A Lower Bound Framework for Binary Search Trees with Rotations

Jonathan Derryberry **Daniel Dominic Sleator**
Chengwen Chris Wang

November 22, 2005
CMU-CS-05-187

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

This paper considers the problem of bounding below the cost of accessing a sequence of keys in a binary search tree. We develop a lower bound framework for this problem that applies to any binary search tree algorithm, including self-adjusting and offline ones. This new framework can be used to derive two previously known lower bounds.

This research was sponsored by National Science Foundation (NSF) grant no. CCR-0122581.

Keywords: Dynamic Optimality, Binary Search Tree, Splay Tree, Competitive Algorithms, Lower Bound

1 Introduction

Binary search trees (BSTs) are a common class of data structures for storing a set of keys from a totally ordered universe. BSTs support, at a minimum, the operations of accessing, insertion, and deletion of keys. A number of *self-adjusting* BST data structures have been proposed in which the tree is restructured via rotations in response to the pattern of accesses [ST85, DHIP04, WDS06]. To better understand the limits of performance of self-adjusting BSTs, it is useful to consider the case in which the set of keys is static, so that accessing is the only operation allowed.

A well-known open problem is that of devising a *competitive* BST algorithm. This would be an online self-adjusting BST algorithm whose performance on a sequence of accesses is within a constant factor of all other BST algorithms, even offline ones that know all future accesses. The Dynamic Optimality Conjecture for splay trees [ST85] asserts that splay trees are competitive in this sense.

One approach to this problem has been to develop lower bounds on the cost of accesses in a BST. Such a lower bound takes as input a sequence of accesses, and returns a number which is a lower limit on the cost of any BST algorithm for handling that sequence of accesses. If, for example, splay trees were shown to cost at most a constant factor more than some lower bound, then this would prove the Dynamic Optimality Conjecture.

In Section 2 of this paper we define the model carefully. In Section 3 we prove our new *Rectangle Cover Lower Bound*. In Section 4 we show how to use our bound to derive two previously known lower bounds. Section 5 contains conclusions and future work.

2 The Model

Let the sequence of accesses to be processed by a BST algorithm be $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_m$. Each access σ_i is a key in the tree, and the requested keys must be accessed in the specified order.[1] Traditionally, a BST algorithm processes an access by walking down from the root of the tree until the accessed node is reached. The cost is the number of nodes on that path. The algorithm is also allowed (whenever it wants) to restructure the tree by doing rotations, at a cost of 1 per rotation.

There is an alternative model proposed by Wilber [Wil89]. In this model, the accessed node must be moved to the root of the tree by doing rotations. Of course, other rotations may be done as well, which may affect future accesses. The cost is exactly the number of rotations done plus 1. A BST algorithm that does this is called a *standard search algorithm*. This is the model we use in this paper. Because rotations are invertible, the cost in the standard model is at most twice that in the traditional model.

Note that the cost for a BST to process a sequence of accesses will clearly depend on the initial tree. Our lower bound is applicable to any BST algorithm on any initial tree.

[1] We often refer to the node containing key x as x. We assume the tree will never contain two equal keys, so this represents no ambiguity.

3 The Lower Bound

It is useful to view the access sequence as a set of points in two dimensions: the x-coordinate is time, and the y-coordinate is the key space. (See Figure 1.) In this view, access σ_i is mapped to the point (i, σ_i). Let P denote the set of points generated in this fashion by the access sequence σ.

A *box* is an axis-aligned rectangle whose opposing corners are points in P. A box is either *up* or *down*. A box is up if its lower-left and upper-right corners are points in P. Similarly, a box is down if its upper-left and lower-right corners are in P. A box also contains a *divider* which is simply a horizontal line across the box. To avoid ambiguity, we require that the y-coordinate of the divider be distinct from all the keys. Two boxes *conflict* if they are both up or both down, and the intersection of the two boxes contains part or all of both dividers.

Theorem 1. *Let σ be a sequence of accesses, let P be the corresponding point set, and let B be a set of pairwise non-conflicting boxes for P. Then the number of rotations required by any standard BST search algorithm to process σ is at least $|B|$.*

Before we prove the theorem, we need a few preliminaries. Let LCA(x, y) (where x and y are nodes in a BST) denote the least common ancestor of x and y.

