# Multi-Finger Binary Search Trees

**Parinya Chalermsook**[1]
Aalto University, Finland
parinya.chalermsook@aalto.fi

**Mayank Goswami**
Queens College, City University of New York, USA
mayank.goswami@qc.cuny.edu

**László Kozma**[2]
TU Eindhoven, The Netherlands
lkozma@gmail.com

**Kurt Mehlhorn**
MPI für Informatik, Saarbrücken, Germany
mehlhorn@mpi-inf.mpg.de

**Thatchaphol Saranurak**[3]
KTH Royal Institute of Technology, Sweden
thasar@kth.se

## Abstract

We study multi-finger binary search trees (BSTs), a far-reaching extension of the classical BST model, with connections to the well-studied $k$-server problem. Finger search is a popular technique for speeding up BST operations when a query sequence has *locality of reference*. BSTs with *multiple* fingers can exploit more general regularities in the input. In this paper we consider the cost of serving a sequence of queries in an optimal (offline) BST with $k$ fingers, a powerful benchmark against which other algorithms can be measured.

We show that the $k$-finger optimum can be matched by a standard dynamic BST (having a single root-finger) with an $O(\log k)$ factor overhead. This result is tight for all $k$, improving the $O(k)$ factor implicit in earlier work. Furthermore, we describe new *online* BSTs that match this bound up to a $(\log k)^{O(1)}$ factor. Previously only the "one-finger" special case was known to hold for an online BST (Iacono, Langerman, 2016; Cole et al., 2000). Splay trees, assuming their conjectured optimality (Sleator and Tarjan, 1983), would have to match our bounds for all $k$.

Our online algorithms are randomized and combine techniques developed for the $k$-server problem with a multiplicative-weights scheme for learning tree metrics. To our knowledge, this is the first time when tools developed for the $k$-server problem are used in BSTs. As an application of our $k$-finger results, we show that BSTs can efficiently serve queries that are *close to some recently accessed item*. This is a (restricted) form of the *unified property* (Iacono, 2001) that was previously not known to hold for any BST algorithm, online or offline.

**2012 ACM Subject Classification** Theory of computation → Design and analysis of algorithms, Theory of computation → Data structures design and analysis

**Keywords and phrases** binary search trees, dynamic optimality, finger search, k-server

[1] Parinya Chalermsook is supported by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 759557) and by Academy of Finland Research Fellows, under grant No. 310415.
[2] László Kozma is supperted through ERC consolidator grant No. 617951.
[3] Thatchaphol Saranurak is supported by European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 715672, and by the Swedish Research Council (Reg. No. 2015-04659).



29th International Symposium on Algorithms and Computation (ISAAC 2018).
Editors: Wen-Lian Hsu, Der-Tsai Lee, and Chung-Shou Liao; Article No. 55; pp. 55:1–55:26

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

**Digital Object Identifier** 10.4230/LIPIcs.ISAAC.2018.55

**Acknowledgements** We thank Nikhil Bansal and Greg Koumoutsos for insightful discussions.

## 1 Introduction

The binary search tree (BST) is the canonical comparison-based implementation of the dictionary data type for maintaining ordered sets. Dynamic BSTs can be re-arranged after every access via rotations and pointer moves starting from the root. Various ingenious techniques have been developed for dynamically maintaining balanced BSTs, supporting search, insert, delete, and other operations in time $O(\log n)$, where $n$ is the size of the dictionary (see e.g. [31, § 6.2.2], [40, § 5]).

In several applications where the access sequence has strong *locality of reference*, the worst-case bound is too pessimistic (e.g. in list merging, adaptive sorting, or in various geometric problems). A classical technique for exploiting locality is *finger search*. In finger search trees, the cost of an access is typically $O(\log d)$,[4] where $d$ is the difference in rank between the accessed item and a *finger* ($d$ may be much smaller than $n$). The finger indicates the starting point of the search, and is either given by the user, or (more typically) it points to the previously accessed item. Several special purpose tree-like data structures have been designed to support finger search.[5]

In 1983, Sleator and Tarjan [49] introduced Splay trees, a particularly simple and elegant "self-adjusting" BST algorithm. In 2000, Cole et al. [16, 15] showed that Splay matches (asymptotically) the efficiency of finger search, called in this context the *dynamic finger* property. This is remarkable, since Splay uses no explicit fingers; every search starts from the root. The result shows the versatility of the BST model, and has been seen as a major (and highly nontrivial) step towards "dynamic optimality", the conjecture of Sleator and Tarjan that Splay trees are constant-competitive.

BSTs can also adapt to other kinds of locality. The *working set* property [49] requires the amortized cost of accessing $x$ to be $O(\log t)$, where $t$ is the number of distinct items accessed since the last access of $x$. Whereas dynamic finger captures proximity in keyspace, the working set property captures proximity *in time*. In 2001, Iacono [26] proposed a *unified* property that generalizes both kinds of proximity. Informally, a data structure with the unified property is efficient when accessing an item that is *close to some recently accessed item*. It is not known whether any BST data structure has the unified property.

Recently, Iacono and Langerman [28] studied the *lazy finger* property (Bose et al. [8]), and showed that an online algorithm called Greedy BST[6] satisfies it. The lazy finger property requires the amortized cost of accessing $x$ to be $O(d)$, where $d$ is the distance (number of edges) from the previously accessed item to $x$ in the best *static reference tree*. This property is stronger than the dynamic finger property [8], and it is not known to hold for Splay.

In this paper we study a generalization of the lazy finger property; instead of a single finger stationed at the previously accessed item, we allow $k$ fingers to be moved around arbitrarily. An access is performed by moving any of the fingers to the requested item. Cost

[4] To simplify notation, we let $\log(x)$ denote $\log_2(\max\{2, x\})$.

[5] The initial 1977 design of Guibas et al. [23] was refined and simplified by Brown and Tarjan [10] and by Huddleston and Mehlhorn [25]. Further solutions include [51, 50, 32, 30], see also the survey [9]. Randomized treaps [46] and skip lists [43] can also support finger search.

[6] Greedy BST was discovered by Lucas in 1988 [37] and later independently by Munro [42]. Demaine et al. [17] transformed it into an online algorithm.

is proportional to the *total* distance traveled by the fingers. We assume that the fingers move according to an optimal strategy, in an optimally chosen static tree, with a priori knowledge of the entire access sequence. The cost of this optimal *offline* execution with $k$ fingers is an intrinsic measure of complexity of a query sequence, and at the same time a benchmark that algorithms in the classical model can attempt to match. Parameter $k$ describes the strength of the bound: the case $k = 1$ is the lazy finger, at the other extreme, at $k = n$, each item may have its own finger, and all accesses are essentially free.

Our main result is a family of new *online*[7] dynamic BST algorithms (in the standard model, where every access starts at the root), matching the $k$-finger optimum on sufficiently long sequences, up to an overhead factor with moderate dependence on $k$ and no dependence on the dictionary size or on the number of accesses in the sequence.

Our online BST combines three distinct techniques: (1) an offline, one-finger BST simulation of a multi-finger execution (the technique is a refinement of an earlier construction [18]), (2) online $k$-server algorithms that can simulate the offline optimal multi-finger strategy, and (3) a multiplicative-weights scheme for learning a tree metric in an online fashion.

The fact that "vanilla" BSTs can, with a low overhead, simulate a much more powerful computational model further indicates the strength and versatility of the BST model. As an application, we show that our online BST algorithms satisfy a restricted form of the *unified property*; previously no (online or offline) BST was known to satisfy such a property.

If there is a constant-competitive BST algorithm, then it must match our $k$-finger bounds. The two most promising candidates, Splay and Greedy BST (see e.g. [27]) were only shown (with considerable difficulty) to satisfy variants of the one-finger, i.e. lazy finger property. To obtain our online BSTs competitive for other values of $k$, we combine sophisticated tools developed for other online problems, as well as our refinement of a previous (highly nontrivial) construction for simulating multiple fingers. These facts together may hint at the formidable difficulty (more pessimistically: the low likelihood) of attaining dynamic optimality by simple and natural BST algorithms such as Splay or Greedy.

**BST and finger models. Main results.** Now, we introduce the formal statements of our results. In the dynamic BST model a sequence of keys is accessed in a binary search tree (BST), and after each access, the tree can be reconfigured via a sequence of rotations and pointer moves starting from the root. (There exist several alternative but essentially equivalent models, see [52, 17].) Denote the space of keys (or elements) by $[n]$. For a sequence $X = (x_1, \ldots, x_m) \in [n]^m$, denote by $\mathsf{OPT}(X)$ the cost of the optimal offline BST for accessing $X$.[8] Arguably the most important question in the BST model is the dynamic optimality conjecture, i.e. the existence of an online BST whose cost is $O(\mathsf{OPT}(X))$ for every $X$.

A BST *optimality property* is an inequality between $\mathsf{OPT}(X)$ and some function $f(X)$, that holds in the BST model. (If $\mathsf{OPT}(X) \leq f(X)$ for all $X$ is a BST optimality property, then every $O(1)$-competitive algorithm must cost at most $O(f(X))$.)

Several natural BST properties have been suggested over the last few decades. For instance, the *static finger* property [49] states $\mathsf{OPT}(X) = O(\mathsf{SF}(X))$, for $\mathsf{SF}(X) = \sum_t \log |x_t - j|$, where $j \in [n]$ is a fixed element (finger). The *static optimality* property [49] is $\mathsf{OPT}(X) = O(\mathsf{SO}(X))$, where $\mathsf{SO}(X) = \min_R \sum_i d_R(x_i)$. Here $R$ is a *static* BST, and $d_R(x)$ is the depth of $x$ in $R$.

[7] An *online* BST algorithm can base its decisions only on the current and past accesses. An *offline* algorithm knowns the entire access sequence in advance.

[8] To avoid technicalities, we only consider *access* (i.e. successful search) operations and assume $m \geq n$.

For the *dynamic finger* property [49], $\mathsf{DF}(X) = \sum_t \log|x_t - x_{t+1}|$, and for *working set* [49], $\mathsf{WS}(X) = \sum_t \log \rho_t(x_t)$, where $\rho_t(a)$ is the number of distinct keys accessed between time $t$ and the last time at which $a$ was accessed (all keys assumed accessed at time zero).

In 2001, Iacono [26] initiated the study of a property that would "unify" the latter two notions of efficiency and exhibited a data structure (not a BST) achieving this property. This *unified bound* is defined as $\mathsf{UB}(X) = \sum_t \min_{t'<t} \log(|x_t - x_{t'}| + \rho_t(x_{t'}))$. Dynamic finger and working set are in general, not comparable. On the other hand, $\mathsf{UB}(X) \le \mathsf{DF}(X)$, and $\mathsf{UB}(X) \le \mathsf{WS}(X)$ clearly hold, justifying the name of the unified bound.

Despite several attempts, the question whether the unified bound is a valid BST property remains unclear; it was shown in [20] that $\mathsf{OPT}(X) = O(\mathsf{UB}(X) + m \log\log n)$, and in [11, 26] that the unified bound is valid in some other (non-BST) models[9].

We show that a unified bound with "bounded time-window" holds in the BST model:

▶ **Theorem 1.** *For every integer $\ell \ge 1$, every sequence $X$ and some fixed function $\beta(\cdot)$,*

$$\mathsf{OPT}(X) \le \beta(\ell) \cdot \mathsf{UB}^\ell, \quad \textit{where} \quad \mathsf{UB}^\ell = \sum_t \min_{t' \in [t-\ell, t)} \log\big(|x_t - x_{t'}| + \rho_t(x_{t'})\big).$$

Observe that $\mathsf{UB}(X) = \mathsf{UB}^m(X) \le \cdots \le \mathsf{UB}^1(X) = \mathsf{DF}(X)$. Prior to our work it was not known whether the theorem holds when $\ell = 2$, i.e. no known BST property subsumes this property even when $\ell = 2$. Thus, Theorem 1 establishes the first BST property that combines the efficiencies of time- and keyspace-proximity without an additive term.[10]

Recently Bose et al. [8] introduced the *lazy finger* property, $\mathsf{LF}(X) = \min_R \sum_i d_R(x_i, x_{i+1})$. Here distance is measured in a static reference BST $R$, optimally chosen for the entire sequence. The lazy finger bound can be visualized as follows: accesses are performed in the reference tree by moving a unique finger from the previously accessed item to the requested item. The lazy finger property is rather strong: Bose et al. show that it implies the dynamic finger and static optimality properties, which in turn imply static finger.

Our main tool in proving Theorem 1 is a generalization of the lazy finger property allowing multiple fingers. The model is motivated by the famous $k$-server problem. For an input sequence $X \in [n]^m$ and a static BST $R$ with nodes associated with the keys in $[n]$, we have $k$ servers located initially at arbitrary nodes in $R$. At time $t = 1, \ldots, m$, the request $x_t$ arrives, and we move a server of our choice to the node of $R$ that stores $x_t$. The cost for serving a sequence $X$ is equal to the total movement in $R$ to serve the sequence $X$.

