B-Treaps Revised: Write Efficient Randomized Block Search Trees with High Load

Roodabeh Safavi
Institute of Science and Technology Austria (ISTA), Klosterneuburg, Austria

Martin P. Seybold
Faculty of Computer Science, Theory and Applications of Algorithms, University of Vienna, Austria

Abstract

Uniquely represented (UR) data structures represent each logical state with a unique storage state. We study the problem of maintaining a dynamic set of n keys from a totally ordered universe in this context. UR structures are also called "strongly history independent" structures in the literature.

We introduce a two-layer data structure called (α, ε)-Randomized Block Search Tree (RBST) that is uniquely represented and suitable for external memory (EM). Though RBSTs naturally generalize the well-known binary Treaps, several new ideas are needed to analyze the *expected* search, update, and storage efficiency in terms of block-reads, block-writes, and blocks stored. We prove that searches have $O(\varepsilon^{-1} + \log_\alpha n)$ block-reads, that dynamic updates perform $O(\varepsilon^{-1} + \log_\alpha(n)/\alpha)$ block-writes and $O(\varepsilon^{-2} + (1 + \frac{\varepsilon^{-1}+\log n}{\alpha}) \log_\alpha n)$ block-reads, and that (α, ε)-RBSTs have an asymptotic load-factor of at least $(1 - \varepsilon)$ for every $\varepsilon \in (0, 1/2]$.

Thus (α, ε)-RBSTs improve on the known, uniquely represented B-Treap [Golovin; ICALP'09]. Compared with non-UR structures, the RBST is also, to the best of our knowledge, the first external memory structure that is storage-efficient and has a non-amortized, write-efficient update bound.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Unique Representation, Randomization, Top-Down Analysis, Block Search Tree, Write-Efficiency, Storage-Efficiency

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.47

Related Version *Full Version*: `https://arxiv.org/abs/2303.04722`

Funding This work was supported under the Australian Research Council Discovery Projects funding scheme (project number DP180102870). This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101019564) "The Design of Modern Fully Dynamic Data Structures (MoDynStruct)" and from the Austrian Science Fund (FWF) project Z 422-N and project "Fast Algorithms for a Reactive Network Layer (ReactNet)" P 33775-N, with additional funding from the *netidee SCIENCE Stiftung*, 2020–2024.

1 Introduction

Organizing a set of keys such that insertions, deletions, and searches are supported is one of the most basic problems in computer science that drives the development and analysis of structures with various guarantees and performance trade-offs. Binary search trees for example have several known height balancing methods (e.g. AVL and Red-Black trees) and weight balancing methods (e.g. BB[a] trees), some providing $\mathcal{O}(1)$ writes for *each* update [17].

Block Search Trees are generalizations of binary search trees that store in every tree node up to α keys and $\alpha + 1$ child pointers, where the array size α of the blocks is a (typically) fixed parameter. Since such layouts enforce a certain data locality, block structures are central for read- and write-efficient access in machine models with a memory hierarchy.

External Memory and deterministic B-Trees variants. In the classic External Memory (EM) model, the n data items of a problem instance must be transferred in blocks of a fixed size B between a block-addressible external memory and an internal Main Memory (MM), which can store up to M data items and perform computations. Typically, $n \gg M > B$ and the MM can only store a small number of blocks throughout processing, say $M/B = \mathcal{O}(1)$. Though Vitter's Parallel Disk Model can formalize more general settings [30, Section 2.1], we are interested in the most basic case with one, merely block-addressible, EM device and one computing device with MM. The cost of a computation is typically measured in the number of *block-reads* from EM (Is), the number of *block-writes* to EM (Os), or the total number of IOs performed on EM. For basic algorithmic tasks like the range search, block search trees with $\alpha = \Theta(B)$ support algorithms with asymptotically optimal total IOs (e.g. [30, Section 10]). Classic B-Trees [4] guarantee that every block, beside the root, has a load-factor of at least $1/2$ and that all leaves are at equal depth, using a deterministic balance criteria. The B*-Tree variant [23, Section 6.2.4] guarantees a load-factor of at least $2/3$ based on a (deterministic) overflow strategy that shares keys among the two neighboring nodes on equal levels. Yao's Fringe Analysis [32] showed that inserting keys under a random order in (deterministic balanced) B-Trees yields an expected asymptotic load-factor of $\ln(2) \approx 69\%$, and the expected asymptotic load-factor is $2\ln(3/2) \approx 81\%$ for B*-Trees. For general workloads however, maintaining higher load-factors (with even wider overflow strategies) further increases the update write-bounds in block search trees [3, 24]. The B$^+$-Tree variant for key-value pairs, which stores the values separate from the keys to increase the load-factor and thus *fan-outs*, is heavily used in practical file systems, see e.g. [14].

The IO-optimized[1] B$^\varepsilon$-Tree [13, 7] provides smooth trade-offs between $\mathcal{O}(\log_{1+\alpha^\varepsilon}(n)/\alpha^{1-\varepsilon})$ amortized IOs per update and $\mathcal{O}(\log_{1+\alpha^\varepsilon} n)$ block-reads (Is) for searches. E.g. tuning parameter $\varepsilon = 1$ retains the bounds of B-Trees and $\varepsilon = 1/2$ provides improved bounds. In the fully update IO-optimized case ($\varepsilon = 0$) however, searches have merely a $\mathcal{O}(\log n)$ bound for the number of block-reads. The main design idea to achieve this is to augment the non-leaf nodes of a B-Tree with additional "message buffers" so that a key insertion can be stored close to the root. Messages then only need propagation, further down the tree, once a message buffer is full (cf. [13, Section 3.4] and [7]). Clearly, the state of either of those (deterministic) structures depends *heavily* on the actual sequence in which the keys were inserted in them.

Uniquely Represented Data Structures. For security or privacy reasons, some applications, see e.g. [19], require data structures that do not reveal any information about historical states to an observer of the memory representation, i.e. evidence of keys that were deleted or evidence of the keys' insertion sequence. Data structures are called *uniquely represented* (UR) if they have this strong history independence property. For example, sorted arrays are UR, but the aforementioned search trees are not UR due to deterministic rebalancing. Early results [28, 29, 1] show lower and upper bounds on comparison-based search in UR data structures on the pointer machine. Using UR hash tables [9, 26] in the RAM model, pointer structures can also be mapped uniquely into memory. (See [9, Thm. 4.1], [10, Sec. 2].)

The Randomized Search Trees are defined by inserting keys, one at a time, in order of a random permutation into an initially empty *binary* search tree. The well-known Treap [27, 31] maintains the tree shape, of the permutation order, and supports efficient insertions, deletions, and searches (see also [25, Chapter 1.3]). Searches read with high probability $\mathcal{O}(\log n)$ tree

[1] B^ε-trees are usually called "write-optimized", though they aim to minimize IOs and not writes (Os).

Table 1 Bounds of bottom-up B-Treaps and the proposed top-down RBSTs.

UR Tree	Model		[# Blocks]	
B-Treap [20]	BEM	Size	$\mathcal{O}(n/\alpha)$	
		Update **(Os)**	$\mathcal{O}\left(\frac{1}{\delta}\log_\alpha n\right)$	, if $\alpha = \Omega\left(\ln^{\frac{1}{1-\delta}} n\right)$.
		Search **(Is)**		
(α,ε)-RBST	EM	Size	$(1+\mathcal{O}(\varepsilon))\ n/\alpha$	
		Update **(Os)**	$\mathcal{O}(\varepsilon^{-1}+\log_\alpha(n)/\alpha)$	
		Update **(Is)**	$\mathcal{O}(\varepsilon^{-2}+(1+\frac{\varepsilon^{-1}+\log n}{\alpha})\log_\alpha n)$	
		Search **(Is)**	$\mathcal{O}(\varepsilon^{-1}+\log_\alpha n)$	

nodes, any insertion (or deletion) performs expected $\mathcal{O}(1)$ rotations (writes), and the expected subtree weight, beneath the updated key, is $\mathcal{O}(\log n)$. Beside UR, these remarkably strong bounds are central in the design of simple MM algorithms for dynamic problems. The *constant writes bound* of Treaps, and some Red-Black Tree variants, is particularly valuable for persistent data structures [16, 22], that is $\mathcal{O}(1)$ amortized writes are *not sufficient* to design fully-persistent data structures with linear space. The *logarithmic weight bound* for example is particularly valuable for obtaining a $\mathcal{O}(1)$ query time for order-queries in dynamic sets with $\mathcal{O}(\log n)$ expected update time [12, Section 5]. This is a central sub-problem in designing fully-persistent data structures, see [18] for example. Further important examples include asymmetric read/write cost settings [8, 5] and parallel computation [11], or simply storage media that can only endure few write-cycles.

Golovin [20] introduced the B-Treap as the first UR data structure for Byte-addressable External Memory (BEM). The B-Treap originates from a certain block storage layout of an associated (binary) Treap, i.e. the block tree is obtained, bottom-up, by iteratively placing subtrees, of a certain size, in one block node[2]. The author shows bounds of B-Treaps, under certain parameter conditions: If α is at least polylogarithmic, specifically $\alpha = \Omega((\ln n)^{\frac{1}{1-\delta}})$ for some $\delta \in (0,1)$, then updates have expected $\mathcal{O}(\frac{1}{\delta}\log_\alpha n)$ block-writes and range-queries have expected $\mathcal{O}(\frac{1}{\delta}\log_\alpha n + k/\alpha)$ block-reads, where k is the output size (see Theorem 1 in [20]). The paper also discusses experimental data, from a non-dynamic implementation, that shows an expected load-factor close to 1/3 [20, Sec.6]. Though this storage layout approach allows leveraging known bounds of Treaps to analyze B-Treaps, the block size conditions are limiting for applications of B-Treaps in algorithms. Since the update algorithm is rather involved, Golovin [21] proposed a simpler, Skip-List based EM approach for a UR solution.

The known Skip-List based approaches [21] and [6] avoid the restrictions on α, however both require an involved form of history independent hash-table for strings of key values (see [21, Section 2.2.2]). Moreover, the B-Skip-List [21] provides only a logarithmic $\mathcal{O}(\log n)$ depth bound (see [6, Lemma 15]) and the EM Skip-List in [6] offers only a weaker notion of History Independence than UR and has *logarithmic* block-writes (cf. [6, proof of Lem.19]).

Contribution and Paper Outline. Our work provides the first UR EM search tree that is generally applicable for all block sizes $\alpha \geq 2$, whilst providing bounds that improve on the only known UR EM search tree solution, B-Treap [20].

[2] E.g. byte-addressable memory is required for intra-block child pointers and each key maintains a size field that stores its subtree size (in a binary Treap), resulting in logarithmic writes (cf. Os in Table 1).

We introduce a *new design technique*, and provide a self-contained analysis, that leads to a two-layer randomized data structure called (α, ε)-Randomized Block Search Trees (RBSTs), where α is the block size and $\varepsilon \in (0, 1/2]$ a tuning parameter for the space-efficiency of our secondary layer (see Section 2). Without the secondary structures, the RBSTs are precisely the distribution of trees that are obtained by inserting the keys, one at a time, under a random order into an initially empty block search tree (e.g. $\alpha = 1$ yields the Treap distribution). RBSTs are UR and the algorithms for searching are simple. In Section 2.1, we give a partial-rebuild algorithm for dynamic updates that occupies $\mathcal{O}(1)$ blocks of temporary MM storage and performs a number of writes (Os) to EM that is proportional to the structural change in the RBST. Its number of reads (Is) is bounded within a small factor of the writes. We prove three performance bounds for (α, ε)-RBSTs in terms of block-reads, block-writes, and blocks stored.

