New Binary Search Tree Bounds via Geometric Inversions

Parinya Chalermsook
Aalto University, Finland
chalermsook@gmail.com

Wanchote Po Jiamjitrak
Aalto University, Finland
wanchotej@gmail.com

Abstract

The long-standing dynamic optimality conjecture postulates the existence of a dynamic binary search tree (BST) that is $O(1)$-competitive to all other dynamic BSTs. Despite attempts from many groups of researchers, we believe the conjecture is still far-fetched. One of the main reasons is the lack of the "right" potential functions for the problem: existing results that prove various consequences of dynamic optimality rely on very different potential function techniques, while proving dynamic optimality requires a single potential function that can be used to derive all these consequences. In this paper, we propose a new potential function, that we call *extended (geometric) inversion*. Inversion is arguably the most natural potential function principle that has been used in competitive analysis but has never been used in the context of BSTs. We use our potential function to derive new results, as well as streamlining/strengthening existing results.

First, we show that a broad class of BST algorithms (including Greedy and Splay) are $O(1)$-competitive to Move-to-Root algorithm and therefore have simulation embedding property – a new BST property that was recently introduced and studied by Levy and Tarjan (SODA 2019). This result, besides substantially expanding the list of BST algorithms having this property, gives the first potential function proof of the simulation embedding property for BSTs (thus unifying apparently different kinds of results). Moreover, our analysis is the first where the costs of two dynamic binary search trees are compared against each other directly and systematically. Secondly, we use our new potential function to unify and strengthen known BST bounds, e.g., showing that *Greedy* satisfies the weighted dynamic finger property within a multiplicative factor of $(5 + o(1))$.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases Binary Search Tree, Potential Function, Inversion, Data Structures, Online Algorithms

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.28

Funding *Parinya Chalermsook*: Part of this work was done while Parinya was visiting the Simons Institute for the Theory of Computing. It was partially supported by the DIMACS/Simons Collaboration on Bridging Continuous and Discrete Optimization through NSF grant #CCF-1740425. This project has received funding from European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 759557). Parinya is also funded by Academy of Finland Research Fellowship, under grant number 310415.

Acknowledgements We would like to thank Thatchaphol Saranurak for his contributions in the early stage of this paper and for many insightful discussions. We also thank anonymous reviewers for many detailed comments and suggestions.

28th Annual European Symposium on Algorithms (ESA 2020).
Editors: Fabrizio Grandoni, Grzegorz Herman, and Peter Sanders; Article No. 28; pp. 28:1–28:16

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

1 Introduction

The dynamic optimality conjecture [23] is among the most fundamental problems in data structures. The conjecture postulates the existence of an online[1] and dynamic[2] binary search tree (BST) that is $O(1)$-competitive (or simply *competitive*) to the optimal offline BST algorithm. So far, the main candidates for being dynamically optimal are Splay and Greedy[17, 12, 18] (a.k.a. *Greedy Future* and *Greedy ASS*) since they possess many desirable properties that are consequences of dynamic optimality [5, 14, 11, 10, 19], although the best known competitive ratio of $O(\log \log n)$ is given by Tango trees [13]. Despite various attempts from many groups of researchers, the conjecture remains elusive. There are several observed reasons that make the conjecture long-standing, and one such reason (that this work tries to address) is the difficulty of comparing the behavior of two dynamic BSTs (see, for instance, [16]), which can be attributed to the lack of "generic" and "intuitive" potential function in this context: The simplest bound, static optimality, was derived via sum-of-logs [23, 6], and Splay's dynamic finger (which extends static optimality) uses sophisticated potential function that in some way extends sum-of-logs [10, 11]; Greedy's weighted dynamic finger [14] relies on very different potential function that neither seems related to sum-of-logs nor Splay's dynamic finger; the recent simulation embedding properties [16, 21] as well as best known bound for Splay's deque property [19, 20] do not even use potential function. Finally, none of these techniques was used to prove that Greedy or Splay is $o(\log n)$-competitive. Given this state of the art, it is relatively unclear which potential function should be used/extended for proving dynamic optimality. This paper is inspired by the following question:

> Is there a natural, generic potential function technique that allows us to compare the cost of two dynamic BSTs in a modular way?

1.1 Our Contribution

Our main conceptual contribution is a new potential function that, in our opinion, seems to be the right way to handle binary search trees. Our idea is inspired by *inversions*[3], which are arguably the most natural potential function for the purpose of analyzing online algorithms (see, for instance, [2, 22, 1] in the context of list update and [9] in the context of the k-server problem). We illustrate the power of our techniques in two directions.

Let us first introduce some notation before stating our results. We consider keys in $[n] = \{1, 2, \ldots, n\}$ and access sequence $X = (x_1, x_2, \ldots, x_m) \in [n]^m$. For an online dynamic BST algorithm $\mathcal{A}$, denoted by $cost_{\mathcal{A}}(X)$ the cost of serving sequence X using algorithm $\mathcal{A}$. We illustrate the power of our new concept in two ways.

First contribution: MTR-Competitiveness and Simulation Embedding

Recently, Levy and Tarjan [16] (and independently Russo [21]) noted the difficulty of comparing two dynamic BSTs and proposed an alternative path towards dynamic optimality. They rephrase dynamic optimality as two intrinsic properties (which they call, *simulation*

[1] A BST is online if an input sequence is revealed one at a time, i.e. the request for x_t appears at time t.
[2] A binary search tree is dynamic if it is allowed to change its form after each access, paying the cost of pointer movements.
[3] In general, the inversion potential function (or its generalization to "distance potential") measures the difference between the algorithm and the optimal, so it is suitable for analyzing the case when the optimal can change.

embeddings and *approximately monotone*) of an algorithm. Roughly speaking, a BST has simulation embeddings if it can "simulate"[4] any other BST algorithm, and it is approximately monotone if the cost of running the algorithm on an input sequence X is asymptotically at least the cost of running it on an arbitrary subsequence of X. An algorithm is $O(1)$-competitive if and only if it has both properties. They argue that Splay trees satisfy simulation embeddings and outlined a plan to prove that Splay trees are approximately monotone.

In this paper, we show that a broad class of algorithms (as defined in [6]) in fact satisfies simulation embeddings. This result is derived as a corollary to the following theorem.

▶ **Theorem 1.** *All BST algorithms (including Greedy and Splay) described in [6] are $O(1)$-competitive to Move-to-Root.*

Move-to-Root (MTR) is a classical BST algorithm that always rotates the requested key up until it becomes the root of the tree. It is known to be sub-optimal but not subsumed by any existing BST bounds (such as working set [23], lazy finger [14], pattern avoidance [5], or multiple fingers [8]). See discussion in the full version. Interestingly, so far no dynamic BSTs have been shown to be competitive even to this simplest dynamic algorithm.

▶ **Corollary 2.** *Let $\mathcal{A}$ be any BST algorithm according to [6]. Then, $\mathcal{A}$ has simulation embedding property. Therefore, it is dynamically optimal if and only if $\mathcal{A}$ is approximately monotone.*

Corollary 2 gives the first potential function proof of simulation property of any BST algorithm, therefore unifying the classical potential function techniques with the new attempt by Levy, Tarjan and Russo. The fact that infinitely many BST algorithms (that have simple description) have simulation embedding property can be interpreted in many ways. For an optimist, this could give us an access to a large design toolbox for studying the Levy-Tarjan approach: Instead of focusing on Splay or Greedy, we have the freedom to seek an algorithm that is approximately monotone by fine-tuning.

