithm of its left depth and then monotonely decreases. The nonessential potential of x equals 2 units if x is not on the right path and x's left child has the same right inner height as x, and equals 0 otherwise.

We compute the amortized cost of kernel tree operations. PUSH has amortized cost 2, to provide for the nonessential potential that may be needed by the parent of the inserted node. Consider a POP. Let x denote the lowest node on the left path such that $\lceil \log ld(x) \rceil \leq rih(x)$. Every double rotation of a splay step that involves two nodes with identical right inner heights is paid for using the nonessential potential of the node leaving the left path. The number of remaining double rotations is at most $\lfloor ld(x)/2 \rfloor + \lceil \log(ld(x)+1) \rceil$. The first term accounts for the double rotations involving two proper ancestors of x and the second term accounts for the double rotations involving the descendents of x. Each latter category rotation increases the potential by at most 1, contributing to a net increase of at most $\lceil \log(ld(x)+1) \rceil$ units of potential. The halving of the left depths of the ancestors of x caused by the splay operation decreases the potential by exactly $ld(x) - 1$. The amortized cost of POP is therefore bounded by

$$\lfloor ld(x)/2 \rfloor + 2\lceil \log(ld(x)+1) \rceil - (ld(x) - 1) \leq 5.$$

We conclude that at most $7n$ double rotations, hence at most $15n$ single rotations, are performed by the sequence of operations on the kernel tree. This proves the Scanning Theorem.

4.2. An inductive proof

In this section, we describe an inductive proof of a generalization of the Scanning Theorem. The proof technique is similar to the method used to derive the upper bounds in Section 2.1. The binary tree is partitioned into blocks of constant size so that the total number of single rotations within blocks is linear. The induction is applied to a global tree consisting of a constant fraction of the tree nodes. Since a splay on the original tree translates into a much weaker rotation on the global tree, we have to incorporate the strength of the rotations into the inductive hypothesis.

We state the result. For any positive integer k and real number d, such that $1 \leq d \leq n$, a right k-twist is called *d-shallow* if the lowest node involved in the twist has a left depth of at most dk. Let $S^{(d)}(n)$ denote the maximum number of single rotations performed by d-shallow right twists in any right twist sequence executed on an n-node binary tree. We prove that $S^{(d)}(n) = O(dn)$. The Scanning Theorem follows from $S^{(2)}(n) = O(n)$.

We estimate the number of d-shallow right twists in a right twist sequence:

Lemma 11. *For any $d \geq 1$, the total number of d-shallow right twists in any right twist sequence is at most $4dn$.*

Proof. Consider any d-shallow right twist that rotates a sequence of edges, say $[x_1, y_1], [x_2, y_2], \ldots, [x_k, y_k]$, such that the left depths of the sequence of nodes

$x_k, y_k, x_{k-1}, y_{k-1}, \ldots, x_1, y_1$ is nonincreasing. Let $ld(z)$ and $ld'(z)$ denote, respectively, the left depths of any node z before and after the twist. For all $i \in [\lceil k/2 \rceil, k]$, we have

$$\frac{ld'(x_i)}{ld(x_i)} = 1 - \frac{i}{ld(x_i)} \leq 1 - \frac{i}{kd} \leq 1 - \frac{1}{2d}.$$

In order to pay unit cost for a twist, we charge each node x_i, such that $i \in [\lceil k/2 \rceil, k]$, $\min\{2d/ld(x_i), 1\}$ debits. Let us prove that the total charge is at least 1. If $ld(x_{\lceil k/2 \rceil}) \leq 2d$, then $x_{\lceil k/2 \rceil}$ is charged 1 debit. Otherwise, we have $1 \geq 2d/ld(x_i) \geq 2/k$ for all $i \geq \lceil k/2 \rceil$. Since $\lfloor k/2 \rfloor + 1$ nodes are each charged at least $2/k$ debits, the net charge to all the nodes is at least 1.

