Greedy Is an Almost Optimal Deque

P. Chalermsook[1], M. Goswami[1], L. Kozma[2], K. Mehlhorn[1], and T. Saranurak[3]*

[1] Max-Planck Institute for Informatics, Saarbrücken, Germany 66123
[2] Department of Computer Science, Saarland University, Saarbrücken, Germany 66123
[3] KTH Royal Institute of Technology, Stockholm, Sweden 11428

Abstract. In this paper we extend the geometric binary search tree (BST) model of Demaine, Harmon, Iacono, Kane, and Pătraşcu (DHIKP) to accommodate for insertions and deletions. Within this extended model, we study the online GREEDY BST algorithm introduced by DHIKP. GREEDY BST is known to be equivalent to a maximally greedy (but inherently offline) algorithm introduced independently by Lucas in 1988 and Munro in 2000, conjectured to be dynamically optimal.

With the application of forbidden-submatrix theory, we prove a quasilinear upper bound on the performance of GREEDY BST on deque sequences. It has been conjectured (Tarjan, 1985) that splay trees (Sleator and Tarjan, 1983) can serve such sequences in linear time. Currently neither splay trees, nor other general-purpose BST algorithms are known to fulfill this requirement. As a special case, we show that GREEDY BST can serve output-restricted deque sequences in linear time. A similar result is known for splay trees (Tarjan,1985; Elmasry, 2004).

As a further application of the insert-delete model, we give a simple proof that, given a set U of permutations of $[n]$, the access cost of any BST algorithm on "most" of the permutations from U is $\Omega(\log |U| + n)$. In particular, this implies that the access cost for a random permutation of $[n]$ is $\Omega(n \log n)$ with high probability.

Besides the splay tree noted before, GREEDY BST has recently emerged as a plausible candidate for dynamic optimality. Compared to splay trees, much less effort has gone into analyzing GREEDY BST. Our work is intended as a step towards a full understanding of GREEDY BST, and we remark that forbidden-submatrix arguments seem particularly well suited for carrying out this program.

1 Introduction

Binary search trees (BST) are among the most popular and most thoroughly studied data structures for the dictionary problem. There remain however, several outstanding open questions related to the BST model. In particular, what is the best way to adapt a BST in an online fashion, in reaction to a sequence of operations (e.g. access, insert, and delete), and what are the theoretical limits of such an adaptation? Does there exist a "one-size-fits-all" BST algorithm, asymptotically as efficient as any other dynamic BST algorithm, regardless of the input sequence?

Splay trees have been proposed by Sleator and Tarjan [11] as an efficient BST algorithm, and were shown to be competitive with any *static* BST (besides a number of other attractive properties, such as the *balance*, *working set*, and *static finger* properties). The question whether splay trees are competitive with any *dynamic* BST algorithm, indeed,

*Work mostly done while at Saarland University whether there exists such an algorithm at all is the subject of the famous dynamic optimality conjecture [11].

A different BST algorithm (later called GREEDYFUTURE) has been proposed independently by Lucas [6] and by Munro [7]. GREEDYFUTURE is an offline algorithm: it anticipates future accesses, preparing for them according to a greedy strategy. In a breakthrough result, Demaine, Harmon, Iacono, Kane, and Pătraşcu (DHIKP) transformed GREEDYFUTURE into an online algorithm (called here GREEDY BST), and presented a geometric view of BST that facilitates the analysis of access costs (while abstracting away many details of the BST model).

At present, our understanding of both splay trees and GREEDY BST is incomplete. For splay trees, besides the above-mentioned four properties (subsumed by a single statement called the access lemma), a few other corollaries of dynamic optimality have been shown, including the *sequential access* [13] and the *dynamic finger* [1,2] theorems. The only known proof of the latter result uses very sophisticated arguments, which makes one pessimistic about the possibility of proving even stronger statements.

A further property conjectured for splay trees is a linear cost on deque sequences (stated as the "deque conjecture" by Tarjan [13] in 1985). Informally, a deque sequence consists of insert and delete operations only at *minimum* or *maximum* elements of the current dictionary. The only known upper bounds for the cost of splay on a sequence of n deque operations are $O(n\alpha(n))$ by Sundar [12] and $O(n\alpha^*(n))$ by Pettie [8]. Here α is the *extremely* slowly growing inverse Ackermann function, and α^* is its iterated version. A linear bound for *output-restricted* deque sequences (i.e. where deletes can only occur at the minimum) has been shown by Tarjan [13], and later improved by Elmasry [4].

In general, our understanding of GREEDY BST is even more limited. Fox [5] has shown that GREEDY BST satisfies the access lemma and the sequential access theorem, but no other nontrivial bounds appear to be known. One might optimistically ascribe this to a (relative) lack of trying, rather than to insurmountable technical obstacles. This motivates our attempt at the deque conjecture for GREEDY BST.

As mentioned earlier, a deque sequence consists of insert and delete operations. In the tree-view, e.g. for splay trees, such operations have a straightforward implementation. Unfortunately, the geometric view in which GREEDY BST can be most naturally expressed only concerns with accesses. Thus, prior to our work there was no way to formulate the deque conjecture in a managable way for GREEDY BST.

Our contributions. We augment the geometric model of DHIKP to allow insert and delete operations (exemplified by the extension of the GREEDY BST algorithm), and we show the offline and online equivalence of a sequence of operations in geometric view with the corresponding sequence in tree-view. This extended model allows us to formulate the deque conjecture for GREEDY BST. We transcribe the geometric view of GREEDY BST in matrix form, and we apply the forbidden-submatrix technique to derive the quasilinear bound $O(m2^{\alpha(m,m+n)} + n)$ on the cost of GREEDY BST, while serving a deque sequence of length m on keys from $[n]$.

