# Hardness Amplification for Dynamic Binary Search Trees


**Shunhua Jiang**
Columbia University, New York, NY, USA

**Victor Lecomte**
Stanford University, CA, USA

**Omri Weinstein**
The Hebrew University of Jerusalem, Israel

**Sorrachai Yingchareonthawornchai**
The Hebrew University of Jerusalem, Israel



## Abstract

We prove direct-sum theorems for Wilber's two lower bounds [Wilber, FOCS'86] on the cost of access sequences in the binary search tree (BST) model. These bounds are central to the question of dynamic optimality [Sleator and Tarjan, JACM'85]: the *Alternation* bound is the only bound to have yielded online BST algorithms beating $\log n$ competitive ratio, while the *Funnel* bound has repeatedly been conjectured to exactly characterize the cost of executing an access sequence using the optimal tree [Wilber, FOCS'86, Kozma'16], and has been explicitly linked to splay trees [Levy and Tarjan, SODA'19]. Previously, the direct-sum theorem for the Alternation bound was known only when approximation was allowed [Chalermsook, Chuzhoy and Saranurak, APPROX'20, ToC'24].

We use these direct-sum theorems to amplify the sequences from [Lecomte and Weinstein, ESA'20] that separate between Wilber's Alternation and Funnel bounds, increasing the Alternation and Funnel bounds while optimally maintaining the separation. As a corollary, we show that Tango trees [Demaine et al., FOCS'04] are optimal among any BST algorithms that charge their costs to the Alternation bound. This is true for *any* value of the Alternation bound, even values for which Tango trees achieve a competitive ratio of $o(\log \log n)$ instead of the default $O(\log \log n)$. Previously, the optimality of Tango trees was shown only for a limited range of Alternation bound [Lecomte and Weinstein, ESA'20].



**2012 ACM Subject Classification** Theory of computation → Data structures design and analysis; Theory of computation → Online algorithms

**Keywords and phrases** Data Structures, Amortized Analysis

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2024.42

**Related Version** *Full Version*: https://arxiv.org/pdf/2411.14387

**Funding** This research was supported by the European Research Council (ERC) Starting grant CODY 101039914.

**Acknowledgements** This work was done in part while the authors (SJ, OW, SY) were visiting the Simons Institute for the Theory of Computing, UC Berkeley.


## 1 Introduction

Direct Sum theorems assert a lower bound on a certain complexity measure $\mathcal{C}$ of a *composed*[1] problem $f \circ g$ in terms of the individual complexities of $f$ and $g$, ideally of the form $\mathcal{C}(f \circ g) \approx \mathcal{C}(f) + \mathcal{C}(g)$. Direct Sums have a long history in complexity theory, as they provide

[1] Formally speaking, direct-sum problems pertain to the complexity of solving $k$ separate copies of a problem $f$, rather than computing a composed function of $k$ copies $g(f(x_1), \ldots, f(x_k))$, but it is common to refer to both variations of the $k$-fold problem as direct-sums [23].

a *black-box* technique for amplifying the hardness of computational problems $\mathcal{C}(f^{\circ k}) \gtrsim k \cdot \mathcal{C}(f)$, and are the most promising approach for proving several holy-grail lower bounds in complexity theory [23, 20, 35, 2, 24]. Moreover, a "tensorization" property of $\mathcal{C}$ under composition allows to "lift" the problem and leverage its asymptotic behavior (e.g., concentration), which is not present in the single-copy problem – this feature has been demonstrated and exploited in various models, including combinatorial Discrepancy [28, 38], Richness of data structure problems [33], decision trees [35] and rank [24] to mention a few. Despite their powerful implications, (strong) direct-sum scaling of composed problems are often simply false [34, 39, 37], and highly depend on the underlying computational model.

In this paper, we study direct sums in the *online BST model*, motivated by the *dynamic optimality* conjecture of Sleator and Tarjan [40]. The dynamic optimality conjecture postulates the existence of an *instance optimal* binary search tree algorithm (BST), namely, an online self-adjusting BST whose running time[2] matches the best possible running time *in hindsight* for any sufficiently long sequence of queries. More formally, denoting by $\mathcal{T}(X)$ the operational time of a BST algorithm $\mathcal{T}$ on an access sequence $X = (x_1, \ldots, x_m) \in [n]^m$ of keys to be searched, the conjecture says that there is an online BST $\mathcal{T}$ such that $\forall X$, $\mathcal{T}(X) \le O(\mathsf{OPT}(X))$, where $\mathsf{OPT}(X) := \min_{\mathcal{T}'} \mathcal{T}'(X)$ denotes the optimal offline cost for $X$. In their seminal paper, Sleator and Tarjan [40] conjectured that *splay trees* are $O(1)$-competitive; A more recent competitor, the *GreedyFuture* algorithm [30, 15, 31], also forms a compelling candidate for constant-competitive dynamic optimality. However, the near-optimality of both Splay trees and GreedyFuture was proven only in special cases [41, 18, 32, 7, 8, 21, 10, 13], and they are not known to be $o(\log n)$-competitive for general access sequences $X$ (note that every balanced BST is trivially $O(\log n)$-competitive). After 35 years of active research, the best provable bound to date is an $O(\log \log n)$-competitive BST, starting with *Tango trees* [16], among other $O(\log \log n)$-competitive BST algorithms [5, 42, 6]. Interestingly, this progress was made possible due the development of *lower bounds* in the BST model, as we discuss next.

Indeed, a remarkable feature of the BST model – absent from general computational models (e.g., word-RAM) – is that it allows for nontrivial lower bounds on the search time of a *fixed* query sequence $X$: In general models, lower bounds against a specific input $X$ do not make much sense as the best algorithm in hindsight can simply "store and read-off the answer" for $X$. Nevertheless, in the BST model, even an all-knowing binary search tree must pay the cost of traversing the root-to-leaf path to retrieve keys. For example, there are classical examples of deterministic access sequences (e.g., *bit-reversal* sequence [43]) that require the worst case $\Omega(m \log n)$ total search time. This feature is what makes instance-optimality in the BST model an intriguing possibility. Our work focuses on two classic lower bounds due to Wilber [43], the Alternation and Funnel bounds (a.k.a, Wilber's first and second bounds), which are central to the aforementioned developments.

**The Alternation and Funnel Bounds.** Essentially all BST lower bounds are derived from a natural geometric interpretation of the access sequence $X = (X_1, \ldots, X_m)$ as a point set on the plane, mapping the $i^{\text{th}}$ access $X_i$ to point $(X_i, i)$ ([15, 22], see Figure 1). The earliest lower bounds on $\mathsf{OPT}(X)$ were proposed in an influential paper of Wilber [43]. The *alternation bound* $\mathsf{Alt}_{\mathcal{T}}(X)$ counts the total number of left/right alternations obtained by searching the keys $X = (X_1, \ldots, X_m)$ on a *fixed* (static) binary search tree $\mathcal{T}$, where alternations are summed up over all nodes $v \in \mathcal{T}$ of the "reference tree" $\mathcal{T}$ (see Figure 3

[2] i.e. the number of pointer movements and tree rotations performed by the BST

and the formal definition in Section 2). Thus, the Alternation bound is actually a family of lower bounds, optimized by the choice of the reference tree $\mathcal{T}$, and we henceforth define $\mathsf{Alt}(X) := \max_{\mathcal{T}} \mathsf{Alt}_{\mathcal{T}}(X)$. The Alternation bound plays a key role in the design and analysis of Tango trees and their variants [16, 42]. In fact, *all* non-trivial $o(\log n)$-competitive BST algorithms [16, 5, 42, 6, 11] rely on the Alternation bound.

In the same paper, Wilber proposed another lower bound – the *Funnel Bound* $\mathsf{Funnel}(X)$ – which is less intuitive and can be defined by the following process: Consider the geometric view $\{(X_i, i)\}_{i\in[m]}$ of the simple "move-to-root" algorithm that simply rotates each searched key $X_i$ to the root by a series of single rotations. Then $\mathsf{Funnel}(X_i, i)$ is exactly the number of *turns* on the path from the root to $X_i$ right before it is accessed [1, 22]. The Funnel bound is then defined as $\mathsf{Funnel}(X) := \sum_{i=1}^{m} \mathsf{Funnel}(X_i, i)$. This view emphasizes the *amortized* nature of the Funnel bound: at any point, there could be linearly many keys in the tree that are only *one* turn away from the root, so one can only hope to achieve this bound in some amortized fashion.

The Funnel bound has been repeatedly conjectured to tightly characterize the cost of an offline optimal algorithm [43, 25, 6, 27]. Recently, Lecomte and Weinstein [27] proved that the funnel bound is *rotation-invariant*, meaning that the bound is preserved when the geometric representation of the input sequence is rotated by 90 degrees. This property also holds for an optimal algorithm [15], giving another evidence that the Funnel bound might give a tight characterization of the cost of an offline optimal algorithm. While the Funnel bound does not have an algorithmic interpretation like $\mathsf{Alt}(X)$, Levy and Tarjan [29] recently observed interesting similarities between Splay trees and the Funnel bound. The core difficulty in converting $\mathsf{Funnel}(X)$ into a BST algorithm is its highly *amortized* nature (also a feature of Splay trees), compared to the Alternation bound which gives a point-wise lower bound on the retrieval time of $X_i \in X$. As such, understanding the mathematical properties of the Funnel bound is important in its own right.