Another interesting aspect of MTR-competitiveness is perhaps a conceptual resemblance between Move-to-Root and the second Wilber bound [24] which is believed to be stronger than the first Wilber bound but so far no algorithm has ever exploited such bound[5]. Being able to charge the cost of an algorithm to Move-to-Root is a very first step towards this direction. (Informally, the second Wilber bound is equal to *crossing* Move-to-Root, see [16] for a more detailed discussion).

Second contribution: Streamlining known bounds

Now, we discuss how to use our potential function to streamline the BST bounds. Our second main result is an improved bound for the lazy finger property of Greedy. For a sequence $X = (x_1, \ldots, x_m)$ and a fixed BST R, denoted by $\mathsf{LF}_R(X) = \sum_t d_R(x_t, x_{t+1})$ where $d_R(a, b)$ denotes the number of edges on the unique path in R from a to b. Let $\mathsf{LF}(X) := \min_R \mathsf{LF}_R(X)$, we say that an algorithm $\mathcal{A}$ has lazy finger property if $cost_{\mathcal{A}}(X) \leqslant O(\mathsf{LF}(X))$. For any sequence X, denote by $G(X)$ the cost of Greedy in the geometric view[6]. Iacono and Langerman showed that $G(X) \leqslant C' \cdot (\mathsf{LF}(X) + m)$ where C' is around 50 (A quick glance at

[4] The definition of simulation is quite technical and we will only discuss this formally later.

[5] The best known competitive ratio is due to Tango Trees and its variants [13] which charge the cost to the first Wilber bound, which is provably insufficient for dynamic optimality [15, 4]

[6] Recall that, when turning the Greedy algorithm into a standard BST view, there is a constant factor blowup in the cost, that is, $cost_{Greedy}(X)$ (in the tree view) is at most $O(G(X))$.

their paper would show that the value of C' is 24, but there is also another multiplicative factor hidden in converting the result from "leaf-oriented" tree[7] setting to the BST setting). We show the following.

▶ **Theorem 3.** *For each access sequence X, $G(X) \leqslant 5 \cdot \mathsf{LF}(X) + O(m+n)$.*

In particular, when the input sequence is sufficiently costly, this bound converges to a multiplicative factor of 5 in front of the lazy finger term. We highlight that the interesting aspect of this result is not the improvement of constant but rather (1) the fact that our lazy finger proof directly and intuitively extends the proof of static optimality, and (2) the fact that we are able to charge the cost directly to the reference BST R instead of a leaf-oriented tree, in contrast to [14]. We hope that these two points open up natural new paths to adapt our techniques to handle stronger BST bounds where leaf-oriented settings become unnatural (e.g. BST rotation is less natural in the leaf-oriented setting).

Conclusion & Open Problems. We introduce a new potential function that allows us to, for the first time, compare two dynamic BSTs directly in a systematic way. Moreover, to our knowledge, ours is the first potential function in the BST context that uses a natural concept of inversions. We show many applications of our potential function. We note that, once the potential function is formally defined, the proofs of our results do not require any ground-breaking ideas but rather a careful adaptation of existing proofs [23, 6, 14] to our potential function.

Though the most intriguing open question is to prove dynamic optimality, we feel that it is still very far from the current understanding. We list here some open questions that we believe they are not overly far-fetched.

- **(i)** Can we use our method to analyze weighted dynamic finger (or even just dynamic finger) for Splay trees?
- **(ii)** Can we show that Greedy satisfies multi-finger properties as defined in [8]? We find this non-trivial even when there are two fingers.
- **(iii)** The property of being $O(\sqrt{\log n})$-competitive BST can be cast as a "local" BST property, e.g. see a survey paper [7]. Can we show that Greedy satisfies this property?

Further Related Work. Most prior works in this area focus on proving consequential properties of dynamic optimality. In particular, dynamic BSTs satisfy various forms of locality of reference properties that allow efficient access when the input sequence is in some way "local". For instance, the dynamic finger bound allows an efficient access when the access is close to the previous one (on average). It is often not so difficult to prove that these locality properties are satisfied by an optimal offline BST, so a candidate for optimal online BST must satisfy them as well. Proving that an online BST algorithm satisfies such properties has, however, been very challenging (for instance, Cole's proof [11, 10] of Splay's dynamic finger property spans 80 pages in total). Recently, a much stronger locality of reference bound, called Lazy Finger [3], was proved in a breakthrough result of Iacono and Langerman [14]. One way to view these locality of reference properties is as a "mildly dynamic" BST algorithm, i.e. each such property can be described by a restricted way of using the power of dynamic BST algorithms. Therefore, proving these bounds has naturally been seen as intermediate steps to study the dynamic optimality conjecture.

[7] A leaf-oriented binary search tree is one where all keys in $[n]$ are maintained as leafs, and each internal BST node is auxiliary (corresponding to a subset of leafs in the subtree under it).

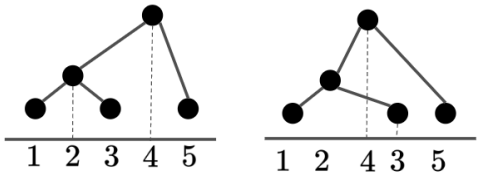


Figure 1 Examples of good (left) and bad (right) drawing of the same binary search tree.

2 Overview of Techniques

Let us recall the BST search model. The algorithm $\mathcal{A}$ maintains a binary search tree T on keys $[n]$, and when access $x_t \in [n]$ comes at time t, the cost incurred is equal to the depth of x_t in T. Elements on the search path of x_t (including x_t) are said to be *touched* by T. After that, the algorithm may adjust the shape of the tree, paying the cost which is equal to the number of keys that are involved in the adjustment. The total cost is then equal to the sum of search cost and adjustment cost. In all algorithms we consider, it suffices to analyze only the search cost (in particular, for BST algorithms that only change the search path, the update cost is at most a constant factor of the length of the search path. Therefore, such cost can be charged to the search cost). We follow this standard practice.

For convenience, we will abuse notation and use $\mathcal{A}$ to stand for both the BST algorithm and the state of BST at certain time.

2.1 Interval geometry for BST & Extended Inversion

Our potential function is defined based on geometry that requires correct drawing of BSTs. When drawing a BST, one should always use two rules: (1) imagine placing node containing key z on the plane at point p where $p.x = z$ (the x-coordinate is z). (2) If u is a parent of v, always draw u higher than v (See Figure 1)

Let $\mathcal{A}$ be a BST. For each key $z \in [n]$, let $I_{\mathcal{A}}(z)$ be the largest open interval in $(0, n+1)$ containing only all keys in the subtree of $\mathcal{A}$ rooted at z (in Figure 1 we have $I_{\mathcal{A}}(2) = (0, 4)$). More formally, $I_{\mathcal{A}}(z) = (\mathsf{left}_{\mathcal{A}}(z), \mathsf{right}_{\mathcal{A}}(z))$ where $\mathsf{left}_{\mathcal{A}}(z)$ is the nearest left ancestor of z (0 if not exist) and $\mathsf{right}_{\mathcal{A}}(z)$ is the nearest right ancestor of z ($n+1$ if not exist). When drawing the BST correctly, we have that the intervals $\{I_{\mathcal{A}}(z)\}_{z\in[n]}$ form a laminar family (that is, each pair of intervals is either disjoint or nested); see Figure 2.