Denote by $\mathsf{F}^k_R(X)$ the cost of the optimal (offline) strategy that serves sequence $X$ in $R$ with $k$ servers, minimized over all possible initial server locations. Let $\mathsf{F}^k(X) = \min_R \mathsf{F}^k_R(X)$. We call $\mathsf{F}^k(X)$ the *$k$-finger cost* of $X$. We remark that the value of $\mathsf{F}^k_R(X)$ is polynomial-time computable for each $R$, $k \in \mathbb{N}$, and $X \in [n]^m$ by dynamic programming. Clearly, $\mathsf{F}^1(X) \ge \mathsf{F}^2(X) \ge \cdots \ge \mathsf{F}^n(X)$ holds for all $X$.

We first show that one can simulate any $k$-finger strategy in the BST model, in a near-optimal manner. In particular, we prove the following tight result.

▶ **Theorem 2.** $\mathsf{OPT}(X) \le O(\log k) \cdot \mathsf{F}^k(X)$.

The proof of Theorem 2 is a refinement of an earlier argument [18], improving the overhead factor from $O(k)$ to $O(\log k)$. The logarithmic dependence on $k$ is, in general, the best possible. To see this, consider a sequence $S$ of length $m$, over $k$ distinct items with average cost $\Omega(\log k)$ (e.g. a random sequence from $[k]^m$ does the job). While $\mathsf{OPT}(S) = \Theta(m \log k)$, clearly $\mathsf{F}^k(X) = O(m)$, as each of the $k$ items can be served with its own private finger.

[9] Another attempt to study the bounds related to the unified bound was done in [24].

[10] The proof of Theorem 1 implies in fact a stronger, *weighted* form, which we omit for ease of presentation.

In the definition of $\mathsf{F}^k(X)$ we assume a *static* reference tree $R$ for the $k$-finger execution. The offline BST simulation in the proof of Theorem 2 works in fact (with the same overhead) even if $R$ is *dynamic*, i.e. if the multi-finger adversary can perform rotations at any of the fingers. In this case, however, the $k$-finger bound is too strong to be useful; already the $k = 1$ case captures the dynamic BST optimum. Our next result is the online counterpart of Theorem 2. In this case, the restriction that $R$ is static is essential.

▶ **Theorem 3.** *There exists an online randomized BST algorithm whose cost for serving $X \in [n]^m$, is $O\big((\log k)^7\big) \cdot \mathsf{F}^k(X) + \rho(n)$, for some fixed function $\rho(\cdot)$.*

The result can be interpreted as follows. On sufficiently long access sequences, there is an online BST algorithm (in fact, a family of them) competitive with the $k$-finger bound, up to an overhead factor with moderate dependence on $k$. The randomized algorithm (as is standard in the online setting) assumes an oblivious adversary that does not know in advance the outcomes of the algorithm's random coin-flips. The use of randomness seems essential to our approach. We propose as intriguing open questions to find a deterministic online BST with comparable guarantees and to narrow the gap between the online and offline results.

Due to its substantial amount of computation (outside the BST model), our online algorithm is of theoretical interest only. Nonetheless, the connection with the $k$-server problem allows us to "import" several techniques to the BST problem; some of these, such as the *double coverage* heuristic for $k$-server [14] are remarkably simple and may find their way to practical BST algorithms.

The strength of the $k$-finger model lies in the $k$-server abstraction. In order to establish a BST property of the form $\mathsf{OPT}(X) \le \beta(\ell) \cdot O(g(X))$, it is now sufficient to prove $\mathsf{F}^\ell(X) \le (\beta(\ell)/\log \ell) \cdot O(g(X))$. In other words, our technique reduces the task of bounding the cost in the BST model to designing $k$-server strategies, which typically admits much cleaner combinatorial arguments. We illustrate this approach by showing that the unified property with a fixed time-window holds in the BST model.

▶ **Theorem 4.** *For some fixed functions $\alpha(\cdot), \gamma(\cdot)$, we have: $\mathsf{F}^{\alpha(\ell)}(X) \le \gamma(\ell) \cdot \mathsf{UB}^\ell$.*

Theorems 4 and 2 together imply Theorem 1. Moreover, Theorem 3 implies that the property holds for *online* BST algorithms (we later specify the involved functions).

The $k$-finger approach can be used to show further BST properties. For example, we connect *decomposability* (refer to § 4 for definitions) and finger properties by showing that even one finger is enough to obtain the *traversal* property in significantly generalized form.

▶ **Theorem 5.** *Let $X$ be a $d$-decomposable sequence. Then $\mathsf{F}^1(X) = O(\log d) \cdot |X|$.*

As a corollary, using the recent result by Iacono and Langermann [28], we resolve an open problem in [13], showing that Greedy costs at most $O(\log d) \cdot |X|$ on every $d$-decomposable sequence, matching the lower bound in [13].[11]

In another direction, we connect multiple fingers and generalized monotone sequences. In [13], we showed that $\mathsf{OPT}(X) \le |X| \cdot 2^{O(d^2)}$ on every $d$-monotone sequence $X$; a sequence is $d$-monotone if it can be decomposed into $d$ increasing or $d$ decreasing sequences. Using the $k$-finger technique, we show the stronger BST property $\mathsf{OPT}(X) \le O(d \log d) \cdot |X|$.

Concerning simple and natural BST algorithms (Splay and Greedy), we give evidence that the strongest results in the literature may still be far from settling the dynamic optimality conjecture. To this end, we describe a class of sequences for which increasing the number of fingers by one can create an $\Omega(\log n)$ gap. More precisely, we show the following:

[11] Independently of our work, Goyal and Gupta [22] showed the same result using a charging argument.

▶ **Theorem 6.** *For every integer $k$, there is a sequence $S_k$ such that $\mathsf{F}^{k-1}(S_k) = \Omega(\frac{n}{k}\log(n/k))$ but $\mathsf{F}^k(S_k) = O(n)$.*

Theorem 6 shows that the multi-finger bounds form a fine-grained hierarchy. For small $k$, our online algorithm (Theorem 3) can match these bounds (up to a constant factor). However, any online BST (such as Splay or Greedy) must also match the dependence of $O(\log k)$ in the upper bound of $O(\log k) \cdot F^k(X)$, in order to be constant-competitive.

**Techniques. The *k*-server problem.** The $k$-server problem, introduced by Manasse, McGeoch, and Sleator [38] in 1988 is a central problem in online algorithms: Is there an online deterministic strategy for serving a sequence of requests by moving $k$ servers around, with a total movement cost at most $k$ times the optimal offline strategy? The question in its original form, for arbitrary metric spaces, remains open. Nonetheless, the problem has inspired a wealth of results and a rich set of techniques, many of which have found applications outside the $k$-server problem. A full survey is out of our scope, we refer instead to some prominent results [21, 34, 47, 44, 3, 2], and the surveys [6, § 10, § 11], [33]. Most relevantly for us, Chrobak and Larmore [14] gave in 1991, an intuitive, deterministic, $k$-competitive algorithm for *tree metrics*, and the very recently announced breakthrough of Lee [35], building on Bubeck et al. [12], gives an $O\big((\log k)^6\big)$-competitive randomized algorithm for arbitrary metrics.

Our online BST algorithm relies on an online $k$-server in an almost black box fashion (the metric space underlying the $k$-server instance is induced by a static reference BST). Thus, improvements for $k$-server would directly yield improvements in our bounds. Despite the depth and generality of $k$-server (e.g. it also models the caching/paging problem), to our knowledge it has previously not been related to the BST problem.[12]

It is known that in an arbitrary metric space with at least $k+1$ points, no deterministic online algorithm may have a competitive ratio better than $k$. In the randomized case the lower bound $\Omega(\log k/\log\log k)$ holds, see e.g. [33]. (The lower bounds thus apply for a metric induced by a BST, for all $k < n$.) These results imply a remarkable separation between the $k$-server and BST problems. Dynamic optimality would require, by Theorem 2, a BST cost of $O(\log k) \cdot \mathsf{F}^k$. To match this, an online BST may not implicitly perform a deterministic $k$-server execution, since, in that case its overhead would have to be $\Omega(k)$. This indicates that improving Theorem 3 will likely require tools significantly different from $k$-server, which is surprising, given the similarity of the two formulations.

Our online BST learns the metric induced by the optimal reference tree using a multiplicative weights update (MWU) scheme. The technique has a rich history, and a recent emergence as a powerful algorithmic tool (we refer to the survey of Arora, Hazan, and Kale [1]). MWU or closely related techniques have been used previously in data structures (including for BST-related questions), see e.g. [5, 4, 27, 29]. Specifically, Iacono [27] obtains, using MWU, an online BST that is constant-competitive on sufficiently long sequences, *if any online BST is constant-competitive.* As we relate online BSTs with an offline strategy, the results are not directly comparable.

[12] In his work on a generalized $k$-server problem, Sitters [48] asks whether the work-function (WF) technique [34] for $k$-server may have relevance for BSTs. Indeed, we can use WF as an $O(k)$-competitive component of our online BSTs, but for our special case of tree-metrics, the technique of [14] is much simpler. Whether WF may be used (in different ways) to obtain competitive BSTs remains open.

**Further open questions and structure of the paper.** The main open question raised by our work is whether natural algorithms such as Splay or Greedy match the properties of our new BST algorithms. (This must be the case, if Splay and Greedy are, as conjectured, $O(1)$-competitive). We suggest the following easier questions. Do Splay or Greedy satisfy the unified bound with a time-window of 2 steps? Does Splay satisfy the lazy finger or the 2-monotone bounds? Does Greedy satisfy the 2-finger bound?

Except for Theorems 2 and 5, the factors in our results are not known to be tight. Improving them may reveal new insight about the power and limitations of the BST model.

In § 2 we describe our offline BST simulation. In § 3 we describe our new family of online algorithms. In § 4 we prove the main applications and further observations.

## 2 Offline simulation of multi-finger BSTs (Theorem 2)

Let $k \in \mathbb{N}$ , let $T$ be a BST on $[n]$, and let $X = (x_1, \ldots, x_m) \in [n]^m$ be an access sequence. A *$k$-finger strategy* consists of a sequence $\vec{f} \in [k]^m$ where $f_t \in [k]$ specifies the finger that serves access $x_t$. Let $\vec{\ell} \in [n]^k$ be the *initial vector*, where $\ell_i \in [n]$ gives the initial location of finger $i$. The cost of strategy $(\vec{f}, \vec{\ell})$ is $\mathsf{F}^k_{T,\vec{f},\vec{\ell}}(X) = \sum_{t=1}^{m}(1 + d_T(x_t, x_{\sigma(f_t,t)}))$ where $\sigma(i,t) = \max\{j < t \mid f_j = i\}$ is the location of finger $i$ before time $t$, and $\sigma(i,1) = \ell_i$. Let $\mathsf{F}^k_T(X) = \min_{\vec{f},\vec{\ell}} \mathsf{F}^k_{T,\vec{f},\vec{\ell}}(X)$. In other words, for a fixed BST $T$ on keyset $[n]$, $\mathsf{F}^k_T(X)$ is the *$k$-server* optimum for serving $X$ in the metric space of the tree $T$. (Note that the tree is unweighted, and the distance $d_T(\cdot,\cdot)$ counts the number of edges between two nodes in $T$.) We define $\mathsf{F}^k(X) = \min_T \mathsf{F}^k_T(X)$. It is clear form the definition that $\mathsf{F}^1(X) \geq \mathsf{F}^2(X) \geq \cdots \geq \mathsf{F}^n(X) = m$ for all $X$.

Observe that we implicitly assume that during every access at most one server moves. In addition, we may assume that if some server is already placed at the requested node, then no movement happens. Algorithms with these two restrictions are called *lazy*. As argued in the $k$-server literature (see e.g. [33]), non-lazy server movements can always be postponed to a later time, keeping track of the "virtual" locations of servers. In other words, every $k$-server algorithm can be simulated by a lazy algorithm, without additional cost. We therefore assume throughout the paper that $k$-server/$k$-finger executions are lazy.

Consider some (lazy) $k$-finger execution $(\vec{f}, \vec{\ell})$ in tree $T$, for access sequence $X$. We can view $\vec{f}$ as an explicit sequence of elementary steps $\mathcal{S} = \mathcal{S}^k_{T,\vec{f},\vec{\ell}}$, where in each step we move one of the fingers to its parent or to one of its children in $T$. We further allow $\mathcal{S}$ to contain rotations at a finger in $T$ (although $k$-finger strategies as described above do not generate rotations). The position of a finger is maintained during a rotation.

We show how $\mathcal{S}$ can be simulated in a standard dynamic BST. If in $\mathcal{S}$ a finger visits a node, then the (single) pointer in the BST also visits the corresponding node, therefore all accesses are correctly served in the BST. Every elementary step in $\mathcal{S}$ is mapped to (amortized) $O(\log k)$ elementary steps (pointer moves and rotations) in the BST. This immediately implies Theorem 2, since, if we can simulate an arbitrary $k$-finger execution, then indeed we can simulate the optimal $k$-finger execution on the best static tree. Assuming that the intial conditions $T$ and $\vec{\ell}$ are known, the steps of $\mathcal{S}$ are simulated one-by-one, without any lookahead. Thus, insofar as the $k$-finger execution is *online*, the BST execution is also online (this fact is used in § 3).

Let us describe simulation by a standard BST $T'$ of a $k$-finger execution $\mathcal{S}$ in a BST $T$. The construction is a refinement of the one given by Demaine et al. [18], see also [19]. (We improve the overhead factor from $O(k)$ to $O(\log k)$.) The main ingredients are: (1) Making sure that each item with a finger on it in $T$ has depth at most $O(\log k)$ in $T'$. (In [18], each

finger may have depth up to $O(k)$ in $T'$.) (2) Implementing a deque data structure within $T'$ so that each finger in $T$ can move to any of its neighbors, or perform a rotation, with cost $O(\log k)$ amortized. (In [18], this cost is $O(1)$ amortized.)

