Dynamic Binary Search Trees: Improved Lower Bounds for the Greedy-Future Algorithm

Yaniv Sadeh
Tel Aviv University, Israel

Haim Kaplan
Tel Aviv University, Israel

Abstract

Binary search trees (BSTs) are one of the most basic and widely used data structures. The best static tree for serving a sequence of queries (searches) can be computed by dynamic programming. In contrast, when the BSTs are allowed to be dynamic (i.e. change by rotations between searches), we still do not know how to compute the optimal algorithm (OPT) for a given sequence. One of the candidate algorithms whose serving cost is suspected to be optimal up-to a (multiplicative) constant factor is known by the name Greedy Future (GF). In an equivalent geometric way of representing queries on BSTs, GF is in fact equivalent to another algorithm called Geometric Greedy (GG). Most of the results on GF are obtained using the geometric model and the study of GG. Despite this intensive recent fruitful research, the best lower bound we have on the competitive ratio of GF is $\frac{4}{3}$. Furthermore, it has been conjectured that the additive gap between the cost of GF and OPT is only linear in the number of queries. In this paper we prove a lower bound of 2 on the competitive ratio of GF, and we prove that the additive gap between the cost of GF and OPT can be $\Omega(m \cdot \log \log n)$ where n is the number of items in the tree and m is the number of queries.

2012 ACM Subject Classification Theory of computation → Online algorithms; Theory of computation → Sorting and searching

Keywords and phrases Binary Search Trees, Greedy Future, Geometric Greedy, Lower Bounds, Dynamic Optimality Conjecture

Digital Object Identifier 10.4230/LIPIcs.STACS.2023.53

Related Version *Full Version*: `https://arxiv.org/abs/2301.03084`

Funding The work of the authors is partially supported by Israel Science Foundation (ISF) grant number 1595-19, German Science Foundation (GIF) grant number 1367 and the Blavatnik research fund at Tel Aviv University.

1 Introduction

Binary search trees (BSTs) are one of the most basic and widely used data-structures. They are used to store a sorted set of keys from a totally ordered universe. Traversing BSTs is usually done by using a single pointer, initially pointing to the root, and moving to the left or right child according to the order of the searched key and the key of the item at the current node. Therefore, we typically define the cost[1] of a search to be the length of the search path. The data structure itself may be static, or change dynamically throughout time, in response to insertions and deletions of items, and possibly even restructured during queries.

Static BSTs are well understood. One can guarantee that the longest path from the root to a leaf is of length $O(\log n)$ if the number of keys is n, by using a balanced tree. If the access sequence is known in advance (in fact only the frequency of accesses of each key matters) then an $O(n^2)$ time algorithm computing the optimal static tree for the particular set of

[1] Our cost model is formally defined in Definition 1, in Section 2.

frequencies was given by Knuth [13]. It is also notable that the lower bound on the cost when the known frequencies are $\vec{f} = [f_1, f_2, \ldots, f_n]$ and the number of queries is m, is $\Omega(m \cdot H(\vec{f}))$ where $H(\vec{f}) = \sum_{i=1}^{n} f_i \log \frac{1}{f_i}$ is the entropy function. A simple way with $O(n \log n)$ running time to construct a near-optimal static (centroid) tree whose cost is $O(m \cdot H(\vec{f}))$, has been described by Mehlhorn [17]. The running time has been improved to $O(n)$ by Fredman [10].

In contrast to the static case, the dynamic case is less understood. One can, of course, serve the sequence with a static tree. But, for many sequences we must change the structure of the tree as we make the searches in order to be efficient. For example, the requested items may be different in different parts of the sequence so a different set of items has to be placed near the root during different parts of the sequence. Restructuring is done by rotations that maintain the symmetric order. When rotations are allowed, the cost is defined to be the size of the subtree that contains the search path and all edges which we rotate.

Here, we assume that the set of values stored in the tree does not change (no insertions or deletions), yet restructuring the tree is allowed to speed up future searches. One famous dynamic algorithm for doing this is the *Splay* algorithm of Sleator and Tarjan [20]. After each query, the splay algorithm moves the queried item to the root of the tree, according to three simple rules called *zig-zag*, *zig-zig* and *zig*. The splay algorithm is efficient in the sense that it is able to exploit the structure of many families of sequences. In particular splay is proven to be as good as the static optimum (up to a constant factor), which also implies that the cost of splay on any given sequence is at most $O(\log n)$ times the (dynamic) optimum cost. Sleator and Tarjan conjectured that splay is in fact dynamically-optimal, meaning that its cost is like the cost of an optimal algorithm that knows the whole sequence of queries in advance, up to some constant factor. However, this dynamic-optimality conjecture of splay is still open. In fact, it is open whether there is any dynamically-optimal online binary search tree algorithm. The best competitive ratio achievable to date is $O(\log \log n)$, and it is obtained by Tango [8], Multi-splay [21] and Chain-splay [11] trees, and a geometric divide-and-conquer approach of [1].

While seeking for (better) guaranteed competitiveness, other dynamic algorithms were considered. A promising candidate was independently proposed by Lucas [16] and Munro [18], which is now commonly referred to as *Greedy Future*, henceforth: GF in short. As its name suggests, GF is a greedy algorithm that rearranges the nodes on the path from the root to the current queried item as a treap whose priorities are according to the future accesses[2] (as this paper deals with analyzing GF, we detail it formally in Algorithm 1). Note that unlike splay, GF, by definition, is required to know the future in order to restructure the tree. Surprisingly however, Demaine et al. [7] showed that one can simulate GF without knowing the future by a hierarchy of split-trees while losing only a constant factor in performance.

Additionally, [7] presented a geometric view of an algorithm serving queries by a dynamic binary search tree using a two dimensional grid on which we mark the sequence as well as the items accessed by the algorithm. In this presentation there is yet another natural promising candidate for dynamic optimality, which is commonly known as *Geometric Greedy* and sometimes simply *Greedy*, which we shall refer to as GG. [7] showed that GG is in fact the same algorithm as GF.

[2] Each item in a treap has two keys: *value* and *priority*. The treap is a binary search tree with respect to the values of the items and a heap with respect to their priorities. That is, the priority of an item is no larger than the priorities of its children. In our case, the priorities are deterministically defined by future requests in a way that we define precisely in Algorithm 1.

The geometric view proved useful to obtain new results regarding GG and hence GF. Fox [9] proved that an *access-lemma* that is analogues to the so called *access-lemma* of splay trees holds for GG. From this follows that most of the nice properties that hold for splay also hold for GF. In particular, it follows that GF is $O(\log n)$ competitive. Chalermsook et al. [3] analyzed upper bounds on the cost of GG for access patterns which are permutations, and in particular found that for highly structured permutations, which they called *k-decomposable*, the cost is $n \cdot 2^{\alpha(n)^{O(k)}}$ where $\alpha(n)$ is the inverse-Ackermann function. Chalermsook et al. [5] study special access patterns that belong to a broader family of *pattern-avoiding* permutations. See [4] for a survey of currently known properties of greedy and splay.

Our Contributions.

1. It is known that GF is not exactly optimal, but it is conjectured, like splay, to be optimal up to a constant factor. In fact, it has been even more strongly conjectured by Demaine et al. [7] to be optimal up to an additive $O(m)$ term, and possibly even exactly m. Kozma [14] refuted the second part and gave a specific sequence for which this additive gap is $m + 1$. In this paper we refute the linear gap conjecture and show a family of sequences for which the additive gap is at least $\Omega(m \log \log n)$.
2. The largest lower bound on the competitive ratio of GF is $\frac{4}{3}$ by Demaine et al. [7]. They show a family of sequences on which after an initial query, the optimum pays 1.5 on average per query while GF pays 2.[3] We describe a technique that allows us to improve this lower bound to 2. We note that the best known lower bound on the competitive ratio of splay is 2 (see [15, Section 2.5]). In both cases, the construction requires a rather large number of items (large n).
3. Based on the multiplicative lower bound described above we show the following two interesting properties of GF: (1) There are sequences X such that the cost of GF on the reverse sequence is twice larger than the cost of GF on X. (2) There are sequences X such that we can remove some queries from them and get a subsequence X', such that the cost of GF on X' is twice larger than the cost of GF on X.

We study subsequences and reversal (contribution 3) since any dynamically-optimal algorithm A must have a "nice" behavior in these cases. Concretely, A must satisfy the approximately-monotone property (Definition 8) which states that there is a fixed constant c such that the cost of A on any subsequence of any sequence is never more than c times the cost on the whole sequence. As for reversal, the optimum can process a sequence and its reversal with similar costs up to a difference of n, thus any dynamically-optimal algorithm must be able to do so with costs that differ by at most a constant factor. We discuss this motivation in more detail in Section 3 (right after stating Theorem 7).

Our contributions are all based on the same technique, which is quite simple. We enforce GF to maintain a static tree and only query the leaves of this tree. Although being dynamic in general, there are some access-patterns that cause GF not to change the tree. By studying these patterns, we can study GF on a static tree, and the analysis of its cost simplifies to the weighted-average of the depth of the queries (weighted by frequency). To lower-bound the gap between GF and OPT, we analyze the average cost that can be saved by promoting the items in the leaves to locations closer to the root. Note that any other item can be placed further away from the root since it is never queried by the sequence.

[3] Reddmann [19] found an example in which the cost ratio between GF and the optimum is $\frac{26}{17} \approx 1.53$. But this is for one particular sequence of a fixed length so it does not rule out any competitive ratio if we allow an additive constant.

2 Model

In this section we describe the model which we use, and define our notations. First, we note that throughout the paper $\lg x$ is used to denote the base two logarithm of x.

We consider a totally ordered universe of (fixed size) n items. For simplicity, one may think of the values $\{1, \ldots, n\}$. The items are organized in some initial BST which we denote by T_0. Then, a sequence of queries, denoted by $X = [x_1, x_2, \ldots, x_m]$, is given, one query at a time. We reserve m to denote the length of the sequence. The tree before serving x_t is denoted by T_{t-1}. An algorithm has to find the queried value x_t, by traversing T_{t-1} from its root. After finding x_t, the algorithm is allowed to re-structure T_{t-1} to get T_t. We define the cost of the algorithm at time t to be the total number of nodes that were touched at time t, both on the path to x_t and for restructuring. The cost of an algorithm for the whole sequence is simply the sum of its costs over all times. We define it formally below.