**Access Sequence Composition and Known Direct-Sum Results.** Informally, Direct Sum theorems assert a lower bound on the complexity measure of solving $R$ copies of a problem in a given computational model, in terms of the cost of solving a single copy, ideally $C(f^{\circ R}) \gtrsim \Omega(R) \cdot C(f)$, where $f^{\circ R}$ denotes certain $R$-copy *composed* problem. Indeed, the precise notion of composition we use here (a-la [9]) is crucial, as direct-sum theorems are subtle and often turn out to be *false* [34].

A natural definition of sequence-composition in the BST model was introduced by Chalermsook et al. [9]. Let $X^{(1)}, \ldots, X^{(\ell)}$ be a sequence of $\ell$ access sequences where $X^{(i)} \in [n_i]^{m_i}$. That is, each sequence $X^{(i)}$ has $n_i$ keys and $m_i$ accesses where $m := \sum_i m_i, n := \sum_i n_i$. We view each sequence $X^{(i)}$ as the $i^{\text{th}}$ queue where we dequeue elements by the order of the sequence $X^{(i)}$ (i.e., in FIFO order). Let $\widetilde{X} \in [\ell]^m$ be a sequence with keys in $[\ell]$ such that every $j \in [\ell]$ appears exactly $m_j$ times. We can view $\widetilde{X}$ as a *template* which defines the ordering of dequeue operations among the $\ell$ queues. We define the *composed sequence* $X := \widetilde{X}(X^{(1)}, \ldots, X^{(\ell)}) \in [n]^m$ as follows. For each $t = 1$ to $m$, $X_t := q_t + \sum_{i<\tilde{X}_t} n_i$ where $q_t$ is the next element dequeued from $X^{(\tilde{X}_t)}$. We refer the reader to Section 2 for the precise definition.

The direct sum results for the optimal cost are well understood with applications to proving non-trivial bounds of binary search trees. In [9], they prove (approximate) *subadditivity* of the optimal cost on composed sequences. That is,

$$\mathsf{OPT}(X) \leq 3 \cdot \mathsf{OPT}\Big(\widetilde{X}\Big) + \sum_j \mathsf{OPT}\Big(X^{(j)}\Big).$$

The subadditivity of optimal cost finds application in proving the linear optimal bounds for "grid" sequences, and a strong separation in the hierarchy of *lazy finger bounds* [9]. On the other hand, [6] recently proved *superadditivity* of the optimal cost on composed sequences. That is,

$$\mathsf{OPT}(X) \geq \mathsf{OPT}\Big(\widetilde{X}\Big) + \sum_j \mathsf{OPT}\Big(X^{(j)}\Big). \tag{1}$$

The superadditivity of optimal cost finds an application in designing a new $O(\log \log n)$-competitive online BST algorithm based on purely geometric formulation [6].

However, the direct sum results for Wilber's bounds are poorly understood despite their importance to the pursuit of dynamic optimality. The only published work that we are aware of is the approximate subadditivity of the Alternation bound [6]. That is, they proved that

$$\mathsf{Alt}(X) \leq 4 \cdot \mathsf{Alt}\Big(\widetilde{X}\Big) + 8 \cdot \sum_j \mathsf{Alt}\Big(X^{(j)}\Big) + O(|X|). \tag{2}$$

Their proof is quite involved, and it is based on geometric arguments and the probabilistic method. This finds applications in proving the separation between the Alternation and Funnel bounds. In [6], they used approximate subadditivity of the Alternation bound (Equation (2)) to prove a near-optimal separation between the Alternation and Funnel bounds. That is, they constructed a sequence $Y$ such that the gap between $\mathsf{Alt}(Y)$ and $\mathsf{Funnel}(Y)$ is as large as $\Omega(\frac{\log \log n}{\log \log \log n})$. This gap is nearly optimal because the upper bound of Tango tree [16] implies that the gap must be $O(\log \log n)$. This gap has been closed by an independent work by Lecomte and Weinstein [27], proving the optimal separation between the Alternation and Funnel bounds. That is, they constructed an instance $Y$ such that the gap between $\mathsf{Alt}(Y)$ and $\mathsf{Funnel}(Y)$ is as large as $\Omega(\log \log n)$.

Furthermore, [6] also used the approximate subadditivity of the Alternation bound (Equation (2)) to prove the $\Omega(\frac{\log \log n}{\log \log \log n})$ gap between $\mathsf{Alt}(Y)$ and $\mathsf{cGB}(Y)$ where $\mathsf{cGB}(Y)$ denotes the *Consistent Guillotine Bound* (cGB), a lower bound measure that is an extension of $\mathsf{Alt}(Y)$.

## 1.1 Our Results

We prove that the Alternation bound is subadditive whereas the Funnel bound is superadditive for composed sequences. More precisely, we prove the following theorem.

▶ **Theorem 1** (Direct-Sum Theorem for Wilber's Bounds). *Let* $X := \widetilde{X}(X^{(1)}, \ldots, X^{(l)})$ *be a composed sequence. Then*

- $\mathsf{Alt}(X) \leq \mathsf{Alt}\Big(\widetilde{X}\Big) + \sum_j \mathsf{Alt}\big(X^{(j)}\big) + O(|X|)$, *and*
- $\mathsf{Funnel}(X) \geq \mathsf{Funnel}\Big(\widetilde{X}\Big) + \sum_j \mathsf{Funnel}\big(X^{(j)}\big) - O(|X|)$.

Our proof of the subadditivity of the Alternation bound is simpler and yields stronger bounds than the proof in [6] (Theorem 3.6 in their arXiv version). Direct sum theorems are a natural black-box tool for hardness amplification, as they effectively reduce complex lower bounds to a simpler "one-dimensional" problem. Indeed, as a showcase application, we use the base-case separation proved in [27] along with Theorem 1 to amplify both Wilber's bounds. Let $\overline{\mathsf{Alt}}(X) := \mathsf{Alt}(X)/|X|$, and $\overline{\mathsf{Funnel}}(X) := \mathsf{Funnel}(X)/|X|$. They proved that there is a sequence $Y$ such that

$$\overline{\mathsf{Alt}}(Y) \leq O(1), \text{but}$$
$$\overline{\mathsf{Funnel}}(Y) \geq \Omega(\log \log n).$$

Note that the sequence $Y$ is *easy* w.r.t. the Alternation bound since $\overline{\mathsf{Alt}}(Y) \leq O(1)$. We use the sequence $Y$ as a base-case and apply Theorem 1 to construct *hard* sequence w.r.t. the funnel bound while maintaining the separation as in the following theorem [3].

▶ **Theorem 2** (Hardness Amplification). *There is a constant $K > 0$ such that for any $n$ of the form $2^{2^r}$ and any power-of-two $R \leq \frac{\log n}{K}$, there is a sequence $Y_n^{\circ R} \in [n]^{m'}$ with $m' \leq \mathrm{poly}(n)$ such that*

$$\overline{\mathsf{Alt}}(Y_n^{\circ R}) \leq O(R)$$
$$\overline{\mathsf{Funnel}}(Y_n^{\circ R}) \geq \Omega\left(R \log\left(\frac{\log n}{R}\right)\right).$$

**Remark 1.** We emphasize that the approximate subadditivity of the Alternation bound (Equation (2)) is not sufficient for such hardness amplification. On the other hand, one could also use the superadditivity of the optimal cost (Equation (1)) instead of the Funnel bound to prove hardness amplification.

**Tightness of the Separation.** As a corollary of Theorem 2, we can derive the following trade-offs between the multiplicative and additive factors for the Alternation bound.

▶ **Theorem 3.** *Let $\alpha, \beta : \mathbb{N} \to \mathbb{R}_{\geq 1}$ be any functions such that some BST algorithm achieves an amortized cost of $\alpha(n)\overline{\mathsf{Alt}}(X) + \beta(n)$ for all access sequences $X$ over $n$ keys. Then $\alpha(n) \geq \Omega\left(\log\left(\frac{\log n}{\beta(n)}\right)\right)$.*

As we discuss below, the trade-offs are tight with the matching upper bounds from Tango Trees (which can be derived directly from Tango trees). For convenience, we present self-contained BST algorithms with the matching upper bounds .

▶ **Lemma 4.** *There is a BST algorithm that takes an integer $k \leq \log n$ as a parameter and serve the sequence $X = (X_1, \cdots, X_m)$ with the total access time of*

$$O\left(\left(\mathsf{Alt}(X) + m \cdot \frac{\log n}{k}\right) \cdot (\log k + 1)\right).$$

For the reason of space, we defer the proof of Lemma 4 to the full version.

In this algorithm, the additive cost is $\Theta\left(\frac{\log n \log k}{k}\right)$ and multiplicative cost $\Theta(\log k)$. By Theorem 3, if $\beta(n) = \Theta\left(\frac{\log n \log k}{k}\right)$, then we have

$$\alpha(n) \geq \Omega\left(\log\left(\frac{\log n}{\beta(n)}\right)\right) = \Omega\left(\log\left(\frac{k}{\log k}\right)\right) = \Omega(\log k),$$

so our trade-off is optimal for any sufficiently large $k \leq \log n$.