Given these ingredients, to move a finger $f$ to its neighbor $x$ in $T$, we can simply access $f$ from the root of $T'$ in $O(\log k)$ steps, and then move $f$ to $x$ in $T'$ in $O(\log k)$ amortized steps, with a similar approach for a rotation at $f$. Hence, the overhead factor is $O(\log k)$. We sketch the main technical ideas, postponing the details to Appendix A.

Consider the tree $S$ induced by the current fingers and the paths connecting them in $T$. The tree $S$ consists of finger-nodes and non-finger nodes of degree 3 (both types of nodes are called *pseudo-fingers*), and paths of non-finger nodes of degree 2 connecting pseudo-fingers with each other, called *tendons*. Tendons can be compressed into a BST structure that allows their traversal between the two endpoints in $O(1)$ steps.

We maintain $S$ as a root-containing subtree of our BST $T'$, called the *hand*. Due to the compression of the tendons, the relevant part of $S$ has size $O(k)$. The description so far, including the terminology, is identical to the one in [18, § 2]. Our construction differs in the fact that it maintains the hand, i.e. the compressed representation of $S$ as a *balanced* BST. This guarantees the reachability of fingers in $O(\log k)$ instead of $O(k)$ steps, i.e. property (1).

When a finger in $T$ moves or performs a rotation, the designation of some (pseudo)finger, or tendon nodes may change. Such changes can be viewed as the insertion or deletion of items in the tendons. As these operations happen only at certain places within the tendons, they can be implemented efficiently. We implement tendons with the same BST-based *deque* as [18]. The construction appears to be folklore, we describe it in Appendix A.1 for completeness.

We depart again from [18], as the operation affecting the (pseudo)finger and tendon nodes can trigger a re-balancing of the hand, which may again require $O(\log k)$ operations to fix, i.e. property (2). Any efficient balancing strategy (e.g. red-black tree) may be used.

## 3 Online simulation of multi-finger BSTs (Theorem 3)

Consider the optimal (offline) $k$-finger execution $\vec{f}$ for access sequence $X \in [n]^m$, with static reference tree $T$ and initial finger-placement $\vec{\ell}$. We wish to simulate it by a dynamic *online* BST. The construction proceeds in two stages: (1) A simulation of $\vec{f}$ by a sequence $\mathcal{S}$ of steps that describe finger-movements and rotations-at-fingers, starting from an arbitrary BST $T_0$ and arbitrary finger locations $\vec{\ell}_0$. The sequence $\mathcal{S}$ is *online*, i.e. it is constructed without knowledge of the optimal initial state $T$,$\vec{\ell}$, and it correctly serves the sequence $X$, as its elements are revealed one-by-one. (2) A step-by-step simulation of $\mathcal{S}$ by a standard BST algorithm using the result of § 2. Since $\mathcal{S}$ is online, the BST algorithm is also online.

As before, we denote by $\mathsf{F}^k(X) = \mathsf{F}^k_{T,\vec{f},\vec{\ell}}(X)$ the cost of the optimal offline execution. Observe that this is exactly the $k$-server optimum with the tree metric defined by $T$ and initial configuration of servers $\vec{\ell}$. If $T$ and $\vec{\ell}$ were known, we could conclude part (1) by running an arbitrary *online* $k$-server algorithm defined on tree metrics.

To this end, we mention two online $k$-server algorithms, the deterministic "double coverage" algorithm of Chrobak and Larmore [14] (Algorithm A) and the very recently announced randomized algorithm of Lee [35, 12] (Algorithm B). It is known that the cost of Algorithms A, resp. B is at most $k$-times, resp. $O((\log k)^6)$ times $\mathsf{F}^k$. We only describe Algorithm A, as it is particularly intuitive. To obtain the claimed result, we need the much more complex Algorithm B. (By using Algorithm A we get an overall factor $O(k \log k)$.)

During the execution of Algorithm A, given a current access request $x_t$, call those servers (fingers) *active*, whose path to $x_t$ in $T$ does not contain another server. If several servers are in the same location, one of them is chosen arbitrarily to be active. Algorithm A serves $x_t$ as follows: as long as there is no server on $x_t$, move all active servers one step closer to $x_t$. Observe that as servers move, some of them may become inactive. Algorithm A (as described) may need to move multiple servers during one access. It can, however, easily be transformed into a lazy algorithm, as discussed in § 2.

Remains the issue that the optimal initial $T$ and $\vec{\ell}$ are not known. Let $B_1, \ldots, B_N$ be instances of an online $k$-server algorithm (in our case Algorithm B), one for each combination of initial tree $T$ and initial server-placement $\vec{\ell}$. Note that $N = O(4^n \cdot n^k)$. Let $\mathcal{M}$ be a "meta-algorithm" that simulates all $B_j$'s for $j = 1, \ldots, N$, competitive on sufficiently long input with the best $B_j$. Algorithm $\mathcal{M}$ processes $X$ in epochs of length $M = n \log n$, executing in the $i$-th epoch, for $i = 1, \ldots, \lceil m/M \rceil$, some $B_{\tau(i)}$ according to a (randomized) choice $\tau(i)$.

Suppose that $\vec{\ell}^*$ and $T^*$ describe the state of $B_{\tau(i)}$ chosen by $\mathcal{M}$ at the beginning of the $i$-th epoch. To switch to the state $\vec{\ell}^*$, $T^*$, $\mathcal{M}$ takes $O(n \log n)$ elementary steps: (1) rotate the current tree to a *balanced* tree using any of the fingers ($O(n)$ steps), (2) move all fingers to their location in $\vec{\ell}^*$ ($k$ times $O(\log n)$ steps), (3) use an arbitrary finger $f$ to rotate the tree to $T^*$ ($O(n)$ steps), (4) move $f$ back to its location in $\vec{\ell}^*$ ($O(n)$ steps). Since $M = n \log n$, the cost of switching can be amortized over the epoch.

The choice of $B_{\tau(i)}$ for epoch $i$ is done according to the multiplicative-weights (MW) technique [1], based on the past performance of the various algorithms. Our *experts* are the online executions $B_1, \ldots, B_N$, our $i$-th *event* is the portion of $X$ revealed in the $i$-th epoch, the *loss* of the $j$-th expert for the $i$-th event is the *cost* of $B_j$ in the $i$-th epoch. Let $C_{max}$ denote the maximum possible loss of an expert for an event (we may assume $C_{max} \leq n \cdot M$).

It follows from the standard MW-bounds [1, Thm. 2.1], that for an arbitrary $\varepsilon \in (0, 1)$, the cost of $\mathcal{M}$ on $X$ is at most $\min_j (1+\varepsilon)\mathcal{C}_j + \dfrac{C_{max} \cdot \ln N}{\varepsilon}$, where $\mathcal{C}_j$ is the cost of expert $B_j$ for the entire $X$; in particular, $B_j$ may correspond to the optimal offline choice $\vec{\ell}$, $T$, in which case $\mathcal{C}_j = O((\log k)^6) \cdot \mathsf{F}^k(X)$.

Thus, for e.g. $\varepsilon = 1/2$, we obtain that the cost of $\mathcal{M}$ on $X$ is at most $O((\log k)^6) \cdot \mathsf{F}^k(X) + O(n^3 \log^2 n)$. The output of $\mathcal{M}$ is an *online* sequence $\mathcal{S}_{\mathcal{M}}$ of rotations and finger moves, starting from an arbitrary initial state $T_0$ and $\vec{\ell}_0$. Note that while $\mathcal{M}$ needs to evaluate the costs and current states for all experts in all epochs (an extraordinary amount of computation), only one of the experts interacts with the tree at any time. Thus, $\mathcal{S}_{\mathcal{M}}$ is a standard sequence of steps which can be simulated by a standard BST algorithm according to Theorem 2, at the cost of a further $O(\log k)$ factor. This concludes the proof of Theorem 3.

## 4 Applications of the multi-finger property

In this section we show that every BST algorithm that satisfies the $k$-finger property also satisfies the unified bound with fixed time-window (Application 1), is efficient on decomposable sequences (Application 2), and on generalized monotone sequences (Application 3).

**Application 1. Combined space-time sensitivity (Theorem 4).** Recall the definition of $\mathsf{UB}^\ell$ in Theorem 1 for a sequence $X = (x_1, \ldots, x_m) \in [n]^m$. We connect this quantity with the $k$-finger cost, from which Theorem 4 immediately follows.

▶ **Theorem 7.** *For every $\ell$, $F^{(\ell!)}(X) = O(\ell!) \cdot \mathsf{UB}^\ell(X)$.*

Since we are only concerned with the case when $\ell$ is constant, we may drop the term $\rho_t(x_{t'})$ in the definition of $\mathsf{UB}^\ell$ (whose value is always between 1 and $\ell$).

We prove Theorem 7 via another bound in which distances are measured in a static reference BST: $\ell\text{-DistTree}_T(X) = \sum_{i=1}^{m} \min_{i-\ell \le j < i} \{d_T(x_i, x_j) + 1\}$. [13]

▶ **Lemma 8.** $\min_T \ell\text{-}DistTree_T(X) = O(\mathsf{UB}^\ell(X))$.

**Proof.** By [46, Thm. 4.7], there is a randomized BST $\tilde{T}$ such that the expected distance between elements $i$ and $j$ is $E[d_{\tilde{T}}(i,j)] = \Theta(\log|i-j|)$. Therefore,

$$\min_T \ell\text{-DistTree}_T(X) \le E[\sum_{i=1}^{m} \min_{i-\ell \le j < i} \{d_{\tilde{T}}(x_i, x_j) + 1\}] = \sum_{i=1}^{m} E[\min_{i-\ell \le j < i} \{d_{\tilde{T}}(x_i, x_j) + 1\}]$$
$$\le \sum_{i=1}^{m} \min_{i-\ell \le j < i} \{E[d_{\tilde{T}}(x_i, x_j) + 1]\} = \sum_{i=1}^{m} \min_{i-\ell \le j < i} \{O(\log|x_i - x_j|)\} = O(\mathsf{UB}^\ell(X)). \quad \blacktriangleleft$$

It is now sufficient to show that $\mathsf{F}_T^{(\ell!)}(X) = O(\ell!) \cdot \ell\text{-DistTree}_T(X)$, for all $X$ and $T$, i.e. to describe an $(\ell!)$-finger strategy in $T$ for serving $X$ with the given cost.

At a high level, our strategy is the following: (1) Define a *virtual tree* $\mathcal{T}(X)$ whose nodes are the *requests* $x_i$ for $i = 1, \ldots, m$. The virtual tree captures the *proximities* between the requests, with each $x_i$ having as parent the *nearest* request $x_j$ within a fixed time-window before time $i$. Edges in $\mathcal{T}(X)$ are given as weights the distances between requests in $T$. Note that the virtual tree is not necessarily binary. (2) Define a recursive structural decomposition of the tree $\mathcal{T}(X)$, with the property that certain blocks of this decomposition contain requests in non-overlapping time-intervals. (3) Describe a multi-finger strategy on $\mathcal{T}(X)$ for serving the requests, which induces a multi-finger strategy on $T$ with the required cost. (The strategy takes advantage of the decomposition in (2).)

We describe the steps more precisely, deferring some details to Appendix B.

**The virtual tree.** Given a number $\ell$, $X \in [n]^m$, and a BST $T$ over $[n]$ with root $r$, the virtual tree $\mathcal{T} = \mathcal{T}(\ell, T, X)$ is a rooted tree with vertex-set $\{(i, x_i) \mid i \in [m]\} \cup \{(0, x_0)\}$, where $x_0 = r$ is the root of $T$ and $(0, x_0)$ is the root of $\mathcal{T}$. The *parent* of a non-root vertex $(i, x_i)$ in $\mathcal{T}$ is $(j, x_j) = \arg\min_{j \in [i-\ell, i)} \{d_T(x_i, x_j)\}$. In words, $(j, x_j)$ is the request at most $\ell$ steps before $(i, x_i)$, closest to $x_i$ (in $T$).

For each edge $e = ((j, x_j), (i, x_i))$, we define the weight $w_{\mathcal{T}}(e) = d_T(x_i, x_j) + 1$. For each subtree $H$ of $\mathcal{T}$, let $w_{\mathcal{T}}(H)$ be the total weight of its edges. Observe that $w_{\mathcal{T}}(\mathcal{T}) = \ell\text{-DistTree}_T(X)$.

**Structure and decomposition of the virtual tree.** We say that a vertex $(i, x_i)$ is *before* (or *earlier than*) $(j, x_j)$ if $i < j$, otherwise it is *after* (or *later than*). For every subtree $H$ of $\mathcal{T}$ we denote the earliest vertex in $H$ as $\mathsf{start}(H)$ and the latest vertex in $H$ as $\mathsf{end}(H)$. The *time-span* of $H$, denoted $\mathsf{span}(H)$, is $(t_1, t_2]$ where $(t_1, x_{t_1}) = \mathsf{start}(H)$ and $(t_2, x_{t_2}) = \mathsf{end}(H)$, and $H$ is *active* at time $t$ if $t \in \mathsf{span}(H)$.