▶ **Definition 1** (Cost). *Let X be a sequence of queries, and let T_0 be an initial tree. Let A be an algorithm that serves X and let T_t be the tree that A has after serving x_t. Let P_t be the set of nodes on the path from the root to x_t in T_{t-1} and let U_t be the set of nodes of the minimal subtree that contains all the edges that were rotated by A to transform T_{t-1} to T_t. Then the cost of A for serving X at time t is $|P_t \cup U_t|$, and the cost of A for serving X is the sum of costs over $t = 1, \ldots, m$. We denote the cost of A to serve X starting with T_0 by $cost(A, X, T_0)$. We denote the average cost per query by $\hat{c}(A, X, T_0) = \frac{cost(A,X,T_0)}{m}$. When T_0 is clear from the context, or immaterial, we write $cost(A, X)$ and $\hat{c}(A, X)$.*

▶ **Definition 2** (Depth). *Let T be a tree. The depth of a node $v \in T$, denoted by $d(v)$, is the number of edges in the path from the root to v (in particular $d(root) = 0$). Note that the cost of querying v (without restructuring) is $d(v) + 1$. We also define the depth of the tree, denoted by $d(T)$, as the maximum depth of a node in T, that is $d(T) = \max_{v \in T} d(v)$.*

▶ **Definition 3** (Competitiveness). *We say that an algorithm A is (α, β)-competitive for initial tree T_0 if for any sequence of queries X, it holds that $cost(A, X, T_0) \leq \alpha \cdot cost(OPT, X, T_0) + \beta$ where OPT is a best algorithm to serve X given T_0 (with full knowledge of X). When we do not specify T_0 we mean that the relation holds for all initial trees. We refer to α as the* multiplicative term *and to β as the* additive term. *For ease of language, we regard the multiplicative term as the* competitive ratio, *and also write "the competitive ratio of" instead of "the multiplicative term of the competitiveness of". In such cases, we assume that the additive term is $o(m)$. It is easiest to think of $\beta = O(n)$ while assuming that $m = \omega(n)$.*

To conclude this section, we give a precise description of the GF algorithm, in Algorithm 1. We emphasize that its implementation is complex and probably would not be good in practice. However, its main benefit is its theoretical value, as a candidate for dynamic optimality. Should it be proven to be dynamically-optimal, then we would get a better understanding of the problem and also a stepping-stone to analyze simpler algorithms, such as splay, in comparison to GF rather than against some "vague" optimum that depends on the sequence.

3 Stable Sequences and Lower Bounds

In this section we properly define the family of *stable sequences* (Definition 11) for which the tree maintained by GF is never changed (i.e. the access path of the current query is a treap with respect to the suffix of the sequence). To prove our lower bounds we use such sequences in which only the items at the leaves of GF are requested, and the internal nodes

Algorithm 1 GreedyFuture (GF) Algorithm.

Input: A sequence of queries $X \in [n]^m$ and an initial BST T_0. We restructure T_{t-1} to T_t after serving the request x_t with T_{t-1} for $t = 1, \ldots, m$.

Function `Restructure`(*query value* v*, current tree* T_{t-1}*, future accesses* X'**):**

Let $v_1 < v_2 < \ldots < v_k$ be the nodes on the path from the root of T_{t-1} to the queried value v (including v and the root). We also define $v_0 = -\infty$ and $v_{k+1} = +\infty$. Denote the subtrees hanging off this path by $R_0, \ldots, R_k$.

For each $i = 1, \ldots, k$, let $\tau(v_i)$ be the index of the first appearance of a query of a value $x \in (v_{i-1}, v_{i+1})$ in X'. Restructure the nodes $v_1, \ldots, v_k$ as a treap: maintain a BST ordering, while the heap's priorities are set to be the τ values, where the root's τ is smallest. Tie-break arbitrarily, e.g. in favor of smaller values, or smaller depth prior to restructuring. Then, hang the subtrees $R_0, \ldots, R_k$ unchanged at their appropriate locations. The resulting tree is T_t.

cause some extra cost that OPT avoids. We use a natural way to represent such sequences as trees, and use this representation to prove the following lower bounds, which are the main results of this section.

▶ **Theorem 4.** *If GF is (c, d)-competitive where the additive term d is sublinear in the length of the sequence, i.e. $d = o(m)$, then $c \geq 2$.*

▶ **Theorem 5.** *For every $n \geq 2$ there exist sequences $X \in [n]^m$ such that $cost(GF, X) = cost(OPT, X) + \Omega(m \cdot \lg \lg n)$. Among these sequences, there exists a sequence whose length is $m = n^{\Theta(\frac{\lg \lg n}{\lg \lg \lg n})}$. (There exist other longer sequences too.)*

Theorem 4 enables us to prove the following two theorems, proven in Appendix A.2.

▶ **Theorem 6.** *For any $\epsilon > 0$ there exists a sequence X with a subsequence (not necessarily consecutive) $X' \subseteq X$ such that $cost(GF, X') \geq (2 - \epsilon) \cdot cost(GF, X)$.*

▶ **Theorem 7.** *Let S be a sequence, we define $rev(S)$ to be the sequence S in reverse. For any $\epsilon > 0$ there exists a sequence X such that $cost(GF, rev(X)) \geq (2 - \epsilon) \cdot cost(GF, X)$.*

The motivation for studying subsequences (Theorem 6) is the fact that OPT always saves costs when queries are removed from its sequence. Formally, if $X' \subseteq X$, then $cost(OPT, X') \leq cost(OPT, X)$. Indeed, OPT can serve X' by simulating a run on X. More generally, this relation of costs when comparing a sequence to a subsequence of it, is an important property which even has a name:

▶ **Definition 8** (Approximate-monotonicity [12, 15])**.** *An algorithm A is approximately-monotone with a constant c if for any sequence X, subsequence $X' \subseteq X$, and initial tree T, it holds that $cost(A, X', T) \leq c \cdot cost(A, X, T)$.*

▶ **Corollary 9.** *If GF is approximately-monotone with a constant c, then $c \geq 2$.*

As noted, OPT is approximately-monotone with $c = 1$ (strictly monotone). The reason that approximate-monotonicity is of interest, in particular for GF, is because it is one of two properties that together are necessary and sufficient for any dynamically-optimal algorithm. The complementing property, which GF is known to satisfy, is simulation-embedding:

▶ **Definition 10** (Simulation-Embedding [15]). *An algorithm A has the simulation-embedding property with a constant c if for any algorithm B and any sequence X, there exists a supersequence $Y \supseteq X$ such that $cost(A, Y) \leq c \cdot cost(B, X)$. ($X$ is a subsequence of Y, not necessarily of consecutive queries.)*

An algorithm A which is approximately-monotone with a constant c_1 and has the simulation-embedding property with a constant c_2 is dynamically-optimal with a constant $c_1 \cdot c_2$. Indeed, for any sequence X, there is some supersequence $Y(X) \supseteq X$ such that $cost(A, X) \leq c_1 \cdot cost(A, Y(X)) \leq c_1 \cdot c_2 \cdot cost(OPT, X)$. Harmon [12] proved that GG, and hence GF, has the simulation-embedding property, hence GF is dynamically-optimal if and only if it is approximately-monotone. An alternative indirect proof was given by [6], proving that GG is $O(1)$-competitive versus the move-to-root algorithm, therefore inheriting the property from move-to-root.

The motivation for studying reversal (Theorem 7) is that OPT is oblivious to reversing the sequence of queries, up to an additive difference of n. Indeed, to serve a sequence X in reverse, we can pay n to restructure the initial tree T_0 to the final tree T_m, and then "reverse the arrow of time": when serving query x_t, also modify the tree from T_t to T_{t-1} where T_i is the tree that OPT would get by the end of processing the i-th query of X, in order. This means that any dynamically-optimal algorithm must be able to serve a sequence of requests and its reverse with the same cost up to a constant factor. Theorem 7 does not disprove dynamic-optimality for GF, but gives some insight of how reversal affects GF.

3.1 Maintaining a Static Tree for *GF*

In this section we describe the basic "tool" which we use to fix a tree structure for GF despite its dynamic nature. That is, we describe a class of sequences which we call *mixed-stable sequences* such that GF never restructures its tree when serving a sequence in this class. For the sake of simplicity, we assume that the initial tree is structured as we need it to be. Appendix A.1 explains how to enforce a specific "initial" tree given an arbitrary initial tree, and also argues why this minor issue does not affect the competitive ratio of GF.

As noted, our objective is to produce a sequence that "tricks" GF into having unnecessary nodes in the core of the tree, such that the requested values are only at the leaves. As an example, consider the classic sequence of queries $X = [1, 3, 1, 3, \ldots]$ with an initial tree containing 2 at the root, 1 as a left child of the root and 3 as a right child of the root. Because of the alternating pattern, GF never re-structures the tree, and the cost per query is 2 rather than 1.5 on average (e.g. when 1 is in the root, and 3 is its right child).

▶ **Definition 11** (Stable Nodes and Sequences). *Let T be a full binary search tree, and let X be a sequence of queries over the items in the leaves of T. We define the stability of nodes as follows, see also Figure 1.*

We say that an inner node v in T is strongly-stable *if it has two children, and the subsequence of X consisting only of the items in the subtree of v, alternates between accesses to the left and right subtrees of v.*

We say that an inner node v with a left child u in T is weakly-stable with a left-bias *if both v and u have two children, and the subsequence of X consisting only of the items in the subtree of v, repeats the following 3-cycle. First it accesses the left-subtree of u, then the right subtree of u, and finally right subtree of v. (It is left-biased because $\frac{2}{3}$ of the accesses are to the left of v). Symmetrically, we say that v is* weakly-stable with a right-bias *if v has two children, its right child u has two children, and the restriction of X to accesses in the*

subtree of v repeats a 3-cycle consisting of an access to the right subtree of u, the left subtree of u, and the left subtree of v. Notice that u is a strongly-stable node by definition, and we refer to it as the favored-child *of v.*

We regard the sequence X as being induced by the tree T with stability "attached" to its inner nodes. We assume that every node is stable, and refer to X as a mixed-stable *sequence and to T as a* mixed-stable *tree. We distinguish two special cases: If all inner nodes are strongly-stable then we refer to X and T as* strongly-stable*, and if exactly half of the inner nodes of T are weakly-stable then we refer to X and T as* weakly-stable *(recall that each weakly-stable node has a strongly-stable favored-child).*

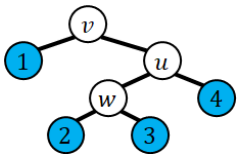


Figure 1 Node and sequence stability (Definition 11). First, consider the repeated sequence 421, i.e. $X = 421421421\ldots$. Then v is a weakly-stable right-biased node because its visits pattern is a repetition of $right(u), left(u), left(v)$. u is a strongly-stable node because its visits pattern is $right(u), left(u)$. w is not stable at all, because its visits pattern is always $left(w)$. Second, consider the repetition of the access pattern 12141314. One can verify that all three inner nodes are strongly-stable. Hence, this is a strongly-stable sequence. Third, note that no weakly-stable sequence corresponds to the figure, because it requires an even number of inner nodes, but if we make w a leaf (removing $2, 3$), then the repeated access pattern of $4w1$ is a weakly-stable sequence.