Section 3 shows that searching reads expected $\mathcal{O}(\varepsilon^{-1} + \log_\alpha n)$ blocks, and that RBST depth is with high probability $\mathcal{O}(\varepsilon^{-1} + \sqrt{\varepsilon^{-1} \ln n} + \log_\alpha n)$ for all $\alpha \geq 2$. If $\alpha = \text{polylog}(n)$, RBST depth is with high probability $\mathcal{O}(\varepsilon^{-1} + \log_\alpha n)$. Section 4 shows that (α, ε)-RBSTs have an expected asymptotic load-factor of at least $(1 - \varepsilon)$ for every $\varepsilon \in (0, 1/2]$. Combining both analysis techniques, we show in Section 5 that dynamic updates perform expected $\mathcal{O}(\varepsilon^{-1} + \log_\alpha(n)/\alpha)$ block-writes and expected $\mathcal{O}(\varepsilon^{-2} + (1 + \frac{\varepsilon^{-1} + \log n}{\alpha}) \log_\alpha n)$ block-reads.

See Table 1 for an overview of the write bounds (Os), and read bounds (Is), of the known bottom-up B-Treap and our proposed top-down RBST. To compare the update ***write-bounds*** (Os), note that RBSTs have *constant* $\mathcal{O}(\varepsilon^{-1})$ writes for all α that are admissible in B-Treaps, whereas B-Treaps write at least *logarithmic* $\Omega(\log_\alpha n)$ blocks. This is a significant asymptotic improvement. To compare the update ***read-bounds*** (Is), note that the factor $(1 + \frac{\log n}{\alpha}) = \mathcal{O}(1)$ for *all* $\alpha = \Omega(\ln^{\frac{1}{1-\delta}} n)$ with $\delta \in (0, 1)$ that are admissible in B-Treaps, whereas the factor $\frac{1}{\delta} \to \infty$ for α close to $\ln(n)$. That is, the RBST bounds improve on *all* B-Treap bounds. Thus, RBSTs are simple UR search trees, for *all* block sizes $\alpha \geq 2$, that simultaneously provide improved search, storage utilization, and write-efficiency, compared to the bottom-up B-Treap [20].

Comparing also with non-UR structures, (α, ε)-RBSTs are, to the best of our knowledge, the first EM structure that provides storage-efficiency and write-efficiency for *every* update. The decisive novelty of our tree structure is that all block-nodes of the primary layer use the *maximum* fan-out and that the block-nodes of the secondary layer use a *weight-proportional* fan-out, i.e. proportional to the number of keys that are stored in the subtree. The design of our secondary layer structures has thus *nothing* in common with the "message-buffers" in B^ε-trees. Our result is incomparable to IO-optimized B^ε-trees: Though RBSTs are UR and provide better bounds for space, searching (Is), and writes (Os), the block-reads (Is) for updates are $\Omega(\log_\alpha n)$ and thus worse than the IO-bound of the (non-UR) B^ε-Tree.

2 Randomized Block Search Trees

Unlike the well-known B-Trees that store between α and $\alpha/2$ keys in every block and have all leaves at equal depth, our definition of (α, ε)-RBSTs does not strictly enforce a minimum load for every single node. Like binary Treaps, the shape of an RBST $T_X(\pi)$, over a set X of n keys, is defined by an incremental process that inserts the keys $x \in X$ in a random order π into an initially empty block search tree structure, i.e. with ascending priority values $\pi(x)$. The actual algorithms for dynamic updates are discussed in Section 2.1.

This section defines the basic tree structure that supports both, reporting queries (i.e. range search) and order-statistic queries (i.e. range counting and searching the k-th smallest value). Section 2.1 then shows how to avoid explicitly maintaining the subtree weights

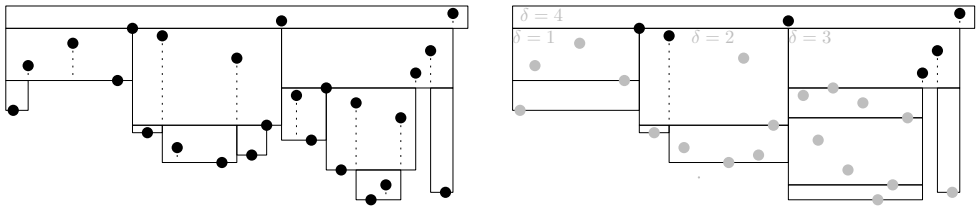

Figure 1 Cartesian representations of two RBSTs that shows the i-th smallest key $x_i \in X$ with priority value $\pi(i)$ as point $(i, -\pi(i))$ in the plane ($n = 25$ and $\alpha = 3$). (Cf. [31] for Cartesian Trees.) Active separators of a block v are shown with dotted vertical lines and the horizontal lines are $p^\dagger(v)$. The left tree has $\beta = 0$ and the right tree has buffering subtrees. Note that all, except the last block, on a root-to-leaf path are full.

of nodes close to the root, which yields improved update write-bounds for the case that order-statistic queries are not required in applications. Next, we define the structure of RBSTs and state their invariants.

Let array size $\alpha \geq 1$ and buffer threshold $\beta \geq 0$ be fixed integer parameters. Every RBST block node stores a fixed-size array for α keys, $\alpha + 1$ child pointers, and one parent pointer. We use the key-value x of minimum priority $\pi(x)$ from those keys contained in the node's array as the UR label of the block-node. Every child pointer is organized as a *pair*, storing the pointer to the child block and the total number of keys $s \geq 1$ contained in that subtree[3]. All internal blocks of an RBST are full, i.e. they store exactly α keys. Any block has one of two possible states, either ***non-buffering*** ($s \geq \alpha + \beta$) or ***buffering*** ($s < \alpha + \beta$). Though (α, ε)-RBSTs will use $\beta = \Theta(\alpha^2/\varepsilon)$, we first give the definition of the primary tree without the use of buffers ($\beta = 0$), i.e. every internal block remains non-buffering.

For $\beta = 0$, the tree shape is defined by the following incremental process that inserts the keys in ascending order of their priority $\pi(\cdot)$. Starting at the root block, the insertion of key x first locates the leaf block of the tree using the search tree property. If the leaf is non-full, x is emplaced in the sorted key array of the node. Otherwise, a new child block is allocated (according to the search tree property) and x is stored therein. This concludes the definition of the tree structure for $\beta = 0$.

(Invariant 1:) Thus, any internal block stores those α keys that have the smallest priority values in its subtree[4].

See Figure 1 (left) for an example on 25 keys with $\beta = 0$ that consists of 12 blocks. It demonstrates that leaves may merely store one key in their block. We address this storage inefficiency issue next.

The "high-level idea" for $\beta > 0$ is to delay generating new leaves until the subtree contains a sufficiently large number of keys. To this end, we use secondary structures, which we call ***buffers***[5], that replace the storage layout of all those subtrees that contain $< \alpha + \beta$ keys.

(Invariant 2:) Thus, all remaining block nodes of the primary structure are full and their children are either a primary block or a buffer.

There are several possible ways for UR organization of buffers, for example using a list of at most $\lceil (\alpha + \beta)/\alpha \rceil = \mathcal{O}(\alpha/\varepsilon)$ blocks that are sorted by key values. We propose the following structure that leads to stronger bounds.

[3] E.g. subtree weights are immediately known to the parent, without reading the respective child blocks.
[4] For example, $\alpha = 1$ and $\beta = 0$ yields the well-known Treap and $\alpha > 1$ search trees with fan-out $\alpha + 1$.
[5] Though we use the word "buffer" as it describes this purpose best, our definition has nothing to do with the "message buffers" in [13, 2].

Secondary UR Trees: Buffers with weight proportional fan-out. Our buffers are search trees that consist of nodes similar to the primary tree. They also store the keys with the α smallest priority values, from the subtree, in their block. However, the fan-out of internal nodes varies based on the number of keys $n \leq \alpha + \beta$ in the subtree. Define $\delta(n) = \min\{\alpha + 1, \max\{1, \lceil \frac{n-\alpha}{\rho} \rceil\}\}$ as the ***fan-out bound*** for a subtree of weight $n < \alpha + \beta =: \alpha + (\alpha + 1)\rho$. We defer the calibration of the parameter $\rho = \Theta(\alpha/\varepsilon)$ to the start of the proof of Theorem 9. The subtree weight of buffering nodes is maintained explicitly, with the aforementioned in-pointers. To obtain UR, the buffer layout is again defined recursively solely based on the set of keys in the buffer and the random permutation π.

For fan-out bound $\delta = 1$, the subtree root has at most one child, which yields a list of blocks. The keys inside each block are stored in ascending key order, and the blocks in the list have ascending priority values. That is, the first block contains those keys with the α smallest priorities, the second block contains, from the remaining $n - \alpha$ keys, those with the α smallest priorities, and so forth.

For fan-out bound $\delta \in [2, \alpha + 1]$, we also store the keys with the α smallest priorities in the root. We call those keys with the $\delta - 1$ smallest priority values the ***active separators*** of this block. The remaining $n - \alpha$ keys in the subtree are partitioned in δ sets of key values, using the active separators only. Each of these key sets is organized recursively, in a buffer, and the resulting trees are referenced as the children of the active separators.

This concludes the definition of our tree structure.

See the right part of Figure 1 for an example of the (α, ε)-RBST on the same permutation (y-coordinates) and 25 keys (x-coordinates) that consists of 11 blocks. (See [31] for Cartesian Tree representations.) The active separators are shown in black, non-active separators in gray, and the block nodes as rectangles.

Some remarks on the definition of primary and secondary RBST nodes are in order. The buffer is UR since the storage layout only depends on the priority order π of the keys, the number of keys in the buffer, and the order of the key values in it. Note that summing the state values of the child pointers yields $s_1 + \ldots + s_\delta + \alpha = n$ the weight of the subtree, without additional block reads. Moreover, whenever an update in the secondary structure brings the buffer's root above the threshold, i.e. $s_1 + \ldots + s_{\alpha+1} \geq \beta$, we have that its root contains those keys with the α smallest priorities form the set and all keys are active separators, i.e. the buffer root immediately provides both invariants of blocks from the primary tree.

Note that this definition of RBSTs not only naturally contains the randomized search trees of block size α, but also allows a *smooth trade-off* towards a simple list of blocks (respectively $\varepsilon = \infty$ and $\varepsilon = 0$). Though leaves are non-empty, it is possible that the tree shape defined by the random permutation π yields leaves with load as low as $1/\alpha$ in RBSTs.

As stated at the beginning of this section, the definition above does not yield an efficient algorithm to actually maintain RBSTs. The remainder of this section specifies our algorithms for searching, dynamic insertion, and dynamic deletion in RBSTs. Our analysis is presented in Sections 3, 4, and 5.