We describe a procedure to decompose $\mathcal{T}(\ell, T, X)$ into directed paths (for the purpose of analysis), defining the key notions of *$i$-body* and *$i$-core*. The procedure is called on a subtree $H$ of $\mathcal{T}$, and the top-level call is $\mathsf{decompose}(\mathcal{T}, \ell)$.

[13] We let $x_0$ denote the root of $T$, and distances involving negative indices are defined to be $+\infty$.

**procedure** decompose$(H, i)$:

1. If $H$ has no edges, return.
2. Let $C(H)$ be the path from $\mathsf{start}(H)$ to $\mathsf{end}(H)$.
3. Call $C(H)$ an $i$-core of $H$, and call $H$ the $i$-body of $C(H)$.
4. For each connected component $H'$ in $H \setminus C(H)$ invoke $\mathsf{decompose}(H', i-1)$.

Observe that $\mathcal{T}$ itself is an $\ell$-body. Each $i$-body $H$ consists of its $i$-core $C(H)$ and a set of $(i-1)$-bodies that are connected components in $H \setminus C(H)$. For each of those $(i-1)$-bodies $H'$, we say that $H$ is a *parent* of $H'$, defining a tree-structure over bodies. Observe that the number of ancestor bodies of an $i$-body (excluding itself) is $\ell - i$. We make a sequence of further structural observations about the virtual tree and its decomposition.

▶ **Lemma 9** (B.1).

**(i)** *At every time $t$, there are at most $\ell$ active edges in $\mathcal{T}(\ell, T, X)$.*

**(ii)** *The $i$-cores of the decomposition, for $1 \le i \le \ell$, partition the vertices of $\mathcal{T}$.*

**(iii)** *Let $H$ be an $i$-body. At any time during the time-span of $H$, among the $(i-1)$-bodies with parent $H$ at most $i-1$ are active.*

**(iv)** *Let $H$ be an $i$-body. The $(i-1)$-bodies with parent $H$ can be partitioned into $(i-1)$ groups $\mathcal{H}_1, \ldots, \mathcal{H}_{i-1}$ such that, for $1 \le j \le i-1$ and $H', H'' \in \mathcal{H}_j$, the time-spans of $H'$ and $H''$ are disjoint.*

**The strategy for moving fingers.** For two vertices $(i, x_i)$ and $(j, x_j)$ in the virtual tree $\mathcal{T} = \mathcal{T}(\ell, T, S)$, *moving a finger* $f$ from $(i, x_i)$ to $(j, x_j)$ means the following: let $P = ((i_1, x_{i_1}), \ldots, (i_k, x_{i_k}))$ be the unique path from $(i, x_i) = (i_1, x_{i_1})$ to $(j, x_j) = (i_\ell, x_{i_\ell})$ in $\mathcal{T}$. For $j = 1, \ldots, k-1$, we iteratively move a finger $f$ from $x_{i_j}$ to $x_{i_{j+1}}$ using $d_T(x_{i_j}, x_{i_{j+1}})$ steps. Hence, the total number of steps is at most $w_{\mathcal{T}}(P)$.

By *serving an access in an $i$-body $H$*, we mean that, for each $(j, x_j) \in V(H)$, at time $j$ there is a finger move to $x_j$ in $T$. For each $i \le \ell$, let $\mathsf{nf}(i)$ be the number of fingers used for serving accesses in an $i$-body. We define $\mathsf{nf}(1) = 1$ and $\mathsf{nf}(i) = 1 + (i-1) \cdot \mathsf{nf}(i-1)$, thus, by induction, $\mathsf{nf}(i) \le i!$ for all $i \le \ell$.

We now describe the strategy for moving fingers. Let $F$ be a set of fingers where $|F| = \mathsf{nf}(\ell)$. At the beginning all fingers are at $(0, x_0)$. (In the reference tree $T$, all fingers are initially at the root $x_0$.) For $1 \le j \le m$, we call $\mathsf{access}(\mathcal{T}, F, (j, x_j))$, defined below for an $i$-body $H$, set of fingers $F$, and $u \in V(H)$.

**procedure** access$(H, F, u)$:

Let $C = C(H)$ be the $i$-core of $H$, with $C = \{u_1, \ldots, u_{k'}\}$, where $u_k$ is before $u_{k+1}$ for each $k$. For $1 \le j \le i-1$, let $\mathcal{H}_j$ be the $j$-th group of the $(i-1)$-bodies with parent $H$ ($\mathcal{H}_j$ defined in Lemma 9(iv)). The $i$-bodies in $\mathcal{H}_j$ are ordered by their time-span. That is, suppose $\mathcal{H}_j = \{H'_1, \ldots, H'_{\ell'}\}$. For each $\ell$, if $\mathsf{span}(H'_\ell) = (a_1, a_2]$ and $\mathsf{span}(H'_{\ell+1}) = (b_1, b_2]$, then $a_2 \le b_1$. Fingers in $F$ are divided into $i$ groups $F_1, \ldots, F_{i-1}, \{f_i\}$, where $|F_j| = \mathsf{nf}(i-1)$, for $j \le i-1$, and $f_i$ is a single finger.

1. If $u \in C$, then move $f_i$ to $u$ from the predecessor node of $u$ in $C$. If $u = \mathsf{end}(H)$, then move $F$ from $\mathsf{end}(H)$ to $\mathsf{start}(H)$.
2. Else let $u \in V(H') \setminus V(C)$ where $H' \in \mathcal{H}_j$. If $u = \mathsf{start}(H')$ and $H'$ is the first $(i-1)$-body in $\mathcal{H}_j$, move $F_j$ from $\mathsf{start}(H)$ to $\mathsf{start}(H')$. Perform $\mathsf{access}(H', F_j, u)$. If $u = \mathsf{end}(H')$ and if $H'$ is the last in $\mathcal{H}_j$ then move $F_j$ from $\mathsf{start}(H')$ to $\mathsf{end}(H)$. Otherwise, if $u = \mathsf{end}(H')$ and there is a next $(i-1)$-body $H''$ in $\mathcal{H}_j$, then move $F_j$ from $\mathsf{start}(H')$ to $\mathsf{start}(H'')$.

In order to give the reader more intution, we give an alternative description. A 1-body $H$ consists only of its 1-core $C(H)$. We use one finger and move it through $C(H)$. For $i > 1$, an $i$-body $H$ decomposes in its $i$-core $C(H)$ and $i-1$ groups $\mathcal{H}_1$ to $\mathcal{H}_{i-1}$ of $(i-1)$-bodies. Initially, we have $\mathsf{nf}(i)$ fingers on $\mathsf{start}(H)$. We use one finger to move down the $i$-core. We use a group $F_j$ of $\mathsf{nf}(i-1)$ fingers for the $j$-group $\mathcal{H}_j$. Let $H_1, \ldots H_p$ be the $(i-1)$-cores in $\mathcal{H}_j$. We first move $F_j$ to $\mathsf{start}(H_1)$. Then we use the strategy recusively to move $F_j$ through $H_1$. Once the group of fingers has reached $\mathsf{end}(H_1)$, we move them to $\mathsf{start}(H_2)$, and so on. Once the fingers have reached $\mathsf{end}(H_p)$, we move them back to $\mathsf{start}(H)$. We coordinate (this is not really necessary) the movement of the fingers by the order of the accesses in the access sequence $X$.

From the description of $\mathsf{access}$ it is clear that all accesses in $\mathcal{T}$ are served and that $\mathsf{nf}(\ell)$ fingers are sufficient. It remains to bound the total number of steps all fingers move. For an $i$-body $H$, let $\mathsf{cost}(H)$ be the total cost of calling $\mathsf{access}(H, F, u)$ for all $u \in H$. Let $\mathcal{H}$ denote the set of $(i-1)$-bodies with parent $H$. Let $C^+(H)$ denote the $i$-core $C(H)$ augmented with the edges connecting $C(H)$ to the $(i-1)$-bodies in $\mathcal{H}$. Then:

▶ **Lemma 10** (B.2). $\mathsf{cost}(H) \leq 2 \cdot \mathsf{nf}(i) \cdot w_{\mathcal{T}}(C^+(H)) + \sum_{H' \in \mathcal{H}} \mathsf{cost}(H')$.

By induction, we obtain $\mathsf{cost}(H) \leq 2 \cdot i! \cdot w_{\mathcal{T}}(H)$. (For $i = 1$ we have $H = C(H)$.) Since $\mathsf{nf}(\ell) \leq \ell!$, we have that $\mathsf{F}_T^{(\ell!)}(X) \leq \mathsf{F}_T^{\mathsf{nf}(\ell)}(X) \leq \mathsf{cost}(\mathcal{T})$. By the previous claim we have $\mathsf{cost}(\mathcal{T}) \leq 2 \cdot (\ell!) \cdot w_{\mathcal{T}}(\mathcal{T}) = 2 \cdot (\ell!) \cdot \ell\text{-DistTree}_T(X)$, concluding the proof.

**Application 2. Decomposable sequences (Theorem 5).** Let $\sigma = (\sigma(1), \ldots, \sigma(n))$ be a permutation. For $a, b : 1 \leq a < b \leq n$, we say that $[a, b]$ is a *block* of $\sigma$ if $\{\sigma(a), \ldots, \sigma(b)\} = \{c, \ldots, d\}$ for some integer $c, d \in [n]$. A *block partition* of $\sigma$ is a partition of $[n]$ into $k$ blocks $[a_i, b_i]$ such that $(\bigcup_i [a_i, b_i]) \cap \mathbb{N} = [n]$. For such a partition, for each $i = 1, \ldots, k$, consider a permutation $\sigma_i \in S_{b_i - a_i + 1}$ obtained as an order-isomorphic permutation when restricting $\sigma$ on $[a_i, b_i]$. For each $i$, let $q_i \in [a_i, b_i]$ be a representative element of $i$. The permutation $\tilde{\sigma} \in [k]^k$ that is order-isomorphic to $\{\sigma(q_1), \ldots, \sigma(q_k)\}$ is called a *skeleton* of the block partition. We may view $\sigma$ as a *deflation* $\tilde{\sigma}[\sigma_1, \ldots, \sigma_k]$.

A permutation $\sigma$ is $d$-decomposable if $\sigma = (1)$, or $\sigma = \tilde{\sigma}[\sigma_1, \ldots, \sigma_{d'}]$ for some $d' \leq d$ and each permutation $\sigma_i$ is $d$-decomposable (we refer to [13] for alternative definitions). Permutations that are 2-decomposable are called *separable* [7], and this class includes preorder traversal sequences [49] as a special case.

To show Theorem 5, it is sufficient to define a reference tree $T$ and a one-finger strategy for serving a $d$-decomposable sequence $X$ in $T$ with cost $O(\log d) \cdot |X|$. (Appendix C.)

Combined with the Iacono-Langerman result [28] that Greedy BST has the lazy finger property, we conclude that the cost of Greedy on any $d$-decomposable sequence $X$ is at most $O(\log d) \cdot |X|$. The result is tight and strengthens our earlier bound [13] of $|X| \cdot 2^{O(d^2)}$.

**Application 3. Generalized monotone sequences.** A sequence $X \in [n]^m$ is *$k$-monotone*, if it can be partitioned into $k$ subsequences (not necessarily contiguous), all increasing or all decreasing. This property has been studied in the context of adaptive sorting, and special-purpose structures have been designed to exploit the $k$-monotonicity of input sequences (see e.g. [41, 36]). Our results show that BSTs can also adapt to such structure.

▶ **Theorem 11.** *Let $X$ be a $k$-monotone sequence. Then $\mathsf{F}^k(X) = O(k) \cdot |X|$.*

It follows that $\mathsf{OPT}(X) \leq O(k \log k) \cdot |X|$ for $k$-monotone sequences.[14] The simulation is straightforward. Let $\{X_1, \ldots, X_k\}$ be a partitioning of $X$ into increasing sequences (such a partition can be found online). Let $T$ be an arbitrary static BST over $[n]$. Consider $k$ fingers $f_1, \ldots, f_k$, initially all on 1. For accessing $x_j \in X_i$, move finger $f_i$ to $x_j$. Observe that over the entire sequence $X$, each finger does only an in-order traversal of $T$, taking $O(n)$ steps. Thus, $\mathsf{F}^k_T(X) = O(nk)$.

A lower bound of $\Omega(n \log k)$ follows from enumerative results: for sufficiently large $n$, the number of $k$-monotone permutations $X \in [n]^n$ is at least $k^{\Omega(n)}$ (implied by e.g. [45]). Therefore, by a standard information-theoretic argument (see e.g. [5, Thm. 4.1]), there exists a $k$-monotone permutation $X \in [n]^n$ with $\mathsf{OPT}(X) = \Omega(n \log k)$.

**Further results.** We state our hierarchy result (Theorem 6), also implying a weak separation between $k$-finger bounds and "monotone" bounds.

▶ **Theorem 12** (Appendix E). *For all $k$ and infinitely many $n$, there is a $k$-monotone sequence $S_k$ of length $n$, such that:*

- $F^{k-1}(S_k) = \Omega(\frac{n}{k} \log(n/k))$
- $F^k(S_k) = O(n)$ *(independent of $k$).*

In addition, we show a separation between the $k$-finger property and the working set property, showing that for all $k$ and infinitely many $n$, there are sequences $S$ and $S'$ of length $n$, such that $\mathsf{WS}(S) = o(\mathsf{F}^k(S))$, and $\mathsf{F}^k(S') = o(\mathsf{WS}(S))$. (Appendix F.)