▶ **Observation 4.** *If z' is a left child of z, then $I_{\mathcal{A}}(z') = (\mathit{left}_{\mathcal{A}}(z'), \mathit{right}_{\mathcal{A}}(z')) = (\mathit{left}_{\mathcal{A}}(z), z)$. If z' is a right child of z, then $I_{\mathcal{A}}(z') = (\mathit{left}_{\mathcal{A}}(z'), \mathit{right}_{\mathcal{A}}(z')) = (z, \mathit{right}_{\mathcal{A}}(z))$. Also, for any keys $y, z \in [n]$, we have $y \in I_{\mathcal{A}}(z)$ iff y is in the subtree rooted at z.*

Let $\mathcal{A}$ be an algorithm we want to analyze and $\mathcal{O}$ be an optimal algorithm. In this paper, we only consider $\mathcal{A}$ that changes only search path (this includes Greedy and Splay). Both $\mathcal{A}$ and $\mathcal{O}$ store the keys in $[n]$. We want to have a potential function that captures the "difference" between $\mathcal{A}$ and $\mathcal{O}$. The most natural scheme (which does not always work) often used in the context of online algorithm for this purpose is *inversion*.

▶ **Definition 5** (Inversion). *Let $z, \alpha \in [n]$. We say that z forms an* inversion *with α if $z \in I_{\mathcal{O}}(\alpha)$ and $\alpha \in I_{\mathcal{A}}(z)$.*

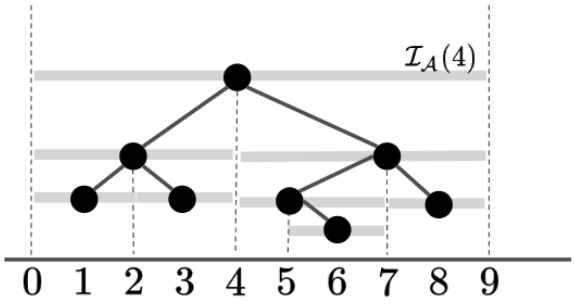


Figure 2 Example of intervals (in grey) for all keys. In this figure, $I_{\mathcal{A}}(4) = (0, 9)$ and $I_{\mathcal{A}}(5) = (4, 7)$.

Notice that an inversion is an indicator of α being in the subtree of z in one BST, but z is in the subtree of α in the other. See Figure 3. Unfortunately, we are unable to use this natural and intuitive scheme to prove any meaningful result. We will use *extended inversion* instead.

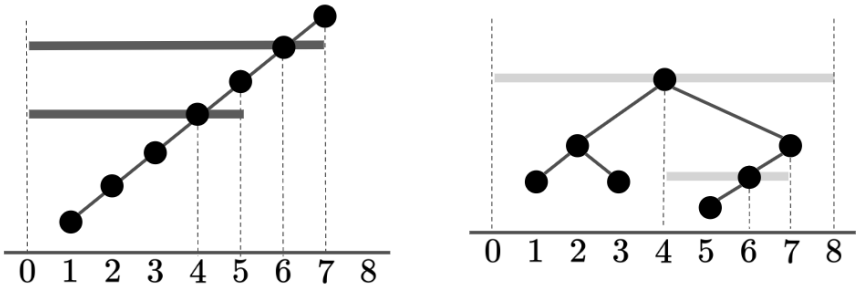


Figure 3 An example of inversion between the keys 4 and 6. The intervals of $\mathcal{A}$ are shown in dark grey and intervals of $\mathcal{O}$ in light grey. The left and right BSTs are $\mathcal{A}$ and $\mathcal{O}$ respectively.

For each interval in BST $\mathcal{O}$, $I_{\mathcal{O}}(\alpha)$, we define three **important points** of α in $\mathcal{O}$ as $\mathcal{P}_{\mathcal{O}}(\alpha) = \{\alpha, \mathsf{left}_{\mathcal{O}}(\alpha), \mathsf{right}_{\mathcal{O}}(\alpha)\}$.

▶ **Definition 6** (Extended Inversion). *Let $z, \alpha \in [n]$. We say that z forms an extended inversion with α if and only if $z \in I_{\mathcal{O}}(\alpha)$ and $\exists \beta \in \mathcal{P}_{\mathcal{O}}(\alpha)(\beta \in I_{\mathcal{A}}(z))$. See Figure 4.*

Unlike inversion, the notion of extended inversion is not symmetric. Clearly, if z forms an inversion with α, then z also forms an extended inversion with α.

Figure 4 The darker intervals are those intervals $I_{\mathcal{A}}(z)$, and the lighter ones are $I_{\mathcal{O}}(z)$. In the first figure from left, z forms an inversion with α. In the second, z forms an extended inversion with α but not an inversion. In the 3rd figure, there is no extended inversion, since $I_{\mathcal{A}}(z)$ does not contain any point in $\mathcal{P}_{\mathcal{O}}(\alpha)$. In the 4th figure, there is no extended inversion because $z \notin I_{\mathcal{O}}(\alpha)$.

2.2 Our potential function and its basic properties

All proofs in this paper build upon the following base function.

▶ **Definition 7** (Base Potential function). *Define the potential $\Phi = \Phi_{\mathcal{A},\mathcal{O}}$ at any state of execution of our algorithm $\mathcal{A}$ and optimal $\mathcal{O}$ as follows. Let $\Phi(z,\alpha) = 1$ if z forms an extended inversion with α; otherwise, $\Phi(z,\alpha) = 0$. The potential is defined as $||\Phi|| = \sum_{z,\alpha} \Phi(z,\alpha)$. This is the total number of extended inversions.*

We "visualize" our potential function value as a collection of "coins". Whenever $\Phi(z,\alpha) = 1$, one can imagine there is a coin of label z (or z-coin) placed at node α in $\mathcal{O}$ whenever z forms an extended inversion with α (see Figure 5). By definition, z always forms an extended inversion with itself, so there is always a z-coin at z. We use $\Phi(\bullet,\alpha) = \sum_{z\in[n]} \Phi(z,\alpha)$ to denote the total number of coins at α, and $\Phi(z,\bullet) = \sum_{\alpha\in[n]} \Phi(z,\alpha)$ the total number of z-coins. The coin interpretation will be used crucially in our analysis.

The main properties we would need are the following:

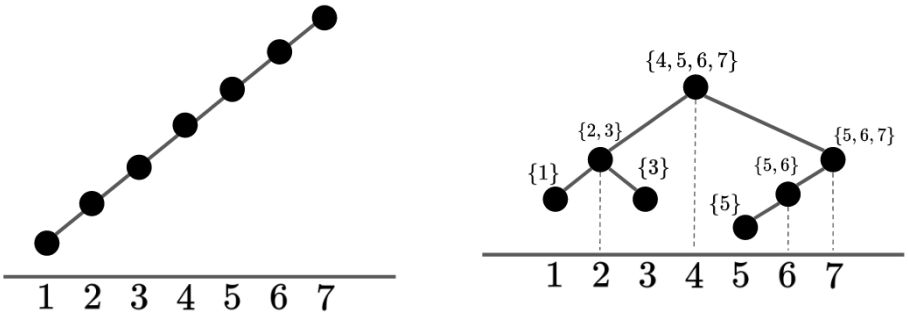


Figure 5 The left and right BSTs are $\mathcal{A}$ and $\mathcal{O}$ respectively. The set notation shown at each node is a collection of coins placed at that node. For instance, there are 5- and 6-coins at node 6. The path of 3-coins (according to Lemma 8) is a path containing nodes 3 and 2.