Successor and Range Search in (α, ε)-RBSTs. As with ordinary B-Trees, we determine with binary search on the array the successor of search value q, i.e. the smallest active separator key in the block that has a value of at least q, and follow the respective child pointer. If the current search node is buffering, then we additionally check the set of non-active separators in the block (i.e. all keys) during the search to find the actual successor of q from X. The search descends until the list, i.e. block nodes with fan-out ≤ 1, in the respective RBST buffer is found. Since the lists are sorted by priority, we check all keys, by scanning

the $\mathcal{O}(1/\varepsilon)$ blocks of the list, to determine the successor of q. This point-search extends to range-queries in the ordinary way by considering the two search paths of the range's boundary $[q, q']$.

To summarize, successor and range search occupies $\mathcal{O}(1)$ blocks in main memory, does not write blocks, and the number of block-reads is $\mathcal{O}(D)$, where D is the depth of the RBST.

2.1 Insertion and Deletion via Partial-Rebuilds

As with the Treaps, the tree shape of (α, ε)-RBSTs is solely determined by the permutation π of the keys. For $X \cup \{x\} = X'$, any update method that transforms $T_X(\pi)$ into $T_{X'}(\pi)$ and vice-versa are suitable algorithms for insertions and deletions. Unlike Treaps, that use rotations to always perform insertions and deletions at the leaf level, rotations in block search trees seem challenging to realize and analyze. We propose an update algorithm that rebuilds subtrees in a top-down fashion, which has vague similarities to rebalancing Scapegoat Trees.

The main difficulty for RBST update algorithms is to achieve IO-bounds for EM-access that are (near) proportional to the structural change in the trees, *while* having an $\mathcal{O}(1)$ worst-case bound on the temporary blocks needed in main memory. The *partial*-rebuild algorithm we introduce in this section has bounds on the number of block-writes (Os) in terms of the structural change of the update, the expected Os and Is are analyzed in Section 5.

▶ **Observation 1.** *Let π be a permutation on the key universe U and $X' = X \cup \{x\}$ the keys in the tree, after and before the insertion of x. Let m be the number of blocks in $T_X(\pi)$ that are different from the blocks in $T_{X'}(\pi)$ and m' the number of blocks in $T_{X'}(\pi)$ that are different from the blocks in $T_X(\pi)$. Let D and D' be the depth of the subtrees in $T_X(\pi)$ and $T_{X'}(\pi)$ that contain those blocks.*

The rebuild algorithm in this section performs $\mathcal{O}(m + m')$ block-writes to EM, reads $\mathcal{O}(m' + D' \cdot m)$ blocks, and uses $\mathcal{O}(1)$ blocks of temporary MM storage during the update.

Our rebuild-algorithm aims at minimizing the number of block-writes (Os) in an update, using a greedy top-down approach that builds the subtree of $T_{X'}$ in auxiliary EM storage while reading the keys in T_X from UR EM. On completion, we delete the m obsolete blocks from UR memory and write the m' result blocks back to UR EM to obtain the final tree $T_{X'}$. This way, our update algorithm will only require $\mathcal{O}(1)$ blocks of temporary storage in MM. The basic idea is as follows. Given a designated separator-interval $(\ell(u), r(u))$ when assembling block u for writing to auxiliary EM, we determine those keys with the α-smallest priority values by searching in T_X, place them in the array of u, determine the fan-out δ_u and the active separators of block node u, and recursively construct the remaining subtrees for those keys within each section between active separators of u. Eventually, we free the obsolete blocks of T_X form UR and move the resulting subtree from the auxiliary to UR memory.

Next we discuss all possible rebuild cases when ***inserting*** a new key x to the tree, their realization is discussed thereafter. For a block node v, let $X(v)$ denote the set of keys stored in the subtree of v, $B(v) \subseteq X(v)$ the keys stored in the array of v itself, $p^\dagger(v) = \max\{\pi(y) : y \in B(v)\}$, and $p^*(v) = \min\{\pi(y) : y \in B(v)\}$[6].

Consider the search path for x in T_X. Starting at the root, we stop at the first node v that i) must store x in its array (i.e. $p^\dagger(\text{parent}(v)) < \pi(x) \leq p^\dagger(v)$) or ii) that requires rebuilding (i.e. must increase its fan-out). To check for case ii), we use the augmented child

[6] For example, the key y with $\pi(y) = p^*(v)$ is the UR label of the block v. See also Figure 1 for $p^\dagger(v)$.

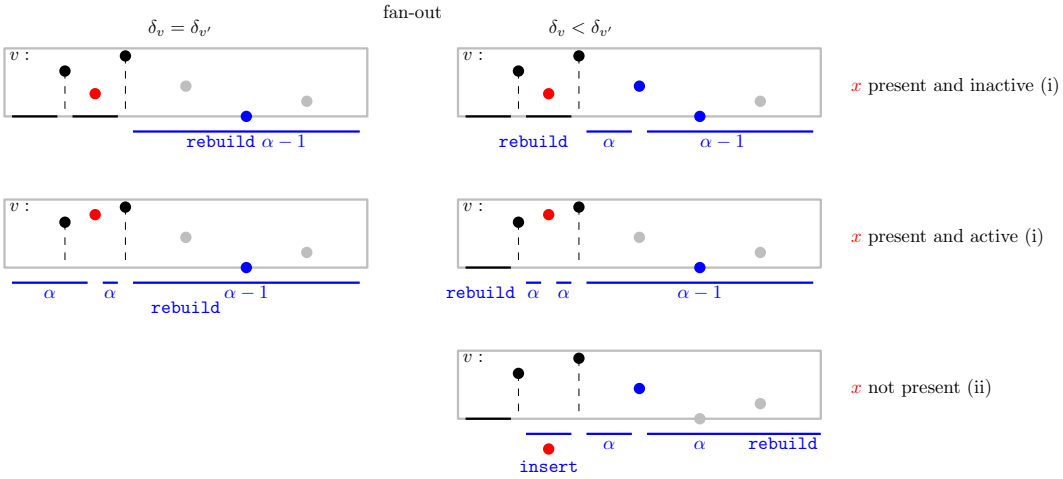

Figure 2 Rebuild cases (blue) for insertion of key x (red) in block v of an RBST.

references of buffering node v to determine the number of keys in the resulting subtree v'. Thus, both fan-outs, δ_v pre-insertion and $\delta_{v'}$ post-insertion, are known without additional block reads from the stored subtree weights.

If neither i) nor ii) occurs, the search halts in a buffer list ($\delta_v = \delta_{v'} \leq 1$) and we simply insert the key and rewrite all subsequent blocks of the list in one linear pass. Note that, if the list is omitted (as there are no keys in the active separator interval), the insertion must simply allocate a new block that solely stores x to generate the output RBST. It remains to specify the insertion algorithm for case i) and ii). See Figure 2 for an overview of all cases.

In ii), v is a buffer node that must increase its fan-out. It suffices to build two trees from one subtree, each containing the top α keys in their root. If the new key x is contained in this interval, we pass x as additional "carry" parameter to the procedure. If x is not contained in this interval, the insertion continues on the respective child of v that must contain x. (See Figure 2 bottom.) Note that ≤ 3 subtrees of v' are built in auxiliary EM, regardless of α.

In i), v is an internal node that stores α keys and there are four cases; depending on if x is an active separator in v' and if the fan-out stays or increases (see Figure 2). In all cases, it suffices to rebuild at most three subtrees. Two trees for the two separator-intervals that are incident to x and one tree that contains the two subtrees of the separator-intervals incident to the key y that is displaced from block $B(v)$ by the insertion of x, i.e. $\pi(y) = p^{\dagger}(v)$.

To complete the insertion algorithm, it remains to specify our top-down greedy rebuild procedure for a given interval between two active separators. First, we isolate the task of finding the relevant keys, needed for writing the next block to auxiliary EM within the following range-search function $\texttt{top}(k, v, (\ell, r))$. Given a subtree root v in T_X, a positive integer $k \leq \alpha$, and query range (ℓ, r), we compute those keys from $Y = X(v) \cap (\ell, r)$ that have the k smallest priority values, together with their respective new subtree weights $w_0 + \ldots + w_k = |Y| - k$, based on a naïve search that visits (in ascending key order) all descendants of v that overlap with the query range.

For our $k = \alpha$ and $k = \alpha - 1$ rebuilds, we run the respective `top`-query on the designated interval range and check if x is stored in the output block or beneath it. Then we determine the fan-out from the range count results $\{w_i\}$, which determines the active separators in the output block. We allocate one block in auxiliary EM for each non-empty child-reference and set their parent pointers. Finally, we recursively rebuild the subtrees for the intervals of the active sections, passing them the reference to the subtree root in T_X that holds all necessary

keys for rebuilding its subtree. Note that for this form of recursion, using parent pointers allows to implement the rebuild procedure such that only a constant number of temporary blocks of MM storage suffice. After the subtrees are built in the auxiliary EM, we delete the m old blocks from UR and move the m' blocks to UR to obtain the tree T'.

Note that ***deleting*** has the same cases for rebuilding sections of keys between active separators. Moreover, if a deletion leads to a buffering block ($\delta_{v'} < \alpha + 1$), then all its children are buffering and thus store their subtree weight together with the reference to it. Thus, determining the fan-out for deletions requires no additional block reads.

Clearly, the number of block writes is $\mathcal{O}(m + m')$ and the total number of block reads of all `top`-calls is $\mathcal{O}(D' \cdot m)$, from a charging argument: That is, any one block u of the m blocks in old tree is only read by a `top` call if the key range $(\ell(u), r(u))$ of u intersects the query range (ℓ, r) of the `top` call, i.e. $(\ell, r) \cap (\ell(u), r(u)) \neq \emptyset$. Consequently (ℓ, r) contains the smallest key, the largest key, or all keys of $B(u)$. The number of either such reads that is charged to a block u (from the m blocks) is bounded by D', since those blocks from the output tree that issue the calls to `top` have nested intervals $(\ell_1, r_1) \supset (\ell_2, r_2) \supset \dots$ that are contained in each other, by the search tree property (of the m' result blocks).

Our bounds for Observation 1 in terms of writes (Os) and reads (Is) will follow from our analysis of the depth and size of RBSTs in the next sections.

3 Bounds for Depth

Since successful tree searches terminate earlier, it suffices to analyze the query cost for keys $q \notin X$, i.e. unsuccessful searches. In this section, we bound the expected block reads for searching for some fixed q in the block trees T_X, where the randomness is over the permutations $\pi \in \mathrm{Perm}(n)$, i.e. the set of bijections $\pi : X \to \{1, \dots, n\}$.

Consider the sequence $v_1, v_2, \dots$ of blocks of an RBST on the search path from the root to some fixed q. Since the subtree weight is a (strictly) decreasing function, we have that the fan-out δ of the nodes is a non-increasing function that eventually assumes a value ≤ 1. From the definition of δ, we have that there are (in the worst case) at most $O(1/\varepsilon)$ blocks in the search path with $\delta = 1$. Thus, it suffices to bound the expected number of internal blocks with $\delta \geq 2$, which all have the search tree property $[\ell(v_i), r(v_i)] \supset (\ell(v_{i+1}), r(v_{i+1})) \ni q$.