## References


1. Sanjeev Arora, Elad Hazan, and Satyen Kale. The Multiplicative Weights Update Method: a Meta-Algorithm and Applications. *Theory of Computing*, 8(1):121–164, 2012. doi:10.4086/toc.2012.v008a006.
2. Nikhil Bansal, Niv Buchbinder, Aleksander Madry, and Joseph Naor. A Polylogarithmic-Competitive Algorithm for the $k$-Server Problem. *J. ACM*, 62(5):40:1–40:49, 2015. doi:10.1145/2783434.
3. Yair Bartal and Eddie Grove. The harmonic $k$-server algorithm is competitive. *J. ACM*, 47(1):1–15, 2000. doi:10.1145/331605.331606.
4. Avrim Blum and Carl Burch. On-line Learning and the Metrical Task System Problem. In *Proceedings of the Tenth Annual Conference on Computational Learning Theory, COLT 1997, Nashville, Tennessee, USA, July 6-9, 1997.*, pages 45–53, 1997. doi:10.1145/267460.267475.
5. Avrim Blum, Shuchi Chawla, and Adam Kalai. Static Optimality and Dynamic Search-Optimality in Lists and Trees. *Algorithmica*, 36(3):249–260, 2003. doi:10.1007/s00453-003-1015-8.
6. Allan Borodin and Ran El-Yaniv. *Online computation and competitive analysis*. Cambridge University Press, 1998.
7. Prosenjit Bose, Jonathan F Buss, and Anna Lubiw. Pattern matching for permutations. *Information Processing Letters*, 65(5):277–283, 1998.
8. Prosenjit Bose, Karim Douïeb, John Iacono, and Stefan Langerman. The Power and Limitations of Static Binary Search Trees with Lazy Finger. In *ISAAC*, pages 181–192, 2014. doi:10.1007/978-3-319-13075-0_15.


[14] The result holds, in fact, for the more general case, when each $X_i$ is either increasing or decreasing.

9 Gerth Stølting Brodal. Finger Search Trees. In *Handbook of Data Structures and Applications*. Chapman and Hall/CRC, 2004. doi:10.1201/9781420035179.ch11.

10 Mark R. Brown and Robert Endre Tarjan. Design and Analysis of a Data Structure for Representing Sorted Lists. *SIAM J. Comput.*, 9(3):594–614, 1980. doi:10.1137/0209045.

11 Mihai Bădoiu, Richard Cole, Erik D. Demaine, and John Iacono. A Unified Access Bound on Comparison-Based Dynamic Dictionaries. *Theoretical Computer Science*, 382(2):86–96, August 2007.

12 Sébastien Bubeck, Michael B. Cohen, James R. Lee, Yin Tat Lee, and Aleksander Madry. k-server via multiscale entropic regularization. In *STOC*, 2018.

13 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Pattern-Avoiding Access in Binary Search Trees. In *FOCS*, pages 410–423, 2015.

14 Marek Chrobak and Lawrence L. Larmore. An Optimal On-Line Algorithm for k-Servers on Trees. *SIAM J. Comput.*, 20(1):144–148, 1991. doi:10.1137/0220008.

15 R. Cole. On the Dynamic Finger Conjecture for Splay Trees. Part II: The Proof. *SIAM Journal on Computing*, 30(1):44–85, 2000. doi:10.1137/S009753979732699X.

16 Richard Cole, Bud Mishra, Jeanette Schmidt, and Alan Siegel. On the Dynamic Finger Conjecture for Splay Trees. Part I: Splay Sorting Log n-Block Sequences. *SIAM J. Comput.*, 30(1):1–43, April 2000. doi:10.1137/S0097539797326988.

17 Erik D. Demaine, Dion Harmon, John Iacono, Daniel M. Kane, and Mihai Pătraşcu. The geometry of binary search trees. In *SODA 2009*, pages 496–505, 2009. URL: http://dl.acm.org/citation.cfm?id=1496770.1496825.

18 Erik D. Demaine, John Iacono, Stefan Langerman, and Özgür Özkan. Combining Binary Search Trees. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part I*, pages 388–399, 2013. doi:10.1007/978-3-642-39206-1_33.

19 Erik D. Demaine, Stefan Langerman, and Eric Price. Confluently Persistent Tries for Efficient Version Control. *Algorithmica*, 57(3):462–483, 2010. doi:10.1007/s00453-008-9274-z.

20 Jonathan Derryberry and Daniel Dominic Sleator. Skip-Splay: Toward Achieving the Unified Bound in the BST Model. In *WADS 2009, Banff, Canada, August 21-23, 2009. Proceedings*, pages 194–205, 2009.

21 Amos Fiat, Yuval Rabani, and Yiftach Ravid. Competitive k-Server Algorithms. *J. Comput. Syst. Sci.*, 48(3):410–428, 1994. doi:10.1016/S0022-0000(05)80060-1.

22 Navin Goyal and Manoj Gupta. On Dynamic Optimality for Binary Search Trees. *CoRR*, abs/1102.4523, 2011. arXiv:1102.4523.

23 Leonidas J. Guibas, Edward M. McCreight, Michael F. Plass, and Janet R. Roberts. A New Representation for Linear Lists. In *Proceedings of the 9th Annual ACM Symposium on Theory of Computing, May 4-6, 1977, Boulder, Colorado, USA*, pages 49–60, 1977. doi:10.1145/800105.803395.

24 John Howat, John Iacono, and Pat Morin. The Fresh-Finger Property. *CoRR*, abs/1302.6914, 2013. arXiv:1302.6914.

25 Scott Huddleston and Kurt Mehlhorn. A New Data Structure for Representing Sorted Lists. *Acta Inf.*, 17:157–184, 1982. doi:10.1007/BF00288968.

26 John Iacono. Alternatives to splay trees with O(log n) worst-case access times. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms, January 7-9, 2001, Washington, DC, USA.*, pages 516–522, 2001. URL: http://dl.acm.org/citation.cfm?id=365411.365522.

27 John Iacono. In Pursuit of the Dynamic Optimality Conjecture. In *Space-Efficient Data Structures, Streams, and Algorithms*, volume 8066 of *Lecture Notes in Computer Science*, pages 236–250. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-40273-9_16.

28 John Iacono and Stefan Langerman. Weighted dynamic finger in binary search trees. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 672–691, 2016.

29 Adam Tauman Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *J. Comput. Syst. Sci.*, 71(3):291–307, 2005. doi:10.1016/j.jcss.2004.10.016.

30 Haim Kaplan and Robert Endre Tarjan. Purely Functional Representations of Catenable Sorted Lists. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 202–211, 1996. doi:10.1145/237814.237865.

31 Donald E. Knuth. *The Art of Computer Programming, Volume 3: (2nd Ed.) Sorting and Searching*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 1998.

32 S. Rao Kosaraju. Localized Search in Sorted Lists. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing, May 11-13, 1981, Milwaukee, Wisconsin, USA*, pages 62–69, 1981. doi:10.1145/800076.802458.

33 Elias Koutsoupias. The k-server problem. *Computer Science Review*, 3(2):105–118, 2009. doi:10.1016/j.cosrev.2009.04.002.

34 Elias Koutsoupias and Christos H. Papadimitriou. On the k-Server Conjecture. *J. ACM*, 42(5):971–983, 1995. doi:10.1145/210118.210128.

35 James R. Lee. Fusible HSTs and the randomized k-server conjecture. *CoRR*, abs/1711.01789, 2017. arXiv:1711.01789.

36 Christos Levcopoulos and Ola Petersson. Sorting Shuffled Monotone Sequences. *Inf. Comput.*, 112(1):37–50, 1994. doi:10.1006/inco.1994.1050.

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

38 Mark S. Manasse, Lyle A. McGeoch, and Daniel Dominic Sleator. Competitive Algorithms for Server Problems. *J. Algorithms*, 11(2):208–230, 1990. doi:10.1016/0196-6774(90)90003-W.

39 K. Mehlhorn and P. Sanders. *Algorithms and Data Structures: The Basic Toolbox*. Springer, 2008.

40 Kurt Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*, volume 1 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1984. doi:10.1007/978-3-642-69672-5.

41 Alistair Moffat and Ola Petersson. An Overview of Adaptive Sorting. *Australian Computer Journal*, 24(2):70–77, 1992.

42 J.Ian Munro. On the Competitiveness of Linear Search. In Mike S. Paterson, editor, *Algorithms - ESA 2000*, volume 1879 of *Lecture Notes in Computer Science*, pages 338–345. Springer Berlin Heidelberg, 2000. doi:10.1007/3-540-45253-2_31.

43 William Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Commun. ACM*, 33(6):668–676, 1990. doi:10.1145/78973.78977.

44 Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms. *IBM Journal of Research and Development*, 38(6):683–708, 1994. doi:10.1147/rd.386.0683.

45 Amitai Regev. Asymptotic values for degrees associated with strips of Young diagrams. *Advances in Mathematics*, 41(2):115–136, 1981.

46 Raimund Seidel and Cecilia R. Aragon. Randomized Search Trees. *Algorithmica*, 16(4/5):464–497, 1996. doi:10.1007/BF01940876.

47 Steven S. Seiden. A General Decomposition Theorem for the k-Server Problem. *Inf. Comput.*, 174(2):193–202, 2002. doi:10.1006/inco.2002.3144.

48 René Sitters. The Generalized Work Function Algorithm Is Competitive for the Generalized 2-Server Problem. *SIAM J. Comput.*, 43(1):96–125, 2014. doi:10.1137/120885309.

49 Daniel Dominic Sleator and Robert Endre Tarjan. Self-Adjusting Binary Search Trees. *J. ACM*, 32(3):652–686, 1985. doi:10.1145/3828.3835.

50 Robert Endre Tarjan and Christopher J. Van Wyk. An O(n log log n)-Time Algorithm for Triangulating a Simple Polygon. *SIAM J. Comput.*, 17(1):143–178, 1988. doi:10.1137/0217010.

51 Athanasios K. Tsakalidis. AVL-Trees for Localized Search. *Information and Control*, 67(1-3):173–194, 1985. doi:10.1016/S0019-9958(85)80034-6.

52 R. Wilber. Lower Bounds for Accessing Binary Search Trees with Rotations. *SIAM Journal on Computing*, 18(1):56–67, 1989. doi:10.1137/0218004.


# A Offline BST simulation

## A.1 BST simulation of a deque

▶ **Lemma 13.** *The minimum and maximum element from a BST-based deque can be deleted in $O(1)$ amortized operations.*

**Proof.** The simulation is inspired by the well-known simulation of a queue by two stacks with constant amortized time per operation ([39, Exercise 3.19]). We split the deque at some position (determined by history) and put the two parts into structures that allow us to access the first and the last element of the deque. It is obvious how to simulate the deque operations as long as the sequences are non-empty. When one of the sequences becomes empty, we split the other sequence at the middle and continue with the two parts. A simple potential function argument shows that the amortized cost of all deque operations is constant. Let $\ell_1$ and $\ell_2$ be the length of the two sequences, and define the potential $\Phi = |\ell_1 - \ell_2|$. As long as neither of the two sequences are empty, for every insert and delete operation both the cost and the change in potential are $O(1)$. If one sequence becomes empty, we split the remaining sequence into two equal parts. The decrease in potential is equal to the length of the sequence before the splitting (the potential is zero after the split). The cost of splitting is thus covered by the decrease of potential.

The simulation by a BST is easy. We realize both sequences by chains attached to the root. The right chain contains the elements in the second stack with the top element as the right child of the root, the next to top element as the left child of the top element, and so on. ◀

## A.2 Extended hand

To describe the simulation precisely, we borrow terminology from [18, 19]. Let $T$ be a BST with a set $F$ of $k$ fingers $f_1, \ldots, f_k$. For convenience we assume the root of $T$ to be one of the fingers. Let $S(T, F)$ be the Steiner tree with terminals $F$. A *knuckle* is a connected component of $T$ after removing $S(T, F)$, i.e. a hanging subtree of $T$. Let $P(T, F)$ be the union of fingers and the degree-3 nodes in $S(T, F)$. We call $P(T, F)$ the set of *pseudofingers*. A *tendon* $\tau_{x,y}$ is the path connecting two pseudofingers $x, y \in P(T, F)$ (excluding $x$ and $y$) such that there is no other $z \in P(T, F)$ inside. We assume that $x$ is an ancestor of $y$.

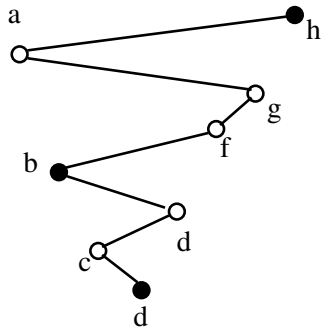


**Figure 1** The pseudofingers are $b$, $d$, and $h$. The half-tendons $\tau^{<}_{h,b}$ and $\tau^{>}_{h,b}$ are $a$ and $g, f$. The intervals in $E(T,F)$ are $[a,a]$, $[b,b]$, $[c,c]$, $[d,d]$, $[f,g]$, and $[h,h]$.