▶ **Lemma 8.** *[Upward path property] For each $z \in [n]$, the set of nodes having z-coin (that is, $Q_z = \{\alpha : \Phi(z,\alpha) = 1\}$) is a contiguous subpath of the path from z to the root of $\mathcal{O}$.*

Proof. Let $\alpha_1, \alpha_2 \in Q_z$. Since $z \in I_{\mathcal{O}}(\alpha_1) \cap I_{\mathcal{O}}(\alpha_2)$, we must have that $I_{\mathcal{O}}(\alpha_1) \subseteq I_{\mathcal{O}}(\alpha_2)$ (from laminar property). Since $z \in Q_z$, we have that Q_z is a subset of path from z to the root. Next, we argue that Q_z is connected. Suppose α' is on the path from z to the root such that $\alpha' \notin Q_z$. See Figure 6. We argue that the parent of α' (say α, where α' is the right child of α) is also not in Q_z: From Observation 4, $I_{\mathcal{O}}(\alpha') = (\alpha, \mathsf{right}_{\mathcal{A}}(\alpha))$. Recall that $\mathcal{P}_{\mathcal{O}}(\alpha) = \{\alpha, \mathsf{left}_{\mathcal{A}}(\alpha), \mathsf{right}_{\mathcal{A}}(\alpha)\}$. Since $\alpha' \notin Q_z$, we have that $I_{\mathcal{A}}(z)$ is completely contained in $(\alpha, \mathsf{right}_{\mathcal{O}}(\alpha))$. This means that it does not contain any point in $\mathcal{P}_{\mathcal{O}}(\alpha)$. ◀

By the above lemma, we view the potential function $\Phi(z,\bullet)$ as the coins on an upward path in $\mathcal{O}$ and Φ as a collection of upward paths.

▶ **Lemma 9.** *Consider an access to key $x \in [n]$. Let z be a key on the search path $\mathcal{S}_{\mathcal{A}}(x)$ of our algorithm. Then x forms an inversion with $LCA_{\mathcal{O}}(x,z)$. Hence, there is a z-coin at node $LCA_{\mathcal{O}}(x,z)$.*

Proof. Fix $z \in \mathcal{S}_{\mathcal{A}}(x)$. Recall that $x \in I_{\mathcal{A}}(z)$. Denote by $\ell = LCA_{\mathcal{O}}(x,z)$. Since ℓ is between x and z, $\ell \in I_{\mathcal{A}}(z)$. Since ℓ is an ancestor of z in $\mathcal{O}$, $z \in I_{\mathcal{O}}(\ell)$, and thus an inversion occurs between z and ℓ. ◀

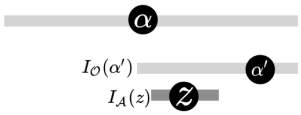


Figure 6 Illustration of the proof of Lemma 8.

The above lemma is the main advantage of the inversion principle: Each $z \in \mathcal{S}_{\mathcal{A}}(x)$ contributes $+1$ to the cost of $\mathcal{A}$ when accessing x. The lemma shows that such cost can be "deducted" from the z-coin at $LCA_{\mathcal{O}}(x, z)$.

▶ **Definition 10** (Canonical payment function)**.** *Consider algorithm $\mathcal{A}$ and optimal $\mathcal{O}$. The* ***canonical payment function*** *for accessing x is a function $\mathcal{CP} = \mathcal{CP}_{\mathcal{A},\mathcal{O},x}$ such that for each $z \in \mathcal{S}_{\mathcal{A}}(x)$, we have $\mathcal{CP}(z, LCA_{\mathcal{O}}(z, x)) = 1$. The function is 0 everywhere else (Figure 7).*

We will simply write $\mathcal{CP}$ instead of $\mathcal{CP}_{\mathcal{A},\mathcal{O},x}$ when it is clear from the context. Notice that, for each $z \in \mathcal{S}_{\mathcal{A}}(x), \mathcal{CP}(z, \alpha) = 1$ only if α is on the search path $\mathcal{S}_{\mathcal{O}}(x)$ of the optimal. From Lemma 9, we have that $\mathcal{CP}(z, \alpha) \leqslant \Phi(z, \alpha)$ for all $z, \alpha \in [n]$.

▶ **Observation 11.** *We have $\sum_{z,\alpha} \mathcal{CP}(z, \alpha) = |\mathcal{S}_{\mathcal{A}}(x)|$.*

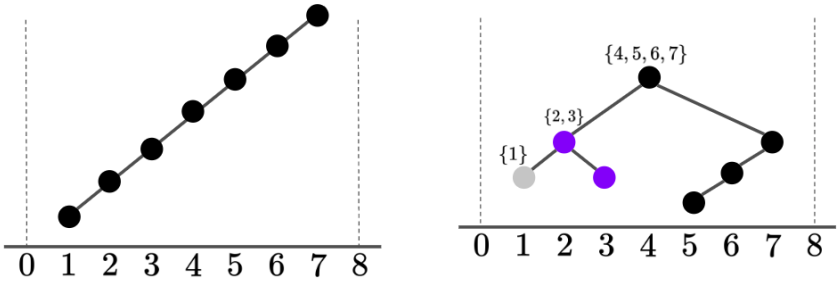


Figure 7 An example of coins in the support of $\mathcal{CP}$. In this example, when access 1, we touch every key in $\mathcal{A}$, but we touch only {1,2,4} in $\mathcal{O}$. Each $z \in \{4, 5, 6, 7\}$ has its coin at node 4 in the support of $\mathcal{CP}$. Each z in $\{2, 3\}$ has its coin at node 2. Finally, 1 has a coin at node 1.

2.3 Overview of our proofs

We first recall the basic ideas of potential function proofs.

▶ **Definition 12.** *We say that potential function Φ* ***proves*** *$\mathcal{A} \leqslant_\eta \mathcal{O}$ if for any state $\mathcal{A}$ and $\mathcal{O}$ of execution, when an access $x \in [n]$ arrives, we have*

$$(||\Phi_{\mathcal{A},\mathcal{O}}|| - ||\Phi_{\mathcal{A}',\mathcal{O}'}||) + \eta|\mathcal{S}_{\mathcal{O}}(x)| \geqslant |\mathcal{S}_{\mathcal{A}}(x)|$$

where $\mathcal{A}'$ and $\mathcal{O}'$ are the BSTs after the algorithm and the optimal update their trees.

The following claim follows from a standard potential function analysis. See, for instance, [23] for the formal proof.

▶ **Proposition 13.** *If potential function Φ proves $\mathcal{A} \leqslant_\eta \mathcal{O}$, then for any input $X \in [n]^m$, we have $cost_{\mathcal{A}}(X) \leqslant \eta \cdot cost_{\mathcal{O}}(X) + \Phi_{\max}$ where $\Phi_{\max}$ is the maximum value of $||\Phi_{\mathcal{A},\mathcal{O}}||$ over all possible states of execution of $\mathcal{A}$ and $\mathcal{O}$.*

Finally, since the size of support of $\mathcal{CP}$ is exactly $|\mathcal{S}_{\mathcal{A}}(x)|$, one can rewrite the condition in Definition 12 as $||\Phi_{\mathcal{A}',\mathcal{O}'}|| \leqslant ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}|| + \eta|\mathcal{S}_{\mathcal{O}}(x)|$. All our proofs follow this high-level idea: (i) Upper bound $||\Phi_{\mathcal{A}',\mathcal{O}}|| - ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}||$, and (ii) upper bound $||\Phi_{\mathcal{A}',\mathcal{O}'}|| - ||\Phi_{\mathcal{A}',\mathcal{O}}||$. The extended inversion potential function allows us to analyze Step (i) of various kinds of BST bounds in a modular way. The following is a key property.