▶ **Lemma 2.** *Let X be a set of n keys, $q \in \mathbb{R} \setminus X$, and $\alpha \geq 2$ an integer. The expected number of primary tree nodes ($\delta = \alpha + 1$) in an (α, ε)-RBST on X that contain q in their interval is $\mathcal{O}(\log_\alpha n)$. The expected number of secondary tree nodes ($\delta < \alpha + 1$) that have q in their interval is $\mathcal{O}(1/\varepsilon)$.*

In particular, the expected number of blocks in the search path of q is $\mathcal{O}(\varepsilon^{-1} + \log_\alpha n)$.

Further, the depth of an (α, ε)-RBST on X is with high probability less than $\mathcal{O}(\varepsilon^{-1} + \log n)$.

The block structure of RBSTs do not allow a clear extension of the Treap analysis based on "Ancestor-Events" [27] or the "Backward-Analysis" of point-location cost in geometric search structures [15, Chapter 6.4] to obtain bounds on block-reads (Is) that are stronger than the high-probability $O(\log n)$-depth bound of ordinary Treaps. Our proof technique is vaguely inspired by a top-down analysis technique of randomized algorithms for *static* problems that consider "average conflict list" size. However, several new ideas are needed to analyze a dynamic setting where the location q is user-specified and not randomly selected by the algorithm. Our proof uses a partition in layers whose sizes increase exponentially with factor α and a bidirectional counting process of keys in a layer of that partition.

Proof. Recall that there are in the worst case $\mathcal{O}(1/\varepsilon)$ nodes with fan-out $\delta \leq 1$ on the search path of q. Consider the active separators $\{x_1, \ldots, x_d\}$ of the RBST blocks that contain q in their search-interval. This set consists of two sequences $\{x_i^+\}$ and $\{x_i^-\}$ of strictly decreasing and increasing key values, respectively. Since their priority values are strictly increasing, i.e. $\pi(x_1) < \pi(x_2) < \ldots < \pi(x_d)$, we have from the known bounds of ordinary Treaps that $d = \mathcal{O}(\log n)$ with high probability [27, Lem. 4.8]. Thus, RBST depth is with high probability less than $\mathcal{O}(\varepsilon^{-1} + \log n)$.

To show our sharper search bound, let random variable D_q be the number of primary tree nodes v that contain q in their interval $[\ell(v), r(v)]$. Partition $\{1, \ldots, n\}$ with intervals of the form $[\alpha^i, \alpha^{i+1})$ for indices $0 \leq i \leq \lfloor \log_\alpha n \rfloor$. E.g. permutation π induces an assignment of keys $x \in X$ to an unique layer index, i.e. i with $\pi(x) \in [\alpha^i, \alpha^{i+1})$. For internal node v let $p^*(v) = \min\{\pi(x) : x \text{ is stored in } v\}$ and $p^\dagger(v)$ is the maximum over the set. Note that $p^*(v) + \alpha - 1 \leq p^\dagger(v)$, since every internal node stores exactly α keys. Thus, D_q counts nodes v that either have both $p^*(v), p^\dagger(v) \in [\alpha^i, \alpha^{i+1})$ or have that $p^\dagger(v)$ has a larger layer index than i. We bound the expected number of nodes of the first kind, since there are in the worst case at most $\lfloor \log_\alpha n \rfloor$ from the second kind. Defining for every layer index i a random variable V_i that counts the tree nodes in that layer

$$V_i(\pi) = \left| \left\{ v \in T_X(\pi) \ : \ \ell(v) < q < r(v) \text{ and } p^*(v), p^\dagger(v) \in [\alpha^i, \alpha^{i+1}) \right\} \right| \quad ,$$

we have that $D_q \leq \lfloor \log_\alpha n \rfloor + V$, where $V = \sum_{i \geq 0} V_i$ is the total number of blocks of the first kind. The remainder of the proof shows bounds for V_i.

Let $K_i = \{x \in X : \pi(x) < \alpha^{i+1}\}$ for each index i, $K_i^- = \{x \in K_i : x < q\}$, and $K_i^+ = K_i \setminus K_i^-$. Thus, $K_i \subset K_{i+1}$, and $|K_i| = \alpha^{i+1} - 1$. Define Y_i^- to be the number of consecutive keys form K_i^- that are less than q but not contained in K_{i-1}. Analogously, random variable Y_i^+ counts those larger than q and $Y_i := Y_i^- + Y_i^+$ is the number of consecutive keys of K_i, whose range contain q, but do not contain elements from K_{i-1}. Since all separators of primary nodes are active and internal nodes store exactly α keys, we have $V_i(\pi) \leq Y_i(\pi)/\alpha$ for every $\pi \in \mathrm{Perm}(n)$.

Next we bound $\mathbb{E}[Y_i^-]$ based on a sequence of binary events: Starting at q, we consider the elements $x \in K_i^-$ in descending key-order and count as "successes" if $x \in K_i \setminus K_{i-1}$ until one "failure" ($x \in K_{i-1}$) is observed. If K_i^- has no more elements, the experiment continues on K_i^+ in ascending key order. Defining Z_i as the number of successes on termination, we have $Y_i^-(\pi) \leq Z_i(\pi)$. The probability to obtain failure, after observing j successes, is $\frac{|K_{i-1}|}{|K_i|-j}$, which is at least $p_i := \frac{|K_{i-1}|}{|K_i|}$ for all $j \geq 0$ and $i > 0$.

Hence $\Pr[Z_i = j] \leq \Pr[Z_i' = j+1]$ where random variable $Z_i' \sim NB(1, p_i)$, i.e. the number of trials to obtain one failure in a sequence of independent Bernoulli trials with failure probability p_i. Since $\mathbb{E}[Z_i'] = 1/p_i$, we have $\mathbb{E}[Y_i^-] \leq \mathbb{E}[Z_i] < \mathbb{E}[Z_i'] = \frac{1}{p_i} = \frac{\alpha^{i+1}-1}{\alpha^i - 1} < \frac{\alpha}{1-1/\alpha^i}$.

We thus have $\mathbb{E}[V_i] \leq \mathbb{E}[Y_i]/\alpha < \frac{2}{\alpha} \frac{\alpha}{1-1/\alpha^i} \leq 4$ for all $\alpha \geq 2$, which shows the lemma's statement for the primary tree nodes.

Since there are in the worst case $\mathcal{O}(1/\varepsilon)$ nodes with fan-out $\delta \leq 1$ on a search path, it remains to bound the expected number of secondary nodes with a fan-out in $[2, \alpha]$ on a search path. If there are any such nodes to count, consider the top-most buffering node v and let $n' := |X(v)|$ be its subtree weight. For any fixed integer n', random variables regarding the subtree only depend on the relative order of the n' keys, which are uniform from $\mathrm{Perm}(n')$. Since the expected number of keys in either of the $\delta(n')$ sections is $(n'-\alpha)/\delta(n')$, the expected number of keys in the section of q has an upper bound of the form $(n' - \alpha)/\delta(n') = \mathcal{O}(\alpha/\varepsilon)$. Since the bound holds for all n', this bound holds unconditionally. The lemma's $\mathcal{O}(1/\varepsilon)$ expectation bound on the nodes follows from the fact that all secondary nodes (with $\delta \geq 2$) of a search path store exactly α keys. Thus, the expected number of such nodes is $\mathcal{O}(1/\varepsilon)$. ◀

Next, we show our depth bound that improves on the simpler $\mathcal{O}(\varepsilon^{-1} + \ln n)$ bound above. We first give a tail-estimate for the number of primary nodes and then an estimate for the number of secondary nodes on a search-path to the fixed query point q.

▶ **Lemma 3.** *Let $\alpha \geq 2$. The random variable D_q is with high probability $\mathcal{O}(\log_\alpha n)$.*

We will tighten above's analysis by considering the relative priority orders of the Y_i keys that are counted in layer i, where $i \leq r = \mathcal{O}(\log_\alpha n)$. Indeed, even if random variable Y_i exceeds its mean $\Theta(\alpha)$ by a large factor, the blocks of a search path to q can only contain all Y_i keys if also their relative priority-order have a, very specific, deep block structure. This observation allows to obtain a much stronger tail bound.

Proof. Let $\hat{Y}_i$ be the number of keys from $K_i \setminus K_{i-1}$ that are stored in the block of a node on the search path to q, i.e. $\hat{Y}_i \leq Y_i$. We will show for the random variables $\hat{Y}_i$ that $\Pr[\hat{Y}_i = c\alpha] < (e^2/c)^{c\alpha}$ to obtain a tailbound that is strong enough for a union bound of all $n+1$ possible queries $q \notin X$. Recall from Stirling's formula, that $1 < \frac{n!}{\sqrt{2\pi n}(n/e)^n} < e^{1/12n}$ for all $n \geq 1$.

To observe $c\alpha$ keys from layer i stored in c blocks of a search path, the α keys stored in the last block must all have priorities larger than the first $(c-1)\alpha$ keys, the α keys in the second to last block must all have priorities larger than the first $(c-2)\alpha$ keys, and so forth. Further, each of the $\alpha!$ relative orders of any one such batch of α keys does not affect them being stored in the block of a search path. Thus the probability that $c\alpha$ keys have relative priorities to form c blocks of a search path is at most

$$\frac{(\alpha!)^c}{(c\alpha)!} < \frac{(e^{\frac{1}{12\alpha}}\sqrt{2\pi\alpha}(\alpha/e)^\alpha)^c}{(c\alpha/e)^{c\alpha}} = e^{\frac{c}{12\alpha}}(2\pi\alpha)^{\frac{c}{2}}\left(\frac{\alpha/e}{c\alpha/e}\right)^{c\alpha} = e^{\frac{c}{12\alpha}}\left(\frac{\sqrt{2\pi\alpha}}{c^\alpha}\right)^c < e^{\frac{c}{12\alpha}}\left(\frac{e}{c}\right)^{\alpha c},$$

where the last inequality is due to $\sqrt{2\pi\alpha} < e\sqrt{\alpha}$ and $\frac{e\sqrt{\alpha}}{c^\alpha} \leq (e/c)^\alpha \Leftrightarrow \sqrt{\alpha} \leq e^{\alpha-1}$ being true for all $\alpha \geq 2$. Thus $\Pr[\hat{Y}_i = c\alpha] < (e^2/c)^{c\alpha}$.

For $c \geq e^3$, $\Pr[\hat{Y}_i = c\alpha] < (\frac{1}{e^\alpha})^c =: z^c$ and we have

$$\Pr[\hat{Y}_i \geq c\alpha] \leq \sum_{j \geq c} \sum_{0 \leq k < \alpha} \Pr[\hat{Y}_i = j\alpha + k] \leq \alpha \sum_{j \geq c} \Pr[\hat{Y}_i = j\alpha] \leq \alpha \sum_{j \geq c} z^j = \alpha \frac{z^c}{1-z} < 2\alpha/e^{\alpha c}.$$

From the layer partition, we have that the outcome of Y_i is independent of Y_{i+1}, i.e. bounds for Y_i are with respect to $|K_i|$ and regardless of the actual set K_i, and all relative priority-orders of the keys in $K_i \setminus K_{i-1}$ are equally likely. Since across all r layers the probability bound for the sum is maximized when all r factors have equal thresholds, we have

$$\Pr[\hat{Y}_1 + \ldots + \hat{Y}_r \geq c\alpha r] \leq \prod_{i=1}^{r} \Pr[\hat{Y}_i \geq c\alpha] < \left(\frac{2\alpha}{e^{\alpha c}}\right)^r \leq \frac{2\alpha \cdot n^{1+1/\log_2 \alpha}}{n^{c\alpha/\ln \alpha}} \leq \frac{n^3}{n^{c\alpha/\ln \alpha}} < n^{3-c},$$

where the third last inequality uses that $\log_\alpha(n+1) \leq r \leq \log_\alpha(n) + 1$, the second last inequality uses that $\alpha \geq 2$ and $2\alpha \leq n$, and and the last inequality uses that $\alpha/\ln \alpha > 1$. ◀

▶ **Lemma 4.** *The depth of a buffering subtree is with high probability $\mathcal{O}(\varepsilon^{-1} + \sqrt{\varepsilon^{-1} \log n})$. If $\alpha = \text{polylog}(n)$, then the depth of a buffering subtree is with high probability $\mathcal{O}(\varepsilon^{-1} + \log_\alpha n)$.*

We believe that Lemma 4 cannot be improved substantially without redesigning the secondary structures, update algorithms, and proofs. Specifically, parameter constellations with small $\delta(n) = \omega(1)$ provide a major obstacle in obtaining better bounds.