Lemma 1. *Let a and b be two distinct nodes in a binary search tree T with $a < b$. Let p and c be two nodes in T, such that p is the parent of c. Suppose a rotation of edge (c, p) is performed and as a result of this rotation, LCA(a, b) changes. It must follow that:*

(1) $a \leq c \leq b$
(2) $a \leq p \leq b$
(3) LCA$(a, b) = p$ *before the rotation*
(4) LCA$(a, b) = c$ *after the rotation*

Proof: Let Π be a permutation of the keys of tree T sorted in increasing order of depth with ties broken arbitrarily. Notice that, starting from an empty tree, inserting the keys of Π in order using the standard insertion algorithm produces T. Moreover, notice that LCA(a, b) is the first node in Π whose key k satisfies $a \leq k \leq b$. We can assume without loss of generality that c immediately follows p in Π.[2] If we swap c and p in the permutation, then the tree that results from the new permutation is exactly T modified by doing a rotation of edge (c, p).

We can now use this alternative characterization of a rotation and of LCA(a, b) to determine the conditions that must hold for LCA(a, b) to change. Since p and c are adjacent in Π and a change in LCA(a, b) implies that Π's first key from the range $[a, b]$ changes, both p and c must be in the range $[a, b]$ (conditions (1) and (2)), and LCA(a, b) must change from p to c (conditions (3) and (4)). $\square$

[2] Actually, even without this assumption, it is still the case that swapping p and c in Π results in T with edge (c, p) rotated.

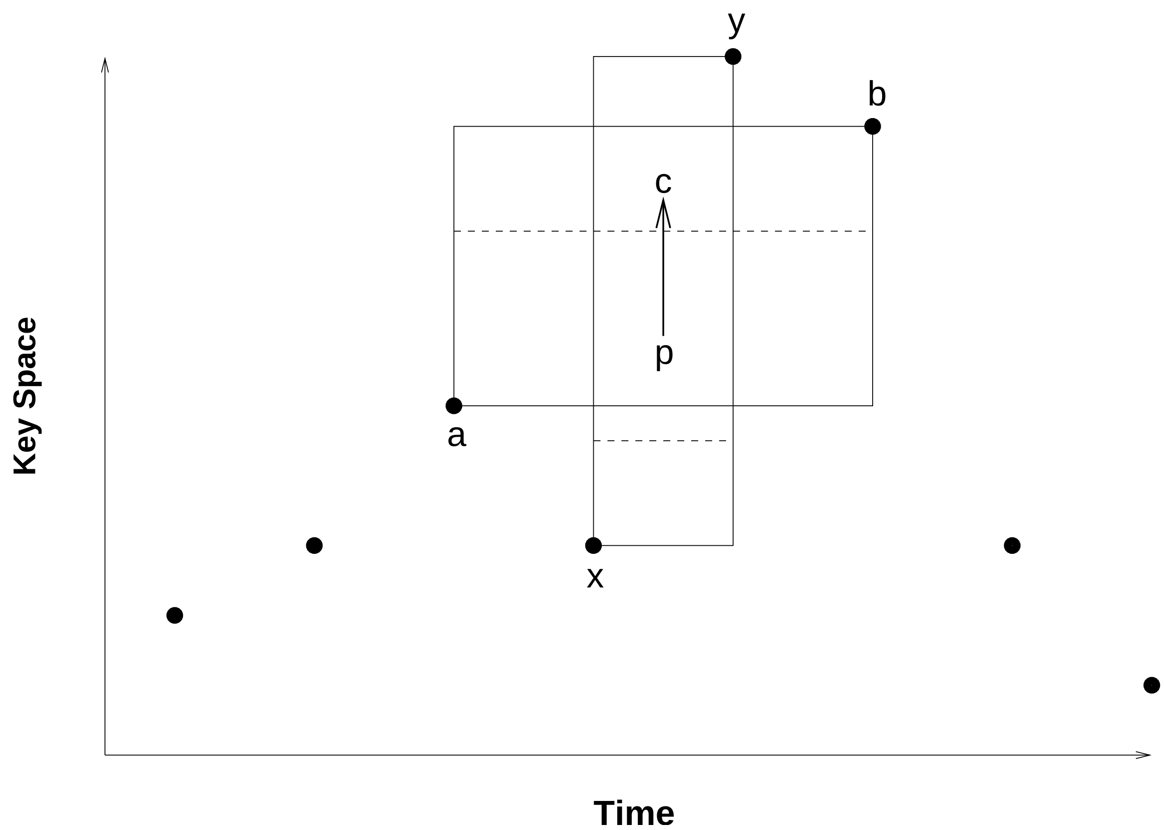

Figure 1: The 2-D representation of an access sequence and some boxes.