The next terms are new. For each tendon $\tau_{x,y}$, there are two *half tendons*, $\tau^{<}_{x,y}, \tau^{>}_{x,y}$ containing all elements in $\tau_{x,y}$ which are less than $y$ and greater than $y$ respectively. Let $H(T,F) = \{\tau^{<}_{x,y}, \tau^{>}_{x,y} \mid \tau_{x,y} \text{ is a tendon}\}$ be the set of all half tendons.

For each $\tau \in H(T,F)$, we can treat $\tau$ as an interval $[\min(\tau), \max(\tau)]$ where $\min(\tau), \max(\tau)$ are the minimum and maximum elements in $\tau$ respectively. For each $f \in P(T,F)$, we can treat $f$ as an trivial interval $[f,f]$.

Let $E(T,F) = P(T,F) \cup H(T,F)$ be the set of intervals defined by all pseudofingers $P(T,F)$ and half tendons $H(T,F)$. We call $E(T,F)$ an *extended hand*[15]. Note that when we treat $P(T,F) \cup H(T,F)$ as a set of elements, such a set is exactly $S(T,F)$. So $E(T,F)$ can be viewed as a partition of $S(T,F)$ into pseudofingers and half-tendons. Figure 1 illustrates these definitions.

We first state two facts about the extended hand.

▶ **Lemma 14.** *Given any $T$ and $F$ where $|F| = k$, there are $O(k)$ intervals in $E(T,F)$.*

**Proof.** Note that $|P(T,F)| \leq 2k$ because there are $k$ fingers and there can be at most $k$ nodes with degree 3 in $S(T,F)$. Consider the graph where pseudofingers are nodes and tendons are edges. That graph is a tree. So $|H(T,F)| = O(k)$ as well. ◀

▶ **Lemma 15.** *Given any $T$ and $F$, all the intervals in $E(T,F)$ are disjoint.*

**Proof.** Suppose that there are two intervals $\tau, x \in E(T,F)$ that intersect each other. One of them, say $\tau$, must be a half tendon. Because the intervals of pseudofingers are of length zero and they are distinct, they cannot intersect. We write $\tau = \{t_1, \ldots, t_k\}$ where $t_1 < \cdots < t_k$. Assume w.l.o.g. that $t_i$ is an ancestor of $t_{i+1}$ for all $i < k$, and so $t_k$ is an ancestor of a pseudofingers $f$ where $t_k < f$.

Suppose that $x$ is a pseudofinger and $t_j < x < t_{j+1}$ for some $j$. Since $t_j$ is the first left ancestor of $t_{j+1}$, $x$ cannot be an ancestor of $t_{j+1}$ in $T$. So $x$ is in the left subtree of $t_{j+1}$. But then $t_{j+1}$ is a common ancestor of two pseudofingers $x$ and $f$, and $t_{j+1}$ must be a pseudofinger which is a contradiction.

Suppose next that $x = \{x_1, \ldots, x_\ell\}$ is a half tendon where $x_1 < \cdots < x_\ell$. We claim that either $[x_1, x_\ell] \subset [t_j, t_{j+1}]$ for some $j$ or $[t_1, t_k] \subset [x_{j'}, x_{j'+1}]$ for some $j'$. Suppose not. Then there exist two indices $j$ and $j'$ where $t_j < x_{j'} < t_{j+1} < x_{j'+1}$. Again, $x_{j'}$ cannot be an ancestor of $t_{j+1}$ in $T$, so $x_{j'}$ is in the left subtree of $t_{j+1}$. We know either $x_{j'}$ is the first left

[15] In [18], the hand is defined only over the pseudofingers.

ancestor of $x_{j'+1}$ or $x_{j'+1}$ is the first right ancestor of $x_{j'}$. If $x_{j'}$ is an ancestor of $x_{j'+1}$, then $x_{j'+1} < t_{j+1}$ which is a contradiction. If $x_{j'+1}$ is the first right ancestor of $x_{j'}$, then $t_{j+1}$ is not the first right ancestor of $x_{j'}$ and hence $x_{j'+1} < t_{j+1}$ which is a contradiction again. Now suppose w.l.o.g. $[x_1, x_\ell] \subset [t_j, t_{j+1}]$. Then there must be another pseudofinger $f'$ in the left subtree of $t_{j+1}$, hence $\tau$ cannot be a half tendon, which is a contradiction. ◀

## A.3 The structure of the simulating BST

In this section, we describe the structure of the BST $T'$ that we maintain given a $k$-finger BST $T$ and the set of fingers $F$.

For each half tendon $\tau \in H(T, F)$, let $T'_\tau$ be the tree with $\min(\tau)$ as a root which has $\max(\tau)$ as a right child. $\max(\tau)$'s left child is a subtree containing the remaining elements $\tau \setminus \{\min(\tau), \max(\tau)\}$. We implement a *BST simulation of a deque* on this subtree as defined in Appendix A.1. By Lemma 15, intervals in $E(T, F)$ are disjoint and hence they are totally ordered. Since $E(T, F)$ is an ordered set, we can define $T'_{E_0}$ to be a balanced BST such that its elements correspond to elements in $E(T, F)$. Let $T'_E$ be the BST obtained from $T'_{E_0}$ by replacing each node $a$ in $T'_{E_0}$ that corresponds to a half tendon $\tau \in H(T, F)$ by $T'_\tau$. That is, suppose that the parent, left child, and right child are $a_{up}, a_l$ and $a_r$ respectively. Then the parent in $T'_E$ of the root of $T'_\tau$ which is $\min(\tau)$ is $a_{up}$. The left child in $T'_E$ of $\min(\tau)$ is $a_l$ and the right child in $T'_E$ of $\max(\tau)$ is $a_r$.

The BST $T'$ has $T'_E$ as its top part and each knuckle of $T$ hangs from $T'_E$ in a determined way.

▶ **Lemma 16.** *Each element corresponding to pseudofinger $f \in P(T, F)$ has depth $O(\log k)$ in $T'_E$, and hence in $T'$.*

**Proof.** By Lemma 14, $|E(T, F)| = O(k)$. So the depth of $T'_{E_0}$ is $O(\log k)$. For each node $a$ corresponding to a pseudofinger $f \in P(T, F)$, observe that the depth of $a$ in $T'_E$ is at most twice the depth of $a$ in $T'_{E_0}$ by the construction of $T'_E$. ◀

## A.4 The cost for simulating the $k$-finger BST

We finally prove the claim on the cost of our BST simulation, which immediately implies Theorem 2. That is, we prove that whenever one of the fingers in a $k$-finger BST $T$ moves to its neighbor or rotates, we can update the maintained BST $T'$ to have the structure as described in the last section with cost $O(\log k)$.

We state two observations which follow from the structure of our maintained BST $T'$ described in A.3. The first observation follows immediately from Lemma 13.

▶ **Lemma 17.** *For any half tendon $\tau \in H(T, F)$, we can insert or delete the minimum or maximum element in $T'_\tau$ with cost $O(1)$ amortized.*

Next, it is convenient to define a set $A$, called *active set*, as a set of pseudofingers, the roots of knuckles whose parents are pseudofingers, and the minimum or maximum of half tendons.

▶ **Lemma 18.** *When a finger $f$ in a $k$-finger BST $T$ moves to its neighbor or rotates with its parent, the extended hand $E(T, F) = P(T, F) \cup H(T, F)$ is changed as follows.*

1. *There are at most $O(1)$ half tendons $\tau \in H(T, F)$ whose elements are changed. Moreover, for each changed half tendon $\tau$, either the minimum or maximum is inserted or deleted. The inserted or deleted element $a$ was or will be in the active set $A$.*
2. *There are at most $O(1)$ elements added or removed from $P(T, F)$. Moreover, the added or removed elements were or will be in the active set $A$.*

▶ **Lemma 19.** *Let $a \in A$ be an element in the active set. We can move $a$ to the root with cost $O(\log k)$ amortized. Symmetrically, the cost for updating the root $r$ to become some element in the active set is $O(\log k)$ amortized.*

**Proof.** There are two cases. If $a$ is a pseudofinger or a root of a knuckle whose parent is pseudofinger, we know that the depth of $a$ was $O(\log k)$ by Lemma 16. So we can move $a$ to root with cost $O(\log k)$. Next, if $a$ is the minimum or maximum of a half tendon $\tau$, we know that the depth of the root of the subtree $T'_\tau$ is $O(\log k)$. Moreover, by Lemma 17, we can delete $a$ from $T'_\tau$ (make $a$ a parent of $T'_\tau$) with cost $O(1)$ amortized. Then we move $a$ to root with cost $O(\log k)$ worst-case. The total cost is then $O(\log k)$ amortized. The proof for the second statement is symmetric. ◀

▶ **Lemma 20.** *When a finger $f$ in a $k$-finger BST $T$ moves to its neighbor or rotates with its parent, the BST $T'$ can be updated accordingly with cost $O(\log k)$ amortized.*

**Proof.** According to Lemma 18, we separate our cost analysis into two parts.

For the fist part, let $a \in A$ be the element to be inserted into a half tendon $\tau$. By Lemma 19, we move $a$ to root with cost $O(\log k)$ and then insert $a$ as a minimum or maximum element in $T'_\tau$ with cost $O(\log k)$. Deleting $a$ from some half tendon with cost $O(\log k)$ is symmetric.

For the second part, let $a \in A$ be the element to be inserted into a half tendon $\tau$. By Lemma 19 again, we move $a$ to root and move back to the appropriate position in $T'_{E_0}$ with cost $O(\log k)$. We also need rebalance $T'_{E_0}$ but this also takes cost $O(\log k)$. ◀

Finally, we describe the BST simulation of a $k$-finger execution with overhead $O(\log k)$. Let $A$ be an arbitrary $k$-finger execution in BST $T$. Whenever there is an update in $T$ (i.e. a finger moves to its neighbor or rotates), we update the BST $T'$ according to Lemma 20 with cost $O(\log k)$ amortized. The BST $T'$ is maintained so that its structure is as described in Appendix A.3. By Lemma 16, we can access any finger $f$ of $T$ from the root of $T'$ with cost $O(\log k)$. Therefore, the cost of the BST execution is at most $O(\log k)$ times the cost of $A$. This concludes the proof.

# B Missing proofs for Application 1

## B.1 Proof of Lemma 9

**Part (i).**

Suppose that there is some time $t$ when there are $\ell' > \ell$ edges $\{(j_k, x_{j_k}), (i_k, s_{i_k})\}_{k=1}^{\ell'}$ such that $j_k < t \le i_k$ for all $k \le \ell'$. Since each node has a unique parent, $i_1, \ldots, i_{\ell'-1}, i_{\ell'}$ must be distinct and hence $\max_{1\le k\le \ell'} i_k \ge t + \ell' - 1 \ge t + \ell$. Thus $\max_{1\le k\le \ell'} j_k \ge t$, a contradiction.

**Part (ii).**

By construction, the cores are edge-disjoint, and every vertex belongs to some core (the recurrence ends on singleton vertices only). It remains to show that when $\mathsf{decompose}(H, 0)$ is called during the execution of $\mathsf{decompose}(\mathcal{T}, \ell)$, $H$ has no edges, i.e. there is no $i$-core or $i$-body with $i \le 0$.

To see this, define the sequence of graphs $H_0, \ldots, H_\ell$ where $H_\ell = \mathcal{T}(\ell, T, X)$, $H_{i-1}$ is a connected component of $H_i \setminus C(H_i)$, and $H_0 = H$. Recall that $\mathsf{span}(K)$ denotes the time-span of $K$. By definition of $C(H_i)$, we have $\mathsf{span}(H_{i-1}) \subseteq \mathsf{span}(H_i)$.

Suppose for contradiction that $H_0$ has an edge. Denote $\mathsf{span}(H_0) = (t_1, t_2]$, where $t_1 < t_2$. For all $0 \le i \le \ell$, it holds that $\mathsf{span}(H_i) \supseteq (t_1, t_2]$. Let $t \in (t_1, t_2]$. We have that $C(H_i)$ contains an edge $((a_i, x_{a_i}), (b_i, x_{b_i}))$ where $a_i < t \le b_i$ for all $0 \le i \le \ell$. Since $C(H_i)$ are edge-disjoint, this contradicts part (i).

**Part (iii).**

Suppose there are $i$ active $(i-1)$-bodies $H'_1, \ldots, H'_i$ of $H$ at time $t$. Since $H$ is an $i$-body, there are $\ell - i$ ancestors $A_1, \ldots, A_{\ell-i}$ of $H$. For each of the cores $C \in \{C(H'_1), \ldots, C(H'_i), C(H), C(A_1), \ldots, C(A_{\ell-i})\}$ which is a set of size $\ell + 1$, there is an edge $(a, s_a), (b, s_b)$ where $a < t \le b$. This contradicts part (i).

**Part (iv).**

We construct the decomposition greedily. Consider the $(i-1)$ bodies $H'$ ordered by $\mathsf{start}(H')$ and put $H'$ into the group $\mathcal{H}_j$ for the smallest index $j$ such that the time-span of $H'$ is disjoint from the time-spans of all members of the group. Assume that this process opens up $i' > i - 1$ groups. Then there are $(i-1)$-bodies $H'_1$ to $H'_{i'}$ (one per group) such that the time-span of the $i$-body $H$ intersects the time-spans of $H'_1$ to $H'_{i'}$, contradicting part (iii).