Proof. For the number of secondary tree nodes, which have $\delta_v \in [2, \alpha]$ and store exactly α keys, we observe that having d such nodes on a search path requires that the topmost node has one child with weight $N_1 \geq (d-1)\alpha + \rho$, and the second topmost node must have one child with weight $N_2 \geq (d-2)\alpha + \rho$, and so forth, where the RBST parameter $\rho = \Theta(\alpha/\varepsilon)$. Thus, at least $d/4$ nodes must have weight $N_i \geq (d/2)\alpha + \rho$, and consequently fan-out value at least $k \geq \delta((d/2)\alpha + \rho) = \Omega(\varepsilon d)$. Using our $\Pr[\delta(N_i) \geq k] \leq e^{2-k}$ bound of Corollary 17 on the fan-out of a child, we obtain the probability of having d such nodes on a search path is at most $\prod_{i=d/4}^{d/2} \exp(2 - \delta(N_i)) \leq e^{-\Omega(\varepsilon d^2)}$, i.e. a high-probability bound for all $d = \omega(1)$.

Thus, the number secondary nodes ($\delta_v \geq 2$) is at most $d = O(\log_\alpha n)$ with probability at least $1 - n^{-\Omega(\varepsilon \ln(n)/(\ln \alpha)^2)}$, and $d = O(\sqrt{\varepsilon^{-1} \ln n})$ with probability at least $1 - n^{-\Omega(1)}$. ◀

4 Bounds for Size using a Top-Down Analysis

Our analysis will frequently use the following characterization of the partitions of a set X of n keys that are induced by the α elements of the smallest priority values from the set.

▶ **Observation 5.** *There is a bijective mapping between the partitions on n keys, induced by the first α keys from $\pi \in \mathrm{Perm}(n)$, and the solutions to the equation $X_1 + X_2 + \cdots + X_{\alpha+1} = n - \alpha$, where the variables X_i are non-negative integers. This bijection implies that the solutions of the equation happen with the same probability.*

In other words, X_i is the number of keys in the i-th section beneath the root of T_X, where $1 \leq i \leq \alpha + 1$. Thus, X_i can be considered as the number of consecutive keys from X that are between the i-th and $(i+1)$-th key stored in the root. For example, in an RBST on the keys $\{1, \ldots, 10\}$ and block size $\alpha = 3$, a root block consisting of the key values 4, 7, 8 is characterized by the assignment $X_1 = 3,\ X_2 = 2,\ X_3 = 0,\ X_4 = 2$. Next we analyze the effect of our secondary structures, since the size of RBSTs without buffer ($\beta = 0$) can be dramatically large. E.g. the expected number of blocks $\mathbb{E}[S]$ for subtrees of size $n = 2\alpha$ is

$$\begin{aligned}\mathbb{E}[S] &= 1 + \Pr[X_1 > 0] + \cdots + \Pr[X_{\alpha+1} > 0] \\ &= 1 + (\alpha+1)\left(1 - \binom{n-1}{\alpha-1} / \binom{n}{\alpha}\right) = 1 + (\alpha+1)(1 - \alpha/n) = 1 + (\alpha+1)/2 \quad .\end{aligned}$$

In this example, buffering stores the whole subtree using only one additional block.

4.1 Buffering for UR trees with load-factor $1 - \varepsilon$

Our top-down analysis of the expected size, and thus load-factors, of (α, ε)-RBSTs frequently uses the following counting and index exchange arguments.

▶ **Lemma 6** (Exchange). *We have $\Pr[X_i < c] = \Pr[X_j < c]$ for all $i, j \in \{1, \ldots, \alpha + 1\}$.*

Proof. Due to Observation 5, each solution to $X_1 + \cdots + X_{\alpha+1} = n - \alpha$ occurs with the same probability. Hence, we can calculate $\Pr[X_i \geq c]$ by counting the number of solutions where $X_i \geq c$ and dividing it by the total number of solutions.

Thus, $\Pr[X_i < c] = 1 - \Pr[X_i \geq c] = 1 - \binom{n-c}{\alpha} / \binom{n}{\alpha} = 1 - \Pr[X_j \geq c] = \Pr[X_j < c]$. ◀

Next, we show our size bound for an (α, ε)-RBST with n keys, where the priorities are from a uniform distribution over the permutations in $\mathrm{Perm}(n)$. Our analysis crucially relies on the basic fact that restricting $\mathrm{Perm}(n)$ on an arbitrary key subset of cardinality $n' < n$ yields the uniform distribution on $\mathrm{Perm}(n')$. Random variable S_n denotes the space, i.e. the

number of blocks used by the RBST on a set of n keys, F_n denotes the number of full, and $E_n := S_n - F_n$ the number of non-full blocks. Next we show a probability bound for the event that a given section, say the k-th, contains a number of keys X_k of a certain range.

▶ **Lemma 7.** *For any $1 \le k \le \alpha + 1$ and $i < j$, we have $\sum_{x=i}^{j} \Pr[X_k = x] = \frac{\binom{n-i}{\alpha} - \binom{n-1-j}{\alpha}}{\binom{n}{\alpha}}$.*

Proof. We have $\Pr[X_k \ge i] = \binom{n-i}{\alpha}/\binom{n}{\alpha}$ and $\Pr[X_k \ge j+1] = \binom{n-(j+1)}{\alpha}/\binom{n}{\alpha}$. ◀

These basic facts allow us to compute the following expressions.

▶ **Lemma 8.** *We have $\sum_{x=1}^{m} \Pr[X_k = x]x = \left(\binom{n}{\alpha+1} - \binom{n-m}{\alpha+1} - m\binom{n-1-m}{\alpha}\right)/\binom{n}{\alpha}$.*

Proof. By elementary computation, we have

$$\begin{aligned}\sum_{x=1}^{m} \Pr[X_k = x]x &= \sum_{x=1}^{m} \Pr[X_k = x] + \sum_{x=2}^{m} \Pr[X_k = x] + \ldots + \sum_{x=m}^{m} \Pr[X_k = x] \\ &= \frac{\binom{n-1}{\alpha} - \binom{n-1-m}{\alpha}}{\binom{n}{\alpha}} + \ldots + \frac{\binom{n-m}{\alpha} - \binom{n-1-m}{\alpha}}{\binom{n}{\alpha}} \\ &= \sum_{i=1}^{m} \frac{\binom{n-i}{\alpha}}{\binom{n}{\alpha}} - m\frac{\binom{n-1-m}{\alpha}}{\binom{n}{\alpha}} \ .\end{aligned} \tag{1}$$

Note that an RBST on $n' = n+1$ keys with block size $\alpha' = \alpha + 1$ has $\Pr[X'_1 = i] = \frac{\binom{n-i}{\alpha}}{\binom{n+1}{\alpha+1}}$. Thus, we have from Lemma 7 that

$$\begin{aligned}\sum_{i=1}^{m} \Pr[X'_1 = i] = \sum_{i=1}^{m} \frac{\binom{n-i}{\alpha}}{\binom{n+1}{\alpha+1}} &= \frac{\binom{(n+1)-1}{\alpha+1} - \binom{(n+1)-1-m}{\alpha+1}}{\binom{n+1}{\alpha+1}} \\ \Rightarrow \sum_{i=1}^{m} \binom{n-i}{\alpha} &= \binom{n}{\alpha+1} - \binom{n-m}{\alpha+1}\end{aligned} \tag{2}$$

and the lemma follows by using (2) for the last summation in (1). ◀

We are now ready to state our size bound.

▶ **Theorem 9** (Size). *The expected number of non-full blocks in an n-key (α, ε)-RBST is at most $\mathbb{E}[E_n] \le \max\{\varepsilon n/\alpha,\ 1\}$.*

Our proof is by induction on n, where the base cases are due to the number of non-full blocks that are occupied by the secondary buffer structures (Appendix A).

Proof. Observation 18 states that $E_n \le 1$ for all n with $\delta(n) \le 1$. Moreover, Theorem 21 shows that $\mathbb{E}[E_n] \le 27\delta(n)$ for $n \le \beta + \alpha = (\alpha+1)\rho + \alpha$. (Recall that $\beta = (\alpha+1)\rho$.) Next, we set ρ sufficiently large for the following proof. For simplicity, we will use t and γ instead of $\rho + \alpha$ and $(\alpha+1)\rho + \alpha$ respectively. We can conclude from Theorem 21 that for $t < n \le \gamma$,

$$\mathbb{E}[E_n] \le 27\left\lceil \frac{n-\alpha}{\rho} \right\rceil = 27\left\lceil \frac{n-\alpha}{108\alpha/\varepsilon} \right\rceil \le 27\left(\frac{n-\alpha}{108\alpha/\varepsilon} + 1\right) \le 27 \cdot 2\frac{n-\alpha}{108\alpha/\varepsilon} = \frac{n-\alpha}{2\alpha/\varepsilon} \ .$$

The last inequality holds because $n > \rho + \alpha = 108\alpha/\varepsilon + \alpha$ and $\frac{n-\alpha}{108\alpha/\varepsilon} > 1$.