Now, we bound the total charge to a node over the entire sequence. Call a node *deep* if its left depth is greater than $2d$ and *shallow* otherwise. Suppose that a node receives a sequence of charges $2d/L_k, 2d/L_{k-1}, \ldots, 2d/L_0$ while it is deep. Then

$$L_i > \frac{2d}{(1 - 1/2d)^i} \quad \text{for all } i \geq 0.$$

Therefore the total charge to a node while it remains deep is at most

$$(1 - 1/2d)^k + (1 - 1/2d)^{k-1} + \cdots + 1 \leq 2d.$$

A node receives at most $2d$ debits while it is shallow. This implies that any node is charged at most $4d$ debits, giving a bound of $4dn$ for the total number of d-shallow right twists. ∎

The upper bound for $S^{(d)}(n)$ is given by:

Theorem 8. *$S^{(d)}(n) \leq 87dn$ for all $d \geq 1$ and $n \geq 1$.*

Proof. The proof uses induction on n.

Case 1. $n \leq 174d$: $S^{(d)}(n) \leq \binom{n}{2} \leq 87dn$.

Case 2. $n > 174d$: Divide the tree into a sequence of $\lceil n/K \rceil$ blocks such that each block except the first contains exactly $K = 29d$ nodes. The first block may contain fewer nodes. In each block except the first, the nodes with preorder numbers 1 to $4d$ within the block are global. The first block does not contain any global nodes. Notice that the global nodes of a block form a connected subtree within the block whose root coincides with the root of the block. Further, if the left path of the block contains more than $4d$ nodes then all global nodes lie on the left path of the block. Otherwise all nodes on the left path of the block are global. The global nodes in the tree form a global tree as in the previous upper bound constructions. The size of the global tree is at most $n/7.25$.

We analyze the effect of a original tree rotation on the global tree. An interblock right single rotation translates into a corresponding rotation on the global tree if both nodes of the rotation are global. Otherwise the global tree is not affected. The analysis of an intrablock right single rotation involves the following cases:

Local-local: The global tree is unaffected.

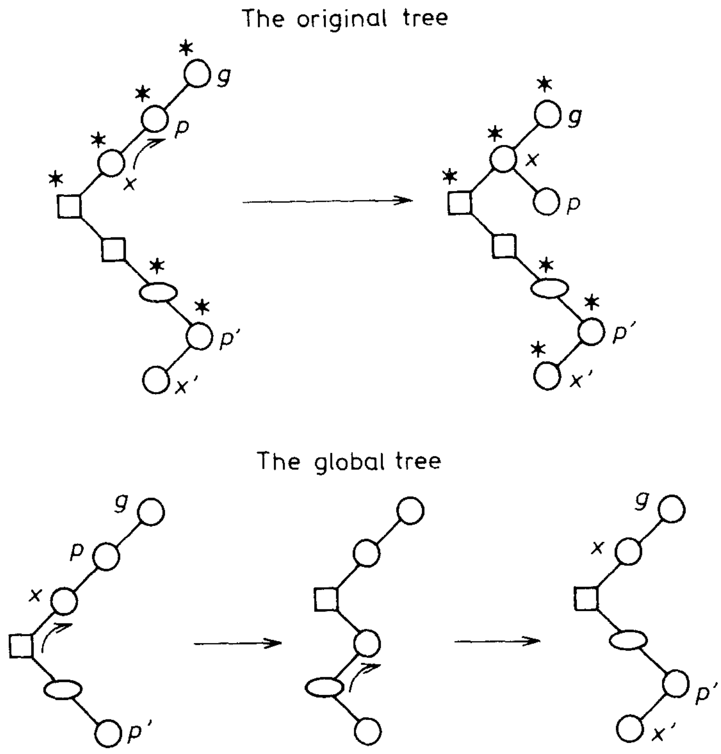

Fig. 8. The transfer of global role in an intrablock global-global rotation. Circles denote the nodes of the block. The starred nodes of the original tree are the global nodes

Local-global: Again, the global tree is not affected, but the global role is transferred from the global node to the local node.