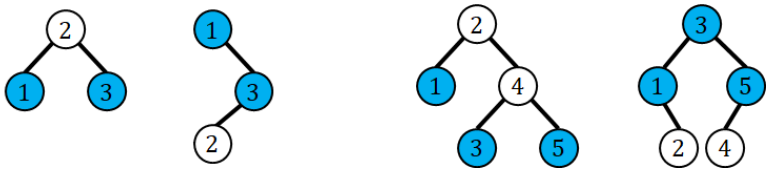
To motivate Definition 11 a little, note that the sequence $X = [1, 3, 1, 3, \ldots]$ is a strongly-stable sequence that corresponds to a tree over the items $\{1, 2, 3\}$ where 2 is in the root. X yields a lower-bound of $\frac{4}{3}$ on the competitive ratio of GF. Similarly, the sequence $X' = [5, 3, 1, 5, 3, 1, \ldots]$ is a weakly-stable sequence that corresponds to the tree over $\{1, 2, 3, 4, 5\}$ with 2 at the root and 4 its right-child. X' yields a lower-bound of $\frac{8}{5}$ on the competitive ratio of GF, which is already an improvement over the best known lower bound, see also Figure 2. The distinction between strongly-stable and weakly-stable nodes is that GF may modify the structure of the tree when a weakly-stable node is considered, but only temporarily and without affecting the cost. In our example with X', after querying 5, GF may put 4 in the root instead of 2, but following the query of 3 this change will be reverted.

Motivated by the power of stable sequences over small trees, we proceed to a more general analysis of stable sequences.

▶ **Definition 12** (Atomic Sequence). *A tree T, along with stability type (weak/strong) for each node, and a subtree of each node to be accessed initially, induce a stable sequence. This sequence is unique up to its length, which can be extended indefinitely. We define the "atomic unit" of this sequence as the shortest sequence X such that any repetition of X is also a stable sequence that corresponds to T.*

Throughout the paper we work with whole multiples of the atomic sequence. Moreover, unless stated otherwise, we work with the atomic sequence itself (a single repetition).

▶ **Lemma 13.** *Let X be a mixed-stable sequence with respect to a tree T. Then every leaf u is visited once every $2^{a(u)} \cdot 3^{b(u)}$ queries where $a(u)$ and $b(u)$ are non-negative integers. In particular, the atomic length of X is $2^{\max_{leaf\ u} a(u)} \cdot 3^{\max_{leaf\ u} b(u)}$ (the lcm). Moreover, if X is strongly-stable then $\forall u : b(u) = 0$, and if X is weakly-stable then $\forall u : a(u) = 0$.*



(a) $X = [1, 3, 1, 3, \ldots]$. **(b)** $X' = [5, 3, 1, 5, 3, 1, \ldots]$.

Figure 2 Examples of the simplest strongly-stable (a) and weakly-stable (b) sequences. Their corresponding trees are the left tree in each pair while the right tree in each pair is an optimized static tree to serve the same sequence. Queried nodes are colored in blue. One can verify that $\hat{c}(X, GF) = 2$ and $\hat{c}(X', GF) = \frac{8}{3}$ while based on the optimized tree, $\hat{c}(X, OPT) \leq \frac{3}{2}$ and $\hat{c}(X', OPT) \leq \frac{5}{3}$.

Proof. Consider a leaf u. Define the frequency of visiting an ancestor w of u to be the frequency of accessing a leaf in the subtree of w. If w is a strongly-stable ancestor then the frequency of visiting a child of w is $\frac{1}{2}$ of the frequency of visiting w. If w is weakly-stable, v is its favored-child, and x is a child of v then the frequency of visiting x is $\frac{1}{3}$ of the frequency of visiting w. Similarly if w is weakly-stable, v is its non-favored-child then the frequency of visiting v is $\frac{1}{3}$ of the frequency of visiting w. It follows that u is visited exactly once every $2^{a(u)} \cdot 3^{b(u)}$ queries where $a(u)$ is the number of strongly-stable nodes that are not favored-children (there are no such nodes if X is weakly-stable), and $b(u)$ is the number of weakly-stable nodes (no such nodes if X is strongly-stable), on the path to u. Finally, since every leaf u is visited with a specific period, the whole sequence has a period which is the lcm of all periods. ◀

▶ **Lemma 14.** *Let X be a mixed-stable sequence with respect to a tree T. If GF serves X with T as initial tree, and breaks ties in favor of nodes of smaller-depth, then it never restructures T.*

Proof. The proof is by induction on the size of the tree. If T has a single node, then it is trivial. Otherwise, the root r is an inner-node, and we prove that it always remains the root. It then follows, by restricting the access sequence to values within each subtree, that the rest of the tree remains fixed as well. We use the notations of $\tau(v)$ and v_i as in Algorithm 1.

First, consider the case that r is a strongly-stable node (Definition 11). Given an access to some value x in the left subtree of r, by definition, the next access would be to a value in the right subtree of r, hence $\tau(r) < \tau(v_i)$ for any $v_i \neq r$ on the path from r to x, and therefore GF will keep r in the root. The same argument holds if x is in the right subtree of r, and the next access is in the left subtree.

Next, consider the case that r is a weakly-stable node. Without loss of generality, assume that it is left-biased, and denote its favored-child (left child) by u. Denote the left and right subtrees of u by A and B respectively, and the right subtree of r by C. The access pattern of subtrees is $ABC(ABC\ldots)$.

- If the current access was to some $x \in A$, both r and u have been touched. The next access queries in B, so $\tau(u) = \tau(r) < \tau(v_i)$ for any $v_i \neq u, r$ on the access path to x. Since GF tie-breaks in favor of smaller-depth, it will keep r in the root.[4]

[4] This is the reason we defined this kind of access pattern as weakly-stable, because the stability can be chosen, but is not forced. We emphasize that putting u as a parent of r will not make the next access cheaper as both u and r will be touched anyway, and then r will be reinstated as the root.

- If the current access was to some $x \in B$, then both r and u have been touched. The next access touches C, so $\tau(r) < \tau(v_i)$ for any $v_i \neq r$ on the access path to x, including u, thus r must remain the root.
- If the current access was to some $x \in C$, since the next access touches A, $\tau(r) < \tau(v_i)$ for any $v_i \neq r$ on the access path to x, thus r must remain the root. In this case u was not touched, but nonetheless it remains the left child of r. ◀

▶ **Lemma 15.** *If X is a mixed-stable sequence, the frequency of accessing $x \in X$ is in the range of $[\frac{1}{3^{d(x)}}, \frac{1}{3^{d(x)/2}}]$. In particular, if X is strongly-stable then the frequency equals $\frac{1}{2^{d(x)}}$.*

Proof. The frequency of visiting a node depends on the path to it. The frequency is multiplied by $\frac{1}{2}$ when passing through a strongly-stable node, and multiplied by either $\frac{1}{3}$ or $\frac{2}{3}$ when passing through a weakly-stable node. Every factor of $\frac{2}{3}$ is followed by $\frac{1}{2}$, due to the strongly-stable favored-child of the weakly-stable node. Thus the frequency is bounded between $\frac{1}{3^{d(x)}}$ and $\frac{1}{2^{d(x)/2}} \cdot \left(\frac{2}{3}\right)^{d(x)/2} = \frac{1}{3^{d(x)/2}}$. ◀

▶ **Corollary 16.** *Let X be a strongly-stable sequence, then: $\hat{c}(GF, X) = \sum_{x \in X} \frac{d(x)+1}{2^{d(x)}}$.*

3.2 Promotions and Recursive Trees

The way in which we show our lower bounds relies on the fact that serving the leaves of a static tree is sub-optimal, since a trivial static optimization is to move the leaves closer to the root. We refer to this operation as a *promotion* of the leaf that we move. We emphasize that for the purpose of our result, we analyze the improvement one gets from promotions, but the actual OPT, which is dynamic, may be able to reduce the cost further.

▶ **Definition 17** (Promotion). *Consider trees T and T'. We say that a node x was promoted in T' by h (with respect to T), if $d_T(x) - d_{T'}(x) = h$. Given a mixed-stable sequence X, the average promotion of T to T' is the weighted average promotion in T' of the nodes of T, weighted by the query frequencies of the nodes.*

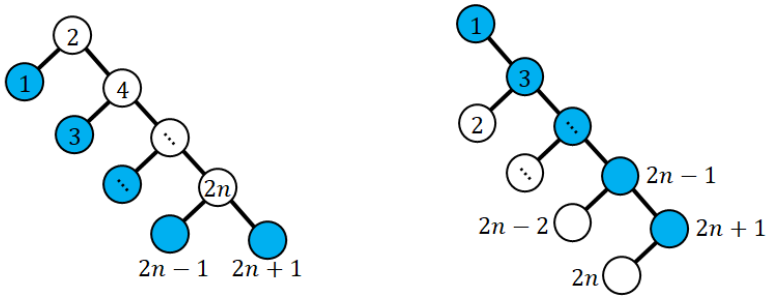
By definition, static optimization of a tree T to T' for a mixed-stable sequence X, implies a cost improvement for OPT which is at least the average promotion of T to T', per query. Intuitively, promoting leaves that are closer to the root contributes more to the average promotion than promoting deeper leaves since the access frequencies decrease exponentially with depth. That being said, our promotion scheme will be relatively uniform, promoting most leaves by roughly the same amount, as in the following example.

▶ **Example 18.** To clarify promotions, consider Figure 3. There, we can safely promote every node by one, except for one of the deepest nodes. Therefore, we immediately conclude that for the corresponding strongly-stable sequence X, we have: $\hat{c}(GF, X) \geq \hat{c}(OPT, X) + (1 - \frac{1}{2^n})$.

We define our trees using recursive structures.

▶ **Definition 19.** *A recursive tree, T_r, of depth r is defined by a specific full binary tree T (independent of r) such that at least one of its leaves is an actual leaf, and some of its leaves are roots of recursive trees, T_{r-1}, of depth $r-1$. We refer to the inner nodes of T as the* trunk *of T_r, and define T_0 to be a single node. See Figure 4 for two examples.*[5]

[5] The name of the pattern F in Figure 4, stands for Fibonacci: One can verify that for $r \geq 2$, the number of leaves at depth $1 \leq d \leq r-1$ is the $(d-1)$th Fibonnaci number F_{d-1} (we define $F_0 = 0$). Moreover, this can be used to prove the nice equation: $\sum_{d=0}^{\infty} \frac{F_d}{2^d} = 2$.