▶ **Proposition 14.** *Consider algorithm $\mathcal{A}$ that, after access x arrives, updates the trees into $\mathcal{A}'$, while $\mathcal{O}$ does not change. Let $\widehat{\Phi} = \Phi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}$.*

(i) *For each $z \notin \mathcal{S}_{\mathcal{A}}(x)$ and $\alpha \in [n]$, we have $\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) \leqslant \widehat{\Phi}(z,\alpha)$.*

(ii) *For each $\alpha \notin \mathcal{S}_{\mathcal{O}}(x)$ and $z \in [n]$, we have $\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) \leqslant \widehat{\Phi}(z,\alpha)$.*

These imply that the potential change $||\Phi_{\mathcal{A}',\mathcal{O}}|| - ||\widehat{\Phi}||$ is upper bounded by the "change on search paths", that is, $||\Phi_{\mathcal{A}',\mathcal{O}}|| - ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}|| \leqslant \sum_{z\in\mathcal{S}_{\mathcal{A}}(x)} \sum_{\alpha\in\mathcal{S}_{\mathcal{O}}(x)} \left(\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) - \widehat{\Phi}(z,\alpha)\right)$

Proof. First, for each such $z \notin \mathcal{S}_{\mathcal{A}}(x)$, the intervals $I_{\mathcal{A}}(z) = I_{\mathcal{A}'}(z)$, so the number of extended inversions for them remains the same. Since $\mathcal{CP}(z,\alpha) = 0$, $\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) \leqslant \widehat{\Phi}(z,\alpha) = \Phi_{\mathcal{A},\mathcal{O}}(z,\alpha)$.

Now we prove the second item. Fix $\alpha \notin \mathcal{S}_{\mathcal{O}}(x)$. Due to the previous observation, only $z \in \mathcal{S}_{\mathcal{A}}(x)$ may create a new inversion. By Lemma 9, $\Phi_{\mathcal{A},\mathcal{O}}(z, LCA_{\mathcal{O}}(x,z)) = 1$. Since $LCA_{\mathcal{O}}(x,z)$ is an ancestor of α, by Lemma 8, $\Phi_{\mathcal{A},\mathcal{O}}(z,\alpha) = 1$. This means that there was already a z-coin at α. Since $\mathcal{CP}(z,\alpha) = 0$, $\Phi_{\mathcal{A}',\mathcal{O}}(z,\alpha) \leqslant \widehat{\Phi}(z,\alpha) = \Phi_{\mathcal{A},\mathcal{O}}(z,\alpha)$. ◀

All our proofs rely on this proposition to upper bound the change from $\mathcal{A}$ to $\mathcal{A}'$.

2.4 The First Showcase: Splay's Ziz-zig

We consider the Splay tree currently maintaining a path with n as a root and 1 as a leaf; and $\mathcal{O}$ does not change (that is, $\mathcal{O} = \mathcal{O}'$). Define the potential $\Psi(z,\alpha) = 2\Phi(z,\alpha)$ for all $z,\alpha \in [n]$ (there are 2 coins for each extended inversion). Let us consider the situation when an access $x = 1$ arrives. We will upper bound the change of potential function Ψ as follows:

$$||\Psi_{\mathcal{A}',\mathcal{O}}|| \leqslant ||\Psi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}|| + 6|\mathcal{S}_{\mathcal{O}}(x)|$$

Deducting $\mathcal{CP}$ from Ψ can be seen as removing some coins from the search path $\mathcal{S}_{\mathcal{O}}(x)$. Recall that in $\Psi - \mathcal{CP}$, for each z on the search path of $\mathcal{A}$, we had removed one z-coin at $LCA(z,x)$. The proof has two steps. In the first step, we move the coins around from "bad" to "good" keys. Refer to Figure 8 for the shape of $\mathcal{A}'$. For simplicity assume that n is odd. Except for the root, we pair i with $i+1$ for $i = 2,4,\ldots,n-1$. We say that the even keys are good and the odd keys are bad. For each i, we turn one $(i+1)$-coin at $LCA(i+1,x)$ into an i-coin at $LCA(i,x)$. Let $\widehat{\Psi}$ be the function after moving the coins, so $||\widehat{\Psi}|| = ||\Psi_{\mathcal{A},\mathcal{O}} - \mathcal{CP}||$.

▶ **Lemma 15.** *We have $||\Psi_{\mathcal{A}',\mathcal{O}}|| - ||\widehat{\Psi}|| \leqslant O(|\mathcal{S}_{\mathcal{O}}(x)|)$. (In other words, we can add at most $O(|\mathcal{S}_{\mathcal{O}}|)$ to achieve the coin level of $\Psi_{\mathcal{A}',\mathcal{O}}$.)*

The rest of this section is devoted to proving the lemma. Denote $\Psi_{\mathcal{A}',\mathcal{O}}$ by Ψ' for convenience. From Proposition 14, we have $||\Psi_{\mathcal{A}',\mathcal{O}}|| - ||\widehat{\Psi}|| \leqslant \sum_{z\in\mathcal{S}_{\mathcal{A}}(x)} \sum_{\alpha\in\mathcal{S}_{\mathcal{O}}(x)} \left(\Psi'(z,\alpha) - \widehat{\Psi}(z,\alpha)\right)$. The following claim therefore finishes the proof.

▷ Claim 16. We have $\sum_{z\in\mathcal{S}_{\mathcal{A}}(x)} \sum_{\alpha\in\mathcal{S}_{\mathcal{O}}(x)} \left(\Psi'(z,\alpha) - \widehat{\Psi}(z,\alpha)\right) \leqslant 14|\mathcal{S}_{\mathcal{O}}(x)|$.

Proof. Refer to Figure 8 that illustrate the shape of $\mathcal{A}'$ (Splay) after the access of x. For $z = 1$, we use the crude bound of $\sum_{\alpha\in\mathcal{S}_{\mathcal{O}}(x)} \left(\Psi'(1,\alpha) - \widehat{\Psi}(1,\alpha)\right) \leqslant 2|\mathcal{S}_{\mathcal{O}}(x)|$. For $z \geqslant 2$, we divide the analysis into two cases:

(i) new extended inversions, i.e. $\Psi'(z,\alpha) = 1$ and $\widehat{\Psi}(z,\alpha) = \Psi_{\mathcal{A},\mathcal{O}}(z,\alpha) = 0$

(ii) existing extended inversions that pay their coins out, i.e. $\Psi'(z,\alpha) = 1, \widehat{\Psi}(z,\alpha) = 0$ and $\Psi(z,\alpha) = 1$.

We upper bound the potential increase by the notion of *witness intervals*. For the first case, a new extended inversion can appear in $\Psi'(z,\alpha)$ due to the fact that the interval $I_{\mathcal{A}'}(z)$ becomes larger than $I_{\mathcal{A}}(z)$. In this case, we define $J(z,\alpha) = I_{\mathcal{A}'}(z) \backslash I_{\mathcal{A}}(z)$ as a **witness** for the new inversion (z,α). From Figure 8, these witness intervals (drawn in red) are non-overlapping. Since there are 3 important points for each $\alpha \in \mathcal{S}_{\mathcal{O}}(x)$ and each point can be in at most one witness interval, there are at most $3|\mathcal{S}_{\mathcal{O}}(x)|$ witness intervals. Each witness interval requires 2 coins, so we need at most $6|\mathcal{S}_{\mathcal{O}}(x)|$ coins in total.