We also prove an $O(m + n)$ upper bound for the special case of output-restricted deque sequences. We find this proof considerably simpler than the corresponding proofs for splay trees, and we observe that a slight modification of the argument gives a new (and perhaps simpler) proof of the sequential access theorem for GREEDY BST.

As a further application of the insert-delete model we show through a reduction to sorting that most representatives from a set U of permutations on $[n]$ have an access cost of $\Omega(\log |U| + n)$, for *any* BST algorithm. In particular, this implies that a random permutation of $[n]$ has access cost $\Omega(n \log n)$ with high probability. The latter observation appears to be folklore, but we are not aware of a published proof. Our argument is very simple, not relying on lower bounds in the geometric BST model or on balls-to-bins-type probabilistic arguments. Permutation access sequences are important, since it is known that the existence of a BST algorithm that is constant-competitive on permutations implies the existence of a dynamically optimal algorithm (on arbitrary access sequences).

Related work. Most relevant to our work is the deque bound of Pettie for splay trees [8]. That result relies on bounds for Davenport-Schinzel sequences, which can be reformulated in the forbidden-submatrix framework. Indeed, the use of forbidden-submatrix theory for proving data structure bounds was pioneered by Pettie, who reproved the sequential access theorem for splay trees [9] (among other data structure results). Our application of forbidden-submatrix theory is somewhat simpler and perhaps more intuitive: the geometric view of GREEDY BST seems particularly suitable for these types of arguments, as the structure of BST accesses is readily available in a matrix form, without the need for an extra "transcribing" step.

2 Geometric Formulation of BST with Insertion/Deletion

In this section we extend the model of DHIKP [3] to allow for insertions and deletions. After defining our geometric model, we prove the equivalence of the arboral (i.e. tree-view) and the geometric views of BSTs.

2.1 Rotations and Updates

Definition 1 (Valid Reconfiguration). *Given a BST T_1, a (connected) subtree τ of T_1 containing the root, and a tree τ' on the same nodes as τ, except that one node may be missing or newly added, we say that T_1 can be reconfigured by an operation $\tau \to \tau'$ to another BST T_2 if T_2 is identical to T_1 except for τ being replaced by τ', meaning that the child pointers of elements not in τ do not change. The cost of the reconfiguration is* $\max\{|\tau|, |\tau'|\}$.

This definition differs from [3, Def.2] in that τ' need not be defined on the same nodes as τ. Note that, according to the definition, if an operation $\tau \to \tau'$ changes a child pointer of an element x, then $x \in \tau$. See Figure 1 for examples.

Definition 2 (Execution of Update Sequence). *Given an update sequence*

$$S = \langle (s_1, \mathtt{op}_1), (s_2, \mathtt{op}_2), \ldots, (s_m, \mathtt{op}_m) \rangle, \text{ where } \mathtt{op}_i \in \{\mathtt{access}, \mathtt{insert}, \mathtt{delete}\},$$

we say that a BST algorithm executes S by an execution $E = \langle T_0, \tau_1 \to \tau_1', \ldots, \tau_m \to \tau_m' \rangle$ if all reconfigurations $\tau_t \to \tau_t'$ transforming T_{t-1} to T_t are valid, and for all t

- *if* $\mathtt{op}_t = \mathtt{access}$, *then* $s_t \in \tau_t$ *and* $\tau_t' = \tau_t$ *as a set,*

– *if* $\mathtt{op}_t = \mathtt{insert}$, *then* $\tau'_t = \{s_t\} \dot{\cup} \tau_t$ *as a set,*
– *if* $\mathtt{op}_t = \mathtt{delete}$, *then* $\tau_t = \{s_t\} \dot{\cup} \tau'_t$ *as a set.*

We also say that E *executes* S*. The* cost of execution *of* E *is the sum over all reconfiguration costs. If an element* $x \in \tau_t \cup \tau'_t$*, we say that* x *is* touched *at time* t*.*

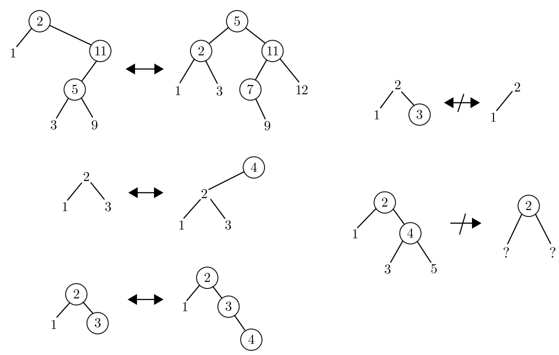

Fig. 1: (left) Examples of valid insert/delete operations. Circled elements indicate τ and τ'; (right) Examples of invalid operations: τ does not contain root (above) and τ' cannot link all pendant trees (below).

We assume that we work over the set $[n]$. Each element can be inserted or deleted many times, but insertions and deletions on the same element must be alternating. We also assume that every element is accessed or updated at least once.

2.2 Valid Sets

Definition 3 (Geometric View of Update Sequence). *The geometric view of an update sequence* S *is a point set* $P(S) = A(S) \dot{\cup} I(S) \dot{\cup} D(S)$ *in the integer grid* $[n] \times [m]$ *consisting of access points* $A(S) = \{(s_t, t) \mid \mathtt{op}_t = \mathtt{access}\}$*, insertion points* $I(S) = \{(s_t, t) \mid \mathtt{op}_t = \mathtt{insert}\}$*, and deletion points* $D(S) = \{(s_t, t) \mid \mathtt{op}_t = \mathtt{delete}\}$*. Update points are* $U(S) = I(S) \dot{\cup} D(S)$*.*

We usually omit the parameter S and simply write A, I, D, U when the choice of S is clear from context. We denote the x-coordinate and t-coordinate of a point p by (p_x, p_t). By element x, we mean the column x. By time t, we mean the row t.