**Optimality of Tango Trees.** As another corollary of Theorem 2, we prove the optimality of Tango Trees among any algorithm charging its cost to Wilber's Alternation bound for all values of the $\overline{\mathsf{Alt}}(X)$. Note that $\overline{\mathsf{Alt}}(X) \leq O(\log n)$. Previously, the optimality of Tango trees is known only when $\overline{\mathsf{Alt}}(X) = O(1)$ [27].

[3] This can be viewed as hardness amplification because the new sequence becomes harder from the optimum's point of view without losing the gap too much.

The basic idea of Tango Trees is to "mimic" Wilber's alternation bound via a BST, by dynamically maintaining a partition of the reference tree $\mathcal{T}$ into *disjoint paths*, formed by designating, for each node $x \in \mathcal{T}$, the unique "preferred" descendant in $\mathcal{T}$ (left or right) which was *acessed most recently.* Since each "preferred path" has length $|p| \leq \text{depth}(\mathcal{T}) \leq \log n$, every *path* can itself be stored in a BST, so assuming these paths can be dynamically maintained (under split and joins), searching for the predecessor of a key $x_i$ *inside* each path only takes $O(\log\log n)$ time, until the search "falls-off" the current preferred path and switches to a different one. The key observation is that this "switch" can be charged to $\mathsf{Alt}_{\mathcal{T}}(X)$, as it certifies a new alternation in Wilber's lower bound, hence OPT must pay for this move as well. This elegant argument directly leads to the aforementioned $O(\log\log n) \cdot \mathsf{OPT}(X)$ search time.

The analysis of Tango trees relies on charging the algorithm's cost to the Alternation bound. One may ask if the bound can be improved using a clever algorithm (not necessarily Tango trees) so that we can charge $o(\log\log n)$ factor to the Alternation bound. Unfortunately, the answer is no as there are known examples of access sequences $\tilde{X}$ with $\mathsf{Alt}(\tilde{X}) = O(m)$ but $\mathsf{OPT}(\tilde{X}) = \Theta(m \log\log n)$ [27, 6]. In light of this, Tango trees are indeed off by a factor $\Theta(\log\log n)$ from $\mathsf{Alt}(\tilde{X})$. Interestingly, when $\mathsf{OPT}(X) \gtrsim m\frac{\log n}{2^{o(\log\log n)}}$, one can do somewhat better. Let $\overline{\mathsf{OPT}}(X) := \mathsf{OPT}(X)/|X|$. The Tango tree, presented by [16] (see the discussion in their Section 1.5), has a competitive ratio of

$$O\left(1 + \log\frac{\log n}{\overline{\mathsf{OPT}}(X)}\right) = o(\log\log n). \tag{3}$$

The condition that allows $o(\log\log n)$-competitiveness is rather narrow: the amortized optimal cost $\overline{\mathsf{OPT}}(X)$ must be very close to $\log n$ to achieve $o(\log\log n)$-competitiveness. Can we achieve $o(\log\log n)$-competitiveness with a wider range of $\overline{\mathsf{OPT}}(X)$ using $\mathsf{Alt}(X)$?

Unfortunately, the answer is no. More generally, we prove a matching lower bound: the competitive ratio of any BST algorithm based on the Alternation bound must be at least $\Omega(\log\frac{\log n}{\overline{\mathsf{OPT}}(X)})$, matching to the upper bound of the Tango tree by Equation (3). More precisely, we prove the following theorem whose proof is a small modification of the proof in Theorem 3.

▶ **Theorem 5.** *Let $\alpha : \mathbb{N} \times \mathbb{R}_{\geq 1} \to \mathbb{R}_{\geq 1}$ be any function such that some BST algorithm achieves an amortized cost of $\alpha\big(n, \overline{\mathsf{OPT}}(X)\big) \cdot \big(\overline{\mathsf{Alt}}(X) + 1\big)$ for all access sequences $X$ over $n$ keys. Then $\alpha(n, s) \geq \Omega\Big(\log\Big(\frac{\log n}{s}\Big)\Big)$.*

As a corollary of Theorem 5,[4] the lower bound of $\Omega(\log\frac{\log n}{\overline{\mathsf{OPT}}(X)})$ follows by setting $s$ to be within constant factor of $\overline{\mathsf{OPT}}(X)$. This holds for every BST algorithm based on the Alternation bound. With the matching upper bound by Equation (3), Tango tree optimally utilizes the Alternation bound.

## 1.2 Further Related Work

Splay trees and GreedyFuture are prime candidates for the dynamic optimality conjecture since they both satisfy many important properties of dynamic trees including static optimality [40], working-set property [40, 19], dynamic finger property [14, 12], and more (see the

[4] We remark that we cannot set $\beta(n) = \overline{\mathsf{OPT}}(X)$ in Theorem 3 because $\beta(n)$ does not depend on $X$ and, even if $\overline{\mathsf{OPT}}(X)$ is a function of $n$, the construction of the sequence $X$ depends on $\beta(n)$. We need the lower bound of Theorem 5.

surveys [22, 25] for an overview of the results in the field). Although they are not yet known to have $o(\log n)$-competitiveness, they have substantially better bounds for special cases. For example, both Splay trees and GreedyFuture are dynamically optimal for *sequential access* sequences [18, 19]. For *deque sequences*, Splay trees are $O(\alpha^*(n))$-competitive [32] whereas GreedyFuture is $O(\alpha(n))$-competitive [10]. Here, $\alpha(n)$ denotes the inverse Ackermann function and $\alpha^*(n)$ is the iterated function of $\alpha(n)$. The sequential and deque sequences are special cases of the *pattern-avoiding access* sequences [8]. For any fixed-size pattern, GreedyFuture is $O(2^{(1+o(1))\alpha(n)})$-competitive for pattern-avoiding access sequences [13]. It was shown recently that an optimal BST algorithm takes $O(n)$ total cost for any fixed pattern [3]. The bounds for specific classes of patterns can be improved if preprocessing is allowed [8, 21] (i.e., the initial tree can be set before executing the online search sequences). Recently, a slight modification of GreedyFuture was shown to be $O(\sqrt{\log n})$-competitive [11]. An important application of GreedyFuture (or any competitive online BSTs) includes adaptive sorting using treesort [4, 13] and heapsort [26].

The lower bounds in the literature other than Wilber's bounds include the *maximum independent rectangle* and *SignedGreedy* bounds [15], which subsume Wilber's Alternation and Funnel bounds. A similar lower bound was presented by Derryberry et al. [17]. Recently, *Guillotine* Bound [6] was introduced, which is a generalization of Wilber's Alternation bound. Unfortunately, it is unclear how to design an algorithm that utilizes these bounds. Recently, Sadeh and Kaplan [36] proved that the competitive ratio of GreedyFuture cannot be less than 2 for the multiplicative factor, or $o(m \log\log n)$ for the additive factor.

### 1.3 Paper Organization

We first describe terminologies and notations in Section 2. We prove the direct-sum results for Wilber's bounds (Theorem 1) in Section 3. In Section 4, we prove the hardness amplification of Wilber's bounds while maintaining their separation (Theorem 2) and we also prove Theorem 3.

## 2 Preliminaries

We follow notations and terminologies from [27].

▶ **Definition 6** (access sequence). *An access sequence is a finite sequence $X = (X_1, \dots, X_m) \in S^m$ of values from a finite set of keys $S \subseteq \mathbb{R}$. Usually, we let $S := [n]$.*

To make our definitions and proofs easier, we will work directly in the geometric representation of access sequences as (finite) sets of points in the plane $\mathbb{R}^2$.

▶ **Definition 7** (geometric view). *Any access sequence $X = (X_1, \dots, X_m) \in S^m$ can be represented as the set of points $G_X := \{(X_t, t) \mid t \in [m]\}$, where the $x$-axis represents the key and the $y$-axis represents time (see Figure 1).*

By construction, in $G_X$, no two points share the same $y$-coordinate. We will say such a set has "distinct $y$-coordinates". In addition, we note that it is fine to restrict our attention to sequences $X$ without repeated values.[5] The geometric view $G_X$ of such sequences also has no two points with the same $x$-coordinate. We will say that such a set has "distinct $x$- and $y$-coordinates".

[5] Indeed, Appendix E in [8] gives a simple operation that transforms any sequence $X$ into a sequence $\text{split}(X)$ without repeats such that $\mathsf{OPT}(\text{split}(X)) = \Theta(\mathsf{OPT}(X))$. Thus if we found a tight lower bound $L(X)$ for sequences without repeats, a tight lower bound for general $X$ could be obtained as $L(\text{split}(X))$.