(a) Before promotions. **(b)** After promotions.

Figure 3 (a) A tree which induces a strongly-stable sequence X, only blue nodes are queried. The frequency of querying an odd number $v = 2i - 1$ in this tree is $\frac{1}{2^i}$ except for $v = 2n + 1$ which has the same frequency as $v = 2n - 1$. (b) An improved static tree, in which each node except for one has been promoted one step closer to the root. The cost of serving X over this tree is cheaper by almost 1 per query.

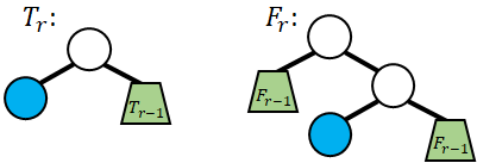


Figure 4 Two recursive trees of depth r. Each of the trees T and F is a full binary tree with at least one actual leaf (in blue), and some hanging subtrees. At the bottom of the recursion (for $r = 0$), the subtrees are nodes. Note that: (a) Expanding T for $r = n$ results in the tree in Figure 3; (b) The pattern F is important for Theorem 4.

3.3 Multiplicative Lower Bound for *GF*

In this section we prove Theorem 4. We do it by describing a concrete weakly-stable sequence, whose average cost per query is 6 while an average promotion of 3 is possible, resulting in an optimal cost of at most 3. We start by stating a purely mathematical lemma that will be used in the analysis.

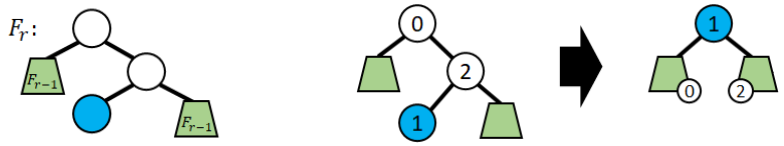
▶ **Lemma 20.** *Let b_r be a sequence defined by an initial value b_0 and the relation $b_r = \alpha \cdot b_{r-1} + \beta + \gamma \cdot \frac{r}{2^r}$ for some constants α, β, γ where $\alpha \neq \frac{1}{2}, 1$. Then $b_r = \frac{\beta}{1-\alpha}(1-\alpha^r) + \alpha^r \cdot b_0 + \frac{2\alpha\gamma}{(2\alpha-1)^2} \cdot (\alpha^r - \frac{1}{2^r}) - \frac{\gamma}{(2\alpha-1)} \cdot \frac{r}{2^r}$. In particular, when $\gamma = 0$ then $b_r = \frac{\beta}{1-\alpha}(1-\alpha^r) + \alpha^r \cdot b_0$.*

Proof Sketch. Either use induction, or "guess" that a geometric sequence y_r with a multiplier of α satisfies $y_r = p \cdot \frac{r}{2^r} + q \cdot \frac{1}{2^r} + s + b_r$, and determine the fixed coefficients p, q, s. ◀

▶ **Lemma 21.** *Let X be a weakly-stable sequence implied by the recursive tree F_r in Figure 4, where the root is a weakly-stable node with a right-bias. Then for any $\epsilon > 0$, there is a sufficiently large recursive depth r such that (1) $\hat{c}(GF, X) > 6 - \epsilon$, (2) a static optimization of the tree saves an average cost of at least $3 - \epsilon$, and (3) regardless of r, $\hat{c}(OPT, X) < 3$.*

Proof. Let c_r denote the average cost of serving X with F_r. Then $c_0 = 1$ and $c_r = \frac{1}{3}(c_{r-1} + 1) + \frac{1}{3} \cdot 3 + \frac{1}{3}(c_{r-1} + 2) = \frac{2}{3}c_{r-1} + 2$, which yields by Lemma 20 that $c_r = \frac{2}{1-2/3}(1 - (2/3)^r) + (2/3)^r \cdot 1 = 6 \cdot (1 - (2/3)^r) + (2/3)^r$. To analyze the average promotion, we re-structure F_r to a new static structure F'_r as follows, see Figure 5. The leaf is moved to the root, whose children are the recursive subtrees, optimized themselves by the same

logic. The old root is moved to be a right child of the maximal value in the new left subtree, and the old right-child (of the old-root) is moved to be a left child of the minimal value in the new right subtree. F'_r maintains the order of values as was in F_r. The demotions of the old root and its right child do not affect the cost, because X does not query these values. Denote by p_r the average promotion of F_r to F'_r. Then $p_0 = 0$ since nothing is promoted for a singleton, and $p_r = \frac{1}{3}p_{r-1} + \frac{1}{3}\cdot 2 + \frac{1}{3}(p_{r-1}+1) = \frac{2}{3}p_{r-1} + 1$. Again by Lemma 20 we get that $p_r = \frac{1}{1-2/3}(1-(2/3)^r) + (2/3)^r \cdot 0 = 3\cdot(1-(2/3)^r)$. Observe that for $r \to \infty$ we get that $c_r \to 6$ and $p_r \to 3$, thus parts (1) and (2) of the claim follow. For part (3), observe that $c_r - p_r = 6\cdot(1-(2/3)^r) + (2/3)^r - 3\cdot(1-(2/3)^r) = 3 - 2\cdot(2/3)^r < 3$. ◀



(a) F-tree pattern. **(b)** Promotion scheme.

Figure 5 The F-tree pattern and its promotion scheme in Lemma 21. Only the top-level promotions are presented in (b), but more promotions are done recursively within each subtree.

▶ **Theorem 4.** *If GF is (c,d)-competitive where the additive term d is sublinear in the length of the sequence, i.e. $d = o(m)$, then $c \geq 2$.*

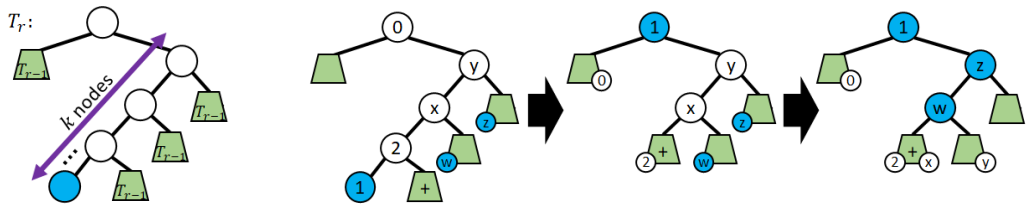
Proof. Assume by contradiction that GF is $(2-\delta, f(m))$-competitive for some $\delta > 0$ and a function $f(m) = o(m)$. Let X' be a sequence that consists of s repetitions of the atomic weakly-stable sequence that corresponds to the recursive tree F_r. It follows that $\hat{c}(GF, X') \leq (2-\delta)\cdot\hat{c}(OPT, X') + \frac{f(|X'|)}{|X'|}$. By Lemma 21, we can choose r large enough such that $\hat{c}(GF, X') > 6-\delta$, and regardless of r, $\hat{c}(OPT, X') < 3$. Then, since f is sub-linear, we can choose the number of repetitions s to be large enough such that $\frac{f(|X'|)}{|X'|} < 2\delta$. But then we also get that $\hat{c}(GF, X') < (2-\delta)\cdot 3 + 2\delta = 6 - \delta$, which is a contradiction. ◀

By Analyzing mixed-stable sequences we proved a lower bound of 2 on the competitve ratio of GF. Theorem 22 gives an upper bound.

▶ **Theorem 22.** *Let X be a mixed-stable sequence and let T be the tree that corresponds to it. Then $cost(GF, X, T) < c \cdot cost(OPT, X, T)$ for $c = \frac{5}{2}$. If X is strongly-stable, then $c = 2$.*

We defer the proof of Theorem 22 to Appendix A.2. The upper-bound in Theorem 22 is clearly not tight, since in the proof of Theorem 22 we neglected a term using the inequality $\hat{c}(GF, X) \leq \frac{2}{\alpha}\cdot\hat{c}(OPT, X) - \frac{1}{\alpha}\left(1 - \frac{n-1}{2m}\right) < \frac{2}{\alpha}\cdot\hat{c}(OPT, X)$, for a constant α. The lack of tightness is more prominent when $\hat{c}(OPT, X)$ is small, like in the sequence studied in Lemma 21 (for Theorem 4). We suspect that the lower bound in Theorem 4 is tight, and more strongly, that the F-tree pattern is the best pattern to use. This is based on studying several other recursive patterns, including those in Figure 4 and Figure 6: None was stronger, and it also seems that patterns with large costs do not "compensate" with large enough promotions.

As a closing remark to the multiplicative results, we note that by the static optimality theorem for GG [9], competitive analysis against a *static* algorithm (i.e. an algorithm that does not change its initial tree) cannot show a super-constant lower bound. Concretely, the theorem states that $cost(GF, X) \equiv cost(GG, X) = O(m + \sum_{i=1}^{n} n_i \lg \frac{m}{n_i})$ and one can



(a) (k, r)-tree pattern. **(b)** Promotion scheme. The main gain is due to the first step.

Figure 6 (a) The recursive pattern of a (k, r)-tree, T_r. The *trunk* of the tree has k nodes: the root, and a chain of $k-1$ nodes leading to an actual leaf. The rest of the leaves are $(k, r-1)$-trees. (b) The promotion scheme used later in Lemma 26, exemplified for $k = 4$ (see also Figure 5 for the degenerate case of $k = 2$). The main gain is from the first step of promoting the actual leaf to the root, and its sibling subtree (marked with $+$) one step upwards. Additional gain is achieved by promoting the left-most node of each hanging right subtree to the trunk at the expense of demoting trunk nodes. More promotions are done recursively within each subtree. The nodes marked $0, 1, 2$ are indeed consecutive, and also: $2 < x$ and $x + 1 = w < y = z - 1$.

verify that the actual constants are $5m + 6\sum_{i=1}^{n} n_i \lg \frac{m}{n_i}$. This bound can be re-written as $5m + 6m \cdot H_2(X)$ where $H_2(X) = \sum_{i=1}^{n} \frac{n_i}{m} \lg \frac{m}{n_i}$ is the base-2 entropy of the frequencies of the values in X. By [17], $cost(OPT^s, X) \geq m \cdot \frac{H_2(X)}{\lg 3}$ where OPT^s is the static optimum, and therefore $cost(GF, X) \leq (5 + 6 \lg 3) \cdot cost(OPT^s, X)$. Thus, no static argument can show a lower bound larger than ≈ 11.59.