Definition 4 (Valid Point). *Given a point set* $P(S)$ *in the integer grid* $[n] \times [m]$*, let* p *be a point (*p *may not be in* $P(S)$*), and let* $p', p'' \in U(S)$ *denote the update points nearest to* p*, below (resp. above)* p*, i.e.* $p'_x = p''_x = p_x$*, and* $p'_t < p_t < p''_t$*. One or both of* p' *and* p'' *might not exist. We say that* p *is* valid *in* $P(S)$*, iff:*

– $p \notin U(S)$*,* $p' \in I(S)$ *(or does not exist), and* $p'' \in D(S)$ *(or does not exist), or*
– $p \in I(S)$*,* $p' \in D(S)$ *(or does not exist), and* $p'' \in D(S)$ *(or does not exist), or*
– $p \in D(S)$*,* $p' \in I(S)$ *(or does not exist), and* $p'' \in I(S)$ *(or does not exist).*

Let T_t denote the resulting tree at time t during an execution of the BST algorithm E on the update sequence S.

Fact 5 *A point x can be touched at time t iff (x,t) is valid.*

Suppose that (x,t) is valid. If (x,t) is a deletion point, then x is in T_{t-1} but not T_t, and it is touched. If (x,t) is an insertion point, then x is in T_t but not T_{t-1}, and it is touched. If (x,t) is not an update point, then x is in both trees, and might or might not be touched. See Figure 2 for an illustration.

Definition 6 (Predecessor/Successor of a Point). *Given $P(S)$, the predecessor $pred(p)$ of a point p is the largest element x' smaller than p_x such that (x', p_t) is valid. The successor $succ(p)$ of p is symmetrically defined. We also write $pred(p) = (x', p_t)$ as a point, as well as $succ(p)$.*

Definition 7 (Valid Set). *A point set $P \supseteq P(S)$ is* valid *iff every point $p \in P$ is valid.*

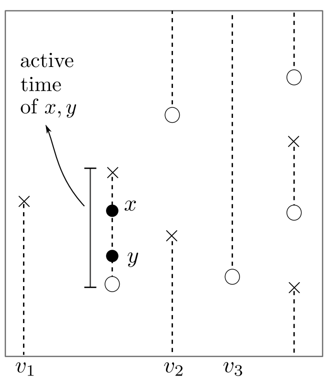

Fig. 2: A point set with insert ($\circ$) and delete ($\times$) points. Dashed lines indicate valid points. Observe that $succ(x) = v_3$, $succ(y) = v_2$, and $pred(x) = pred(y) = v_1$.

For any node x in a tree T, let $pred_T(x)$ denote predecessor of x in T and define the successsor $succ_T(x)$ similarly. The following lemma shows that points in a valid set, and their predecessor and successor, are associated with nodes in the tree at the corresponding time.

Lemma 8. *Let $P \supseteq P(S)$ be a valid point set, and E executes S. For any $p \in U(S)$, we have $pred(p) = pred_{T_{p_t}}(p_x)$ and $succ(p) = succ_{T_{p_t}}(p_x)$.*

Proof: Let $x' = pred(p)$ and hence (x', p_t) is valid by definition. By Fact 5, x' can be touched at time p_t. Since x' is not an updated element, we have $x' \in T_{p_t}$. Moreover, x' is the closest element on the left of p_x at this time. So $x' = pred_{T_{p_t}}(p_x)$. The proof for successor is symmetric. $\square$

Definition 9 (Active Time of Points). *Let p be a point in a valid point set $P \supseteq P(S)$. The active time $act(p)$ of p is the maximal consecutive interval of time $[t_{ins}(p), t_{del}(p)]$ containing p_t such that, for all $t \in act(p)$, (p_x, t) is valid. We call $t_{ins}(p)$ insertion time of p, and $t_{del}(p)$ deletion time of p.*

2.3 Arborally Satisfied Set

Definition 10 (Geometric View of BST Execution). *The geometric view of a BST execution $E = \langle T_0, \tau_1 \to \tau'_1, \ldots, \tau_m \to \tau'_m \rangle$ of some update sequence S is the point set $P(E) = \{(x,t) \mid x \in \tau_t \cup \tau'_t\}$ in the integer grid, indicating which element is touched at which time. Note that $P(E) \supseteq P(S)$.*

Definition 11 (Arborally Satisfied Set). *A valid point set $P \supseteq P(S)$ is* (arborally) satisfied *iff the following holds:*

- *For each pair $p, q \in P$ that are both active from time p_t to q_t (called an active pair), either both p and q lie in the same vertical/horizontal line, or there is a point $r \in \square_{pq} \cap P \setminus \{p, q\}$. If r is on the bottommost row of $\square_{pq}$, then r cannot be a deletion point. If r is on the topmost row of $\square_{pq}$, then r cannot be an insertion point.*
- *For each update point $p \in U$, if both $pred(p)$ and $succ(p)$ exist, then either $pred(p)$ or $succ(p)$ is also in P.*

The first condition is almost the same as the one in [3, Def. 2.3] but focused only on *active pairs* (they are active from p_t to q_t), and with additional technical condition due to update points. The second condition says that if the updated element is not the current minimum/maximum, then one of its adjacent elements must be touched.