Global-global: Let $[x,p]$ denote the rotated edge such that p is the parent of x. If the left subtree of x within the block contains only global nodes, then the rotation simply propagates to the global tree. Otherwise, p's global role is transferred to the node, say x', in p's block that had preorder number $4d+1$ initially. Let p' denote the lowest ancestor of x' in the original tree which is global. The effect of the transfer of global role on the structure of the global tree is to contract edge $[x,p]$ and add a new edge $[x',p']$. We show that the same transformation is realizable through a series of right single rotations in the global tree. These rotations are performed by traversing the path from p to p' in the global tree as follows (See Figure 8):

Start at edge $[p,x]$ and repeat the following operation until the last edge on the path is reached: If the next edge on the path is a left edge, move to the next edge; otherwise, rotate the current edge and move to the next edge after the rotation. Finally, if x' belongs to the right subtree of p' in the original tree, rotate the last global tree edge traversed.

Remark. The operation performs all the rotations within the subtree of the global tree rooted at x. This is seen as follows. If $x = p'$, then no rotations are performed on the global tree. Otherwise, p' lies in the left subtree of x. Hence the successor of edge $[p, x]$ on the global tree path from p to p' is a left edge. This implies that the operation does not rotate edge $[p, x]$. Therefore all the rotations performed by the operation occur in the subtree of the global tree rooted at x.

At any during this traversal, contracting the current edge results in a tree that is identical to the tree obtained by contracting the edge $[x, p]$ in the initial tree, so it follows that the above series of rotations on the global tree correctly simulates the rotation of edge $[x, p]$.

In summary, a right single rotation of an edge $[x, p]$, such that p is the parent of x, either does not affect the global tree, or translates into a rotation of the edge $[x, p]$ in the global tree, or translates into a sequence of right single rotations on the subtree of the global tree rooted at x. The rotation is called *global* if it results in a rotation of the corresponding edge in the global tree and *local* otherwise.

Consider the effect of a right twist in the original tree on the global tree. The sequence of single rotations on the global tree caused by the right twist comprises *l-rotations,* caused by local rotations in the twist, and *g-rotations,* caused by global rotations in the twist. The nodes involved in any l-rotation are distinct from the nodes involved in any previous g-rotation, hence we may transform the sequence of global tree rotations by moving each l-rotation before all the g-rotations without altering the net effect of the sequence on the global tree. The suffix of the sequence consisting of all the g-rotations defines a *global twist* on the global tree. In summary, the effect of a right twist in the original tree on the global tree is a right rotation followed by a global twist corresponding to the subsequence of global rotations in the twist.

We estimate the number of single rotations performed by d-shallow twists in a right twist sequence executed on the tree. Consider any d-shallow twist in the sequence. Define the *left path* of the twist to be the left path resulting from the contraction of the right edges on the access path of the lowest node involved in the twist. We classify the right single rotations performed by the twist as follows:

Type 1. Local, interblock rotation in which the top node is global: There is at most one Type-1 rotation per twist because the left subtree of the bottom node of the rotation consists of only local nodes.

Type 2. Local, interblock rotation in which the top node is local: The top node lies on the left path of its block and, since the node is local, it has $4d$ global ancestors within the block that lie on the left path of the twist. Notice that the top nodes of different Type-2 rotations belong to different blocks. Thus, if k_2 denotes the number of Type-2 rotations performed by the twist, then the left path of the twist contains at least $(4d + 1)k_2$ edges. Since the number of edges on the left path of the twist is bounded by dk, we obtain that $k_2 \leq \lfloor k/4 \rfloor$.

Type 3. Local, intrablock rotation: For each Type-3 rotation, charge $(8/3)$ debits to the block in which the rotation is performed. If the number of Type-3 rotations is at least $(3k - 4)/8$, the total charges to the blocks *plus* a charge of $(4/3)$ debits to the twist itself pays for all the rotations performed by the twist.