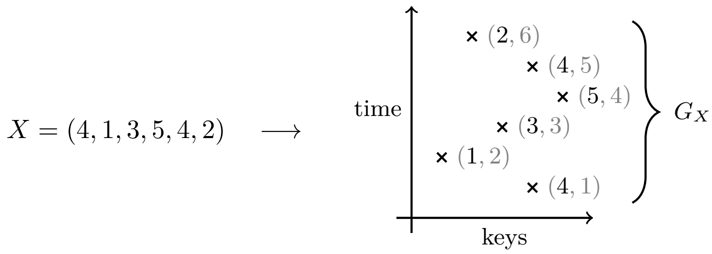



**Figure 1** transforming $X$ into its geometric view $G_X$.

▶ **Definition 8** ($x$- and $y$-coordinates)**.** *For a point $p \in \mathbb{R}^2$, we will denote its $x$- and $y$-coordinates as $p.x$ and $p.y$. Similarly, we define $P.x := \{p.x \mid p \in P\}$ and $P.y := \{p.y \mid p \in P\}$.*

We start by defining the *mixing value* of two sets: a notion of how many two sets of numbers are interleaved. It will be useful in defining both the alternation bound and the funnel bound. We define it in a few steps.

▶ **Definition 9** (mixing string)**.** *Given two disjoint finite sets of real numbers $L, R$, let $\text{mix}(L, R)$ be the string in $\{\texttt{L}, \texttt{R}\}^*$ that is obtained by taking the union $L \cup R$ in increasing order and replacing each element from $L$ by* L *and each element from $R$ by* R*. For example, $\text{mix}(\{2, 3, 8\}, \{1, 5\}) = \texttt{RLLRL}$.*

▶ **Definition 10** (number of switches)**.** *Given a string $s \in \{\texttt{L}, \texttt{R}\}^*$, we define $\#\text{switches}(s)$ as the number of side switches in $s$. Formally,*

$$\#\text{switches}(s) := \#\{t \mid s_t \neq s_{t+1}\}.$$

*For example, $\#\text{switches}(\texttt{LLLRLL}) = 2$. Note that if we insert characters into $s$, $\#\text{switches}(s)$ can only increase.*

▶ **Definition 11** (mixing value)**.** *Let $\text{mixValue}(L, R) := \#\text{switches}(\text{mix}(L, R))$ (see Figure 2).*

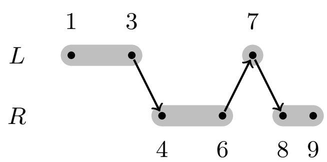



**Figure 2** a visualization of $\text{mixValue}(\{1, 3, 7\}, \{4, 6, 8, 9\}) = 3$.

The mixing value has some convenient properties, which we will use later:

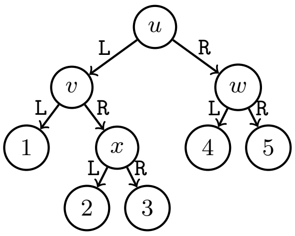
▶ **Fact 12** (properties of mixValue)**.** *Function $\text{mixValue}(L, R)$ is:*

**(a)** *symmetric:* $\text{mixValue}(L, R) = \text{mixValue}(R, L)$;
**(b)** *monotone: if $L_1 \subseteq L_2$ and $R_1 \subseteq R_2$, then $\text{mixValue}(L_1, R_1) \leq \text{mixValue}(L_2, R_2)$;*
**(c)** *superadditive under concatenation: if $L_1, R_1 \subseteq (-\infty, x]$ and $L_2, R_2 \subseteq [x, +\infty)$, then $\text{mixValue}(L_1 \cup L_2, R_1 \cup R_2) \geq \text{mixValue}(L_1, R_1) + \text{mixValue}(L_2, R_2)$.*

*Finally, $\text{mixValue}(L, R) \leq 2 \cdot \min(|L|, |R|) + 1$.*

The definitions of Wilber's Alternation and Funnel bounds $(\mathsf{Alt}(X), \mathsf{Funnel}(X))$ are standard in the literature.

We now give precise definitions of Wilber's two bounds.[6]




| Node | Link used by each access | Group by letter | # |
|---|---|---|---|
| $u$ | R, L, L, R, R, L | [R], [L, L], [R, R], [L] | 3 |
| $v$ | L, R, R | [L], [R, R] | 1 |
| $w$ | L, R, L | [L], [R], [L] | 2 |
| $x$ | R, L | [R], [L] | 1 |
| Total | | | 7 |

**Figure 3** For access sequence $X = (4, 1, 3, 5, 4, 2)$ and reference tree $\mathcal{T}$, $\mathsf{Alt}_{\mathcal{T}}(X) = 7$.

▶ **Definition 13** (alternation bound). *Let $P$ be a point set with distinct $y$-coordinates, and let $\mathcal{T}$ be a binary search tree over the values $P.x$. We define $\mathsf{Alt}_{\mathcal{T}}(P)$ using the recursive structure of $\mathcal{T}$. If $\mathcal{T}$ is a single node, let $\mathsf{Alt}_{\mathcal{T}}(P) := 0$. Otherwise, let $\mathcal{T}_{\mathtt{L}}$ and $\mathcal{T}_{\mathtt{R}}$ be the left and right subtrees at the root. Partition $P$ into two sets $P_{\mathtt{L}} := \{p \in P \mid p.x \in \mathcal{T}_{\mathtt{L}}\}$ and $P_{\mathtt{R}} := \{p \in P \mid p.x \in \mathcal{T}_{\mathtt{R}}\}$ and consider the quantity* $\mathrm{mixValue}(P_{\mathtt{L}}.y, P_{\mathtt{R}}.y)$*, which describes how much $P_{\mathtt{L}}$ and $P_{\mathtt{R}}$ are interleaved with time (we call each switch between $P_{\mathtt{L}}$ and $P_{\mathtt{R}}$ a "preferred child alternation"). Then*

$$\mathsf{Alt}_{\mathcal{T}}(P) := \mathrm{mixValue}(P_{\mathtt{L}}.y, P_{\mathtt{R}}.y) + \mathsf{Alt}_{\mathcal{T}_{\mathtt{L}}}(P_{\mathtt{L}}) + \mathsf{Alt}_{\mathcal{T}_{\mathtt{R}}}(P_{\mathtt{R}}). \tag{4}$$

*The alternation bound is then defined as the maximum over all trees:*

$$\mathsf{Alt}(P) := \max_{\mathcal{T}} \mathsf{Alt}_{\mathcal{T}}(P).$$

*In addition, for an access sequence $X$, let $\mathsf{Alt}_{\mathcal{T}}(X) := \mathsf{Alt}_{\mathcal{T}}(G_X)$ and $\mathsf{Alt}(X) := \mathsf{Alt}(G_X)$.*

For the reason of space, we describe Wilber's Funnel bounds in the full version.

▶ **Definition 14** (amortized bounds). *For any sequence $X \in S^m$, define amortized versions of the optimal cost and the Wilber bounds:*

$$\overline{\mathsf{OPT}}(X) := \frac{\mathsf{OPT}(X)}{m}, \quad \overline{\mathsf{Alt}}(X) := \frac{\mathsf{Alt}(X)}{m}, \quad \overline{\mathsf{Funnel}}(X) := \frac{\mathsf{Funnel}(X)}{m}.$$

▶ **Definition 15** (composed sequence, see [9]). *Let $S_1, \dots, S_l$ be disjoint sets of keys with increasing values (i.e. $\forall x \in S_j, x' \in S_{j+1}$, we have $x < x'$). For each $j \in [l]$, let $X^{(j)} \in S_j^{m_j}$ be an access sequence with keys in $S_j$, and let $\widetilde{X}$ be a sequence with keys in $[l]$ such that every $j \in [l]$ appears exactly $m_j$ times (its total length is $m := m_1 + \cdots + m_l$). Then we define the composed sequence*

$$X = \widetilde{X}(X^{(1)}, \dots, X^{(l)}) \in (S_1 \cup \cdots \cup S_l)^m$$

*as the sequence that interleaves $X^{(1)}, \dots, X^{(l)}$ according to the order given by $\widetilde{X}$: that is, $X_t = X_{\sigma(t)}^{(\widetilde{X}_t)}$ where $\sigma(t) := \#\left\{t' \le t \,\middle|\, \widetilde{X}_{t'} = \widetilde{X}_t\right\}$.*

[6] These definitions may differ by a constant factor or an additive $\pm O(m)$ from the definitions the reader has seen before. We will ignore such differences, because the cost of a BST also varies by $\pm O(m)$ depending on the definition, and the interesting regime is when $\mathsf{OPT}(X) = \omega(m)$.

Note that [6] defines the *decomposition* operation, which is the inverse operation of the composition. We will use Definition 15 throughout this paper.

▶ **Definition 16** ($j_x$). *In the context of Definition 15, for any key* $x \in S_1 \cup \cdots \cup S_l$*, let* $j_x$ *be the unique index such that* $x \in S_{j_x}$.

## 3 Effect of Composition on Wilber's bounds

We prove Theorem 1 in this section. Namely, we show that Wilber's bounds act nicely under composition, allowing us to boost the separation between them in Section 4. We divide the proofs into the following two theorems.