Note that if there are no update points, then all points are active the whole time and our definition is equivalent to [3, Def. 2.3]. We defer the proof of the following fact to the appendix.

Fact 12 *Suppose that P is satisfied. Then, for each pair $p, q \in P$ which are both active from time p_t to q_t and $p_t < q_t$, there exists a point in $P \setminus \{p, q\}$ on a side of $\square_{pq}$ incident to p, that is either a non-deletion point, or the corner (p_x, q_t). Similarly, there exists a point in $P \setminus \{p, q\}$ on a side of $\square_{pq}$ incident to q, that is either a non-insertion point, or the corner (q_x, p_t).*

3 Equivalence of Arboral and Geometric Views

In this section we prove the following theorem:

Theorem 13. *A point set P is satisfied iff $P = P(E)$ for some BST execution E.*

The first direction of the proof involves considering a BST algorithm and showing that it generates a valid point set (tree to geometry). The second direction is showing how to convert a valid point set to a BST algorithm (geometry to tree).

3.1 Tree to Geometry

Lemma 14. *Let x and z be elements with consecutive values in a BST T, with $x < z$. Then one of x and z is an ancestor of the other.*

Proof: Suppose not. Then the lowest common ancestor of x and z is another element y. We know $x < y < z$ which is a contradiction. □

Lemma 15. *Suppose that y is not the minimum or maximum element in a BST T. To insert or delete y in T, either $pred_T(y)$ or $succ_T(y)$ must be touched.*

Lemma 16. *For any execution E, a point set $P(E)$ is satisfied.*

Proof: There are two conditions that need to be checked.

For the first condition, let p, q be a pair of points in $P(E)$ active from time p_t to q_t. Suppose that p, q violate the condition. Hence, they are not vertically or horizontally aligned. We assume that $p_t < q_t$ and $p_x < q_x$. Since p_x and q_x are active at time p_t, by Fact 5 and the statement below the fact, they exist in the tree T_{p_t}. Hence, a lowest common ancestor a of p_x and q_x in T_{p_t} is well-defined. There are two cases.

If $a = p_x$, then p_x is an ancestor of q_x. Since $\square_{pq}$ is not satisfied, q_x is not touched from time p_t to $q_t - 1$ and p_x remains an ancestor of q_x right before time q_t. Thus, to touch q_x at time q_t, p_x must be touched, and so $(p_x, q_t) \in \square_{pq}$. Only insertion point can be in the topmost row of unsatisfied $\square_{pq}$. So (p_x, q_t) an insertion point. But this implies that p and q are not active pair, which is a contradiction.

If $a \neq p_x$, then a must be touched at time p_t. As a has value between p_x and q_x, we have $(a, p_t) \in \square_{pq}$. Since $\square_{pq}$ is not satisfied, (a, p_t) is a deletion point and, moreover, p_x must be its predecessor. Hence p_x becomes an ancestor of q_x right after time p_t and we can use the previous argument again.

For the second condition, suppose that $p \in U$ is an update point. That is, we update p_x in the BST T_{p_t}. If both $pred(p)$ and $succ(p)$ exist, then p_x is not a minimum or maximum in T_{p_t}. By Lemma 15, either $pred_{T_{p_t}}(p_x)$ or $succ_{T_{p_t}}(p_x)$ is touched at time p_t. By Lemma 8, $pred_{T_{p_t}}(p_x) = pred(p)$ and $succ_{T_{p_t}}(p_x) = succ(p)$, and we are done. $\square$

3.2 Geometry to Tree

Now we show how to convert a valid point set to an offline algorithm first. We need the following lemma, which is essentially a converse of Lemma 15, saying that if we touch either $pred_T(y)$ or $succ_T(y)$, then we can insert or delete y. We defer the proofs of the following two statements to the appendix.

Lemma 17. *Suppose either $pred_T(y)$ or $succ_T(y)$ is in a subtree τ containing the root of T, or y is the minimum or maximum element in T. Then (i) any reconfiguration $\tau \to \tau'$, where $\tau' = \tau \dot\cup \{y\}$ as a set, is valid, and (ii) any reconfiguration $\tau \to \tau'$, where $\tau = \tau' \dot\cup \{y\}$ as a set, is valid.*

Lemma 18 (Offline Equivalence). *For any satisfied set X, there is a point set $P(E) = X$ for some execution E. We call E a tree view of X.*

By Lemma 16 and 18, this concludes the proof of Theorem 13.

Observe that if $X = P(E)$, the quantity $|X|$ is exactly the execution cost of E.

3.3 Geometry to Tree: Online

The discussion in § 3.2 assumes that a satisfied set X is available all at once, and we show that there exists an execution E (i.e. an offline BST algorithm) whose point set $P(E)$ is exactly X.

We call an *online geometric algorithm* an algorithm that, given a geometric update sequence $P(S) \subseteq [n] \times [m]$, outputs a satisfied superset $P \supseteq P(S)$, with the condition that both the input and output are revealed *row-by-row* (i.e. the decision on which points to touch can depend only on the current and preceding rows of the input). We remark that GREEDY BST (as extended in § 4) is such an algorithm.

Analogously, by an online BST algorithm we mean a procedure that, given an initial set $S_0 \subseteq [n]$, and an update sequence S, outputs an execution E, with the condition that the both the input and output are revealed *item-by-item* (i.e. the decision on which reconfiguration to perform can depend only on the current and preceding update operations).