For the second case, this happened due to z being a bad key that pays its coin out of canonical payment (the good ones received their coins back from the transfer in the first step). In this case, we say that $J(z,\alpha) = I_{\mathcal{A}'}(z)$ is a witness for the inversion (z,α). From Figure 8, the witness intervals of this type (drawn in blue) are disjoint. Therefore, with the same argument as in the first case, we need at most $6|\mathcal{S}_{\mathcal{O}}(x)|$ coins in total. ◁

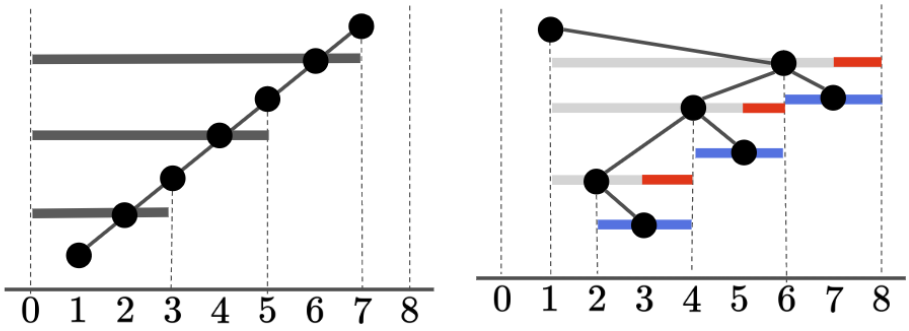


Figure 8 A BST before (left) and after (right) splaying $x = 1$. After splaying, intervals of odd keys (except access key) are disjoint. The red and blue intervals are the witnesses as in Claim 16.

There are two remarks about this proof. First, this proof only gives an analysis of a zig-zig case in Splay trees. A full Splay trees analysis can be found as a special case of our Theorem 1 whose proof is deferred to the full version. Second, if $\mathcal{O}$ is not a static tree, we add one more step to the analysis, that is, upper bounding the change due to updating $\mathcal{O}$; the second step for MTR-competitiveness results is simple, while for lazy finger, the second step is more involved, relying on a modified concept of extended inversions and on some ideas from [14].

2.5 Geometric Inversion for Geometric BST Algorithms

In order to use our scheme, all algorithms must be described in terms of *intervals*. In the previous section, we already explained how BST algorithms $\mathcal{A}$ in the tree view can be described as intervals $I_{\mathcal{A}}(z)$. In this section, we explain how to do the same for BST algorithms in the geometric view. As discussed in [12], the tree and geometric views are equivalent up to constant factor in the competitive ratios.

Let $[n]$ be the set of keys. An access sequence of binary search trees (BST) is specified as searching for key $x_t \in [n]$ at time $t = 1, \ldots, m$. Given an access sequence $X = (x_1, \ldots, x_m) \in [n]^m$, we view X as points $\mathcal{Q}_X = \{(x_t, t) : t = 1, \ldots, m\}$ in the plane where t-th access appears on row t (starting from bottom). We abuse the notation and simply write $\mathcal{Q}_X$ as X.

For $\mathbf{p},\mathbf{q} \in \mathbb{R}^2$, define $\square_{\mathbf{p},\mathbf{q}}$ as the set of integral points in the minimally closed rectangular area defined by $\mathbf{p}$ and $\mathbf{q}$. Let $Z \subseteq \mathbb{R}^2$, we say that such the rectangle $\square_{\mathbf{p},\mathbf{q}}$ is *empty* in Z (or Z-empty) if $Z \cap \square_{\mathbf{p},\mathbf{q}}$ contains no other point than $\mathbf{p}$ or $\mathbf{q}$. Also, the point set Z is *arborally satisfied* if for every pair $\mathbf{p}, \mathbf{q} \in Z$, the rectangle $\square_{\mathbf{p},\mathbf{q}}$ is not Z-empty.

Geometric BST problem (MinASS)

Let $X_{\leqslant t}$ denote the set of points in X in rows t and below. In the MinASS problem, given an input X of points, we are interested in computing a superset $Y \supseteq X$ such that Y is arborally satisfied, while minimizing $|Y|$. In the online version, at any time $t = 1, 2, \ldots, m$, a point in X arrives on the t-th row, and the algorithm must produces a feasible solution $Y_{\leqslant t}$ at each time t by adding points on the t^{th} row. Points in Y are said to be *touched* by the algorithm $\mathcal{A}$. We denote a set of touched keys at time t by $\mathcal{S}_{\mathcal{A}}(x_t)$ (same notation as search path of x_t in the tree view).

▶ **Theorem 17** (Demaine et al. [12]). *Let $\mathcal{A}$ be an algorithm for the MinASS problem. Then for each sequence X, there is a BST algorithm $\mathcal{A}'$ whose total cost on X is at most $O(cost_{\mathcal{A}}(X))$.*

Greedy

For each time t and key $z \in [n]$, let $\tau(z,t)$ denote the last touched time of z on or before time t. At time t, Greedy touches key $z \in [n]$ if and only if $(z, \tau(z,t))$ forms an empty rectangle with (x_t, t). The final output of Greedy on input X is defined as $G(X)$; denote the first t rows of such output by $G_{\leqslant t}(X)$. See Figure 9 for an illustration.

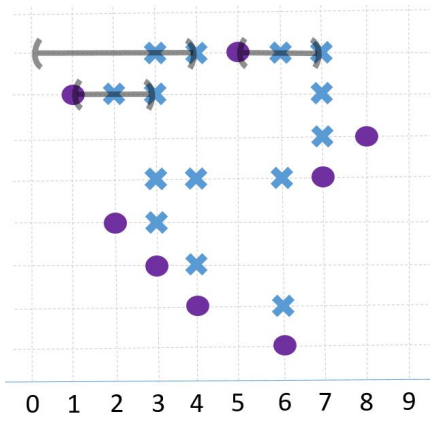


Figure 9 An example of intervals of Greedy. Here, circles represent access keys X and crosses represent touched points $Y \backslash X$. Interval of each key is defined by the top point (i.e. the last touched point) of such key. Intervals of 2,3, and 6 are shown in this figure.

Intervals for Geometric BSTs

Since the intervals of BST algorithms in a tree view depend on sub-trees, for Greedy, we need an analogue of sub-trees in geometric view. Let $z \in [n]$ be a key. The interval $I_G(z)$ is defined as $(\mathsf{left}_G(z), \mathsf{right}_G(z))$ where $\mathsf{left}_G(a)$ is the maximum key less than z that is touched at the same time as when a was last touched (0 if it does not exist). Similarly, we define $\mathsf{right}_G(z)$ as the minimum key greater than $a = z$, touched at the last touched time of z

$(n+1$ if it does not exist). So the interval $I_G(z)$ only change whenever z is touched. See Figure 9. Given an optimal tree $\mathcal{O}$, Lemma 8, Lemma 9 and Proposition 14 still hold for these geometric intervals. The following observations are important. (See Figure 10a)

▶ **Observation 18.** *Suppose key x_t is accessed. Let $\{I_G(b)\}_{b\in[n]}$ and $\{I'_G(b)\}_{b\in[n]}$ be the intervals before and after Greedy updates. Let $b_1 < b_2 < \ldots < b_k$ be the touched keys, $b_0 = 0$ and $b_{k+1} = n+1$. Then we have that $I'_G(b_i) = (b_{i-1}, b_{i+1})$ for all $i = 1, \ldots, k$.*

▶ **Corollary 19.** *For each point $p \in \mathbb{R}$, there are at most two touched keys z for which $p \in I'_G(z)$. In other words, the clique size of this interval graph is 2 and therefore, the touched intervals $I'_G(z)$ can be decomposed into two disjoint sets of intervals.*

2.6 The Second Showcase: Greedy

We show the following static optimality bound for Greedy using our potential.