3.4 Additive Lower Bounds for *GF*

In this section we move on to analyze the additive gap between GF and OPT. For this, we construct and analyze more elaborate patterns of recursively-defined trees, in order to get a large average promotion when optimizing the structure of the trees. The analysis is more involved since we cannot simply assume that the depth of the recurrence, r, approaches infinity. Here n is a function of r and the difference of cost can be meaningful in terms of n only if n is finite.

▶ **Definition 23.** *For $k \geq 2$, and $r \geq 0$ we define a (k, r)-tree T_r as follows. The tree is recursive of depth r (as in Definition 19), such that its trunk is composed of a root and a left-chain of length $k-1$ that starts in the right-child of the root. The left child of the deepest node of the trunk is an actual leaf, and the rest of the leaves are T_{r-1} subtrees. T_0 is a single node. See Figure 6. When k is clear from the context, we also refer to the tree as T_r.*

Observe that the tree F_r that was used to prove Theorem 4 is in fact a (k, r)-tree with $k = 2$. When we conclude the analysis, we will get the two ends of a "tradeoff" such that on the one end we have a relatively high cost ratio, and on the other a relatively high cost difference. Moreover, we will show that the higher the difference of costs on a sequence induced by (k, r)-tree, the closer the cost ratio is to 1 (comparing GF to OPT).

▶ **Lemma 24.** *The depth of a (k, r)-tree is $k \cdot r$, and its left-most node is at depth r.*

Proof. Trivial by induction: For $r = 0$, the deepest node is the root, at depth 0. For $r \geq 1$, observe that the deepest node belongs to the deepest subtree T_{r-1}, which is rooted at depth k since the path to it includes k trunk nodes. Similarly, the depth of the left-most node is increased by 1 per recursive level of the tree. ◀

▶ **Lemma 25.** *Let T_r be a (k,r)-tree. Then $|T_r| = (2+\frac{2}{k-1})k^r - (1+\frac{2}{k-1})$ where $|T_r|$ is the number of nodes in T_r. In rougher terms, $|T_r| = \Theta(k^r)$.*

Proof. Denote $n_r = |T_r|$. By definition, $n_0 = 1$ and $n_r = (k+1) + k \cdot n_{r-1}$. Hence, by Lemma 20 (with $\gamma = 0$): $n_r = \frac{k+1}{1-k}(1-k^r) + k^r = (2+\frac{2}{k-1})k^r - (1+\frac{2}{k-1})$. ◀

▶ **Lemma 26.** *Let X be any mixed-stable sequence corresponding to a (k,r)-tree T_r. Denote the average weighted promotion possible in T_r by p_r, where weighting is according to the frequency of querying each leaf. Then $p_r > k \cdot (1-\alpha^r)$ for $\alpha = 1 - \frac{1}{3^k}$. In particular, if X is a strongly-stable sequence, then $p_r = (k+1)\cdot(1-\alpha^r) + \delta$ for $\alpha = 1 - \frac{1}{2^k}$ and $0 \le \delta < \alpha^r$.*

Proof. We can promote by k every explicit leaf in every $T_{r'}$ for all recursive levels $1 \le r' \le r$, from its location to the root of $T_{r'}$. Only nodes that are T_0 leaves do not contribute an explicit promotion of at least k, therefore $p_r > k \cdot (1-f)$ where f is the sum of query-frequencies of all T_0 leaves (the inequality is strict due to unaccounted subtree promotions). To conclude, we argue that $f \le (1-\frac{1}{3^k})^r$. The frequency of accessing the explicit leaf of T_r is at least $\frac{1}{3^k}$ by Lemma 15, hence with frequency of at most $1-\frac{1}{3^k}$ we query a value in some T_{r-1} subtree. Similarly, within the chosen subtree there is again a relative frequency of at most $1-\frac{1}{3^k}$ to query within some T_{r-2} subtree. Overall, since there are r levels of recursion, we conclude that $f \le (1-\frac{1}{3^k})^r$.

Proving the second part of the claim required a more careful analysis. We define the following method of promotion, depicted in Figure 6. In the (k,r)-tree we promote the (only) explicit leaf to the root, and promote its sibling subtree by 1. Then we apply similar promotions recursively within every $(k,r-1)$-subtree. Finally, we promote the left-most node within each $(k,r-1)$-subtree that hangs as a right-subtree from the trunk to the parent of this subtree. Denote the total average (weighted) promotion by p_r. Note that it does not matter if we promote the left-most nodes of the right subtrees before or after the recursive promotions, because the total order on the items guarantees that there is only one value that can be put instead of every demoted trunk node, and the recursive promotions within a specific subtree do not change the depth of its leftmost leaf.

The promotion of the explicit leaf of T_r saves a cost of k weighted by a factor (query frequency) of $\frac{1}{2^k}$. The promotion of the sibling subtree saves 1 weighted by a factor of $\frac{1}{2^k}$. The recursive promotions are p_{r-1} weighted by $\sum_{i=1}^{k}\frac{1}{2^i}$ (for all the k subtrees), and finally the last promotions are technically negligible (as seen in the analysis below), but for the sake of completeness we consider them in the analysis as well: promoting the left-most node from each subtree saves $(r-1)+1 = r$ since the leaf that we promote last is at depth $r-1$ within the recursive subtree, and this promotion is weighted by $\frac{1}{2^r}\cdot\sum_{i=2}^{k-1}\frac{1}{2^i}$ (factor of $\frac{1}{2^r}$ follows from Lemma 24). We get that: $p_r = \frac{k+1}{2^k} + p_{r-1}\cdot\left(1-\frac{1}{2^k}\right) + \frac{r}{2^r}\cdot\frac{1}{2}\left(1-\frac{1}{2^{k-2}}\right)$. Then by Lemma 20, with $\alpha = 1-\frac{1}{2^k}$ and $\gamma = \frac{1}{2}(1-\frac{1}{2^{k-2}})$, we get:

$$p_r = (k+1)\cdot(1-\alpha^r) + \delta \quad , \quad \delta \equiv \alpha^r \cdot p_0 + \frac{2\alpha\gamma}{(2\alpha-1)^2}\cdot\left(\alpha^r - \frac{1}{2^r}\right) - \frac{\gamma}{(2\alpha-1)}\cdot\frac{r}{2^r}$$

It remains to show that $0 \le \delta < \alpha^r$. It is simple to see that $\delta = 0$ for $k = 2$, because then $\gamma = 0$ and $p_0 = 0$ is the average weighted promotion in a tree with a single node. For $k \ge 3$, by the definition of α and γ we have that $\frac{2\alpha\gamma}{(2\alpha-1)^2} = \frac{(2^k-1)(2^k-4)}{(2^k-2)^2} = 1 - \frac{1}{2^k-4+\frac{4}{2^k}} \in (\frac{3}{4}, 1)$ and $\frac{\gamma}{2\alpha-1} = \frac{1}{2} - \frac{1}{2^k-2} \in [\frac{1}{3}, \frac{1}{2})$. Substituting these bounds and $p_0 = 0$ into the formula for δ gives $\delta < \alpha^r$. Moreover, δ is positive since $\delta > \frac{3}{4}(\alpha^r - \frac{1}{2^r}) - \frac{1}{2}\cdot\frac{r}{2^r} = \frac{3}{4}(\alpha-\frac{1}{2})\cdot\sum_{i=0}^{r-1}\alpha^i\cdot\left(\frac{1}{2}\right)^{r-1-i} - \frac{r}{2^{r+1}} = \frac{3(\alpha-\frac{1}{2})}{2^{r+1}}\cdot\sum_{i=0}^{r-1}(2\alpha)^i - \frac{r}{2^{r+1}} > \frac{3(\alpha-\frac{1}{2})}{2^{r+1}}\cdot r - \frac{r}{2^{r+1}} = (3(\frac{1}{2}-\frac{1}{2^k})-1)\cdot\frac{r}{2^{r+1}} > 0$ for $k \ge 3$.

Note that indeed the gain from promoting the left-most node of each subtree is negligible, since the effect is merely having $\gamma \neq 0$, which only contributes $0 \leq \delta < \alpha^r < 1$. ◀

▶ **Corollary 27.** *The average cost-per-query of GF on a strongly-stable sequence induced by a (k,r)-tree is larger than the optimal cost by at least $(k+1)\cdot\left(1-\left(1-\frac{1}{2^k}\right)^r\right)$. On any mixed-stable sequence, the difference is at least $k\cdot\left(1-\left(1-\frac{1}{3^k}\right)^r\right)$.*

We are ready to prove Theorem 5.

▶ **Theorem 5.** *For every $n \geq 2$ there exist sequences $X \in [n]^m$ such that $cost(GF,X) = cost(OPT,X) + \Omega(m \cdot \lg\lg n)$. Among these sequences, there exists a sequence whose length is $m = n^{\Theta(\frac{\lg\lg n}{\lg\lg\lg n})}$. (There exist other longer sequences too.)*

Proof. Let X be the strongly-stable sequence induced by a (k,r)-tree T_r, and for simplicity assume that the initial tree is T_r.[6] By Lemma 25, $n = (2+\frac{2}{k-1})k^r - (1+\frac{2}{k-1})$ therefore $\lg\lg n = \lg r + \lg\lg k + O(1)$.[7] By Corollary 27, $\hat{c}(GF,X) - \hat{c}(OPT,X) \geq \Delta \equiv (k+1)\cdot(1-(1-\frac{1}{2^k})^r)$. By choosing $r = 2^k$ we get that $\Delta = (k+1)\cdot(1-(1-\frac{1}{2^k})^{2^k}) \approx (1-\frac{1}{e})\cdot(k+1)$.[8] We also get that $\lg\lg n = k + \lg\lg k + O(1)$, therefore $\Delta \approx (1-\frac{1}{e})\lg\lg n$ and we conclude that $\hat{c}(GF,X) - \hat{c}(OPT,X) \geq \Omega(\lg\lg n)$.

By Lemma 13 the length of the atomic strongly-stable sequence of T_r is $m = 2^{d(T_r)}$, hence $m = 2^{rk}$ by Lemma 24. By Lemma 25, $\frac{n+(1+2/(k-1))}{2+2/(k-1)} = k^r = 2^{r\lg k}$. Together we get that $m = 2^{rk} = 2^{(r\lg k)\cdot(k/\lg k)} = \left(\frac{n+(1+2/(k-1))}{2+2/(k-1)}\right)^{(k/\lg k)} = n^{\Theta(\frac{\lg\lg n}{\lg\lg\lg n})}$. ◀

▶ Remark 28. In the proof of Theorem 5, the sequence X does not have to be strongly-stable, and any mixed-stable sequence X induced by a (k,r)-tree T_r works as well. Indeed, Corollary 27 guarantees that $\hat{c}(GF,X) - \hat{c}(OPT,X) \geq \Delta$ for $\Delta = k\cdot\left(1-\left(1-\frac{1}{3^k}\right)^r\right)$, and then by choosing $r = 3^k$ we get that $\Delta = \Theta(k)$, and $k = \Theta(\lg\lg n)$, and $m = 2^{\Theta(rk)} = n^{\Theta(\frac{\lg\lg n}{\lg\lg\lg n})}$.