Theorem 19 (Online Equivalence). *For any online geometric algorithm $\mathcal{A}$, there exists an online BST algorithm $\mathcal{A}'$ such that, on any update sequence, the cost of $\mathcal{A}'$ is bounded by a constant times the cost of $\mathcal{A}$.*

The proof of Theorem 19 is an adaptation of the proof of Lemma 2.3 in [3] to the new geometric setting, and is analogous to the proof of Lemma 18. We omit the proof in this extended abstract.

4 Defining GREEDY BST with Insertion/Deletion

GREEDY BST is an online algorithm for constructing a satisfied set given an update sequence S. At each time t, GREEDY BST minimally satisfies the point set up to time t. Having defined satisfied sets when there are update points, we naturally obtain the extension of GREEDY BST that can handle insertions and deletions.

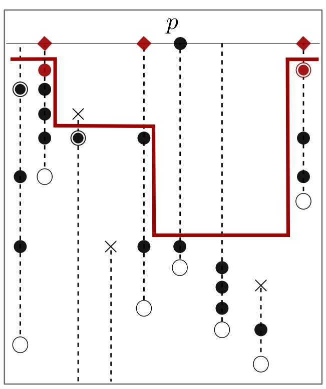

Fig. 3: Sample GREEDY BST execution with insertion ($\circ$), deletion ($\times$), accessed (double circle), and touched ($\bullet$) points. Thick line and red color indicate stair of p, and red ($\blacklozenge$) are the newly touched points at time p_t. Observe that a non-minimum insert or delete must touch a neighbor as well.

We develop some notation for describing the algorithm. A rectangle $\square_{pq}$ is *unsatisfied* if there is no other point in the proper (closed) rectangle formed by points p and q. We say that p and q are an *active pair* if they are active from time p_t to q_t. The *stair* of point p is denoted by $stair(p) = \{p\} \cup \{q \mid \square_{pq}$ is unsatisfied rectangle formed by an active pair p and q where q is below $p\}$. The *stair* of element x at time t is the stair of the point (x, t). *Satisfying/touching* $stair(x, t)$ means visiting, at time t, the elements of points in the stair: $\{(q_x, t) \mid q \in stair(x, t)\}$. These elements visited are then added to the row at time t.

Fact 20 *Touching the stair $stair(p)$ is to minimally satisfy the point p.*

Therefore, when GREEDY BST gets an access point p, it touches only $stair(p)$. For an update point p, if p is not a minimum or maximum, then GREEDY BST chooses the smaller set between $stair(p) \cup stair(pred(p))$ and $stair(p) \cup stair(succ(p))$. This is because of the second condition of satisfied set. If p is a minimum or maximum, then GREEDY BST just touches $stair(p)$. The execution of GREEDY BST is illustrated in Figure 3.

The following observation is useful for deque sequences. For insertion point p, observe that $stair(p) = \{p\}$ because the active time of p begins at time p_t itself (for any point q below p, p and q are not an active pair by definition).

Fact 21 *To insert p such that p is the minimum or maximum,* GREEDY BST *touches only p.*

5 Performance of GREEDY BST on Deque Sequences

Definition 22 (Deque Sequence). *An update sequence is a* deque sequence *if it has only insertions and deletions at the current minimum or maximum element, and no access operations.*

Definition 23 (Output-restricted). *A deque sequence is* output-restricted *if it has deletions only at minimum elements.*

Theorem 24. *The cost of executing a deque sequence on $[n]$ of length m by* GREEDY BST *is at most $O(m2^{\alpha(m,n+m)} + n)$, where α is the inverse Ackermann function.*

Theorem 25. *The cost of executing an output-restricted deque sequence on $[n]$ of length m by* GREEDY BST *is at most $24m + 12n$.*

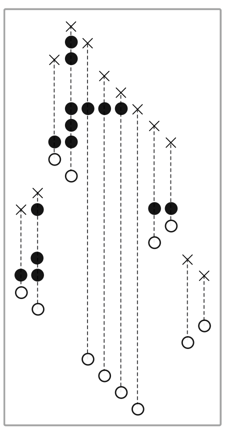

Fig. 4: Sample execution of GREEDY BST on a concentrated deque sequence with insertion ($\circ$), deletion ($\times$), and touched ($\bullet$) points. Dashed lines show the active times of elements.

Remark. The bound in Theorem 25 refers to the cost of the online *geometric* GREEDY BST. In the online tree-view equivalent the constants can be larger, hinging on the details of Theorem 19, but the bound remains of the form $O(m + n)$.

The rest of this section is devoted to the proofs of Theorems 24 and 25.

5.1 Concentrated Deque Sequences

We first reduce the analysis of GREEDY BST on any deque sequence to that on a special type of deque sequence that we call a concentrated deque sequence. Since in a deque sequence we can only delete the current minimum or maximum, we can define two sets of elements as follows: let L_t be the set of elements which are deleted (from the left) before time t when they were the minimum at their deletion time, and R_t be the set of elements which are deleted (from the right) before time t when they were the maximum at their deletion time.

Definition 26 (Concentrated Deque Sequence). *A deque sequence is concentrated if, for any time t, if the inserted element x is the minimum, then $y < x$ for all $y \in L_t$, and if x is the maximum, then $x < y$ for all $y \in R_t$.*

Note that the definition implies that each element in a concentrated deque sequence can be inserted and deleted at most once. We defer the proof of the following lemma to the appendix.