▶ **Theorem 17** (subadditivity of Alt). *Let* $X := \widetilde{X}(X^{(1)}, \ldots, X^{(l)})$ *be a composed sequence with* $|X^{(1)}| = \cdots = |X^{(l)}|$.[7] *Then*

$$\mathsf{Alt}(X) \leq \mathsf{Alt}\left(\widetilde{X}\right) + \sum_j \mathsf{Alt}\left(X^{(j)}\right) + O(m).$$

▶ **Theorem 18** (superadditivity of Funnel). *Let* $X := \widetilde{X}(X^{(1)}, \ldots, X^{(l)})$ *be a composed sequence. Then*

$$\mathsf{Funnel}(X) \geq \mathsf{Funnel}\left(\widetilde{X}\right) + \sum_j \mathsf{Funnel}\left(X^{(j)}\right) - O(m).$$

For the reason of space, we postpone the proof of Theorem 18 to the full version.

### 3.1 Subadditivity of the alternation bound

We prove Theorem 17 in this section.

#### Proof plan

We will show that for any binary search tree $\mathcal{T}$ over $S_1 \cup \cdots \cup S_l$,

$$\mathsf{Alt}_{\mathcal{T}}(X) \leq \mathsf{Alt}\left(\widetilde{X}\right) + \sum_j \mathsf{Alt}\left(X^{(j)}\right) + O(m).$$

We will do this by

- decomposing $\mathcal{T}$ into the corresponding binary trees $\widetilde{\mathcal{T}}$ over $[l]$ and $\mathcal{T}_j$ over $S_j$ for all $j$;
- classifying preferred child alternations in $\mathcal{T}$ into 4 types, which correspond to either
  - alternations in $\widetilde{\mathcal{T}}$,
  - alternations in $\mathcal{T}_j$ for some $j$,
  - or to some other events that happen at most $O(m)$ times in aggregate.

That is, we will show that

$$\mathsf{Alt}_{\mathcal{T}}(X) \leq \mathsf{Alt}_{\widetilde{\mathcal{T}}}\left(\widetilde{X}\right) + \sum_j \mathsf{Alt}_{\mathcal{T}_j}\left(X^{(j)}\right) + O(m).$$

[7] We make this assumption so that the proof is simpler.

### 3.1.1 Decomposing the tree

For a tree $\mathcal{T}$, we write $x \prec_{\mathcal{T}} b$ if $x$ is a descendent of $b$ in $\mathcal{T}$, and we write $S \prec_{\mathcal{T}} b$ if $x \prec_{\mathcal{T}} b$ for all $x \in S$.

▶ **Definition 19.** *Let $\mathcal{T}_j$ be the unique binary search tree over $S_j$ such that if $b, x \in S_j$ and $x \prec_{\mathcal{T}} b$ then $x \prec_{\mathcal{T}_j} b$.*

$\mathcal{T}_j$ is constructed by running the following recursive algorithm, which builds a tree $\mathcal{T}_{\text{out}}$:

- Start at the root of $\mathcal{T}$, and let $x$ be the current node.
- If $x \in S_j$, then
  - make $x$ the root of $\mathcal{T}_{\text{out}}$;
  - form $x$'s left subtree in $\mathcal{T}_{\text{out}}$ by recursing on $x$'s left subtree in $\mathcal{T}$;
  - form $x$'s right subtree in $\mathcal{T}_{\text{out}}$ by recursing on $x$'s right subtree in $\mathcal{T}$.
- If $x \notin S_j$ then since $S_j$ is contiguous, at most one of $x$'s left and right subtrees in $\mathcal{T}$ can contain elements from $S_j$.
  - If there is one such subtree, form $\mathcal{T}_{\text{out}}$ by recursing on it.
  - Otherwise let $\mathcal{T}_{\text{out}}$ be the empty tree.

This algorithm clearly has the desired properties:

- Clearly, by construction, $\mathcal{T}_j$ is a binary search tree and its set of keys is $S_j$.
- If $b, x \in S_j$ and $x \prec_{\mathcal{T}} b$, then $x \prec_{\mathcal{T}_j} b$, because the only way to get to $x$ is to first pass through $b$, add it as a root of the current subtree, then recurse on $b$'s subtree that contains $x$, which eventually adds $x$ to $\mathcal{T}_j$ as a descendent of $b$.
- $\mathcal{T}_j$ is unique since the only nontrivial choice the algorithm makes is to add $x$ as a root, but this is necessary since it must be an ancestor of all of the keys that later get added to this part of $\mathcal{T}_j$.

▶ **Definition 20.** *Let $\widetilde{\mathcal{T}}$ be the unique binary search tree over $[l]$ such that if $x \prec_{\mathcal{T}} b$ and $S_{j_b}, S_{j_x} \prec_{\mathcal{T}} b$ then $j_x \prec_{\widetilde{\mathcal{T}}} j_b$.*

$\widetilde{\mathcal{T}}$ is constructed by running the following recursive algorithm, which builds a tree $\mathcal{T}_{\text{out}}$:

- Start at the root of $\mathcal{T}$, and let $x$ be the current node.
- If $j_x$ hasn't already been seen earlier in the algorithm (which happens iff $x$ is the lowest common ancestor of all of $S_{j_x}$ in $\mathcal{T}$), then
  - make $j_x$ the root of $\mathcal{T}_{\text{out}}$;
  - form $j_x$'s left subtree in $\mathcal{T}_{\text{out}}$ by recursing on $x$'s left subtree in $\mathcal{T}$;
  - form $j_x$'s right subtree in $\mathcal{T}_{\text{out}}$ by recursing on $x$'s right subtree in $\mathcal{T}$.
- If $j_x$ has already been seen earlier in the algorithm, then some ancestor of $x$ was also in $S_{j_x}$, and $S_{j_x}$ is contiguous, so $S_{j_x}$ must contain either $x$'s entire left subtree, or $x$'s entire right subtree. That means that at most one of $x$'s subtrees can contain elements *not* in $S_{j_x}$.
  - If there is one such subtree, form $\mathcal{T}_{\text{out}}$ by recursing on it.
  - Otherwise, let $\mathcal{T}_{\text{out}}$ be the empty tree.

This algorithm clearly has the desired properties:

- Clearly, by construction, $\widetilde{\mathcal{T}}$ is a binary search tree and its set of keys is $[l]$.
- If $x \prec_{\mathcal{T}} b$ and $S_{j_b}, S_{j_x} \prec_{\mathcal{T}} b$ then:
  - We can assume $j_x \neq j_b$ and thus $x \neq b$, otherwise the claim is trivially true.
  - Since $S_{j_b} \prec_{\mathcal{T}} b$, $b$ must be the lowest common ancestor of all of $S_{j_b}$ in $\mathcal{T}$, so $j_b$ gets added to $\widetilde{\mathcal{T}}$ when the algorithm is looking at node $b$.

  - Also, since $S_{j_x} \prec_{\mathcal{T}} b$ and $j_x \neq j_b$, that means that all the elements from $S_{j_x}$ are descendents of $b$, so $j_x$ will be added to $\widetilde{\mathcal{T}}$ in one of the recursive branches launched when looking at $b$.
  - Therefore $j_x$ will be a descendent of $j_b$ in $\widetilde{\mathcal{T}}$.
- $\widetilde{\mathcal{T}}$ is unique since the only nontrivial choice the algorithm makes is to add $j_x$ as root when it first sees an element from $S_{j_x}$, but this is necessary since for any $j$ which will eventually be added to this part of the tree, $S_j$ must have been completely contained in $x$'s subtree, and therefore $j_x$ must be an ancestor of $j$.

### 3.1.2 Stating the classification

Consider some left-to-right[8] preferred child alternation that $X$ produces in $\mathcal{T}$. That is, take some value $b \in S_1 \cup \cdots \cup S_l$ and some times $t_x < t_y \in [m]$ such that

- $x := X_{t_x}$ is in the left subtree of $b$ in $\mathcal{T}$,
- $y := X_{t_y}$ is in the right subtree of $b$ in $\mathcal{T}$,
- and none of the accesses $X_{t_x+1}, \ldots, X_{t_y-1}$ made in the interim were to values that are strict descendents of $b$.

Let $a$ be the lowest ancestor of $b$ such that $a < b$ and $c$ be the lowest ancestor of $b$ such that $b < c$.[9] This means that the left and right subtrees of $b$ correspond to the keys in intervals $(a, b)$ and $(b, c)$. We have $x, y \prec_{\mathcal{T}} b$, $x \in (a, b)$, and $y \in (b, c)$.

▷ Claim 21. One of the following must hold (from most "local" to most "global"):

1. All of $b, x, y$ are in the same range $S_{j_b}$, so $x$ and $y$ are in the left and right subtrees of $b$ in $\mathcal{T}_{j_b}$, and this corresponds to an alternation in $\mathcal{T}_{j_b}$.
2. $b$ is either the highest ancestor of $x$ such that $b \in S_{j_x}$ and $x < b$, or the highest ancestor of $y$ such that $b \in S_{j_y}$ and $b < y$.
3. $S_{j_b} \prec_{\mathcal{T}} b$, and either $j_x$ is the closest (in key value) ancestor of $j_b$ in $\widetilde{\mathcal{T}}$ such that $j_x < j_b$, or $j_y$ is the closest (in key value) ancestor of $j_b$ in $\widetilde{\mathcal{T}}$ such that $j_b < j_y$.
4. All of $b, x, y$ are in different ranges (i.e. $j_x < j_b < j_y$), $j_x$ is in $j_b$'s left subtree in $\widetilde{\mathcal{T}}$, and $j_y$ is in $j_b$'s right subtree in $\widetilde{\mathcal{T}}$, so this corresponds to an alternation in $\widetilde{\mathcal{T}}$.