▶ Remark 29. The choice of $r = 2^k$ in the proof of Theorem 5 maximizes our lower bound on the additive gap $cost(GF,X) - cost(OPT,X)$ (up to constants) for our (k,r)-trees. Indeed, revisiting the proof, we have that Δ and n are both functions of k and r, and we need to choose r and k to maximize Δ as a function of n. Note that $\Delta = O(k)$ regardless of r, and $\lg\lg(n) = \lg r + \lg\lg k + O(1)$. To simplify and eliminate a parameter we define $r = 2^k \cdot f(k)$ for some monotone function f. Now we get simplified relations: $\Delta = (k+1)\cdot(1-(1-\frac{1}{2^k})^{2^k f(k)}) \approx (k+1)\cdot(1-e^{-f(k)})$ and $\lg\lg n = k + \lg f(k) + \lg\lg k + O(1)$. Consider the following two cases.

- If $f(k) = \Omega(1)$: then $\exists c \in \mathbb{R}$ such that $\forall k \geq 1 : \lg f(k) \geq c$, and therefore $\lg\lg n = \Omega(k)$, written differently $k = O(\lg\lg n)$, which yields $\Delta = O(k) = O(\lg\lg n)$.
- If $f(k) = o(1)$: Being $o(1)$ means that $\lim_{k\to\infty} f(k) = 0$, so for sufficiently large values of k we can use the approximation $e^x \approx 1 + x$ (that holds for small x) to get: $\Delta \approx (k+1)\cdot f(k) = \frac{k+1}{1/f(k)}$. If $\frac{1}{f(k)}$ grows faster than $(k+1)$, we get $\Delta = O(1)$ which does not even grow with n. Therefore $\frac{1}{f(k)}$ is increasing, but at a sub-linear rate. Recall that $\lg\lg n = k - \lg\frac{1}{f(k)} + \lg k + O(1)$. Since $\frac{1}{f(k)}$ is sub-linear, we get that $k = \Theta(\lg\lg n)$, which yields $\Delta = O(k) = O(\lg\lg n)$.

---

[6] We remove this assumption in Remark 34.

[7] $k \geq 2 \Rightarrow k^r \leq n < 4k^r \Rightarrow \lg n = r\lg k + c$ for $c \in [0,2)$, and so $\lg\lg n = \lg r + \lg\lg k + O(1)$.

[8] The approximation is off by less than 10% for $k \geq 2$. (60% and 20% for $k = 0,1$ respectively.)

▶ **Corollary 30.** *GF is not* $(1, O(m))$*-competitive. If the multiplicative term is* 1*, then the additive term is at least* $\Omega(m \cdot \lg\lg n)$*.*

We have yet to analyze the cost of OPT on a strongly-stable sequence X corresponding to a (k,r)-tree that produces the gap of $\Omega(m \cdot \lg\lg n)$ in Theorem 5. Allegedly, if the cost is cheap, say linear, we would get a large competitive ratio as well. However, by Theorem 22 we expect a competitive ratio of at most 2, and therefore we can conclude without further analysis, that $cost(OPT, X) = \Omega(m \cdot \lg\lg n)$. In fact, we prove that $cost(OPT, X) = \Theta(m \cdot \frac{\lg n}{\lg\lg\lg n})$. It follows that the competitive ratio deteriorates when the additive gap increases.

▶ **Lemma 31.** *Define the constants* $\alpha \equiv 1 - \frac{1}{2^k}$ *and* $\beta \equiv \sum_{j=1}^{k} \frac{j}{2^j} + \frac{k+1}{2^k}$*. Let* X *be a strongly-stable sequence induced by a* (k,r)*-tree. Then* $\hat{c}(GF, X) = 2^k \cdot \beta \cdot (1 - \alpha^r) + \alpha^r$*. In asymptotic terms:* $\hat{c}(GF, X) = \Theta(2^k \cdot (1 - \alpha^r))$*.*

Proof. We write a recurrence for the average cost, c_r, of GF on the strongly-stable sequence induced by T_r. We have $c_0 = 1$, and

$$c_{r+1} = \frac{1+k}{2^k} + \sum_{j=1}^{k} \frac{j + c_r}{2^j} = \Big(1 - \frac{1}{2^k}\Big)c_r + \sum_{j=1}^{k} \frac{j}{2^j} + \frac{1+k}{2^k} \equiv \alpha \cdot c_r + \beta$$

($\frac{1+k}{2^k}$ is due to the actual leaf, and the summation is the contribution of all the T_r subtrees.) By Lemma 20 (with $\gamma = 0$), $c_r = \frac{\beta}{1-\alpha}(1 - \alpha^r) + \alpha^r \cdot c_0 = 2^k \cdot \beta(1 - \alpha^r) + \alpha^r$. Since $\alpha = 1 - \frac{1}{2^k} \in [\frac{3}{4}, 1)$ clearly $\alpha^r < 1$. Furthermore, $\beta = \Theta(1)$. To see this note that β only depends on k. Denote $\beta = \beta(k)$ and observe that: $\beta(k+1) - \beta(k) = \big(\frac{k+1}{2^{k+1}} + \frac{k+2}{2^{k+1}}\big) - \frac{k+1}{2^k} = \frac{1}{2^{k+1}}$. Therefore, $\beta(k) = \beta(2) + \sum_{i=3}^{k} \Big(\beta(i) - \beta(i-1)\Big) = \Big(\frac{1}{2} + \frac{2}{4} + \frac{3}{4}\Big) + \sum_{i=3}^{k} \frac{1}{2^i} = 2 - \frac{1}{2^k}$, and $\beta(k) \in [\frac{7}{4}, 2) \Rightarrow \beta = \Theta(1)$. Because $2^k \cdot (1 - \alpha^r) \geq 2^k \cdot (1 - \alpha) = 1 > \alpha^r$, we conclude that $c_r = \Theta(2^k \cdot (1 - \alpha^r))$. ◀

▶ **Lemma 32.** *Let* X *be a sequence from the family of sequences in Theorem 5, then* $cost(OPT, X) = \Theta(m \cdot \frac{\lg n}{\lg\lg\lg n})$*.*

Proof. Let X be a strongly-stable sequence induced by querying a (k,r)-tree. We know that $\frac{1}{2}\hat{c}(GF, X) < \hat{c}(OPT, X) \leq \hat{c}(GF, X)$ where the lower-bound is by Theorem 22. Therefore, $\hat{c}(OPT, X) = \Theta(2^k \cdot (1 - \alpha^r))$ by Lemma 31. By Lemma 25, $\lg n = r \cdot \lg k + O(1)$, or $r = \frac{\lg n - O(1)}{\lg k}$. When we substitute $r = 2^k$ as in the proof of Theorem 5, we get that $(1 - \alpha^r) = \Theta(1)$ and $2^k = r = \frac{\lg n - O(1)}{\lg k} = \Theta(\frac{\lg n}{\lg\lg\lg n})$. Therefore, $cost(OPT, X) = \Theta(m \cdot \frac{\lg n}{\lg\lg\lg n})$. ◀

As a concluding remark, we recall that the F_r-tree is a (k,r)-tree for $k = 2$. If we substitute $k = 2$ in the formula of Lemma 31 we get that $\alpha = \frac{3}{4}$, $\beta = \frac{7}{4}$, and $\hat{c}(GF, X) = 7 \cdot (1 - (3/4)^r) + (3/4)^r$. By Lemma 26, the average promotion is $3 \cdot (1 - (3/4)^r)$ (for $k = 2$, we have $\delta = 0$). These values are the strongly-stable analogues of Lemma 21, and can be used to show a weaker lower bound of $\frac{7}{4}$, on the competitive ratio of GF.

4 Conclusions and Open Questions

In this paper we gave improved lower bounds on the competitiveness of the Greedy Future (GF) algorithm for serving a sequence of queries by a dynamic binary search tree (BST). In contrast to many of the previous results on GF that are obtained using the geometric-view by studying the equivalent Geometric Greedy (GG) algorithm, we used the standard "tree-view" and the treap-based definition of GF. We showed that the competitive ratio of GF is at

least 2, and that there are sequences $X \in [n]^m$ for which the cost difference (additive gap) between GF and OPT is $\Omega(m \cdot \lg \lg n)$. These lower bounds enabled us to show that if GF is approximately-monotone (Definition 8) with some constant c then $c \geq 2$. Also, the lower bounds show that the cost of GF on a sequence compared to its cost on its reverse, may differ by a factor as close as we like to 2. In contrast, the cost of OPT on a sequence compared to its reverse may differ by at most n.

Our results give new insights on the "tradeoff" between the additive term and the multiplicative term in the competitiveness of GF, showing that the multiplicative term is typically larger when the total cost of the algorithm on the sequence is smaller. Indeed, our best multiplicative term is achieved for a sequence whose average cost per query is 6. This tradeoff is not surprising since a fixed difference implies a larger ratio when the quantities are small. It may be interesting to figure out if this tradeoff hints of some underlying property of GF, or is just an artifact of our technique that requires high costs on average per query in order to increase the additive gap between GF and OPT.

Clearly, these improved lower bounds still don't settle the deeper question of whether GF (and GG) is dynamically-optimal. Our techniques focused on a smaller family of sequences which we named *mixed-stable sequences*, whereas "most" sequences are not stable. While it is possible that an improved lower bound (larger than 2) can be found by a more clever pattern of mixed-stable sequences, it seems more likely to be found by analyzing sequences for which the tree maintained by GF is not static. In addition, we note that GF was not investigated too deeply directly, as most of the work has been done in the geometric view with respect to its counterpart GG. Therefore, studying other problems in tree-view may give complementing insights. One such problem is the deque conjecture, which has been partially settled for GG, in the case when deletions are only allowed on the minimum item [2].