Lemma 27. *For any deque sequence S, there is a concentrated deque sequence S' such that the execution of any BST algorithm on S' and S have the same cost.*

5.2 Greedy BST on a Concentrated Deque Sequence

Now we analyze the performance of Greedy BST on concentrated deque sequences (see Figure 4 for an example). Because of Lemma 27, we can view the points touched by Greedy BST as an $(m \times (n + m))$ binary matrix (i.e. with entries 0 and 1), with all touched points represented as ones, and all other grid elements as zeroes. Notice that the number of columns is $n + m$ instead of n because of the reduction in Lemma 27 which allows each element to be inserted and deleted at most once.

Definition 28 (Forbidden Pattern). *A binary matrix M is said to* avoid *a binary matrix P (called a pattern) if there exists no submatrix M' of M with same dimensions as P, such that for all 1-entries of P, the corresponding entry in M' is 1 (the 0-entries of P are "don't care" values).*

We denote by $\mathrm{Ex}(P, m, n)$ the largest number of 1s in an $(m \times n)$ matrix M that avoids pattern P. In this work, we refer to the following patterns (as customary, we write dots for 1-entries and empty spaces for 0-entries).

$$P_5 = \begin{pmatrix} \bullet & & \bullet & & \bullet \\ & \bullet & & \bullet & \end{pmatrix} \quad and \quad P_4 = \begin{pmatrix} \bullet & & & \\ & & \bullet & \\ & \bullet & & \\ & & & \bullet \end{pmatrix}$$

Lemma 29. *The execution of* Greedy BST *on concentrated deque sequences avoids the pattern P_5.*

Proof: Suppose that P_5 appears in the Greedy BST execution, and name the touched points matched to the 1-entries in P_5 from left to right as a, b, c, d, and e.

Let $t > b_t$ be smallest such that (c_x, t) is touched. Then $t \leq c_t$ and either b or d must have been deleted within the time interval $[b_t, t]$. Otherwise, any update point in the interval $[b_t, t]$ is outside the interval $[b_x, d_x]$ and c_x is "hidden" by b and d (it cannot be on the stair of any update point).

Assume w.l.o.g. that b is deleted. If b is deleted by a minimum-delete, then a cannot be touched. If b is deleted by a maximum-delete, then e cannot be touched. This is because the sequence is concentrated. □

Lemma 30. *The execution of* Greedy BST *on concentrated output-restricted deque sequences avoids the pattern P_4.*

Proof: Suppose that P_4 appears in the Greedy BST execution, and name the touched points matched to the 1-entries in P_4 from left to right as a, b, c, and d. We claim that in order to touch c, there has to be a deletion point in the interval $[b_x, d_x]$ in the time interval $[d_t, c_t]$. Otherwise, any deletion point in the time interval $[d_t, c_t]$ is left of b_x (as deletes happen only at the minimum). Furthermore, all insertion points in the time interval $[b_t, c_t]$ must be be outside of $[b_x, d_x]$ (since both b and d are active at time b_t). We remind that insertion touches nothing else besides the insertion point itself. This means that c can not be touched: it is "hidden" to deletion points on the left of b_x by b.

Denote the deletion point in the area $[b_x, d_x] \times [d_t, c_t]$ as d'. Observe that a is to the left and above d', and since we only delete minimums, a is not active at time d'_t. In order to be touched, a must become active after d'_t via an insertion, contradicting that the sequence is concentrated. $\square$

Fact 31 **([10, Thm** 3.4] **).** $\mathrm{Ex}(P_5, u, v) = O(u2^{\alpha(u,v)} + v)$.

Fact 32 **([10, Thm** 1.5(5)] **).** $\mathrm{Ex}(P_4, u, v) < 12(u + v)$.

Proof of Theorem 24: By Lemma 27, it is enough to analyze the cost of GREEDY BST on concentrated deque sequences. This cost is bounded by $O(m2^{\alpha(m,m+n)} + n)$ using Lemma 29 and Fact 31.

Proof of Theorem 25: By Lemma 27, it is enough to analyze the cost of GREEDY BST on concentrated deque sequences. This cost is bounded by $24m + 12n$ using Lemma 30 and Fact 32.

Remark. The proof of Theorem 25 can be minimally adjusted to prove the sequential access theorem for GREEDY BST. A sequential access sequence can be simulated as a sequence of minimum-deletions. In this way we undercount the cost by exactly one touched point above each access, which adds a linear term to the bound.

6 A Lower Bound on Accessing a Set of Permutations

Let U be a set of permutations on $[n]$. In this section we prove the following theorem:

Theorem 33. *Fix a BST algorithm $\mathcal{A}$ and a constant $\epsilon < 1$. There exists $U^{'} \subseteq U$ of size $|U^{'}| \geq (1 - \frac{1}{|U|^\epsilon})|U|$ such that $\mathcal{A}$ requires $\Omega(\log|U| + n)$ access cost on any permutation in $U^{'}$.*

Proof: The proof utilizes the geometric view of insertions, and uses two reductions. We first claim that there exists an algorithm $\mathcal{B}$ that is capable of insertions such that the cost of $\mathcal{A}$ to access a permutation π is no less than the cost of $\mathcal{B}$ to insert π. Note that since $\mathcal{A}$ is accessing π, all the points are active by definition. We will describe $\mathcal{B}$ in the geometric view simply by requiring that upon inserting $\pi(t)$ at time t, $\mathcal{B}$ touches all the points that $\mathcal{A}$ touches while accessing $\pi(t)$ at time t. Note that $\mathcal{A}$ touches at least all the points in $stair(\pi(t), t)$, and $\mathcal{B}$ is required only to touch either $pred(\pi(t))$ and its stair, or $succ(\pi(t))$ and its stair (Definition 11). Since $pred(\pi(t))$ belongs to $stair(\pi(t), t)$, one easily sees that $stair(pred(p)) \subset stair(\pi(t), t)$, and this defines a valid insertion algorithm.