### 3.1.3 Proving the classification

Proof of Claim 21. Let us prove that every alternation is of one of these four types.

- First, suppose that $j_x = j_b = j_y$. Then by construction of $\mathcal{T}_{j_b}$, $x$ and $y$ are still descendents of $b$ in $\mathcal{T}_{j_b}$, and since $\mathcal{T}_{j_b}$ is a binary search tree, $x$ must be in $b$'s left subtree and $y$ must be in $b$'s right subtree. So this is type 1.
- Now suppose that exactly one of $j_x = j_b$ and $j_b = j_y$ holds. By symmetry, suppose that it is the former, and thus $j_x = j_b < j_y$. Then we trivially have $b \in S_{j_x}$. And on the other hand, consider any ancestor $b'$ of $x$ which is higher than $b$ and satisfies $x < b'$. Then $b'$ would have to satisfy $y < b'$ as well, and in particular $j_y \leq j_{b'}$, so it could not lie in $S_{j_x}$. Therefore $b$ is the highest ancestor of $x$ which lies in $S_{j_x}$ and satisfies $x < b$. So this is type 2.

[8] The case where the alternation occurs from right to left is analogous.
[9] If either $a$ or $c$ doesn't exist, let $a := -\infty$ or $c := +\infty$ by convention.

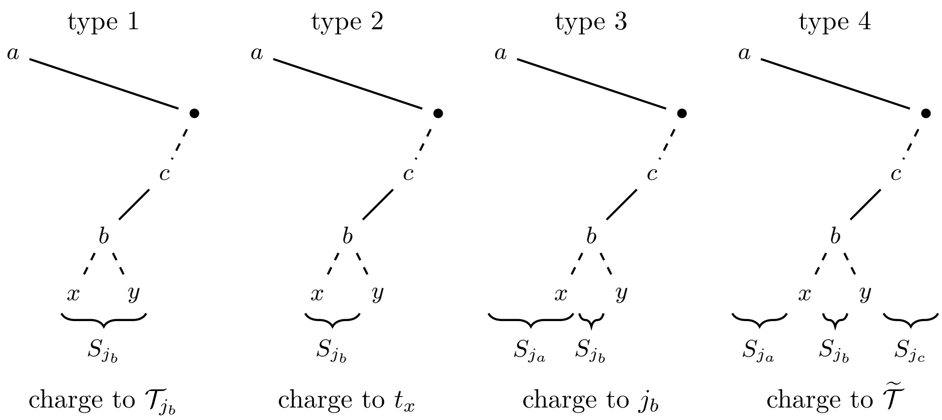



**Figure 4** A preferred child alternation in $\mathcal{T}$: $b$'s preferred child changes from the left side (due to an access to $x$) to the right side (due to an access to $y$). There are four qualitatively different ways in which the alternation can happen depending on which ranges $S_1, \ldots, S_l$ the keys $a, b, c, x, y$ belong.

- From now on we can assume that $j_x < j_b < j_y$, which means in particular that $j_a < j_b < j_c$, that $S_{j_b}$ is contained entirely in $b$'s subtree in $\mathcal{T}$, and therefore $b$ is the highest member of $S_{j_b}$ in $\mathcal{T}$. Now, the lowest common ancestor of $S_{j_a}$ (resp. $S_{j_c}$) must be an ancestor of $a$ (resp. $c$) and therefore an ancestor of $b$, so by the properties of $\widetilde{\mathcal{T}}$, $j_a$ (resp. $j_c$) is an ancestor of $j_b$ in $\widetilde{\mathcal{T}}$. Furthermore, any ancestor of $j_b$ in $\widetilde{\mathcal{T}}$ must be of the form $j_z$ for some ancestor $z$ of $b$ in $\mathcal{T}$, so since $a$ (resp. $c$) is the closest (in key value) ancestors of $b$ on its left (resp. right) side in $\mathcal{T}$, $j_a$ (resp. $j_c$) must be the closest ancestor of $j_b$ on its left (resp. right) side in $\widetilde{\mathcal{T}}$.
  - Suppose that at least one of $j_a = j_x$ or $j_y = j_c$ holds. By symmetry suppose that it is the former. Then just by virtue of the fact that $j_x = j_a$, $j_x$ is the closest ancestor of $j_b$ on its left side in $\widetilde{\mathcal{T}}$. So this is type 3.
  - Otherwise, we have $j_a < j_x < j_b < j_y < j_c$. This implies that $S_{j_x}$ lies entirely within $b$'s left subtree, and $S_{j_y}$ lies entirely within $b$'s right subtree, thus $j_x$ and $j_y$ are descendents of $j_b$ in $\widetilde{\mathcal{T}}$. So this is type 4. ◀

### 3.1.4 Using the classification to prove Theorem 17

**Proof of Theorem 17.** Let $\mathcal{T}$ be any binary search tree over $S_1 \cup \cdots \cup S_l$, and let $\mathcal{T}_j$ and $\widetilde{\mathcal{T}}$ be the corresponding trees defined in Definition 19 and Definition 20. We will show that

$$\mathsf{Alt}_{\mathcal{T}}(X) \leq \mathsf{Alt}_{\widetilde{\mathcal{T}}}\left(\widetilde{X}\right) + \sum_j \mathsf{Alt}_{\mathcal{T}_j}\left(X^{(j)}\right) + O(m).$$

Let us use Claim 21: we charge type 1 alternations to $\mathsf{Alt}_{\mathcal{T}_j}\left(X^{(j)}\right)$, type 4 alternations to $\mathsf{Alt}_{\widetilde{\mathcal{T}}}\left(\widetilde{X}\right)$, and we show below that there are only $O(m)$ alternations of types 2 and 3.

For type 2, this is because we can charge it uniquely to the access made to $x$ or $y$ (formally, we charge it to $t_x$ or $t_y$).

- Let us take the first subcase: $b$ is the highest ancestor of $x$ such that $b \in S_{j_x}$ and $x < b$. $x$ can only have one highest ancestor with a given property, so it has only one highest "ancestor $b$ such that $b \in S_{j_x}$ and $x < b$". So this can apply to at most one of the alternations that occurred when accessing $x$, and thus we can charge it to $t_x$.

- Let us take the second subcase: $b$ is the highest ancestor of $y$ such that $b \in S_{j_y}$ and $b < y$. Again, $y$ can only have one highest "ancestor $b$ such that $b \in S_{j_y}$ and $b < y$". The access to $x$ is the first time that the preferred child switches back from $b$'s left child to $b$'s right child after accessing $y$. So this event is unique from the perspective of this particular access to $y$, and thus we can charge it to $t_y$.

Finally, we will bound the total number of occurrences of type 3 alternations by charging them to $j_b$, not uniquely but in a $\frac{l}{m}$-to-1 manner. Let us take the case where $j_x$ is the closest ancestor of $j_b$ in $\widetilde{\mathcal{T}}$ such that $j_x < j_b$ (the other case is analogous). Clearly, $j_b$ determines $j_x$ uniquely. And since $b$'s subtree contains $S_{j_b}$ in its entirety, $j_b$ determines $b$ uniquely too. So once you know $j_b$, the only uncertainty left about this alternation is *which* access within $X^{(j_x)}$ caused it. So the total number of alternations of this type is bounded by

$$\underbrace{l}_{\text{which } j_b?} \qquad \underbrace{\max_{j_x}\left|X^{(j_x)}\right|}_{\text{which access within } X^{(j_x)}?} \qquad = l\frac{m}{l} = m,$$

where the first equality uses our assumption that $|X^{(1)}| = \cdots = |X^{(l)}|$.

Overall, we have shown that

$$\mathsf{Alt}_{\mathcal{T}}(X) \leq \underbrace{\mathsf{Alt}_{\widetilde{\mathcal{T}}}\left(\widetilde{X}\right)}_{\text{type 4}} + \sum_j \underbrace{\mathsf{Alt}_{\mathcal{T}_j}\left(X^{(j)}\right)}_{\text{type 1}} + \underbrace{O(m)}_{\text{type 2 (charge to } t_x \text{ or } t_y)} + \underbrace{O(m)}_{\text{type 3 (charge to } j_b)},$$

so we can now take the maximum over $\mathcal{T}$ to conclude

$$\begin{aligned}
\mathsf{Alt}(X) &:= \max_{\mathcal{T}} \mathsf{Alt}_{\mathcal{T}}(X) \\
&\leq \max_{\mathcal{T}} \left( \mathsf{Alt}_{\widetilde{\mathcal{T}}}\left(\widetilde{X}\right) + \sum_j \mathsf{Alt}_{\mathcal{T}_j}\left(X^{(j)}\right) + O(m) \right) \\
&\leq \max_{\widetilde{\mathcal{T}}} \mathsf{Alt}_{\widetilde{\mathcal{T}}}\left(\widetilde{X}\right) + \sum_j \max_{\mathcal{T}_j} \mathsf{Alt}_{\mathcal{T}_j}\left(X^{(j)}\right) + O(m) \\
&= \mathsf{Alt}\left(\widetilde{X}\right) + \sum_j \mathsf{Alt}\left(X^{(j)}\right) + O(m).
\end{aligned}$$

◀

## 4 Boosting the separation between Wilber's bounds

We prove Theorem 2 in this section. We now use the composition properties of Alt and Funnel we proved in Section 3 to show that Tango tree makes an optimal trade-off between fixed costs and variable costs that depend on the alternation bound.