▶ **Theorem 20.** *For each X and each static tree $\mathcal{O}$, $G(X) \leqslant 4 \cdot cost_{\mathcal{O}}(X) - m + n^2$.*

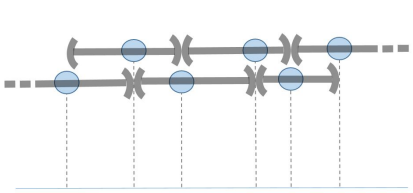
We note that the term n^2 is from the number of initial coins. Later, we will show that these initial coins can be bounded by $cost_{\mathcal{O}}(X)$, therefore we have $G(X) \leqslant 5 \cdot cost_{\mathcal{O}}(X)$.

The potential function we use is simply the extended inversion Φ. Let $\mathcal{O}$ be any tree. After accessing x, the Greedy intervals change from $\{I_G(z)\}_{z\in[n]}$ to $\{I'_G(z)\}_{z\in[n]}$, and the potential changes from Φ to Φ'. We will prove that $||\Phi'|| \leqslant ||\Phi - \mathcal{C}P|| + 4|\mathcal{S}_{\mathcal{O}}(x)| - 1$.

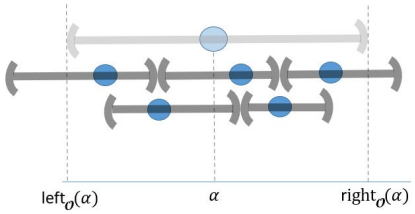
Summing over all accesses over a sequence X of length m will give us the desired bound. Denote $\Phi - \mathcal{C}P$ by $\widehat{\Phi}$. Due to Proposition 14, the change of potential function is only due to the inversions (z, α) where $z \in \mathcal{S}_G(x)$ and $\alpha \in \mathcal{S}_{\mathcal{O}}(x)$.

▶ **Lemma 21.** *For each $\alpha \in \mathcal{S}_{\mathcal{O}}(x)$, we have $||\Phi'(\bullet, \alpha)|| - ||\widehat{\Phi}(\bullet, \alpha)|| \leqslant 4$. Moreover, $||\Phi'(\bullet, x)|| - ||\widehat{\Phi}(\bullet, x)|| \leqslant 3$.*

Proof. Fix $\alpha \in \mathcal{S}_{\mathcal{O}}(x)$. Notice that the term $\Phi'(z, \alpha) - \widehat{\Phi}(z, \alpha) = 1$ only if $z \in \mathcal{S}_G(x)$ and $\Phi'(z, \alpha) = 1$. To have $\Phi'(z, \alpha) = 1$, the interval $I'_G(z)$ must contain a point in $\mathcal{P}_{\mathcal{O}}(\alpha) = \{l, \alpha, r\}$ (where $l = \mathsf{left}_{\mathcal{O}}(\alpha)$ and $r = \mathsf{right}_{\mathcal{O}}(\alpha)$) and point z must be inside $I_{\mathcal{O}}(\alpha)$. From the fact that z is inside $I_O(\alpha)$ and the geometry of Greedy (Observation 18), only one interval can contain l, only one interval can contain r, and at most two intervals can contain α (this is all illustrated in Figure 10b); moreover, when $\alpha = x$, only one interval may contain α ($I_G(x)$ itself). This concludes the proof. ◀



(a) Intervals of keys that are touched at the same time in Greedy. Think of all of them as in the same height, we perturb them in order to show their intervals clearly.



(b) Intervals in G are represented in black, while interval in $\mathcal{O}$ is represented in grey.

Figure 10

To recap, the framework is as follows. Let $\mathcal{A}$ be an algorithm and $\mathcal{O}$ be an optimal algorithm. Suppose, after accesssing key $x \in [n]$, $\mathcal{A}$ changes to $\mathcal{A}'$ and $\mathcal{O}$ to $\mathcal{O}'$. We wan to upper bound $|\mathcal{S}_{\mathcal{A}}(x)| + ||\Phi_{\mathcal{A}',\mathcal{O}'}|| - ||\Phi_{\mathcal{A},\mathcal{O}}|| \leqslant O(|\mathcal{S}_{\mathcal{O}}(x)|)$. We upper bound this in two steps:

- The change of $\mathcal{A}$ to $\mathcal{A}'$ after subtracting the cost: $||\Phi_{\mathcal{A}',\mathcal{O}}|| - ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{C}P||$
- And then the change of $\mathcal{O}$ to $\mathcal{O}'$: $||\Phi_{\mathcal{A}',\mathcal{O}'}|| - ||\Phi_{\mathcal{A}',\mathcal{O}}||$

We have shown how to bound the term $||\Phi_{\mathcal{A}',\mathcal{O}}|| - ||\Phi_{\mathcal{A},\mathcal{O}} - \mathcal{C}P||$ for Greedy and (a special case of) Splay. This is the "systematic" part of our framework that allows us to prove MTR-competitiveness for a family of BSTs. In the next subsection, we outlined how to upper bound $||\Phi_{\mathcal{A}',\mathcal{O}'}|| - ||\Phi_{\mathcal{A}',\mathcal{O}}||$ for Splay and Greedy, thus implying their $O(1)$-competitiveness to MTR. The proof for weighted dynamic finger is omitted due to the lack of space. Full details appear in the full version.

2.7 MTR-Competitiveness

In this section, we present the proof that Greedy and Splay have simulation embeddings. Due limitation of space, we defer the rest of the proofs to the full version.

First, we present necessary preliminaries.

▶ **Definition 22** (Static Optimality). *A BST algorithm $\mathcal{A}$ is statically optimal if, for all input X and static (reference) tree R, $cost_{\mathcal{A}}(X) \leqslant O(cost_R(X))$.*

Remark that a more precise definition of static optimality involves "initial tree" (the initial state of algorithm $\mathcal{A}$). In the context of our potential function, it is not difficult to see that the initial tree only affects the cost by at most an additive factor of $cost_R(X)$. Hence we omit the reference to initial trees for brevity. Denote the move-to-root algorithm by MTR.

▶ **Definition 23** (MTR-Competiveness). *A BST algorithm $\mathcal{A}$ is MTR-competitive if, for all input X, $cost_{\mathcal{A}}(X) \leqslant O(cost_{MTR}(X))$.*

▶ **Definition 24** (Simulation Embeddings). *A BST algorithm $\mathcal{A}$ has simulation embeddings if, for all access sequence X and algorithm $\mathcal{O}$, there exists a supersequence $Y \supseteq X$ such that $cost_{\mathcal{A}}(Y) \leqslant O(cost_{\mathcal{O}}(X))$.*

Levy and Tarjan [16] show that Splay has simulation embeddings by constructing a transition graph G_4 of Splay. Transition graph G_k is a directed graph where each node represents an instance of k-node BST. There is an edge (u, v) if a tree instance u can be changed to a tree instance v using one access. Also, they show that, in order for BST $\mathcal{A}$ to have simulation embeddings, it suffices to show that G_k of $\mathcal{A}$ is strongly connected for some constant $k \geqslant 3$.