We now reduce $\mathcal{B}$ to an algorithm for sorting π. Just by a traversal of the tree maintained by $\mathcal{B}$ at time n, we can produce the sorted order of π after incurring a cost of $O(n)$. However, we know that to sort a set U of permutations, any (comparison-based) sorting algorithm must require $\Omega(\log|U|+n)$ comparisons on at least a $1 - \frac{1}{|U|^\epsilon}$ fraction of the permutations in U. To see this, note that the decision tree of any sorting algorithm must have at least $|U|$ leaves (note that here we are assuming the weaker hypothesis that $\mathcal{A}$ and hence the sorting algorithm, are only designed to work on U; they may fail outside U). The number of leaves at height at most $(1 - \epsilon)\log|U|$ is at most $|U|^{1-\epsilon}$, and hence at least a $1 - \frac{1}{|U|^\epsilon}$ fraction require at least $(1 - \epsilon)\log|U| = \Omega(\log|U|)$ comparisons. Adding the trivial bound of $\Omega(n)$ to scan the input permutation gives us the desired bound.

Remark. Upper bounds proved for our model do not directly translate into bounds for algorithms. For example, when a new maximum is inserted, this can be done at a cost of one by making the element the root of the tree, respectively, only touching the element inserted. Note that this requires the promise that the element inserted is actually a new maximum. A slight extension makes the model algorithmic. This is best described in tree-view. We put all nodes of the tree in in-order into a doubly-linked list. Then, in the case of an insertion one can actually stop the search once the predecessor or the successor of the new element has been reached in the search because by also comparing the new element with the neighboring list element, one can verify that a node contains the predecessor or successor. Thus at the cost of a constant factor, bounds proved for our model are algorithmic. □

References

1. R. Cole. On the dynamic finger conjecture for splay trees. part ii: The proof. *SIAM Journal on Computing*, 30(1):44–85, 2000. 2
2. 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. 2
3. Erik D. Demaine, Dion Harmon, John Iacono, Daniel M. Kane, and Mihai Patrascu. The geometry of binary search trees. In *SODA 2009*, pages 496–505, 2009. 3, 6, 8, 14
4. Amr Elmasry. On the sequential access theorem and deque conjecture for splay trees. *Theor. Comput. Sci.*, 314(3):459–466, 2004. 2
5. Kyle Fox. Upper bounds for maximally greedy binary search trees. In *WADS 2011*, pages 411–422, 2011. 2
6. Joan M. Lucas. Canonical forms for competitive binary search tree algorithms. *Tech. Rep. DCS-TR-250, Rutgers University*, 1988. 2
7. J.Ian Munro. On the competitiveness of linear search. In MikeS. Paterson, editor, *Algorithms - ESA 2000*, volume 1879 of *Lecture Notes in Computer Science*, pages 338–345. Springer Berlin Heidelberg, 2000. 2
8. Seth Pettie. Splay trees, davenport-schinzel sequences, and the deque conjecture. In *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 1115–1124, 2008. 2, 3
9. 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, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pages 1457–1467, 2010. 3
10. Seth Pettie. Generalized davenport–schinzel sequences and their 0–1 matrix counterparts. *Journal of Combinatorial Theory, Series A*, 118(6):1863–1895, 2011. 11
11. Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985. 1, 2
12. Rajamani Sundar. On the deque conjecture for the splay algorithm. *Combinatorica*, 12(1):95–124, 1992. 2
13. Robert Endre Tarjan. Sequential access in play trees takes linear time. *Combinatorica*, 5(4):367–378, 1985. 2

A Proof Omitted from Section 2

A.1 Proof of Fact 12

We give the proof only for p, as it is symmetric for q.

Since P is satisfied, there is a point r in $\square_{pq} \cap P \setminus \{p, q\}$ that satisfies $\square_{pq}$. If r is on the horizontal side incident to p, then it is not a deletion point, otherwise it would not satisfy $\square_{pq}$ by the first condition of Definition 11. If r is on the vertical side incident to p, and not (p_x, q_t), then it is not a deletion point, otherwise p and q would not be an active pair. Thus, if r is on a side of $\square_{pq}$ incident to p, we are done. Suppose this is not the case, and let r be a point in $\square_{pq} \cap P \setminus \{p, q\}$ satisfying $\square_{pq}$ such that $t_{ins}(r)$ is minimum. We claim that $t_{ins}(r) \leq p_t$.

First, we have $t_{ins}(r) < q_t$ because otherwise $t_{ins}(r) = q_t$ and hence r is an insertion point on the topmost row of $\square_{pq}$, and cannot satisfy $\square_{pq}$ due to the first condition of Definition 11. Then there must be some other point in $\square_{pq} \cap P \setminus \{p, q\}$, whose insertion time is before q_t, a contradiction.

Next, suppose $t_{ins}(r) > p_t$ then the point $r' = (r_x, t_{ins}(r))$ is an insertion point, and, by the second condition of Definition 11, there is another point $r'' \in \square_{pq}$ which is either $pred(r')$ or $succ(r')$. Note that $r'' \neq p, q$ because $p_t < t_{ins}(r) < q_t$. Observe that $t_{ins}(r'') < t_{ins}(r)$, contradicting the choice of r.

Now, since $t_{ins}(r) \leq p_t$, p and r are both active from time p_t to r_t, and we can repeat the same argument as long as r is not on the sides of $\square_{pq}$ incident to p.