### 4.1 What boosting can we get?

Lecomte and Weinstein [27] show an $\Omega(\log\log n)$ separation between Alt and Funnel.

▶ **Theorem 22** (Theorem 2 in [27]). *For any $n$ of the form $2^{2^k}$, there is a sequence $Y_n \in [n]^m$ where $m \leq \text{poly}(n)$, each element appears $O(m/n)$ times, and*

$$\begin{aligned}
\overline{\mathsf{Alt}}(Y_n) &\leq O(1) \\
\overline{\mathsf{Funnel}}(Y_n) &\geq \Omega(\log\log n).
\end{aligned}$$

We can use the tight composition results from Section 3 to show the following boosted separation. We emphasize that the approximate subadditivity of the Alternation bound [6] is insufficient to boost the separation.

▶ **Theorem 23** (Hardness Amplification, Restatement of Theorem 2). *There is a constant $K > 0$ such that for any $n$ of the form $2^{2^k}$ and any power-of-two $R \leq \frac{\log n}{K}$, there is a sequence $Y_n^{\circ R} \in [n]^{m'}$ with $m' \leq \mathrm{poly}(n)$ such that*

$$\overline{\mathsf{Alt}}(Y_n^{\circ R}) \leq O(R)$$
$$\overline{\mathsf{Funnel}}(Y_n^{\circ R}) \geq \Omega\left(R \log\left(\frac{\log n}{R}\right)\right).$$

**Proof.** Let $Y_n$ be the sequence stated in Theorem 22. First, pad $Y_n$ so that each key appears *exactly* $m/n$ times, by adding each key one by one the appropriate number of times, in ascending order. It is easy to see that this maintains the bounds

$$\overline{\mathsf{Alt}}(Y_n) \leq O(1)$$
$$\overline{\mathsf{Funnel}}(Y_n) \geq \Omega(\log \log n).$$

Now, let $C_O, C_\Omega > 0$ be constants such that

$$\overline{\mathsf{Alt}}(Y_n) \leq C_O$$
$$\overline{\mathsf{Funnel}}(Y_n) \geq C_\Omega \log \log n$$

(we will allow ourselves to make $C_O$ even larger later on).

Let $Y_n^{\circ 1} := Y_n$, and for all power-of-two $R \geq 1$, let

$$Y_n^{\circ 2R} := \left(Y_{\sqrt{n}}^{\circ R}\right)^{\otimes \sqrt{n}} \left(Y_{\sqrt{n}}^{\circ R}, \ldots, Y_{\sqrt{n}}^{\circ R}\right),$$

where "$X^{\otimes \sqrt{n}}$" means "$X$ repeated $\sqrt{n}$ times", and with an abuse of notation we assume that the $\sqrt{n}$ sequences $Y_{\sqrt{n}}^{\circ R}, \ldots, Y_{\sqrt{n}}^{\circ R}$ that are being composed each contains a distinct range of keys. We can check that

$$\left|Y_n^{\circ R}\right| = n^{1-1/R} |Y_{n^{1/R}}| \leq n^{1-1/R} \, \mathrm{poly}\left(n^{1/R}\right) \leq \mathrm{poly}(n)$$

as desired. We will show by induction that

$$\overline{\mathsf{Alt}}(Y_n^{\circ R}) \leq C_O(2R-1)$$
$$\overline{\mathsf{Funnel}}(Y_n^{\circ R}) \geq C_\Omega \frac{R+1}{2} \log\left(\frac{\log n}{R}\right).$$

**Base case: $R = 1$.** We verify that indeed

$$\begin{aligned}\overline{\mathsf{Alt}}(Y_n) &\leq C_O \\ &= C_O(2 \cdot 1 - 1) \\ &= C_O(2R - 1)\end{aligned}$$

and

$$\begin{aligned}\overline{\mathsf{Funnel}}(Y_n) &\geq C_\Omega \log \log n \\ &= C_\Omega \frac{1+1}{2} \log\left(\frac{\log n}{1}\right) \\ &= C_\Omega \frac{R+1}{2} \log\left(\frac{\log n}{R}\right).\end{aligned}$$

**Inductive case: $R \to 2R$.** Suppose this is true for some $R \geq 1$, for all $n$. Then for Alt, by Theorem 17 we have

$$\begin{aligned}
\overline{\mathsf{Alt}}(Y_n^{\circ 2R}) &\leq \overline{\mathsf{Alt}}\left(Y_{\sqrt{n}}^{\circ R}\right) + \frac{\sqrt{n}\cdot \overline{\mathsf{Alt}}\left(Y_{\sqrt{n}}^{\circ R}\right)}{\sqrt{n}} + O(1) \\
&\leq 2C_O(2R-1) + O(1) \\
&= C_O(4R-1) - C_O + O(1) \\
&\leq C_O(4R-1),
\end{aligned}$$

where the last step holds as long as $C_O$ is large enough.

For Funnel, by Theorem 18 we have

$$\begin{aligned}
\overline{\mathsf{Funnel}}\left(Y_{\sqrt{n}}^{\circ R}\right) &\geq \overline{\mathsf{Funnel}}\left(Y_{\sqrt{n}}^{\circ R}\right) + \frac{\sqrt{n}\cdot \overline{\mathsf{Funnel}}\left(Y_{\sqrt{n}}^{\circ R}\right)}{\sqrt{n}} - O(1) \\
&\geq C_\Omega(R+1)\log\left(\frac{\log\sqrt{n}}{R}\right) - O(1) \\
&= C_\Omega \frac{2R+1}{2}\log\left(\frac{\log n}{2R}\right) + \frac{C_\Omega}{2}\log\left(\frac{\log n}{2R}\right) - O(1) \\
&\geq C_\Omega \frac{2R+1}{2}\log\left(\frac{\log n}{2R}\right) + \frac{C_\Omega}{2}\log\left(\frac{K}{2}\right) - O(1) \\
&\geq C_\Omega \frac{2R+1}{2}\log\left(\frac{\log n}{2R}\right),
\end{aligned}$$

where the penultimate step holds because $R \leq \frac{\log n}{K}$, and the last step holds as long as $K$ is large enough. ◀

## 5 Optimality of Tango Trees

We are ready to prove Theorem 3.

▶ **Theorem 24** (Restatement of Theorem 3). *Let $\alpha, \beta : \mathbb{N} \to \mathbb{R}_{\geq 1}$ be any functions and let $A$ be some BST algorithm. Denote the amortized cost of $A$ on an access sequence $X$ as $\overline{A}(X)$. Suppose that for all access sequences $X$ over $n$ keys, we have*

$$\overline{A}(X) \leq \alpha(n)\overline{\mathsf{Alt}}(X) + \beta(n).$$

*Then $\alpha(n) \geq \Omega\left(\log\left(\frac{\log n}{\beta(n)}\right)\right)$ for infinitely many values of $n$.*

**Proof.** In fact we will show that the result holds under the weaker assumption that the theorem holds for the *optimal* amortized cost:

$$\overline{\mathsf{OPT}}(X) \leq \alpha(n)\overline{\mathsf{Alt}}(X) + \beta(n).$$

The above inequality must in particular hold for the access sequence $Y_n^{\circ R}$ from Theorem 2, where we let $R$ be the largest power of two such that $R \leq \beta(n)$. This gives us

$$\begin{cases}
\overline{\mathsf{OPT}}(Y_n^{\circ R}) \leq \alpha(n)\overline{\mathsf{Alt}}(X) + \beta(n) \leq O(R(\alpha(n)+1)) \\
\overline{\mathsf{OPT}}(Y_n^{\circ R}) \geq \Omega(\overline{\mathsf{Funnel}}(Y_n^{\circ R})) \geq \Omega\left(R\log\left(\frac{\log n}{R}\right)\right).
\end{cases}$$

Combining these inequalities, we obtain

$$R(\alpha(n)+1) \geq \Omega\left(R\log\left(\frac{\log n}{R}\right)\right) \implies \alpha(n) \geq \Omega\left(\log\left(\frac{\log n}{R}\right)\right) \geq \Omega\left(\log\left(\frac{\log n}{\beta(n)}\right)\right)$$