## B.2 Proof of Lemma 10

We analyze the total cost of calling $\mathsf{access}(H, F, u)$ for all $u \in V(H)$. The total cost due to recursive calls in Step 2 is accounted by the term $\sum_{H' \in \mathcal{H}} \mathsf{cost}(H')$. The remaining operations amount to moving $\mathsf{nf}(i)$ fingers from $\mathsf{start}(H)$ to $\mathsf{end}(H)$ and back, along the $i$-core $C(H)$. The cost of this is exactly $2 \cdot \mathsf{nf}(i) \cdot w_{\mathcal{T}}(C(H))$. In addition we need to traverse, using $\mathsf{nf}(i-1)$ fingers, the edges connecting $C(H)$ to $\mathsf{start}(H')$, twice for all $H' \in \mathcal{H}$. The total cost thus becomes at most $2 \cdot \mathsf{nf}(i) \cdot w_{\mathcal{T}}(C^+(H)) + \sum_{H' \in \mathcal{H}} \mathsf{cost}(H')$.

We argue now by induction that for an $i$-body $H$, we have $\mathsf{cost}(H) \le 2 \cdot i! \cdot w_{\mathcal{T}}(H)$. For $i = 1$, $H = C(H) = C^+(H)$. Thus, by the inductive step:

$$\mathsf{cost}(H) \le 2 \cdot \mathsf{nf}(1) \cdot w_{\mathcal{T}}(C^+(H)) \le 2 \cdot w_{\mathcal{T}}(H).$$

For the general inductive step:

$$\begin{aligned}
\mathsf{cost}(H) &\le 2 \cdot \mathsf{nf}(i) \cdot w_{\mathcal{T}}(C^+(H)) + \sum_{H' \in \mathcal{H}} \mathsf{cost}(H') \\
&\le 2 \cdot i! \cdot w_{\mathcal{T}}(C^+(H)) + \sum_{H' \in \mathcal{H}} 2 \cdot (i-1)! \cdot w_{\mathcal{T}}(H') \\
&\le 2 \cdot i! \cdot \Big( w_{\mathcal{T}}(C^+(H)) + \sum_{H' \in \mathcal{H}} w_{\mathcal{T}}(H') \Big) \\
&= 2 \cdot i! \cdot w_{\mathcal{T}}(H).
\end{aligned}$$

# C Decomposable Sequences

▶ **Lemma 21.** *Let $X = (x_1, \ldots, x_n)$ be a $k$-decomposable permutation of length $n$. Then $\mathsf{F}^1(X) \le 4(|X| - 1) \lceil \log k \rceil$.*

**Proof.** It is sufficient to define a reference tree $T$ for which $\mathsf{F}^1_T(X)$ achieves such bound. We remark that the tree will have auxiliary elements. We construct $T$ recursively. If $X$ has length one, then $T$ has a single node and this node is labeled by the key in $X$. Clearly, $\mathsf{F}^1_T(X) = 0$.

Otherwise, let $X = \tilde{X}[X_1, \ldots, X_j]$ with $j \in [k]$ be the outermost partition of $X$. Denote by $T_i$ the tree for $X_i$ that has been inductively constructed. Let $T_0$ be a BST of depth at most $\lceil \log j \rceil$ and with $j$ leaves. Identify the $i$-th leaf with the root of $T_i$ and assign keys to the internal nodes of $T_0$ such that the resulting tree is a valid BST. Let $r_i$ be the root of $T_i$, $0 \le i \le j$ and let $r = r_0$ be the root of $T$. Then

$$
\begin{aligned}
d_T(r, x_1) &\le \lceil \log k \rceil + d_{T_1}(r_1, x_1) \\
d_T(r, x_n) &\le \lceil \log k \rceil + d_{T_j}(r_j, x_n) \\
d_T(x_{t-1}, x_t) &\le \begin{cases} d_{T_\ell}(x_{t-1}, x_t) & \text{if } x_{t-1}, x_t \in X_\ell \\ 2\lceil \log k \rceil + d_{T_\ell}(r_\ell, x_{t-1}) + d_{T_{\ell+1}}(r_{\ell+1}, x_t) & \text{if } x_{t-1} \in X_\ell \text{ and } x_t \in X_{\ell+1}, \end{cases}
\end{aligned}
$$

and hence

$$
\begin{aligned}
\mathsf{F}^1_T(X) &= d_T(r, x_0) + \sum_{t \ge 2} d_T(x_{t-1}, x_t) + d_T(x_n, r) \\
&\le 2j \lceil \log k \rceil + \sum_{1 \le \ell \le j} \mathsf{F}^1_{T_\ell}(X_\ell) \quad \le \quad 2j \lceil \log k \rceil + \sum_{1 \le \ell \le j} 4(|X_\ell| - 1) \lceil \log k \rceil \\
&\le (2j - 4j + 4 \sum_{1 \le \ell \le j} |X_\ell|) \lceil \log k \rceil \quad \le \quad 4(|X| - 1) \lceil \log k \rceil \,,
\end{aligned}
$$

where the last inequality uses $j \ge 2$. ◀

## D Finger bounds with auxiliary elements

Recall that $\mathsf{F}(X)$ is defined as the minimum over all BSTs $T$ on $[n]$ of $\mathsf{F}_T(X)$. It is convenient to define a slightly stronger finger bound that also allows *auxiliary elements*. Define $\widehat{\mathsf{F}}(X)$ as the minimum over all BSTs $T$ that contain the keys $[n]$ (but the size of $T$ can be much larger than $n$). We define $\widehat{\mathsf{F}}^k(X)$ as the $k$-finger bound when the tree is allowed to have auxiliary elements. We argue that the two definitions are equivalent.

▶ **Theorem 22.** *For any integer $k$, $\mathsf{F}^k(X) = \Theta(\widehat{\mathsf{F}}^k(X))$ for all $X$.*

**Proof.** It is clear that $\widehat{\mathsf{F}}^k(X) \le \mathsf{F}^k(X)$. We only need to show the converse.

Let $T$ be a BST (with auxiliary elements) such that $\mathsf{F}^k_T(X) = \widehat{\mathsf{F}}^k(X)$. Denote by $\vec{f}$ the optimal finger strategy on $T$. Let $[n] \cup Q$ be the elements of $T$ where $Q$ is the set of auxiliary elements in $T$. For each $a \in [n] \cup Q$, let $d_T(a)$ be the depth of key $a$ in $T$, and let $w(i) = 4^{-d_T(i)}$. For any two elements $i$ and $j$ and set $Y \subseteq [n] \cup Q$, let $w_Y[i:j]$ be the sum $\sum_{k \in Y \cap [i,j]} w(k)$ of the weights of the elements in $Y$ between $i$ and $j$ (inclusive). For any $i, j \in [n] \cup Q$ such that $i \le j$, we have

$$
\log \frac{w_{[n]\cup Q}[i:j]}{\min(w(i), w(j))} = O(d_T(i, j)),
$$

where $d_T(i, j)$ is the distance from $i$ to $j$ in $T$. So, this same bound also holds when considering only keys in $[n]$. That is, for $i, j \in [n]$, we have

$$
\log \frac{w_{[n]}[i:j]}{\min(w(i), w(j))} = O(d_T(i, j)).
$$

Given the weight of $\{w(a)\}_{a\in[n]}$, the BST $T'$ (without auxiliary elements) is constructed by invoking Lemma 23. We bound the term $\mathsf{F}^k_{T'}(X)$ (using strategy $\vec{f}$) by

$$O(\sum_t d_{T'}(x_{\sigma(f_t,t)}, x_t)) = O(\sum_{t=1}^{m-1} \lg \frac{w_{[n]}[x_t : x_{\sigma(f_t,t)}]}{\min(w(x_i), w(x_{\sigma(f_t,t)}))})$$
$$= O(\sum_{t=1}^{m-1} d_T(x_{\sigma(f_t,t)}, x_t)) = O(\mathsf{F}^k_T(X))$$

where $X = (x_1, \ldots, x_m)$. Therefore, $\mathsf{F}^k(X) \le \mathsf{F}^k_{T'}(X) = O(\mathsf{F}^k_T(X)) = O(\widehat{\mathsf{F}}^k(X))$. ◀

▶ **Lemma 23.** *Given a weight function $w(\cdot)$, and $W = \sum_{i\in[n]} w(i)$, there is a deterministic construction of a BST $T_w$ such that the depth of every key $i \in [n]$ is $d_{T_w}(i) = O(\log \frac{W}{w(i)})$.*

**Proof.** Let $w_1, \ldots w_n$ be a sequence of weights. We show how to construct a tree in which the depth of element $\ell$ is $O(\log w[1:\ell]/\min(w_1, w_\ell))$.

For $i \ge 1$, let $j_i$ be minimal such that $w[1 : j_i] \ge 2^i w_1$. Then $w[1 : j_i - 1] < 2^i w_1$ and $w[j_{i-1} + 1 : j_i] \le 2^{i-1} w_1 + w_{j_i}$.

Let $T_i$ be the following tree. The right child of the root is the element $j_i$. The left subtree is a tree in which element $\ell$ has depth $O(\log 2^{i-1} w_1/w_\ell)$.

The entire tree has $w_1$ in the root and then a long right spine. The trees $T_i$ hang off the spine to the left. In this way the depth of the root of $T_i$ is $O(i)$.

Consider now an element $\ell$ in $T_i$. Assume first that $\ell \ne j_i$. The depth is

$$O\left(i + \log \frac{2^{i-1} w_1}{w_\ell}\right) = O\left(i + \log \frac{2^{i-1} w_1}{\min(w_1, w_\ell)}\right)$$
$$= O\left(\log \frac{2^{i-1} w_1}{\min(w_1, w_\ell)}\right) = O\left(\frac{w[1:\ell]}{\min(w_1, w_\ell)}\right).$$

For $\ell = j_i$, the depth is

$$O(i) = O\left(\log \frac{2^i w_1}{w_1}\right) = O\left(\log \frac{w[1:j_i]}{\min(w_1, w_{j_i})}\right).$$ ◀

## E Proof of Theorem 6

Let $n$ be an integer multiple of $k$ and $\ell = n/k$. Consider the tilted $k$-by-$\ell$ grid $S_k$. More precisely, the access sequence is defined as: $1,\ \ell+1,\ \ldots,\ \ell\cdot(k-1)+1,\ 2,\ \ell+2,\ \ldots,\ (k-1)\ell+2, \ldots,\ (k-1)\ell+\ell$. We denote the elements of $S_k$ as $s_i$, for $i = 1, \ldots, n$. To see the geometry of this sequence, one may view it as a partitioning of the keys $[n]$ into "blocks" $\mathcal{B}_i : i = 1, \ldots, k$ where $\mathcal{B}_i$ contains the keys in $\{\ell(i-1)+1, \ell(i-1)+2, \ldots, \ell i\}$, so we have $|\mathcal{B}_i| = \ell$ and $\bigcup_{i=1}^k \mathcal{B}_i = [n]$. The sequence $S_k$ consists of an interleaving of an increasing traversal of each block.

▶ **Lemma 24.** $\mathsf{F}^k(S_k) = O(n)$.

**Proof.** The main idea is to use each finger to serve only the keys inside blocks and to use a separate finger for each block. (recall that there are $k$ blocks and $k$ fingers.) We create a reference tree $T$ and argue that $\mathsf{F}^k_T(S_k) = O(n)$. Let $T_0$ be a BST of height $O(\log k)$ and with $k$ leaves. Each leaf of $T_0$ corresponds to the keys $\left\{\ell\cdot(i-1)+\frac{1}{2}\right\}_{i=1}^k$. The non-leafs of $T_0$ are assigned arbitrary fractional keys that are consistent with the BST properties. For

each $i \in [k]$, path $P_i$ is defined as a BST with key $\ell \cdot (i-1)+1$ (i.e. the smallest key in block $\mathcal{B}_i$) at the root, where for each $j = 0, \ldots, (\ell-1)$, the key $\ell(i-1)+j$ has $\ell(i-1)+(j+1)$ as its only (right) child. The final tree $T$ is obtained by hanging each path $P_i$ as a left subtree of a leaf $\ell \cdot (i-1) + \frac{1}{2}$. The $k$-finger strategy is simple: The $i$-th finger only takes care of the elements in block $\mathcal{B}_i$. The cost for the first access in block $\mathcal{B}_i$ is $O(\log k)$, and afterwards, the cost is only $O(1)$ per access. So the total access cost is $O(\frac{n}{k} \log k + n) = O(n)$. ◀

The rest of this section is devoted to proving the following:

▶ **Theorem 25.** $\mathsf{F}^{k-1}(S_k) = \Omega(\frac{n}{k} \log(n/k))$

Let $T$ be an arbitrary reference tree. We argue that $\mathsf{F}_T^{k-1}(S_k) = \Omega(\frac{n}{k} \log(n/k))$.

A *finger configuration* $\vec{f} = (f(1), \ldots, f(k-1)) \in [n]^{k-1}$ specifies to which keys the fingers are currently pointing. Any finger strategy can be described by a sequence $\vec{f_1}, \ldots, \vec{f_n}$, where $\vec{f_t}$ is the configuration after element $s_t$ is accessed. As before, we assume w.l.o.g. the following lazy update strategy:

▶ **Lemma 26.** *For each time $t$, the configurations $\vec{f_t}$ and $\vec{f_{t+1}}$ differ at exactly one position. In other words, we only move the finger that is used to access $s_{t+1}$.*

We view the input sequence $S_k$ as having $\ell$ phases: The first phase contains the subsequence $1, \ell+1, \ldots, \ell(k-1)+1$, and so on. Each phase is a subsequence of length $k$ that accesses keys starting in block $bset_1$ and so on, until the block $\mathcal{B}_k$.