This diagram shows a sequence of eight accesses. Each access is a dot. Time is on the horizontal axis. The third access is to point a's second coordinate, the fourth to x's second coordinate, etc.

Two up-boxes are shown: (a, b) and (x, y). The dividers for these boxes are shown as horizontal dashed lines. They are non-conflicting because their intersection contains only one divider.

The rotation of edge (c, p) in the tree is illustrated by the arrow connecting labels p and c. This rotation is the first one in time between a and b that causes $\text{LCA}(a, b)$ to move across the divider of box (a, b). Therefore, it is the rotation that is associated with the box.

Proof of Theorem 1: For each box B_k in B, we are going to associate a rotation R_k done by the BST algorithm processing the access sequence. Two boxes will not be associated with the same rotation. This will prove that the number of rotations done is at least the number of boxes.

Consider an up-box B_k. Let (i, σ_i) be its lower-left corner and (j, σ_j) be its upper-right corner. After accessing σ_i, we know σ_i is at the root of the tree, and after accessing σ_j, we know σ_j is at the root of the tree. Consider the time interval spanned by the box. At the beginning of this interval LCA$(\sigma_i, \sigma_j) = \sigma_i$, and at the end LCA$(\sigma_i, \sigma_j) = \sigma_j$. Thus, LCA$(\sigma_i, \sigma_j)$ must cross from below the divider to above the divider at some time in this interval. The rotation R_k we associate with box B_k is the first one that moves the LCA across the divider.

As shown by Lemma 1, LCA(a, b) only moves by virtue of a rotation between two keys that are inside range $[a, b]$, that is, two keys that are inside the box. So the rotation R_k can be added to our diagram as a vertical line inside the box B_k, crossing the divider. (See Figure 1.) If R_k were inside another up-box B_l, and R_k crossed B_l's divider, the intersection of B_k and B_l would contain R_k and thus contain two dividers. Therefore, the boxes would conflict.

Of course, the situation is analogous for down-boxes. Note that the rotation associated with an up-box is a left-rotation, and the rotation associated with a down-box is a right-rotation, so an up-box and a down-box can never be associated with the same rotation. $\square$

4 Applications

In this section, we show that the Rectangle Cover Lower Bound is a generalization of two previously known lower bounds.

4.1 New Proof of the Interleave Lower Bound

We begin by describing the lower bound, using the description from [WDS06], which we reproduce here for the reader's convenience.

Given an initial tree T_0 and an m-element access sequence σ, for any BST algorithm satisfying these requests there is a cost, as defined in Section 1. Denote by OPT(T_0, σ) the minimum cost any BST algorithm must incur to satisfy these requests starting with initial tree T_0. Wilber [Wil89] derived a lower bound on OPT(T_0, σ), and this was modified to be the *interleave bound* by Demaine *et al.* [DHIP04].

Let IB(R, σ) denote the interleave lower bound on the cost of accessing the sequence σ, where R is a BST over the same set of keys as T_0. Define IB$(R, \sigma) = \sum_{v \in R}$ IB(R, σ, v), where for each node v, IB(R, σ, v) is defined as follows. First, restrict σ to the set of nodes in the subtree of R rooted at v (including v). Next, label each access in this restricted σ as either "left" (or "right") depending on whether the accessed element is in the left subtree (including v) or right subtree of v. Now, IB(R, σ, v) is the number of times the labels switch.

Theorem 2. (Interleave Lower Bound[3])

$$\mathit{OPT}(T_0, \sigma) \geq \mathit{IB}(R, \sigma) + m$$

Proof: To prove this theorem, it suffices to show that we can place an up-box for each left-to-right label change and a down-box for each right-to-left label change, such that no two boxes conflict. Because up-boxes and down-boxes do not interfere, it suffices to show that no two up-boxes for left-to-right label changes conflict.

Given a sequence of queries $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_m$, whenever a node v's label changes from left to right at time r, we place a box in the following way. Let σ_l be the last query in v's left subtree at time l, and σ_r be the query that induces the switch at v. We place a box whose opposing corners are at (l, σ_l) and (r, σ_r), and place its divider at $v - \epsilon$, where ϵ is very small. Let this box be described by the tuple $((l, \sigma_l), (r, \sigma_r), v - \epsilon)$.