For $n > \gamma$, we prove the theorem by induction. For each index $i \in \{1, \ldots, \alpha+1\}$, define Y_i as the number of non-full blocks in the i-th section beneath the root (of an RBST with n keys). Note that Y_i only depends on X_i, i.e. the number of keys in the i-th section, and their relative priorities. We thus have

$$\begin{aligned}
\mathbb{E}[Y_i] &= \sum_{x_i=1}^{n-\alpha} \Pr[X_i = x_i] \cdot \mathbb{E}[E_{x_i}] \\
&\leq \sum_{x_i=1}^{t} \Pr[X_i = x_i] \cdot 1 + \sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \frac{x_i}{2\alpha/\varepsilon} + \sum_{x_i=\gamma+1}^{n-\alpha} \Pr[X_i = x_i] \frac{x_i}{\alpha/\varepsilon} \\
&= \left(\sum_{x_i=1}^{t} \Pr[X_i = x_i] + \sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \right) - \sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \\
&+ \left(\sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \frac{x_i}{2\alpha/\varepsilon} - \sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \frac{x_i}{\alpha/\varepsilon} \right) - 2 \sum_{x_i=1}^{t} \Pr[X_i = x_i] \frac{x_i}{2\alpha/\varepsilon} \\
&+ \left(\sum_{x_i=1}^{t} \Pr[X_i = x_i] \frac{x_i}{\alpha/\varepsilon} + \sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \frac{x_i}{\alpha/\varepsilon} + \sum_{x_i=\gamma+1}^{n-\alpha} \Pr[X_i = x_i] \frac{x_i}{\alpha/\varepsilon} \right) \\
&= \left(\sum_{x_i=1}^{\gamma} \Pr[X_i = x_i] - \sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \frac{x_i}{2\alpha/\varepsilon} - \sum_{x_i=1}^{t} \Pr[X_i = x_i] \frac{x_i}{2\alpha/\varepsilon} \right) \\
&+ \sum_{x_i=1}^{n-\alpha} \Pr[X_i = x_i] \frac{x_i}{\alpha/\varepsilon} - \sum_{x_i=1}^{t} \Pr[X_i = x_i] \frac{x_i}{2\alpha/\varepsilon} - \sum_{x_i=t+1}^{\gamma} \Pr[X_i = x_i] \\
&\leq \sum_{x_i=1}^{\gamma} \Pr[X_i = x_i] \left(1 - \frac{x_i}{2\alpha/\varepsilon} \right) + \frac{\mathbb{E}[X_i]}{\alpha/\varepsilon}
\end{aligned}$$

Using Lemma 6, we have that the summation term has equal value for each section $i \in \{1, \ldots, \alpha+1\}$. Since, for $n \geq \alpha$, the root is full and not counted in E_n, we have

$$\mathbb{E}[E_n] = \mathbb{E}[Y_1 + ... + Y_{\alpha+1}] \leq (1+\alpha) \sum_{x_1=1}^{\gamma} \Pr[X_1 = x_1] \left(1 - \frac{x_1}{2\alpha/\varepsilon} \right) + \frac{n-\alpha}{\alpha/\varepsilon} \ . \tag{3}$$

To proof the inequality it suffices to show that the right-hand side of (3) is at most $\frac{\varepsilon n}{\alpha}$. This is true if and only if $(1+\alpha) \sum_{x_1=1}^{\gamma} \Pr[X_1 = x_1] \left(1 - \varepsilon \frac{x_1}{2\alpha}\right) \leq \varepsilon$. Since $\varepsilon > 0$ and $\alpha + 1 > 0$, it suffices to show that the value of the sum is not positive. This is if and only if

$$\sum_{x_1=1}^{\gamma} \Pr[X_1 = x_1] \leq \frac{\varepsilon}{2\alpha} \sum_{x_1=1}^{\gamma} \Pr[X_1 = x_1] x_1 \ . \tag{4}$$

Using Lemma 7 on the left-hand and Lemma 8 on the right-hand side, it remains to show

$$\frac{2\alpha}{\varepsilon} \binom{n-1}{\alpha} - \frac{2\alpha}{\varepsilon} \binom{n-1-\gamma}{\alpha} \leq \binom{n}{\alpha+1} - \binom{n-\gamma}{\alpha+1} - \gamma \binom{n-1-\gamma}{\alpha} \ . \tag{5}$$

By elementary computation, this inequality holds for all $n > \gamma$. (See Appendix B.) ◀

Since each full block contains α distinct keys, i.e. $F_n \leq n/\alpha$, we showed that $S_n \leq (1+\varepsilon)n/\alpha$. Thus, the load-factor, i.e. the relative utilization of the allocated blocks, has:

▶ **Corollary 10** (Load-Factor). *Let $\varepsilon \in (0, 1/2]$ and $n \geq \alpha + \beta$. The expected number of blocks in an n-key (α, ε)-RBST is at most $(1+\varepsilon)n/\alpha$, i.e. the expected load-factor is at least $1 - \varepsilon$.*

Proof. The load-factor is $L = n/\alpha S_n$. Since $\phi(x) = 1/x$ is a convex function, Jensen's inequality gives $1/\mathbb{E}[X] \leq \mathbb{E}[1/X]$. Using Thm. 9 yields $\mathbb{E}[L] \geq \frac{n}{\alpha}\frac{1}{(1+\varepsilon)n/\alpha} = \frac{1}{1+\varepsilon} \geq 1-\varepsilon$. ◀

Combining Lemma 2 and Theorem 9, we obtained the following for the range-search.

▶ **Corollary 11** (Range-Search). *Let $\varepsilon \in (0, 1/2]$. The expected number of block-reads for range reporting in (α, ε)-RBSTs is $\mathcal{O}(\varepsilon^{-1} + k/\alpha + \log_\alpha n)$, where k is the number of results.*

5 Bounds for Dynamic Updates via Partial-Rebuilding

The weight-proportional fan-out in the design of our secondary UR tree structure has the advantage that it avoids the need of additional restructuring work whenever a subtree must change its state between buffering and non-buffering, due to updates (cf. Section 2.1). Together with the space bound from last section and the observation that the partial-rebuild algorithm for updating RBSTs only rebuilds at most *three* subtrees beneath the affected node, regardless of its fan-out, allows us to show our bound on the expected structural change.

Using the counting technique from Lemma 2, it is easy to show that $\mathbb{E}|X(v)|$ is at most $\frac{2}{n}\sum_{i=1}^{n} \alpha\frac{n}{i} = \mathcal{O}(\alpha \ln n)$. The proof of the theorem will use the following, stronger bound.

▶ **Lemma 12** (Subtree Weight). *Let x be a key and random variable $|X(v)|$ denote the subtree weight of node v that stores $x \in B(v)$. Then $\sum_{N\geq\alpha+\beta}^{n} N \Pr[|X(v)| = N] = \mathcal{O}(\alpha \log_\alpha n)$.*

Proof. Let random variable $T \subseteq X$ be the largest set of consecutive keys that have $\pi(x) = \min_{x'\in T}\{\pi(x')\}$ and $x \in T$. Note that $T \subseteq X(v) \subseteq X$, and $T = X(v)$ iff $\pi(x) = p^*(v)$. Recall from binary Treaps that $\mathbb{E}|T| = \Theta(\ln n)$ for any $x \in X$, i.e. $H(n) \leq \mathbb{E}|T| \leq 2H(n/2)$ where $H(m) = \frac{1}{1} + \ldots + \frac{1}{m}$. We will show an upper bound of $\mathcal{O}(\frac{\alpha}{\log\alpha})\mathbb{E}|T|$ by arguing for the case $|X(v)| = N$. Indeed, if $\mathbb{E}\Big[|T|\Big||X(v)| = N\Big] = N\Omega(\frac{\log\alpha}{\alpha})$ holds for given $N \geq \alpha + \beta$, then

$$\sum_{N=\alpha+\beta}^{n} \Pr[|X(v)| = N]N = \sum_{N=\alpha+\beta}^{n} \Pr[|X(v)| = N]\mathbb{E}\Big[|T|\Big||X(v)| = N\Big]/\Omega(\log(\alpha)/\alpha)$$
$$= \mathcal{O}(\frac{\alpha}{\log\alpha}) \sum_{N=\alpha+\beta}^{n} \Pr[|X(v)| = N]\mathbb{E}\Big[|T|\Big||X(v)| = N\Big] \leq \mathcal{O}(\frac{\alpha}{\log\alpha})\mathbb{E}[T] \ .$$

Observe that x partitions the N keys in $X(v)$ into N^- keys smaller than x and N^+ keys larger than x, where $N^- + 1 + N^+ = N$. Though the priority orders of a set $X(v)$ of N keys are restricted to those that have $x \in B(v)$, we have that all relative orders of the keys in $B(v)$ remain equally likely. Consider the case that $B(v) = \{x_i^-, \ldots, x_1^-, x, x_1^+, \ldots, x_{\alpha-1-i}^+\}$ contains exactly i many separators smaller than x for some given integer $i \in [0, \alpha)$, i.e. there are $\alpha - 1 - i$ separators larger than x. Let the random variables $X_i^-, \ldots, X_0^- \subseteq X(v)$ denote the key sets of the respective sections smaller than x and $X_0^+, X_1^+, \ldots$ for the sections larger than x. That is, we have $\sum_j 1 + |X_j^-| = N^- + 1$ and $\sum_j 1 + |X_j^+| = N^+ + 1$ regardless of the priority orders. We consider the relative π-orders of those $i+1$ separators $\{x_i^-, \ldots, x_1^-\} \cup \{x\}$ in $B(v)$ and use the fact (regarding ancestors in binary Treaps) that the entire keys of the j-th section are in $X_j^- \subseteq T$ if $\pi(x) = \min\{\pi(x), \pi(x_j^-), \ldots, \pi(x_1^-)\}$, which has probability $\frac{1}{j+1}$. Now, using that $\mathbb{E}\Big[|X_j^-|\Big|N, N^-, i\Big] = \mathbb{E}\Big[|X_0^-|\Big|N, N^-, i\Big] = \frac{N^- - i}{i+1} \geq 0$ for all j, we have

$$\mathbb{E}\Big[|T|\Big|N,N^-,i\Big]-1=\mathbb{E}[T^-|N,N^-,i]+\mathbb{E}[T^+|N,N^-,i]$$
$$=\sum_{j=0}^{i}\frac{1}{i+1}\left(1+\mathbb{E}\Big[|X_j^-|\Big|N,N^-,i\Big]\right)+\sum_{j=0}^{\alpha-i}\frac{1}{\alpha-i}\left(1+\mathbb{E}\Big[|X_j^+|\Big|N,N^-,i\Big]\right)$$
$$=\frac{N^-+1}{i+1}\sum_{j=0}^{i}\frac{1}{j+1}+\frac{N^++1}{\alpha-i}\sum_{j=0}^{\alpha-i-1}\frac{1}{j+1}=(N^-+1)\frac{H(i+1)}{i+1}+(N^++1)\frac{H(\alpha-i)}{\alpha-i}.$$

Since $\frac{H(i+1)}{i+1}\geq\frac{H(\alpha)}{\alpha}$, we have that $\mathbb{E}\Big[|T|\Big|N,\ i\Big]=N\Omega(\frac{\ln\alpha}{\alpha})$, thus $\mathbb{E}\Big[|T|\Big|N\Big]=N\Omega(\frac{\ln\alpha}{\alpha})$. ◀

▶ **Theorem 13.** *The expected number of writes (Os) of an update in an n-key (α,ε)-RBST is $\mathcal{O}(\varepsilon^{-1}+\log_\alpha(n)/\alpha)$. Updates have $\mathcal{O}(\varepsilon^{-2}+(1+\frac{\varepsilon^{-1}+\log n}{\alpha})\log_\alpha n)$ expected reads (Is).*

Recall that our algorithm reads blocks only from one search path and during the `top`-queries. To bound $\mathbb{E}[m+m']$ and $\mathbb{E}[D'\cdot m]$ from Obs. 1, we first analyze the writes (Os). This extends to the reads (Is), using a high-probability bound for depth (e.g. Lemma 2 or Lemma 3 and 4), since both random variables are in the worst-case $\mathcal{O}(n)$.