References

1. Parinya Chalermsook, Julia Chuzhoy, and Thatchaphol Saranurak. Pinning down the Strong Wilber 1 Bound for Binary Search Trees. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM)*, pages 33:1–33:21, 2020.
2. Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Greedy is an almost optimal deque. In *14th International Conference on Algorithms and Data Structures (WADS)*, pages 152–165, 2015.
3. Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Pattern-avoiding access in binary search trees. In *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 410–423, 2015.
4. Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. The landscape of bounds for binary search trees. *arXiv*, abs/1603.04892, 2016. `arXiv:1603.04892`.
5. Parinya Chalermsook, Manoj Gupta, Wanchote Jiamjitrak, Nidia Obscura Acosta, Akash Pareek, and Sorrachai Yingchareonthawornchai. Improved pattern-avoidance bounds for greedy BSTs via matrix decomposition. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2023.
6. Parinya Chalermsook and Wanchote Po Jiamjitrak. New binary search tree bounds via geometric inversions. In *28th Annual European Symposium on Algorithms (ESA)*, pages 28:1–28:16, 2020.
7. Erik D. Demaine, Dion Harmon, John Iacono, Daniel Kane, and Mihai Pătraşcu. The geometry of binary search trees. In *20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 496–505, 2009.
8. Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Pătraşcu. Dynamic optimality – almost. *SIAM Journal on Computing*, 37(1):240–251, 2007.

9 Kyle Fox. Upper bounds for maximally greedy binary search trees. In *12th International Conference on Algorithms and Data Structures (WADS)*, pages 411–422, 2011.

10 Michael L. Fredman. Two applications of a probabilistic search technique: Sorting X+Y and building balanced search trees. In *7th Annual ACM Symposium on Theory of Computing (STOC)*, pages 240–244, 1975.

11 George F. Georgakopoulos. Chain-splay trees, or, how to achieve and prove loglogN-competitiveness by splaying. *Information Processing Letters*, 106(1):37–43, 2008.

12 Dion Harmon. *New Bounds on Optimal Binary Search Trees*. PhD thesis, Massachusetts Institute of Technology, 2006.

13 Donald E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1989.

14 László Kozma. *Binary search trees, rectangles and patterns*. PhD thesis, Saarland University, 2016.

15 Caleb Levy and Robert Tarjan. *New Paths from Splay to Dynamic Optimality*. PhD thesis, Princeton, 2019.

16 Joan M. Lucas. Canonical forms for competitive binary search tree algorithms. In *Tech. Rep. DCS-TR-250*. Rutgers University, 1988.

17 Kurt Mehlhorn. Nearly optimal binary search trees. *Acta Informatica*, 5:287–295, 1975.

18 J. Ian Munro. On the competitiveness of linear search. In *8th Annual European Symposium on Algorithms (ESA)*, pages 338–345. Springer-Verlag, 2000.

19 Hauke Reddmann. On the geometric equivalent of instance optimal binary search tree algorithms. Master's thesis, Universität Hamburg, 2021.

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

21 Chengwen Chris Wang, Jonathan Derryberry, and Daniel Dominic Sleator. O(log log n)-competitive dynamic binary search trees. In *17th Annual ACM-SIAM Symposium on Discrete Algorithm (SODA)*, pages 374–383, 2006.

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

A Appendix: Deferred Proofs and Discussions

A.1 Enforcing a Stable Tree for *GF*

We describe how to restructure any initial tree, to a desired tree, when GF is considered. The initial tree cannot simply be re-organized since GF updates the tree in a specific way following each query. Let $P \circ X$ denote the concatenation of the sequences P and X. Note that even if P enforces a desired tree when served alone, serving $P \circ X$ may give a different tree following P when X starts. The reason for this is that GF restructures the tree while serving P according to future queries, therefore the existence of X may affect its decisions while serving P. Nevertheless, we present a simple technique to enforce a tree for GF.

▶ **Theorem 33.** *For any tree T there is a sequence $S(T)$ such that: (1) $|S(T)| = O(n \cdot d(T))$, and more precisely, $|S(T)| \leq \min\left(3n(d(T) - \lfloor \lg n \rfloor + 1), \frac{3}{2}n(n-1)\right)$, and, (2) for any suffix of queries Y, when GF serves $S(T) \circ Y$, its tree when is it done with the last query of $S(T)$ is T. We say that $S(T)$* enforces *T.*

Proof. We enforce the structure of T bottom-up, by first ensuring the position of the leaves, and continuing recursively upwards towards the root. Define $T^{[0]} \equiv T$ and $T^{[i+1]}$ is the tree $T^{[i]}$ stripped of all of its leaves, until the final tree $T^{[h]}$ contains only the root. Observe that $h = d(T)$ (the depth of T). For each tree $T^{[i]}$ we define R_i to be the set of non-leaf nodes in $T^{[i]}$. Note that a non-leaf node may be binary or unary.

We construct $S(T)$ in steps. In step $i \geq 0$ we query R_i in two monotonic phases, where the first phase queries every item twice, and the second phase queries every item once. Denote the queries of this step by S_i. For example, if $R_0 = \{1, 3, 5\}$ then we query in the first step: $S_0 = [1, 1, 3, 3, 5, 5, 1, 3, 5]$. Note that $S_h = \emptyset$ since $R_h = \emptyset$ by definition. The resulting sequence $S(T)$ is the concatenation of the queries in all the (non-empty) steps, that is, $S(T) = S_0 \circ S_1 \circ \ldots \circ S_h$ ($\circ$ for concatenation).

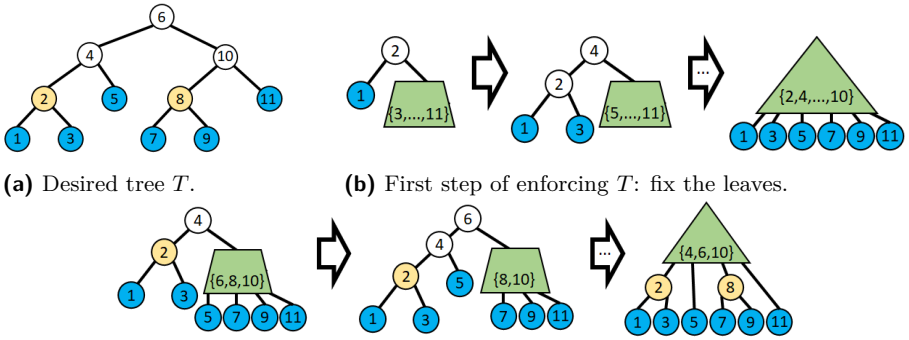
First we analyze $|S(T)|$. Observe that we strip leaves between steps, $|R_{i+1}| \leq |R_i| - 1$. Initially $|R_1| \leq n - 1$ hence $|R_i| \leq n - i$. Also, $h < n$. Therefore we get that: $|S(T)| = \sum_{i=1}^{h} |S_i| = \sum_{i=1}^{h} 3|R_i| \leq 3\sum_{i=1}^{n} (n - i) = \frac{3}{2}n(n - 1)$. To get the other bound, observe that $|R_{h-i}| \leq 2^i - 1$, which is meaningful for the last few steps, i.e., for $i \leq \lfloor \lg n \rfloor$. With that in mind: $|S(T)| = 3\sum_{i=1}^{h-\lfloor \lg n \rfloor} |R_i| + 3\sum_{i=h-\lfloor \lg n \rfloor+1}^{h} |R_i| < 3\sum_{i=1}^{h-\lfloor \lg n \rfloor} n + 3\sum_{j=0}^{\lfloor \lg n \rfloor - 1} 2^j < 3n(h - \lfloor \lg n \rfloor) + 3n$. In conclusion, $|S(T)| < 3n(d(T) - \lfloor \lg n \rfloor + 1)$.

Next we prove that the tree of GF is exactly T when it finishes serving $S(T)$ regardless of any suffix of queries. The proof is by induction on $d(T) = h$. The base case is for $h = 0$. In this case $S(T) = \emptyset$ (because $S_0 = \emptyset$), while also $T^{[h]} = T^{[0]} = T$ contains only the root. Querying nothing is not a problem since there is a unique tree with a single node, and we are done. Now, assume that the claim holds for $h = k$. Consider a tree T of depth $k + 1$. First we show that when GF finishes processing S_0, all the leaves of T are leaves of the tree of GF, and will never be touched (accessed or otherwise), therefore they remain leaves until GF finishes processing $S(T)$. See Figure 7 for a visualization. Indeed, when we query a value u twice in a row, GF brings u to the root when re-ordering the tree after the first query of the couple. Note that since we query R_0 monotonically, by the end of the first phase of S_0 (the queries in pairs), GF has a tree whose left-spine is exactly the items of R_0. Indeed, each item, in turn, is brought to the root and demotes the previous root to the left. Moreover, no value of $T^{[0]} \setminus R_0$ is part of this spine, because of the second phase of queries (monotonously querying the values of R_0). To see this, note that if on the first phase we touch $v \notin R_0$ when r is the root and the pair of queries is to u, such that $r < v < u$, and u is the successor of r in R_0, then the next access to r is closer in the future than that of v, so r will indeed be placed as the left-child of the new root u (and v will be the right child of r). If $r < u < v$ then v does not interfere with r being the left child of u.

Now that we know that all the leaves of $T = T^{[0]}$ are fixed as leaves of the tree of GF when it finishes processing S_0, we can conclude by induction: $S' \equiv S_1 \circ \ldots \circ S_{k+1}$ is exactly the sequence that enforces $T^{[1]}$, and by our inductive assumption, $T^{[1]}$ is enforced correctly. Since the leaves of $T^{[0]}$ are not touched at all during S', we conclude that they must remain hanging off $T^{[1]}$ as "subtrees". The location of each leaf is uniquely determined, and thus we conclude that $S(T)$ indeed enforces $T = T^{[0]}$. ◀

Adding a prefix to our sequence may affect the competitive ratio. However, once we fixed the stable tree, we can repeat the corresponding stable sequence to "amplify" the original competitive ratio making the effect of the prefix negligible. One difficulty raised by repetitions is when we care about the length of the sequence in our claim. This is the case in Theorem 5 where we claim the existence of a sequence of length $n^{\Theta(\frac{\lg \lg n}{\lg \lg \lg n})}$. In the proof of this theorem we assumed for simplicity that we can choose the initial tree. The following remark shows that indeed we can start with an arbitrary initial tree without weakening the theorem.

▶ Remark 34. Let X be an atomic mixed-stable sequence used to prove Theorem 5. Consider the sequence $Z = S \circ X^n$, where S is the prefix (guaranteed by Theorem 33) that is enforcing the desired "initial" tree T that corresponds to X, X^n are n repetitions of X,



(a) Desired tree T.

(b) First step of enforcing T: fix the leaves.

(c) Second step of enforcing T: fix the leaves of $T^{[1]}$.