Type 4. Global rotation: Only the situation where the number of Type-3 rotations is less than $(3k-4)/8$ needs to be considered. In this case at least

$$k-1-\lfloor k/4\rfloor-(\lceil(3k-4)/8\rceil-1)=k-\lfloor k/4\rfloor-\lfloor(3k+3)/8\rfloor\geq 3k/8$$

global rotations are performed. Therefore the global twist performs at least $3k/8$ rotations on the global tree, and it is $(8d/3)$-shallow. If we charge each such global twist $(4/3)$ times the actual cost, then all the rotations can be paid for. This is seen as follows. Let k_3 and k_4 denote, respectively, the number of Type-3 and Type-4 rotations. Then, $k_3+k_4\geq 3k/4-1$. The total charge is $8k_3/3+4/3+4k_4/3$ which is minimized when $k_3=0$. When $k_3=0$, the total charge is at least $4/3+(4/3)(3k/4-1)\geq k$.

Since each d-shallow twist is charged at most $4/3$ debits, the total charge to all the d-shallow twists is at most $16dn/3$ by Lemma 11. The total charge to a block of size s is at most $8\binom{s}{2}/3$. It follows that the total charge to all the blocks is at most $4nK/3\leq 116dn/3$. By the inductive hypothesis, the total charge to all the $(8d/3)$-shallow global twists is at most $(4/3)(87)(8d/3)(n/7.25)=128dn/3$. Therefore the sum total of all the charges is bounded by $16dn/3+116dn/3+128dn/3\leq 87dn$, completing the induction step. ∎

Acknowledgements. I thank my advisor Richard Cole for discussions on this work and on his work on the Finger Conjecture, and for carefully reading many drafts of this paper. I thank Bud Mishra for his comments on the paper. I thank Bob Tarjan for calling my attention to the works of Joan Lucas and S. Rao Kosaraju.

References

[1] R. Cole: On the Dynamic Finger Conjecture for Splay Trees. In Proc. 22nd ACM STOC, 1990, 8–17.

[2] R. Cole: On the Dynamic Finger Conjecture for Splay Trees. Part II: Finger searching. Courant Institute Technical Report No. **472**, 1989.

[3] R. Cole, B. Mishra, J. Schmidt, and A. Siegel: On the Dynamic Finger Conjecture for Splay Trees. Part I: Splay-sorting $\log n$-block sequences. Courant Institute Technical Report No. **471**, 1989.

[4] K. Culik II, and D. Wood: A note on some tree similarity measures. *Info. Proc. Lett.* **15** (1982), 39–42.

[5] S. Hart, and M. Sharir: Nonlinearity of Davenport-Shinzel sequences and of generalized path compression schemes. *Combinatorica* **6** (1986), 151–177.

[6] J. M. Lucas: Arbitrary splitting in Splay Trees. Rutgers University Tech. Rept. No. **TR–234**, June 1988.

[7] D. D. Sleator, and R. E. Tarjan: Self-adjusting binary search trees. *J. ACM* **32** (1985), 652–686.

[8] D. D. Sleator, R. E. Tarjan, and W. P. Thurston: Rotation distance, Triangulations, and Hyperbolic Geometry. *J. Amer. Math. Soc.* **1** (1988), 647–681.

[9] R. E. Tarjan: Sequential access in Splay Trees takes linear time. *Combinatorica* **5** (1985), 367–378.

[10] R. E. TARJAN: Amortized computational complexity. *SIAM J. Appl. Discrete Meth.* **6** (1985), 306–318.

[11] R. WILBER: Lower bounds for accessing binary search trees with rotations. *SIAM J. on Computing* **18** (1989), 56–67.

Rajamani Sundar

Computer Science Department
Princeton University
Princeton, NJ 08544
U. S. A.
sundar@cs.princeton.edu