▷ Claim 25. The transition graph G_3 of MTR is strongly connected. Hence, MTR has simulation embeddings. (See Figure 11)

This implies the following theorem.

▶ **Theorem 26.** *If a BST algorithm $\mathcal{A}$ is $O(1)$-competitive to MTR, then $\mathcal{A}$ has simulation embeddings.*

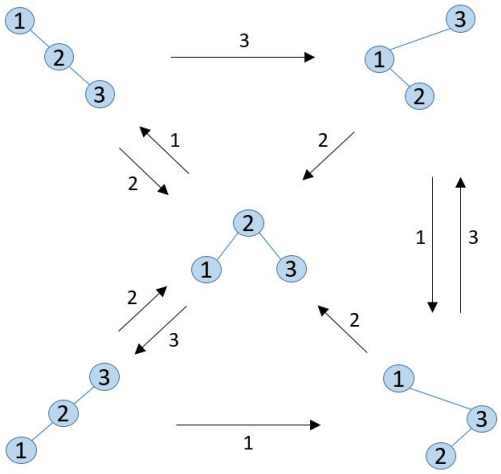


Figure 11 G_3 of MTR. The number on each arrow represents the key that MTR has to access in order to change its tree to the specific structure.

Proof. Suppose $\mathcal{A}$ is c-competitive to MTR. Now fix an input sequence X and a BST algorithm $\mathcal{O}$. From Claim 25, let d be the factor in the simulation embeddings of MTR, so we know that there exists a super-sequence X' such that $cost_{\mathsf{MTR}}(X') \leqslant d \cdot cost_{\mathcal{O}}(X)$. Since algorithm $\mathcal{A}$ is c-competitive to MTR, we have

$$cost_{\mathcal{A}}(X') \leqslant c \cdot cost_{\mathsf{MTR}}(X') \leqslant (cd) \cdot cost_{\mathcal{O}}(X)$$

This implies that $\mathcal{A}$ has simulation embeddings. ◀

Our potential function is particularly suitable for proving MTR-competitiveness. In fact, to show that $\mathcal{A}$ is $O(1)$-competitive to MTR, it suffices to only prove static optimality for $\mathcal{A}$ using extended inversions (together with a function that depends only on n.)

▶ **Lemma 27.** *A BST algorithm that (1) satisfies static optimality via potential function Ψ that depends linearly on extended inversions (i.e. $||\Psi|| = c_1||\Phi|| + c_2 n$), and (2) moves the accessed key to the root, must be $O(1)$-competitive to MTR.*

Proof. If $\mathcal{A}$ satisfies static optimality via Ψ, we have that:

$$||\Psi_{\mathcal{A}',\mathcal{O}}|| - ||\Psi_{\mathcal{A},\mathcal{O}} - \mathcal{C}P|| \leqslant C \cdot |\mathcal{S}_{\mathcal{O}}(x)|$$

Now, since the second term of potential is $c_2 n$ (only depending on n), to upper bound $||\Psi_{\mathcal{A}',\mathcal{O}'}|| - ||\Psi_{\mathcal{A}',\mathcal{O}}||$, it suffices to show that, we have $||\Phi_{\mathcal{A}',\mathcal{O}'}|| - ||\Phi_{\mathcal{A}',\mathcal{O}}|| \leqslant 0$.

For key $z = x$, $||\Phi_{\mathcal{A}',\mathcal{O}'}(z,x)|| = ||\Phi_{\mathcal{A}',\mathcal{O}}(z,x)|| = c_1$. Now, we consider keys $z \neq x$. Observe that intervals in $\mathcal{O}'$ change as follow:

- **(i)** $I_{\mathcal{O}'}(x) = (0, n+1)$.
- **(ii)** $I_{\mathcal{O}'}(y) = (\mathsf{left}_{\mathcal{O}}(y), x)$ for all y such that $y < x$ and $y \in \mathcal{S}_{\mathcal{O}}(x)$.
- **(iii)** $I_{\mathcal{O}'}(y) = (x, \mathsf{right}_{\mathcal{O}}(y))$ for all y such that $y > x$ and $y \in \mathcal{S}_{\mathcal{O}}(x)$.
- **(iv)** $I_{\mathcal{O}'}(y) = I_{\mathcal{O}}(y)$ for all $y \notin S_{\mathcal{O}}(x)$.

In other words, the only new important point in $\mathcal{O}'$ is x. Since $\mathcal{A}'$ also has x as the root, no $I'_{\mathcal{A}}(z)$ can contain x. Since x is the only new ancestor for z, this means $||\Phi_{\mathcal{A}',\mathcal{O}'}(z,x)|| = 0$. ◀

Since Splays and Greedy satisfy static optimality via our potential function, Lemma 27 implies that they are MTR-competitive, and hence have simulation embeddings.

References

1 Susanne Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27(3):682–693, 1998.

2 Susanne Albers. Online algorithms: a survey. *Mathematical Programming*, 97(1-2):3–26, 2003.

3 Presenjit Bose, Karim Douïeb, John Iacono, and Stefan Langerman. The power and limitations of static binary search trees with lazy finger. In *International Symposium on Algorithms and Computation*, pages 181–192. Springer, 2014.

4 Parinya Chalermsook, Julia Chuzhoy, and Thatchaphol Saranurak. Pinning down the strong wilber 1 bound for binary search trees. *APPROX*, 2020.

5 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Pattern-avoiding access in binary search trees. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 410–423. IEEE, 2015.

6 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Self-adjusting binary search trees: What makes them tick? In *Algorithms-ESA 2015*, pages 300–312. Springer, 2015.

7 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. The landscape of bounds for binary search trees. *arXiv preprint*, 2016. `arXiv:1603.04892`.

8 Parinya Chalermsook, Mayank Goswami, László Kozma, Kurt Mehlhorn, and Thatchaphol Saranurak. Multi-finger binary search trees. In *29th International Symposium on Algorithms and Computation (ISAAC 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

9 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`.

10 Richard Cole. On the dynamic finger conjecture for splay trees. part ii: The proof. *SIAM Journal on Computing*, 30(1):44–85, 2000.

11 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 Journal on Computing*, 30(1):1–43, 2000.

12 Erik D Demaine, Dion Harmon, John Iacono, Daniel Kane, and Mihai Patraşcu. The geometry of binary search trees. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 496–505. SIAM, 2009.

13 Erik D Demaine, Dion Harmon, John Iacono, and Mihai Patraşcu. Dynamic optimality—almost. *SIAM Journal on Computing*, 37(1):240–251, 2007.

14 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. SIAM, 2016.

15 Victor Lecomte and Omri Weinstein. Settling the relationship between wilber's bounds for dynamic optimality. *arXiv preprint*, 2019. `arXiv:1912.02858`.

16 Caleb Levy and Robert Tarjan. A new path from splay to dynamic optimality. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1311–1330. Society for Industrial and Applied Mathematics, 2019.

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

18 J Ian Munro. On the competitiveness of linear search. In *European Symposium on Algorithms*, pages 338–345. Springer, 2000.

19 Seth Pettie. Splay trees, davenport-schinzel sequences, and the deque conjecture. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1115–1124. Society for Industrial and Applied Mathematics, 2008.

20 Seth Pettie. Applications of forbidden 0–1 matrices to search tree and path compression-based data structures. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete algorithms*, pages 1457–1467. SIAM, 2010.

21 Luís MS Russo. A study on splay trees. *Theoretical Computer Science*, 2018.

22 Daniel D Sleator and Robert E Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

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

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