Figure 7 Example of enforcing a tree as detailed in Theorem 33. The desired tree T is shown in (a). Blue nodes are the leaves of $T^{[0]} = T$ and golden nodes are the leaves of $T^{[1]}$ (leaves of $T^{[2]}$ and $T^{[3]}$ are not colored). The first step of queries is $S_0 = [2, 2, 4, 4, 6, 6, 8, 8, 10, 10, 2, 4, 6, 8, 10]$; (b) shows the states after the first query of 2, the first query of 4, and by the end of S_0. Following the first step, it remains to enforce the remainder of the tree (the inductive step). Concretely, the second step of queries is $S_1 = [4, 4, 6, 6, 10, 10, 4, 6, 10]$; (c) shows the states after the first query of 4, the first query of 6, and by the end of S_1. The green trapezoids and triangles abstract away the structure of some subtrees. The last non-empty step is $S_2 = [6, 6, 6]$ (not shown), after which we get T.

and $\circ$ represents concatenation. There are no unary nodes in T and X queries every leaf at least once so $\frac{n}{2} < \frac{n+1}{2} \leq |X|$. Together with Theorem 33 we get that $n|X| \leq |Z| < n(|X| + \frac{3n}{2}) < 4n|X|$, therefore we have: $|Z| = \Theta(n|X|) = n^{\Theta(\frac{\lg\lg n}{\lg\lg\lg n})}$ (the second equality is by Theorem 5). Since after processing S the tree of GF is fixed: $cost(GF, Z, T_0) - cost(OPT, Z, T_0) \geq n \cdot (cost(GF, X, T) - cost(OPT, X, T))$. By the proof of Theorem 5 and Remark 28: $cost(GF, X, T) - cost(OPT, X, T) = \Omega(|X| \cdot \lg\lg n)$, and putting everything together we get that: $cost(GF, Z, T_0) - cost(OPT, Z, T_0) = \Omega(|Z| \cdot \lg\lg n)$. Note that if $|X| = \Omega(n^2)$ then it suffices to define $Z = S \circ X$ without repetitions and we get that $|Z| = \Theta(|X|)$, and the rest of the arguments remain the same. There may be additional cases where repetitions are not required, e.g., when the depth of T is $\lg n + O(1)$ (in this case, $|S| = O(n)$).

A.2 Omitted Proofs

In this subsection we restate and prove Lemmas and Theorems that were omitted from the main text. For convenience, we restate the claims in their original numbering.

The proof of Theorem 22 makes use of Wilber's first bound [22]. We use the original presentation of this bound which is a bit tighter than later simplified versions such as [14].

▶ **Definition 35** (Wilber's First Bound [22]). *Let X be a sequence of queries, and let T be a static reference tree such that every query of X is in a leaf of T. An alternation at an inner node u of T is defined to be two queries closest in time such that one accesses either the left or right subtree of u and the other accesses the other subtree of u. Define $ALT(u)$ to be the number of alternations at node u. Then: $cost(OPT, X) \geq m + \frac{1}{2}\sum_{inner\ u \in T} ALT(u)$.*

▶ **Theorem 22.** *Let X be a mixed-stable sequence and let T be the tree that corresponds to it. Then $cost(GF, X, T) < c \cdot cost(OPT, X, T)$ for $c = \frac{5}{2}$. If X is strongly-stable, then $c = 2$.*

Proof. We use the tree that corresponds to the mixed-stable sequence as the reference tree for Wilber's first bound. Arithmetic manipulations will yield an expression that we can tie to the cost of GF, according to the claim.

Let X be a mixed-stable sequence, with a corresponding tree T. Let S be the set of values that are in the leaves of T, and let U be the set of inner nodes, $|U| = \frac{n-1}{2}$. We also denote by $A(i)$ the set of proper ancestors of i. By the definition of the cost of a static tree, we know that $\hat{c}(GF, X) = \sum_{i \in S} (d(i) + 1) \cdot f(i)$ where $d(i)$ is the depth of i and $f(i)$ is the frequency of accessing i. We extend $f(u)$ to refer to the frequency of visiting any node u. Note that $f(u) = \sum_{i \in S \wedge u \in A(i)} f(i)$ and that $\sum_{i \in S} f(i) = 1$.

Now consider Wilber's bound for X, with T as the reference tree. We can use T as the reference tree since X only accesses leaves of T, by definition. We also denote $\alpha_u \equiv \frac{ALT(u)+1}{f(u) \cdot m}$ ($ALT(u)$ is defined in Defintion 35, and note that $0 \leq ALT(u) \leq f(u) \cdot m - 1$). We have $\alpha_u \in (0, 1]$, where $\alpha_u = 1$ corresponds to fully alternating accesses to the subtree rooted at u. The lower bound is $cost(OPT, X) \geq m + \frac{1}{2} \sum_{u \in U} (ALT(u) + 1) - \frac{|U|}{2} = \frac{m}{2} + \frac{m}{2}(1 + \sum_{u \in U} \alpha_u \cdot f(u)) - \frac{n-1}{4} = \left(\frac{m}{2} - \frac{n-1}{4}\right) + \frac{m}{2} \sum_{i \in S} (1 + \sum_{u \in A(i)} \alpha_u) f(i)$. Let $\alpha \leq \min_{u \in U} \alpha_u$, we get that $\hat{c}(OPT, X) \geq \left(\frac{1}{2} - \frac{n-1}{4m}\right) + \frac{\alpha}{2} \sum_{i \in S} (d(i) + 1) \cdot f(i) = \frac{\alpha}{2} \hat{c}(GF, X) + \left(\frac{1}{2} - \frac{n-1}{4m}\right)$ where the equality holds since GF maintains a static tree. Thus $\hat{c}(GF, X) \leq \frac{2}{\alpha} \cdot \hat{c}(OPT, X) - \frac{1}{\alpha}\left(1 - \frac{n-1}{2m}\right) < \frac{2}{\alpha} \cdot \hat{c}(OPT, X)$.

In order to choose a suitable α, recall that a strongly-stable node u has a coefficient of $\alpha_u = 1$, which means that for strongly-stable sequences, in which all inner nodes are stable, we can pick $\alpha = 1$ and conclude that $\hat{c}(GF, X) < 2 \cdot \hat{c}(OPT, X)$. If u is a weakly-stable node, then its coefficient is $\alpha_u = \frac{2}{3}$. So for a mixed-stable sequence we can naively pick $\alpha = \frac{2}{3}$, resulting in $\hat{c}(GF, X) < 3 \cdot \hat{c}(OPT, X)$.

In order to improve from 3 to $\frac{5}{2}$, we observe that by definition, every weakly-stable node has a strongly-stable child. Let u be a weakly-stable node and let w be its (strongly-stable) favored-child (recall Definition 11). Since $ALT(u) = ALT(w)$ (by definition of the access pattern in u), we can present Wilber's bound differently, summing $(ALT(u)+1) \cdot (1+\beta) + (ALT(w)+1) \cdot (1-\beta)$ instead of $(ALT(u)+1) + (ALT(w)+1)$. We get modified coefficients $\alpha'_u = \frac{(ALT(u)+1) \cdot (1+\beta)}{m \cdot f(u)} = \alpha_u \cdot (1 + \beta) = \frac{2(1+\beta)}{3}$ and similarly $\alpha'_w = \alpha_w(1 - \beta) = (1 - \beta)$. Choosing $\beta = \frac{1}{5}$ balances the coefficients: $\alpha'_u = \alpha'_w = \frac{4}{5}$. Now we can choose $\alpha = \frac{4}{5}$, and get $\hat{c}(GF, X) < \frac{5}{2} \cdot \hat{c}(OPT, X)$ for mixed-stable sequences. ◀

▶ **Theorem 6.** *For any $\epsilon > 0$ there exists a sequence X with a subsequence (not necessarily consecutive) $X' \subseteq X$ such that $cost(GF, X') \geq (2 - \epsilon) \cdot cost(GF, X)$.*

Proof. Denote the initial tree by T_0. Let Z be the weakly-stable sequence used for proving Theorem 4. Let T_P be the tree that corresponds to Z and T_Q the optimized tree, in which the leaves are promoted as in Lemma 21. Let P and Q be the sequences that enforce T_P and T_Q by Theorem 33, respectively. Note that ϵ determines Z, P and Q since it tells us how close to a ratio of 2 we need to get.

Revisit Figure 5 to see the (recursive) structures of T_P (on the left) and T_Q (on the right, post-promotions). Observe that T_P remains static when GF serves Z with it, by definition. Moreover, T_Q remains static when GF serves Z with it. Indeed, let r be the root of T_Q. Z queries the item in r every third access and the other accesses are alternating between its left and right subtrees, hence r remains the root of T_Q. The rest of T_Q remains static recursively.

Define $X = P \circ Q \circ Z^k$ for a large k, and $X' = P \circ Z^k \subset X$ ($\circ$ for concatenation). Since GF does not change T_P and T_Q while serving Z we get that $\frac{cost(GF,X')}{cost(GF,X)} = \frac{cost(GF,P,T_0) + k \cdot cost(GF,Z,T_P)}{cost(GF,P \circ Q,T_0) + k \cdot cost(GF,Z,T_Q)}$. This ratio approaches $\frac{cost(GF,Z,T_P)}{cost(GF,Z,T_Q)}$ for large enough k, and

since T_p and T_Q are exactly the trees used in the proof of Lemma 21, we conclude that we can make the resulting ratio as close to 2 as we like (choosing Z, P, Q according to the desired ϵ). ◀

▶ **Theorem 7.** *Let S be a sequence, we define $rev(S)$ to be the sequence S in reverse. For any $\epsilon > 0$ there exists a sequence X such that $cost(GF, rev(X)) \geq (2 - \epsilon) \cdot cost(GF, X)$.*

Proof. The proof is similar to that of Theorem 6, and we define Z, T_0, P, T_P, Q and T_Q the same way. Here we define $X = Q \circ (rev(Z))^{k+1} \circ rev(P)$, for a large k.

We claim that T_Q remains static when GF serves $rev(Z)$ over it, rather than Z, by the same argument as in the proof of Theorem 6, because the interleaving pattern in the root is preserved under reversal. Moreover, $cost(GF, rev(Z), T_Q) = cost(GF, Z, T_Q)$ because the cost on a static tree depends only on the access frequencies. Putting everything together, we get: $\frac{cost(GF,rev(X))}{cost(GF,X)} = \frac{cost(GF,P,T_0)+k\cdot cost(GF,Z,T_P)+cost(GF,Z\circ rev(Q),T_P)}{cost(GF,Q,T_0)+k\cdot cost(GF,rev(Z),T_Q)+cost(GF,rev(Z)\circ rev(P),T_Q)}$. Note that the suffix contains one repetition of Z so that the rest of it ($rev(P)$ or $rev(Q)$) does not affect the restructuring decisions of GF during the earlier repetitions of Z. The limit of this ratio for large k is $\frac{cost(GF,Z,T_P)}{cost(GF,Z,T_Q)}$. We finish the argument as in the proof of Theorem 6. ◀