(where the implication uses the fact that $\alpha(n) \geq 1$). ◀

## References


1. Brian Allen and Ian Munro. Self-organizing binary search trees. *J. ACM*, 25(4):526–535, October 1978. doi:10.1145/322092.322094.
2. Boaz Barak, Mark Braverman, Xi Chen, and Anup Rao. Direct sums in randomized communication complexity. *Electron. Colloquium Comput. Complex.*, TR09-044, 2009. URL: https://eccc.weizmann.ac.il/report/2009/044, arXiv:TR09-044.
3. Benjamin Aram Berendsohn, László Kozma, and Michal Opler. Optimization with pattern-avoiding input. *CoRR*, abs/2310.04236, 2023. doi:10.48550/arXiv.2310.04236.
4. Guy E. Blelloch and Magdalen Dobson. The geometry of tree-based sorting. In *ICALP*, volume 261 of *LIPIcs*, pages 26:1–26:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ICALP.2023.26.
5. Prosenjit Bose, Karim Douïeb, Vida Dujmovic, and Rolf Fagerberg. An $O(\log\log n)$-competitive binary search tree with optimal worst-case access times. In *Algorithm Theory - SWAT 2010, 12th Scandinavian Symposium and Workshops on Algorithm Theory, Bergen, Norway, June 21-23, 2010. Proceedings*, pages 38–49, 2010. doi:10.1007/978-3-642-13731-0_5.
6. Parinya Chalermsook, Julia Chuzhoy, and Thatchaphol Saranurak. Pinning down the strong wilber 1 bound for binary search trees. In *APPROX-RANDOM*, volume 176 of *LIPIcs*, pages 33:1–33:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.APPROX/RANDOM.2020.33.
7. Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Greedy is an almost optimal deque. In *WADS*, volume 9214 of *Lecture Notes in Computer Science*, pages 152–165. Springer, 2015. doi:10.1007/978-3-319-21840-3_13.
8. Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Pattern-avoiding access in binary search trees. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 410–423. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.32.
9. Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. The landscape of bounds for binary search trees. *arXiv preprint*, 2016. arXiv:1603.04892.
10. 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 *SODA*, pages 509–534. SIAM, 2023. doi:10.1137/1.9781611977554.CH22.
11. Parinya Chalermsook, Manoj Gupta, Wanchote Jiamjitrak, Akash Pareek, and Sorrachai Yingchareonthawornchai. The group access bounds for binary search trees. *CoRR*, abs/2312.15426, 2023. doi:10.48550/arXiv.2312.15426.
12. Parinya Chalermsook and Wanchote Po Jiamjitrak. New binary search tree bounds via geometric inversions. In *ESA*, volume 173 of *LIPIcs*, pages 28:1–28:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ESA.2020.28.
13. Parinya Chalermsook, Seth Pettie, and Sorrachai Yingchareonthawornchai. Sorting pattern-avoiding permutations via 0-1 matrices forbidding product patterns. In *SODA*, pages 133–149. SIAM, 2024. doi:10.1137/1.9781611977912.7.
14. Richard Cole. On the dynamic finger conjecture for splay trees. part II: the proof. *SIAM J. Comput.*, 30(1):44–85, 2000. doi:10.1137/S009753979732699X.

15 Erik D. Demaine, Dion Harmon, John Iacono, Daniel M. Kane, and Mihai Patrascu. The geometry of binary search trees. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 496–505, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496825, doi:10.1137/1.9781611973068.55.

16 Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Patrascu. Dynamic optimality - almost. *SIAM J. Comput.*, 37(1):240–251, 2007. doi:10.1137/S0097539705447347.

17 Jonathan Derryberry, Daniel Dominic Sleator, and Chengwen Chris Wang. A lower bound framework for binary search trees with rotations. *Technical report*, 2005.

18 Amr Elmasry. On the sequential access theorem and deque conjecture for splay trees. *Theor. Comput. Sci.*, 314(3):459–466, 2004. doi:10.1016/J.TCS.2004.01.019.

19 Kyle Fox. Upper bounds for maximally greedy binary search trees. In *WADS*, volume 6844 of *Lecture Notes in Computer Science*, pages 411–422. Springer, 2011. doi:10.1007/978-3-642-22300-6_35.

20 Oded Goldreich, Noam Nisan, and Avi Wigderson. On yao's xor-lemma. In Oded Goldreich, editor, *Studies in Complexity and Cryptography. Miscellanea on the Interplay between Randomness and Computation - In Collaboration with Lidor Avigad, Mihir Bellare, Zvika Brakerski, Shafi Goldwasser, Shai Halevi, Tali Kaufman, Leonid Levin, Noam Nisan, Dana Ron, Madhu Sudan, Luca Trevisan, Salil Vadhan, Avi Wigderson, David Zuckerman*, volume 6650 of *Lecture Notes in Computer Science*, pages 273–301. Springer, 2011. doi:10.1007/978-3-642-22670-0_23.

21 Navin Goyal and Manoj Gupta. Better analysis of greedy binary search tree on decomposable sequences. *Theor. Comput. Sci.*, 776:19–42, 2019. doi:10.1016/J.TCS.2018.12.021.

22 John Iacono. In pursuit of the dynamic optimality conjecture. In *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, pages 236–250, 2013. doi:10.1007/978-3-642-40273-9_16.

23 Mauricio Karchmer, Ran Raz, and Avi Wigderson. Super-logarithmic depth lower bounds via the direct sum in communication complexity. *Comput. Complex.*, 5(3/4):191–204, 1995. doi:10.1007/BF01206317.

24 Alexander Knop, Shachar Lovett, Sam McGuire, and Weiqiang Yuan. Log-rank and lifting for and-functions. In Samir Khuller and Virginia Vassilevska Williams, editors, *STOC '21: 53rd Annual ACM SIGACT Symposium on Theory of Computing, Virtual Event, Italy, June 21-25, 2021*, pages 197–208. ACM, 2021. doi:10.1145/3406325.3450999.

25 László Kozma. Binary search trees, rectangles and patterns, 2016.

26 László Kozma and Thatchaphol Saranurak. Smooth heaps and a dual view of self-adjusting data structures. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 801–814, 2018. doi:10.1145/3188745.3188864.

27 Victor Lecomte and Omri Weinstein. Settling the relationship between wilber's bounds for dynamic optimality. In *ESA*, volume 173 of *LIPIcs*, pages 68:1–68:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPICS.ESA.2020.68.

28 Troy Lee, Adi Shraibman, and Robert Spalek. A direct product theorem for discrepancy. In *CCC*, pages 71–80. IEEE Computer Society, 2008. doi:10.1109/CCC.2008.25.

29 Caleb C. Levy and Robert E. Tarjan. A new path from splay to dynamic optimality. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 1311–1330. SIAM, 2019. doi:10.1137/1.9781611975482.80.

30 Joan Marie Lucas. *Canonical forms for competitive binary search tree algorithms*. Rutgers University, Department of Computer Science, Laboratory for Computer Science Research, 1988.

31 J. Ian Munro. On the competitiveness of linear search. In Mike Paterson, editor, *Algorithms - ESA 2000, 8th Annual European Symposium, Saarbrücken, Germany, September 5-8, 2000, Proceedings*, volume 1879 of *Lecture Notes in Computer Science*, pages 338–345. Springer, 2000. doi:10.1007/3-540-45253-2_31.

32 Seth Pettie. Splay trees, davenport-schinzel sequences, and the deque conjecture. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1115–1124, 2008. URL: http://dl.acm.org/citation.cfm?id=1347082.1347204.

33 Mihai Puatracscu and Mikkel Thorup. Higher lower bounds for near-neighbor and further rich problems. *SIAM J. Comput.*, 39(2):730–741, 2009. doi:10.1137/070684859.

34 Ran Raz. A counterexample to strong parallel repetition. *SIAM J. Comput.*, 40(3):771–777, 2011. doi:10.1137/090747270.

35 Ran Raz and Pierre McKenzie. Separation of the monotone NC hierarchy. *Comb.*, 19(3):403–435, 1999. doi:10.1007/S004930050062.

36 Yaniv Sadeh and Haim Kaplan. Dynamic binary search trees: Improved lower bounds for the greedy-future algorithm. In *STACS*, volume 254 of *LIPIcs*, pages 53:1–53:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.STACS.2023.53.

37 Ronen Shaltiel. Towards proving strong direct product theorems. In *Proceedings of the 16th Annual Conference on Computational Complexity*, CCC '01, page 107, USA, 2001. IEEE Computer Society.

38 Alexander A. Sherstov. Strong direct product theorems for quantum communication and query complexity. *SIAM J. Comput.*, 41(5):1122–1165, 2012. doi:10.1137/110842661.

39 Yaroslav Shitov. A counterexample to comon's conjecture. *SIAM J. Appl. Algebra Geom.*, 2(3):428–443, 2018. doi:10.1137/17M1131970.

40 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, July 1985. doi:10.1145/3828.3835.

41 Rajamani Sundar. On the deque conjecture for the splay algorithm. *Comb.*, 12(1):95–124, 1992. doi:10.1007/BF01191208.

42 Chengwen Chris Wang, Jonathan Derryberry, and Daniel Dominic Sleator. O(log log n)-competitive dynamic binary search trees. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 374–383, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics. URL: http://dl.acm.org/citation.cfm?id=1109557.1109600.

43 R. Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM J. Comput.*, 18(1):56–67, February 1989. doi:10.1137/0218004.