B Proof Omitted from Section 3

B.1 Proof of Lemma 15

Since y is not the minimum or maximum, both predecessor $pred_T(y)$ and successor $succ_T(y)$ of y exist. Let $x = pred_T(y)$ and $z = succ_T(y)$. We consider two cases: insertion and deletion.

For insertion of y, $y \notin T$ before we insert. Then x and z are consecutive, and, by Lemma 14, we assume by symmetry that x is an ancestor of z. In particular, x has right child. After insertion, x and y are consecutive. So, by Lemma 14, either x or y is an ancestor of another one. If x is an ancestor of y, then x must have been touched so that we can change pointers below x. Otherwise, y is an ancestor of x and x cannot have a right child. Thus, x's right child pointer must have been changed to null, meaning that x has been touched.

For deletion of y, suppose that we do not touch both x and z. If y has at most one child, then one of x and z is an ancestor of y, but we must touch all the ancestors of y, a contradiction. If y has two children, then x has no right child and z has no left child. Suppose we delete y without touching x and z. Therefore, x still has no right child and z still has no left child even after $y \notin T$. This contradicts Lemma 14.

B.2 Proof of Lemma 17

Suppose that $x = pred_T(y) \in \tau$. The proof when $succ_T(y) \in \tau$ is symmetric. There are two statements to be proved regarding insertion and deletion of y, respectively. Let $child_R(x)$ denote a right child pointer of element x.

For insertion, we show that $\tau \to \tau'$ where $\tau' = \tau \dot{\cup} \{y\}$ is valid. First, we insert y into τ as a right child of x. The only pointer changes are: $child_R(y) \leftarrow child_R(x)$ and $child_R(x) \leftarrow y$. Finally, we rotate the resulting subtree, which includes y, to get τ'.

For deletion, we show that $\tau \to \tau'$ where $\tau = \tau' \dot{\cup} \{y\}$ is valid. Again, we rotate τ such that y is a right child of x, and then remove y. The only pointer change is: $child_R(x) \leftarrow child_R(y)$. Then we rotate the resulting subtree, which excludes y, to get τ'.

B.3 Proof of Lemma 18

We use the almost same argument as in [3, Lemma 2.2] but we need to make sure that we can also update elements while touching all points in X exactly. The argument is as follows.

Define the *next touch time* $N(x, t_0)$ of x at time t_0 in X to be the minimum t-coordinate of any point in X on the ray from (x, t_0) to (x, ∞). If there is no such point, then $N(x, t_0) = \infty$.

Let T_t be the treap defined on all points $(x, N(x, t))$ active right after time t. Recall that a treap is a BST on the first coordinate and a heap on the second. Let X_t denote the set of elements in the row t of X. Since T_t is a treap with heap priority $N(\cdot, t)$, $\tau_t = X_t \cap T_t$ is connected subtree containing the root of T_t. So we have $\tau_t = X_t$, and $\tau_t = X_t \setminus \{y\}$ if we insert y at time t, as desired.

If there is an update element y in X_t, and $pred_T(y)$ and $succ_T(y)$ exist, by Lemma 17, we just need to show that either $pred_{T_t}(y) \in X_t$ or $succ_{T_t}(y) \in X_t$. Since X is satisfied, either $pred(y, t) \in X_t$ or $succ(y, t) \in X_t$, say $pred(y, t) \in X_t$. By Fact 8, $pred_{T_t}(y) = pred(y, t)$. Therefore, $\tau_t \to \tau'_t$ is a valid reconfiguration where $\tau_t \cup \tau'_t = X_t$.

After, we update y in τ_t and get τ'_t, we want to get T_{t+1} which is a treap defined on $N(\cdot, t+1)$. To get this, we just heapify τ'_t based on $N(\cdot, t+1)$. We claim that the whole tree is now T_{t+1}. The following argument is exactly same as in [3, Lemma 2.2]. Suppose there is a parent/child (q, r) that heap property does not hold. Both q, r cannot be in τ'_t by construction. The next touch time of elements outside τ'_t does not change, so both q, r cannot be outside τ'_t.

Now, we have $q \in \tau'_t$ and $r \notin \tau'_t$ where $N(r, t+1) < N(q, t+1)$. The rectangle defined from (q, t) and $(r, N(r, t+1))$ will contradict Fact 12. There are two sides to be considered. First, there is no point on the vertical side $((q, t), (q, N(r, t+1)]$ because $N(r, t+1) < N(q, t+1)$. Next, all elements in T_{t+1} between q and r must be descendants of r, and they cannot be touched as r is not touched at time t. So the horizontal side $((q, t), (r, t)]$ can only have one deletion point s which has q_x as a predecessor/successor. This violates Fact 12 and completes the proof.

C Proof Omitted from Section 5

C.1 Proof of Lemma 27

Suppose that S is not concentrated, and let t_0 be the first time which S violates its condition. We will modify the sequence and obtain another sequence S' such that the first violation time is later than t_0, and the executions of S and S' on any BST algorithm are the same, and repeat the argument.

So, assume w.l.o.g. that the element x is inserted as the minimum at time t_0. Since the condition is violated, $x < y$ for some $y \in L_t$. Let x' be an element such that $y < x'$, for all $y \in L_{t_0}$, and x' is less than all elements in the current tree T_{t_0}. Note that x' must exist, because there is no violation before time t_0.

Now, since BST is a comparison-based model, as long as the relative values of all following update elements are preserved, even when the sequence is modified, the BST algorithm would behave the same.

Therefore, we will modify S such that we set the value of x to be x' while preserving the relative values of all following update elements. So now the condition is not violated at time t_0 while the execution of the modified sequence is unchanged.