Consider any two such up-boxes, $B = ((l, \sigma_l), (r, \sigma_r), v - \epsilon)$ and $B' = ((l', \sigma'_l), (r', \sigma'_r), v' - \epsilon)$. One of the following cases applies (See Figure 2):

1. If $v = v'$ then the boxes do not overlap in time, so they do not conflict.
2. If neither v nor v' is an ancestor of the other, then the ranges (σ_l, σ_r) and (σ'_l, σ'_r) are disjoint.
3. If v' is an ancestor of v and $v < v'$, then $\sigma_l < \sigma_r \leq v' - 1 < v' - \epsilon$, so that B does not contain B''s divider.
4. If v' is an ancestor of v and $v > v'$, then $\sigma_r > \sigma_l \geq v' + 1 > v' - \epsilon$, so that B does not contain B''s divider.

Cases where v is an ancestor of v' are similar to cases 3 and 4. Hence, B and B' do not conflict.

The above argument proves a lower bound of $\mathrm{IB}(P, \sigma)$ on the number of rotations required to access σ. Each access incurs an additional cost of 1, which is adds m to the lower bound. □

4.2 New Proof of Wilber's Second Lower Bound

Wilber's second lower bound [Wil89] is more complicated to explain, and was conjectured by Wilber to be better than his first bound. Informally, the bound can be defined as follows. For each access σ_i, we travel backwards in time starting at time i and keep track of 2 closest accesses in key space to σ_i, one to a larger key than σ_i and one to a smaller key than σ_i. Wilber's second lower bound for access σ_i is the number of times a new 'record closeness' occurs on a different side from where the last record that was set. (We stress that the 2 records are on different sides of σ_i and are independent – the record on one side may remain far away from σ_i in key space while the other comes closer and closer to σ_i.) The lower bound for the sequence σ is the sum of this quantity over all accesses, using each access as a starting point.

[3] This lower bound, the lower bound of Wilber [Wil89] and that of Demaine *et al* [DHIP04] are all identical up to a constant factor. For readers familiar with these papers, our IB() is the same that in [DHIP04] and our OPT() is the same as that in [Wil89].