Proof. We show our bound on the expected total structural change for deleting a key x from the RBST. (Insertion of the key would count the same number of blocks.) Let $X'=X\cup\{x\}$ be the set of keys before and after the deletion of x respectively, where $|X'|=n$. Let v be the subtree root that is subject to the partial rebuild algorithm from Section 2.1. From Lemma 2, determining the search path of v takes expected $\mathcal{O}(\varepsilon^{-1}+\log_\alpha n)$ reads (Is). Either v is a primary node with $\delta_v=\alpha+1$ or v is a buffering node with $\delta_v\leq\alpha+1$.

For $\delta_v\leq 1$, we have in the worst-case at most $\mathcal{O}(1/\varepsilon)$ blocks in the buffer's list.

For a primary node with $\delta_v=\alpha+1$, v contains x in its block $B(v)$. From Lemma 12, we have that $\sum_{N\geq\alpha+\beta}^{n}N\Pr[|X(v)|=N]=\mathcal{O}(\alpha\log_\alpha n)$. Now, for any given integer N of keys in the subtree, we have from Theorem 9 that the expected number of blocks is $\mathcal{O}(N/\alpha)$ and, from linearity of expectation, that three of the $\alpha+1$ subtrees contain at most $\mathcal{O}(N/\alpha^2)$ blocks. Thus, the contribution towards the expected number of block-writes is at most $\sum_{N\geq\alpha+\beta}\Pr[|X(v)|=N]\ N\cdot\mathcal{O}(1/\alpha^2)=\mathcal{O}(\log_\alpha(n)/\alpha)$. Moreover, using the high-probability bound of Lemma 2, we have that the depth of the subtree of v is less than $\mathcal{O}(\varepsilon^{-1}+\ln n)$ for any given integer $N=|X(v)|\leq n$. Thus, the contribution towards the total number of block-reads, issued by the `top`-queries of the update operation (cf. Observation 1), is at most $\sum_{N\geq\alpha+\beta}\Pr[|X(v)|=N]\ N\cdot(\varepsilon^{-1}+\ln n)\mathcal{O}(1/\alpha^2)=\mathcal{O}(\varepsilon^{-1}\log_\alpha(n)/\alpha+\frac{\ln n}{\alpha}\log_\alpha n)$.

For a buffering node with $\delta_v\geq 2$, v has $|X(v)|=\Theta(\alpha\delta_v/\varepsilon)$. By Theorem 21, subtree rooted at v has expected $\mathcal{O}(\delta_v/\varepsilon)$ blocks and writing three, of its δ_v subtrees, has $\mathbb{E}[m]=\mathcal{O}(\varepsilon^{-1})$ expected block-writes. For the block-reads, consider the number of keys in some active section X_i of v. If $X_i=\Theta(c\alpha/\varepsilon)$ for some $c\geq 2$, then the worst-case depth of this child is $\mathcal{O}(c/\varepsilon)$. By Lemma 14, the probability is at most $\mathcal{O}(e^{-c})$. Thus, $\sum_{c=2}^{\delta_v}\Pr[\delta(X_i)=c]\mathcal{O}(c/\varepsilon)\mathbb{E}[\ m\mid\delta(X_i)=c\ ]=\sum_{c=2}^{\delta_v}\frac{c^2}{e^c}\mathcal{O}(1/\varepsilon^2)=\mathcal{O}(\varepsilon^{-2})\sum_{c>1}c^2/e^c$ shows that the expected block-reads are $\mathcal{O}(\varepsilon^{-2})$ for either of the three affected subtrees of v. ◀

References

1 Arne Andersson and Thomas Ottmann. New Tight Bounds on Uniquely Represented Dictionaries. *SIAM J. Comput.*, 24(5):1091–1101, 1995. doi:10.1137/S0097539792241102.

2 Lars Arge. The buffer tree: A technique for designing batched external data structures. *Algorithmica*, 37(1):1–24, 2003. doi:10.1007/s00453-003-1021-x.

3 Ricardo A. Baeza-Yates. The expected behaviour of b+-trees. *Acta Informatica*, 26(5):439–471, 1989. doi:10.1007/BF00289146.

4 Rudolf Bayer and Edward M. McCreight. Organization and maintenance of large ordered indices. *Acta Informatica*, 1:173–189, 1972. doi:10.1007/BF00288683.

5 Naama Ben-David, Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, Charles McGuffey, and Julian Shun. Parallel algorithms for asymmetric read-write costs. In *Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA'16)*, pages 145–156, 2016. doi:10.1145/2935764.2935767.

6 Michael A. Bender, Jonathan W. Berry, Rob Johnson, Thomas M. Kroeger, Samuel McCauley, Cynthia A. Phillips, Bertrand Simon, Shikha Singh, and David Zage. Anti-persistence on persistent storage: History-independent sparse tables and dictionaries. In *Proc. 35th Symposium on Principles of Database Systems (PODS'16)*, pages 289–302. ACM, 2016. doi:10.1145/2902251.2902276.

7 Michael A. Bender, Martin Farach-Colton, Jeremy T. Fineman, Yonatan R. Fogel, Bradley C. Kuszmaul, and Jelani Nelson. Cache-oblivious streaming b-trees. In *Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA'07)*, pages 81–92. ACM, 2007. doi:10.1145/1248377.1248393.

8 Guy E. Blelloch, Jeremy T. Fineman, Phillip B. Gibbons, Yan Gu, and Julian Shun. Sorting with asymmetric read and write costs. In *Proc. Symposium on Parallelism in Algorithms and Architectures (SPAA'15)*, pages 1–12, 2015. doi:10.1145/2755573.2755604.

9 Guy E. Blelloch and Daniel Golovin. Strongly History-Independent Hashing with Applications. In *Proc. 48th Symposium on Foundations of Computer Science (FOCS'07)*, 2007. doi:10.1109/FOCS.2007.36.

10 Guy E. Blelloch, Daniel Golovin, and Virginia Vassilevska. Uniquely Represented Data Structures for Computational Geometry. In Joachim Gudmundsson, editor, *Proc. 11th Scandinavian Workshop on Algorithm Theory (SWAT'08)*, pages 17–28. Springer, 2008. doi:10.1007/978-3-540-69903-3_4.

11 Guy E. Blelloch and Margaret Reid-Miller. Fast set operations using treaps. In *Proc. 10th Symposium on Parallel Algorithms and Architectures (SPAA'98)*, pages 16–26, 1998. doi:10.1145/277651.277660.

12 Milutin Brankovic, Nikola Grujic, André van Renssen, and Martin P. Seybold. A simple dynamization of trapezoidal point location in planar subdivisions. In *Proc. 47th International Colloquium on Automata, Languages, and Programming (ICALP'20)*, pages 18:1–18:18, 2020. doi:10.4230/LIPIcs.ICALP.2020.18.

13 Gerth Stølting Brodal and Rolf Fagerberg. Lower bounds for external memory dictionaries. In *Proc. 14th Symposium on Discrete Algorithms (SODA'03)*, pages 546–554, 2003. URL: https://dl.acm.org/doi/10.5555/644108.644201.

14 Douglas Comer. The ubiquitous b-tree. *ACM Comput. Surv.*, 11(2):121–137, 1979. doi:10.1145/356770.356776.

15 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications, 3rd Edition*. Springer, 2008. doi:10.1007/978-3-540-77974-2.

16 James R. Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989. doi:10.1016/0022-0000(89)90034-2.

17 Amr Elmasry, Mostafa Kahla, Fady Ahdy, and Mahmoud Hashem. Red-black trees with constant update time. *Acta Informatica*, 56(5):391–404, 2019. doi:10.1007/s00236-019-00335-9.

18 Eric Demaine et al. Lecture notes on persistent data structures. https://web.archive.org/web/20230601205149/http://courses.csail.mit.edu/6.851/spring21/scribe/lec1.pdf.

19 Daniel Golovin. *Uniquely represented data structures with applications to privacy*. PhD thesis, Carnegie Mellon University, 2008.

20 Daniel Golovin. B-Treaps: A Uniquely Represented Alternative to B-Trees. In *Proc. 36th International Colloquium on Automata, Languages, and Programming (ICALP'09)*, pages 487–499, 2009. doi:10.1007/978-3-642-02927-1_41.

21 Daniel Golovin. The b-skip-list: A simpler uniquely represented alternative to b-trees. *CoRR*, abs/1005.0662, 2010. arXiv:1005.0662.

22 Dana Jansens. Persistent binary search trees. https://web.archive.org/web/20230526175144/https://cglab.ca/~dana/pbst/, 2014. Accessed: 2023-05-26.

23 Donald E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.

24 Klaus Küspert. Storage Utilization in B*-Trees with a Generalized Overflow Technique. *Acta Informatica*, 19:35–55, 1983. doi:10.1007/BF00263927.

25 Ketan Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, 1994.

26 Moni Naor and Vanessa Teague. Anti-presistence: history independent data structures. In *Proc. 33rd Symposium on Theory of Computing (STOC'01)*, pages 492–501, 2001. doi:10.1145/380752.380844.

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

28 Lawrence Snyder. On uniquely represented data strauctures. In *Proc. 18th Symposium on Foundations of Computer Science (SFCS'77)*, pages 142–146, 1977. doi:10.1109/SFCS.1977.22.

29 Rajamani Sundar and Robert Endre Tarjan. Unique Binary-Search-Tree Representations and Equality Testing of Sets and Sequences. *SIAM J. Comput.*, 23(1):24–44, 1994. doi:10.1137/S0097539790189733.

30 Jeffrey Scott Vitter. External memory algorithms and data structures. *ACM Comput. Surv.*, 33(2):209–271, 2001. doi:10.1145/384192.384193.

31 Jean Vuillemin. A unifying look at data structures. *Commun. ACM*, 23(4):229–239, 1980. doi:10.1145/358841.358852.

32 Andrew Chi-Chih Yao. On Random 2-3 Trees. *Acta Informatica*, 9:159–170, 1978. doi:10.1007/BF00289075.

A Induction Base Case: Expected Size of Buffering Trees

▶ **Lemma 14.** *Let $X_1, \ldots, X_m$ be some non-negative integral random variables where $X_1 + X_2 + \cdots + X_m = n$. For integers $n \geq 1$ and $t \leq n$, we have the tail-bounds*

$$\left(1 - \frac{t}{n+1}\right)^{m-1} \quad \leq \quad \Pr[X_i \geq t] \quad \leq \quad \left(1 - \frac{t}{n+m-1}\right)^{m-1} .$$

Proof.

$$\Pr[X_i \geq t] = \frac{\binom{n-t+m-1}{m-1}}{\binom{n+m-1}{m-1}} = \frac{n-t+m-1}{n+m-1} \, \frac{n-t+m-2}{n+m-2} \, \cdots \, \frac{n-t+m-(m-1)}{n+m-(m-1)}$$
$$= \left(1 - \frac{t}{n+m-1}\right)\left(1 - \frac{t}{n+m-2}\right) \cdots \left(1 - \frac{t}{n+m-(m-1)}\right)$$

Since for $1 \leq i \leq m-1$, $\frac{t}{n+m-1} \leq \frac{t}{n+m-i} \leq \frac{t}{n+m-(m-1)} = \frac{t}{n+1}$, the lemma holds. ◀

▶ **Lemma 15.** *For $0 < c \leq a \leq b$, we have $\frac{a-c}{b-c} \leq \frac{a}{b}$.*

Proof. $\frac{a-c}{b-c} \leq \frac{a}{b} \iff ab - bc \leq ab - ac \iff a \leq b.$ ◀

For a subtree of size $n \leq \beta + \alpha =: (\alpha+1)\rho + \alpha$, fan-out parameter $\delta(n) = \min\{\alpha + 1, \max\{1, \lceil \frac{n-\alpha}{\rho} \rceil\}\}$ is equal to $\max\{1, \lceil \frac{n-\alpha}{\rho} \rceil\}$. Therefore, for n with $2 \leq \delta(n) \leq \alpha + 1$, we have $(\delta(n)-1)\rho + \alpha < n \leq \delta(n)\rho + \alpha$. Moreover, for $k \in \{2, 3, \ldots, \delta_n\}$, $n > (k-1)\rho + \alpha$ if and only if $\delta(n) \geq k$.