▶ **Lemma 27.** *For each phase $p \in \{1, \ldots, \ell\}$, there is a time $t \in [(p-1)k+1, p \cdot k]$ such that $s_t$ is accessed by finger $j$ such that $f_{t-1}(j)$ and $f_t(j)$ are in different blocks, and $f_{t-1}(j) < f_t(j)$. That is, this finger moves to the block $\mathcal{B}_b$, $b = t \mod k$, from some block $\mathcal{B}_{b'}$, where $b' < b$, in order to serve $s_t$.*

**Proof.** Consider the accesses in blocks $\mathcal{B}_1, \ldots, \mathcal{B}_k$ in order. After the access in $\mathcal{B}_1$, we have a finger in $\mathcal{B}_1$ and hence at most $k-2$ fingers in blocks $\mathcal{B}_2, \ldots, \mathcal{B}_k$. If the access to $\mathcal{B}_2$ is served by a finger being in block $\mathcal{B}_1$ before the acces, we are done. Otherwise, it is server by a finger being in blocks $\mathcal{B}_{\geq 2}$ before the access. Then we have two fingers in blocks $\mathcal{B}_{\leq 2}$ after the access and at most $k-3$ fingers in blocks $\mathcal{B}_{\geq 3}$. Continuing in this way, we will find the desired access. ◀

For each phase $p \in [\ell]$, let $t_p$ denote the time for which such a finger moves across the blocks from left to right; if they move more than once, we choose $t_p$ arbitrarily. Let $J = \{t_p\}_{p=1}^{\ell}$. For each finger $j \in [k-1]$, each block $i \in [k]$ and block $i' \in [k] : i < i'$, let $J(j,i,i')$ be the set containing the time $t$ for which finger $f(j)$ is moved from block $\mathcal{B}_i$ to block $\mathcal{B}_{i'}$ to access $s_t$. Let $c(j,i,i') = |J(j,i,i')|$. Notice that $\sum_{j,i,i'} c(j,i,i') = \frac{n}{k} = \ell$, due to the lemma. Let $P(j,i,i')$ denote the phases $p$ for which $t_p \in J(j,i,i')$.

▶ **Lemma 28.** $\sum_{j,i,i' : c(j,i,i') \geq 16} c(j,i,i') \geq n/2k$ *if* $n = \Omega(k^4)$.

**Proof.** There are only at most $k^3$ triples $(j,i,i')$, so the terms for which $c(j,i,i') < 16$ contribute to the sum at most $16k^3$. This means that the sum of the remaining is at least $n/k - 16k^3 \geq n/2k$ if $n$ satisfies $n = \Omega(k^4)$. ◀

From now on, we consider the sets $J'$ and $J'(j,i,i')$ that only concern those $c(j,i,i')$ with $c(j,i,i') \geq 16$ instead.

▶ **Lemma 29.** *There is a constant $\eta > 0$ such that the total access cost during the phases $P(j,i,i')$ is at least $\eta c(j,i,i') \log c(j,i,i')$.*

Once we have this lemma, everything is done. Since the function $g(x) = x \log x$ is convex, we apply Jensen's inequality to obtain:

$$\frac{1}{|J'|} \sum_{j,i,i'} \eta c(j,i,i') \log c(j,i,i') \geq \eta \cdot \frac{n}{2k|J'|} \cdot \log(n/2k|J'|).$$

Note that the left side is the term $\mathbb{E}[g(x)]$, while the right side is $g(\mathbb{E}(x))$. Therefore, the total access cost is at least $\frac{\eta n}{8k} \log(n/2k)$. We now prove the lemma.

**Proof of Lemma 29.** We recall that, in the phases $P(j,i,i')$, the finger-$j$ moves from block $\mathcal{B}_i$ to $\mathcal{B}_{i'}$ to serve the request at corresponding time. For simplicity of notation, we use $\tilde{J}$ and $C$ to denote $J(j,i,i')$ and $c(j,i,i')$ respectively. Also, we use $\tilde{f}$ to denote the finger-$j$. For each $t \in \tilde{J}$, let $a_t \in \mathcal{B}_i$ be the key for which the finger $\tilde{f}$ moves from $a_t$ to $s_t$ when accessing $s_t \in \mathcal{B}_{i'}$. Let $\tilde{J} = \{t_1, \ldots, t_C\}$ such that $a_{t_1} < a_{t_2} < \ldots < a_{t_C}$. Let $R$ be the lowest common ancestor in $T$ of keys in $[a_{t_{\lfloor C/2 \rfloor}+1}, a_{t_C}]$.

▶ **Lemma 30.** *For each $r \in \{1, \ldots, \lfloor C/2 \rfloor\}$, the access cost of $s_{t_r}$ and $s_{t_{C-r}}$ is together at least $\min\{d_T(R, s_{t_r}), d_T(R, s_{t_{C-r}})\}$.*

**Proof.** Let $u_r$ be the lowest common ancestor between $a_{t_r}$ and $s_{t_r}$. Then the cost of accessing $s_{t_r}$ is at least $d_T(u_r, s_{t_r})$. If $s_{t_r}$ is in the subtree rooted at $R$, then $u_r$ must be an ancestor of $R$ (because $a_{t_r} < a_{t_{\lfloor C/2 \rfloor}} < a_{t_C} < s_{t_r}$) and hence $d_T(u_r, s_{t_r}) \geq d_T(R, s_{t_r})$. Thus the cost it at least $d_T(R, s_{t_r})$. Otherwise, we know that $s_{t_r}$ is outside of the subtree rooted at $R$, and so is $s_{t_{C-r}}$. On the other hand, $a_{t_{C-r}}$ is in such subtree, so moving the finger from $a_{t_{C-r}}$ to $s_{t_{C-r}}$ must touch $R$, therefore costing at least $d_T(R, s_{t_{C-r}})$. ◀

Lemma 30 implies that, for each $r = 1, \ldots, \lfloor C/2 \rfloor$, we pay the distance between some element $v_r \in \{s_{t_r}, s_{t_{C-r}}\}$ to $R$. The total such costs would be $\sum_r d_T(R, v_r)$. Applying the fact that (i) $v_r$'s are different and (ii) there are at most $3^d$ vertices at distance $d$ from a vertex $R$, we conclude that this sum is at least $\sum_r d_T(R, v_r) \geq \Omega(C \log C)$. ◀

## F Working set and $k$-finger bounds are incomparable

We show the following theorem.

▶ **Theorem 31.**
**(1)** *There exists a sequence $S$ such that $\mathsf{WS}(S) = o(\mathsf{F}^k(S))$, and*
**(2)** *There exists a sequence $S'$ such that $\mathsf{F}^k(S') = o(\mathsf{WS}(S'))$.*

The sequence $S'$ above is straightforward: For $k = 1$, just consider the sequential access $1, \ldots, n$ repeated $m/n$ times. For $m$ large enough, the working set bound is $\Omega(m \log n)$. However, if we start with the finger on the root of the tree which is just a path, then the lazy finger bound is $O(m)$. The $k$-finger bound is always less than lazy finger bound, so this sequence works for the second part of the theorem.

The existence of the sequence $S$ is slightly more involved (the special case for $k = 1$ was proved in [8]), and is guaranteed by the following theorem, the proof of which comprises the remainder of this section.

▶ **Theorem 32.** *For all $k = O(n^{1/2-\epsilon})$, there exists a sequence $S$ of length $m$ such that $\mathsf{WS}(S) = O(m \log k)$ whereas $F^k(S) = \Omega(m \log(n/k))$.*

We construct a random sequence $S$ and show that while $\mathsf{WS}(S) = O(m \log k)$ with probability one, the probability that there exists a tree $\mathcal{T}$ such that $\mathsf{F}_T^k(S) \leq cm \log_3(n/k)$ is less than $1/2$ for some constant $c < 1$. This implies the existence of a sequence $S$ such that for all trees $\mathcal{T}$, $\mathsf{F}_T^k(S) = \Omega(m \log(n/k))$.

The sequence is as follows. We have $Y$ phases. In each phase we select $2k$ elements $R_i = \{r_j^i\}_{j=1}^{2k}$ uniformly at random from $[n]$. We order them arbitrarily in a sequence $S_i$, and access $[S_i]^{X/2k}$ (access $S_i$ $X/2k$ times). The final sequence $S$ is a concatenation of the sequences $[S_i]^{X/2k}$ for $1 \leq i \leq Y$. Each phase has $X$ accesses, for a total of $m = XY$ accesses overall. We will choose $X$ and $Y$ appropriately later.

**Working set bound.** One easily observes that $\mathsf{WS}(S) = O(Y(2k \log n + (X - 2k) \log(2k)))$, because after the first $2k$ accesses in a phase, the working set is always of size $2k$. We choose $X$ such that the second term dominates the first, say $X \geq 5k \frac{\log n}{\log 2k}$. We then have that the working set bound is $O(XY \log k) = O(m \log k)$, with probability one.

***k*-finger bound.** Fix a BST $\mathcal{T}$. We classify the selection of the set $R_i$ as being $d$-good for $\mathcal{T}$ if there exists a pair $r_j^i, r_\ell^i \in R_i$ such that their distance in $\mathcal{T}$ is less than $d$. The following lemma bounds the probability of a random selection being $d$-good for $\mathcal{T}$.

▶ **Lemma 33.** *Let $\mathcal{T}$ be any BST. The probability that $R_i$ is $d$-good for $\mathcal{T}$ is at most $8k^2 3^d/n$.*

**Proof.** We may assume $8k^2 3^d/n < 1$ as the claim is void otherwise. We compute the probability that a selection $R_i$ is not $d$-good first. This happens if and only if the balls of radius $d$ around every element $r_j^i$ are disjoint. The volume of such a ball is at most $3^d$, so we can bound this probability as

$$\begin{aligned}
P[R_i \text{ is not d-good for } \mathcal{T}] &= \Pi_{i=1}^{2k-1} \left(1 - \frac{i3^d}{n}\right) \\
&\geq \left(1 - \frac{2k3^d}{n}\right)^{2k} \\
\Rightarrow P[R_i \text{ is d-good for } \mathcal{T}] &\leq 1 - \left(1 - \frac{2k3^d}{n}\right)^{2k} \\
&= 1 - \exp\left(2k \ln\left(1 - \frac{2k3^d}{n}\right)\right) \\
&\leq 1 - \exp\left(-8k^2 3^d/n\right) \\
&\leq 8k^2 3^d/n,
\end{aligned}$$

where the last two inequalities follow from $\ln(1-x) > -2x$ for $x \leq 1/2$ (note that $8k^2 3^d/n < 1$ implies $2k3^d/n \leq 1/2$) and $e^x > 1 + x$, respectively. ◀

Observe that if $R_i$ is not $d$-good, then the $k$-finger bound of the access sequence $[S_i]^{X/2k}$ is $\Omega(d(X - k)) = \Omega(dX)$. This is because in every occurrence of $S_i$, there will be some $k$ elements out of the $2k$ total that will be outside the $d$-radius balls centered at the current $k$ fingers.

We call the entire sequence $S$ $d$-good for $\mathcal{T}$ if at least half of the sets $R_i$ are $d$-good for $\mathcal{T}$. Thus if $S$ is not $d$-good, then $\mathsf{F}_{\mathcal{T}}^k(S) = \Omega(XYd)$.

▶ **Lemma 34.** *$P[S$ is $d$-good for $\mathcal{T}] \leq \left(\frac{32k^2 3^d}{n}\right)^{Y/2}$.*

**Proof.** By the previous lemma and by definition of goodness of $S$, we have that

$$P[S \text{ is d-good for } \mathcal{T}] \leq \binom{Y}{Y/2}\left(\frac{8k^2 3^d}{n}\right)^{Y/2} \leq 4^{Y/2}\left(\frac{8k^2 3^d}{n}\right)^{Y/2} = \left(\frac{32k^2 3^d}{n}\right)^{Y/2}. \quad \blacktriangleleft$$

The theorem now follows easily. Taking a union bound over all BSTs on $[n]$, we have

$$P[S \text{ is d-good for some BST } \mathcal{T}] \leq 4^n \left(\frac{32k^2 3^d}{n}\right)^{Y/2}.$$

Now set $Y = 2n$. We have that

$$P[\exists \text{ a BST } \mathcal{T} : \mathsf{F}^k_{\mathcal{T}}(S) \leq md/4] \leq 4^n \left(\frac{32k^2 3^d}{n}\right)^{n}.$$

Putting $d = \log_3 \frac{n}{256k^2}$ gives that for some constant $c < 1$,

$$P[\exists \text{ a BST } \mathcal{T} : \mathsf{F}^k_{\mathcal{T}}(S) \leq c(m \log(n/k))] \leq 4^n \left(\frac{32k^2 3^d}{n}\right)^{n} = 1/2$$

which implies that with probability at least $1/2$ one of the sequences in our random construction will have $k$-finger bound that is $\Omega(m \log(n/k))$. The working set bound is always $O(m \log k)$. This establishes the theorem.