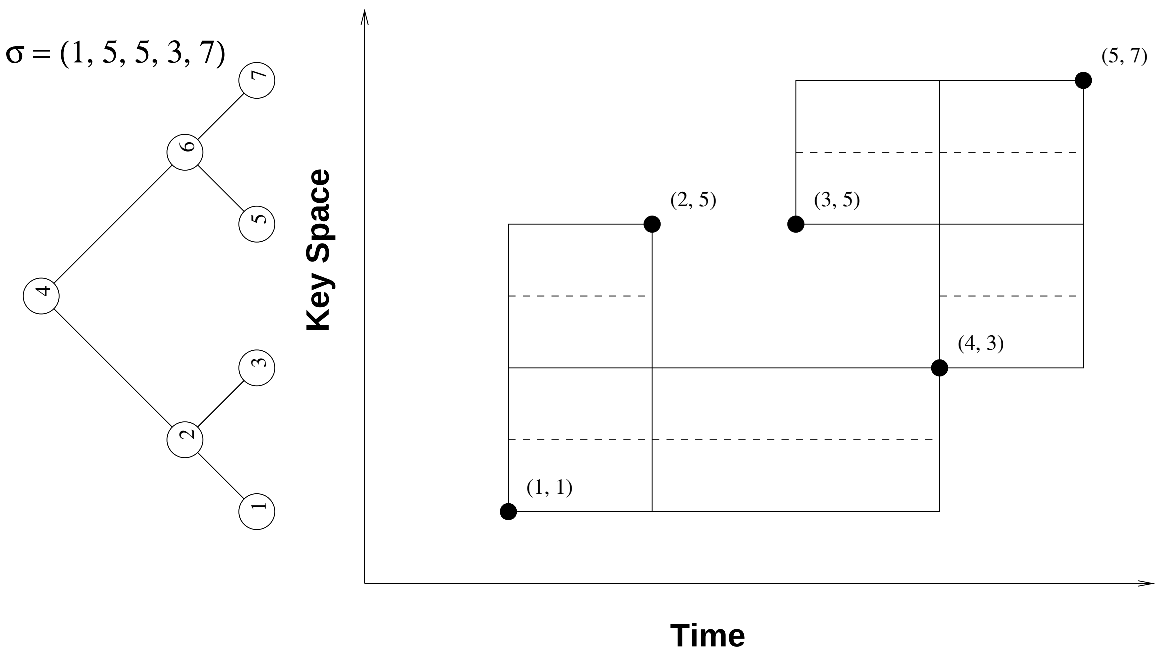

Figure 2: This figure shows the left-to-right switches (and the corresponding up-boxes) induced by the sequence $(1, 5, 5, 3, 7)$. Below is a set of examples for each case in the proof of the interleave lower bound. The boxes $((1,1),(2,5),4-\epsilon)$ and $((4,3),(5,7),4-\epsilon)$ are an example of case 1. The boxes $((1,1),(4,3),2-\epsilon)$ and $((3,5),(5,7),6-\epsilon)$ are an example of case 2. The boxes $((1,1),(2,5),4-\epsilon)$ and $((1,1),(4,3),2-\epsilon)$ are an example of case 3. The boxes $((3,5),(5,7),6-\epsilon)$ and $((4,3),(5,7),4-\epsilon)$ are an example of case 4.

More formally, given a sequence of queries $\sigma = \sigma_1, \ldots, \sigma_m$, for each access σ_i, find a maximum-sized subsequence of *crossing accesses* $\sigma_{c_1}, \ldots, \sigma_{c_{\kappa(i)+1}}$ and *inside accesses* $\sigma_{b_1}, \ldots, \sigma_{b_{\kappa(i)}}$ such that for all $j \in [1, \kappa(i)]$:

1. To ensure that the c_j's and b_j's go backwards in time,
$$c_{j+1} < b_j \leq c_j \leq c_1 = i - 1.$$
2. To ensure that successive c_j's are on different sides of σ_i,
$$(\sigma_{c_{j+1}} - \sigma_i)(\sigma_{c_j} - \sigma_i) \leq 0.$$
3. To help ensure that each σ_{c_j} is a new record, for $j < \kappa(i)$
$$|\sigma_{c_{j+2}} - \sigma_i| < |\sigma_{b_j} - \sigma_i|.$$
4. To define b_j and further ensure that each σ_{c_j} is a new record, for $j \geq 1$,
$$\begin{aligned} S_j &= \{c_{j+1} < k < i \mid (\sigma_{c_j} - \sigma_i)(\sigma_k - \sigma_i) > 0\} \\ b_j &= \operatorname*{argmin}_{k \in S_j} |\sigma_k - \sigma_i|, \end{aligned}$$
where by convention we let b_j be the minimum possible such value if multiple $k \in S_j$ are equally close to σ_i. (Note that this does not conflict with the assertion that $b_j \leq c_j$.)

Theorem 3. *The number of rotations required to access σ is at least $\sum_{1 \leq i \leq m} \kappa(i)$ (so the total cost of σ is at least $m + \sum_{1 \leq i \leq m} \kappa(i)$).*

Proof: We can prove Wilber's second lower bound using our box lower bound as follows. For each access σ_i, find maximum-sized sequences $\sigma_{c_1}, \ldots, \sigma_{c_{\kappa(i)+1}}$ and $\sigma_{b_1}, \ldots, \sigma_{b_{\kappa(i)}}$ satisfying properties 1-4 above. For every pair of accesses $(\sigma_{c_{j+1}}, \sigma_{b_j})$, draw the corresponding box and place the divider at $\sigma_i + \epsilon$ if the box is oriented up or $\sigma_i - \epsilon$ if the box is oriented down, where ϵ is very small.

Note that any two boxes oriented in the same direction created using σ_i's sequence of $\kappa(i)$ boxes cannot conflict because they are disjoint along the time axis by properties 1 and 2. Now, consider two boxes $B = ((c_k, \sigma_{c_k}), (b_{k-1}, \sigma_{b_{k-1}}), \sigma_i + \epsilon)$ and $B' = ((c'_{k'}, \sigma_{c'_{k'}}), (b'_{k'-1}, \sigma_{b'_{k'-1}}), \sigma_{i'} + \epsilon)$ for $i' \neq i$, such that without loss of generality $i < i'$ and both boxes are up-boxes. If $c'_{k'} \geq i$, then B and B' do not overlap in time, so they do not conflict. Else, $c'_{k'} < i$ and by the properties of $\sigma_{i'}$'s sequence of boxes, either $\sigma_{c'_{k'}} < \sigma_i$ so that $\sigma_{b'_{k'-1}} \leq \sigma_i < \sigma_i + \epsilon$ by property 4, or $\sigma_{c'_{k'}} \geq \sigma_i$ so that $\sigma_{c'_{k'}} > \sigma_i + \epsilon$ by property 3 because $\sigma_{c'_{k'}}$ must be strictly closer to $\sigma_{i'}$ than $\sigma_{i'}$, which in this case must be strictly less than $\sigma_{i'}$. Thus, B and B' do not conflict. $\square$

5 Conclusions and Open Problems

It is straightforward to extend the Rectangle Cover Lower Bound to handle insertions and deletions as long as insertions and deletions have to pay for accessing the relevant node, and each deletion has to rotate the relevant node to a leaf. All that needs to be done is to add a point to the diagram for each insertion and deletion. Everything else (i.e., the construction of the boxes) is the same.

Another generalization to the original framework is possible. We can allow the divider for each up-box to be a monotonically non-decreasing function of time, which starts at the left boundary of the box and ends on the right boundary of the box. (For each down-box the divider would be a non-increasing function.) We do not know if this idea leads to an improvement.

Exhaustive search has shown that the Rectangle Cover Bound is not exact.[4] However, it is more general than Wilber's first lower bound, so we know it is within a factor of $O(\log\log n)$ of the cost of the optimal BST algorithm. Perhaps the Rectangle Cover Bound is within a constant factor of optimality. Proving this is an open problem.

Another open problem regarding the Rectangle Cover Lower Bound is whether it can be computed in polynomial time. If not, another interesting problem is whether it can be approximated to within a constant factor, or some factor that is $o(\log\log n)$ (Wilber's first lower bound approximates it to within $O(\log\log n)$). One potentially useful observation is that you can assume without loss of generality that no box contains (inside or on its boundary) any other access point.

Lower bounds sometimes lead to new algorithms. Examples of this are the development of new algorithms for binary search trees based on Wilber's first lower bound [Wil89, DHIP04, WDS06]. There is the possibility that our lower bound formulation could be used in this fashion.

Returning to the original motivation for this research, the problem of finding a $o(\log\log n)$-competitive online (or even offline) BST remains open. Another problem is devising an online comparison-based data structure (that does not necessarily adhere to the BST model) that is within a factor of $o(\log\log n)$ of the optimal offline BST. For example, Iacono devised a non-BST comparison-based data structure called the *unified structure* that exploits temporal and spatial locality of accesses with better bounds than have been proven for any BST [Iac01], but his data structure is only $O(\log n)$-competitive.[5]

References

[DHIP04] Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Pătraşcu. Dynamic Optimality–Almost. *FOCS*, 2004.

[4] There are sequences of six accesses for which the bound is too low by one rotation

[5] Consider the sequence of n accesses $0, \sqrt{n}, 2\sqrt{n}, \ldots, (\sqrt{n}-1)\sqrt{n}, 0, \sqrt{n}, 2\sqrt{n}, \ldots, (\sqrt{n}-1)\sqrt{n}, \ldots$. The unified structure requires $\Omega(\log n)$ time per access while splay trees require only $O(1)$ time per access. To see that splay trees require only $O(1)$ time per access for this sequence, notice that this first round of $\sqrt{n}$ accesses costs $O(n)$ by the Dynamic Finger Theorem. After the first round, at most $2\sqrt{n}$ nodes remain on the left spine and the nodes $0, \sqrt{n}, 2\sqrt{n}, \ldots, (\sqrt{n}-1)\sqrt{n}$ are all among them. Thus, all following rounds will not touch any nodes that were not on the left spine at the end of first round. Applying the Dynamic Finger Theorem on this smaller tree with at most $2\sqrt{n}$ nodes shows that successive rounds cost only $O(\sqrt{n})$.

[Iac01] John Iacono. Alternatives to splay trees with o(log n) worst-case access times. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 516–522, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, July 1985.

[WDS06] Chengwen Wang, Jonathan Derryberry, and Daniel D. Sleator. $O(\log\log n)$-Competitive Dynamic Binary Search Tree. *SODA*, 2006.

[Wil89] Robert Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM Journal on Computing*, 18(1):56–67, 1989.