▶ **Lemma 16.** *For a subtree of size $n \leq \beta+\alpha$, where $\delta(n) \leq \alpha+1$, and all $k \in \{2, 3, \ldots, \delta(n)\}$, we have*

$$\Pr[\delta(X_i) \geq k] = \Pr[X_i > (k-1)\rho + \alpha] \leq \left(1 - \frac{k-1}{\delta(n)}\right)^{\delta(n)-1},$$

where X_i is the number of keys in the i-th section beneath the root.

Proof.

$$\Pr[X_i > (k-1)\rho + \alpha] = \Pr[X_i \geq (k-1)\rho + \alpha + 1]$$

Set variables m, t, and n of Lemma 14 to $\delta(n)$, $(k-1)\rho + \alpha + 1$, and $n - \alpha$ respectively. We have

$$\Pr[X_i \geq (k-1)\rho + \alpha + 1] \leq \left(1 - \frac{(k-1)\rho + \alpha + 1}{n - \alpha + \delta(n) - 1}\right)^{\delta(n)-1}.$$

Since $n \leq \delta(n)\rho + \alpha$, we have

$$\Pr[X_i > (k-1)\rho + \alpha] \leq \left(1 - \frac{(k-1)\rho + \alpha + 1}{\delta(n)\rho + \alpha - \alpha + \delta(n) - 1}\right)^{\delta(n)-1}$$

Using Lemma 15, we subtract $\delta(n) - 1$ from the numerator and denominator of $\frac{(k-1)\rho+\alpha+1}{\delta(n)\rho+\alpha-\alpha+\delta(n)-1}$. Thus

$$\Pr[X_i > (k-1)\rho + \alpha] \leq \left(1 - \frac{(k-1)\rho + \alpha + 1 - (\delta(n)-1)}{\delta(n)\rho}\right)^{\delta(n)-1}.$$

Since $\delta(n)$ is at most $\alpha + 1$, we have $\alpha + 1 - (\delta(n) - 1)$ is positive and conclude

$$\Pr[X_i > (k-1)\rho + \alpha] \leq \left(1 - \frac{(k-1)\rho}{\delta(n)\rho}\right)^{\delta(n)-1} = \left(1 - \frac{k-1}{\delta(n)}\right)^{\delta(n)-1},$$

as stated. ◀

We will combine the results of Lemmas 16 and 15 and get the following result.

▶ **Corollary 17.** *For $2 \leq k \leq \delta(n)$, we have*

$$\Pr[\delta(X_i) \geq k] = \Pr[X_i > (k-1)\rho + \alpha] \leq \left(1 - \frac{k-1-1}{\delta(n)-1}\right)^{\delta(n)-1} \leq e^{-(k-2)}.$$

▶ **Observation 18.** *For a subtree with $\delta(n) = 1$, the data structure is a simple list and has at most one non-full block.*

▶ **Lemma 19.** *For a subtree with $\delta(n) = 2$, the expected number of non-full blocks $\mathbb{E}[E_n] \leq 5$.*

Proof. We will design an algorithm calculating an upper bound on the number of non-full blocks. The root has α keys, and the remaining keys are randomly split into two sections with X_1 and X_2 keys, i.e. $X_1+X_2 = n-\alpha$. For $i \in \{1,2\}$, if $\delta(X_i) = 1$, there is at most one non-full block, and the algorithm does not need to proceed in this section anymore. If $\delta(X_i) = 2$, to observe non-full blocks, the $X_i - \alpha$ keys should be partitioned again. It is impossible for both X_1 and X_2 to have $\delta(X_i) = 2$ since it implies that $n = \alpha + X_1 + X_2 > \alpha + 2(\rho+\alpha) = 2\rho + 3\alpha$, which is a contradiction. So at each iteration, either the algorithm stops or continues with one of the sections. In the last step, there are two sections each with at most one non-full block. Thus, the expected number of iterations plus 1 is an upper bound on $\mathbb{E}[E_n]$.

Clearly, $\mathbb{E}[E_n]$ is indeed an increasing function of n, so losing α keys of the root at each iteration results in fewer non-full blocks. To obtain a weaker upper bound, we assume that at each step, the key with the highest priority splits the $n-1$ remaining keys, instead of $n-\alpha$ keys, into two new sections. These keys include the $\alpha - 1$ other keys of the root as well. It suffices to show that the expected number of iterations is at most 4.

At iteration k, k keys with the highest priorities split n keys into k sections with $Y_1^{(k)}, Y_2^{(k)}, \ldots, Y_k^{(k)}$ keys, where $Y_1^{(k)} + Y_2^{(k)} + \cdots + Y_k^{(k)} = n - k$. Round $k+1$ occurs if any of $Y_i^{(k)}$s exceeds $\rho + \alpha$. Since k top keys are chosen uniformly at random, all solutions to the equation $Y_1^{(k)} + Y_2^{(k)} + \cdots + Y_k^{(k)} = n - k$ have equal probabilities. Next, we find an upper bound on $\Pr[Y_i^{(k)} > \rho + \alpha]$ for $i \in \{1, \ldots, k\}$. Set parameters m, t, and n of Lemma 14 to k, $\rho + \alpha$, and $n - k$ respectively. Thus

$$\Pr[Y_i^{(k)} > \rho + \alpha] \leq \left(1 - \frac{\rho + \alpha + 1}{n - k + k - 1}\right)^{k-1} \leq \left(1 - \frac{\rho + \alpha - 1}{n - 1}\right)^{k-1} .$$

Since n is at most $2\rho + \alpha$, we have

$$\Pr[Y_i^{(k)} > \rho + \alpha] \leq \left(1 - \frac{\rho + \alpha - 1}{2\rho + \alpha - 1}\right)^{k-1} \leq \left(1 - \frac{\rho}{2\rho}\right)^{k-1} = 2^{-(k-1)} .$$

The last inequality is an application of Lemma 15. Using the union bound, we have

$$\Pr[\exists Y_i^{(k)} > \rho + \alpha] \leq k/2^{k-1} .$$

This is indeed the probability of round k+1 occurring, conditioned on the probability of rounds k having occurred. Define Z to be the number of iterations and Z_k to be an indicator random variable of whether the k-th iteration happened. If

$$\begin{aligned}\Pr[Z_k = 1] &= \Pr[Z_{k-1} = 1] \Pr[Z_k = 1 | Z_{k-1} = 1] + \Pr[Z_{k-1} = 0] \Pr[Z_k = 1 | Z_{k-1} = 0] \\ &= \Pr[Z_{k-1} = 1] \Pr[Z_k = 1 | Z_{k-1} = 1] \leq k/2^{k-1}\end{aligned}$$

It follows that

$$\begin{aligned}\mathbb{E}[Z] &= \sum_{k=1}^{\infty} \mathbb{E}[Z_k] = \sum_{k=1}^{\infty} \Pr[Z_k = 1] \\ &\leq \sum_{k=1}^{\infty} k/2^{k-1} = 2(\sum_{j=1}^{\infty} \sum_{k=j}^{\infty} 2^{-k}) = 2(\sum_{j=1}^{\infty} \frac{2^{-j}}{1 - \frac{1}{2}}) = 2\frac{\frac{2^{-1}}{1-1/2}}{1/2} = 4 .\end{aligned}$$

◀

▶ **Lemma 20.** *For a subtree with $\delta(n) = 3$, the expected number of non-full blocks $\mathbb{E}[E_n] \leq 10$.*

Proof. $n-\alpha$ keys beneath the root are separated into three sections with X_1, X_2, and X_3 keys. Same as Lemma 19, we can prove that it is impossible to have $\delta(X_i) \geq 2$ for every $i \in \{1,2,3\}$, or $\delta(X_i) = \delta(X_j) = 3$ for some $i \neq j \in \{1,2,3\}$. Due to the symmetric property of the random variables X_i, we can conclude

$$\begin{aligned}1 = \;& 3\Pr[\delta(X_1)=3 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1] \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=2 \wedge \delta(X_3)=1] \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1] \\ & + \Pr[\delta(X_1)=1 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1]\ .\end{aligned}$$

Next, we perform an induction on $2\rho+\alpha < n \leq 3\rho+\alpha$. Using Observation 18, Lemma 19, and induction hypothesis, we get

$$\begin{aligned}\mathbb{E}[E_n] \leq \;& 3\Pr[\delta(X_1)=3 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1](10+1+1) \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=2 \wedge \delta(X_3)=1](5+5+1) \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1](5+1+1) \\ & + \Pr[\delta(X_1)=1 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1](1+1+1) \\ < \;& 3\Pr[\delta(X_1)=3 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1](12-7) \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=2 \wedge \delta(X_3)=1](11-7) \\ & + 7(3\Pr[\delta(X_1)=3 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1] \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=2 \wedge \delta(X_3)=1] \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1] \\ & + \Pr[\delta(X_1)=1 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1]) \\ = \;& 3\Pr[\delta(X_1)=3 \wedge \delta(X_2)=1 \wedge \delta(X_3)=1]5 \\ & + 3\Pr[\delta(X_1)=2 \wedge \delta(X_2)=2 \wedge \delta(X_3)=1]4 + 7 \\ \leq \;& 7 + 3\Pr[\delta(X_1)\geq 3]5 + 3\Pr[\delta(X_1)\geq 2 \wedge \delta(X_2)\geq 2]4\ .\end{aligned}$$

Lemma 16 implies that $\Pr[\delta(X_1) \geq 3] \leq \frac{1}{9}$. (Set $\delta(n)=3$ and $k=3$.) To complete the proof we show that $\Pr[\delta(X_1)\geq 2 \wedge \delta(X_2)\geq 2] \leq \frac{1}{9}$. Consequently, we get the desired inequality $\mathbb{E}[E_n] \leq 7 + 3\frac{5}{9} + 3\frac{4}{9} = 10$.

The two top-priority keys split the keys into three parts with Y_1, Y_2, and Y_3 keys, where $Y_1+Y_2+Y_3 = n-2$. The i-th part includes all X_i keys beneath the root and some extra keys from the root, so for each $i \in \{1,2,3\}$, we have $X_i \leq Y_i$.

$$\begin{aligned}\Pr[\delta(X_1)\geq 2 \wedge \delta(X_2)\geq 2] &= \Pr[X_1 \geq \rho+\alpha+1 \wedge X_2 \geq \rho+\alpha+1] \\ &\leq \Pr[Y_1 \geq \rho+\alpha+1 \wedge Y_2 \geq \rho+\alpha+1] = \binom{n-2(\rho+\alpha+1)}{2} \Big/ \binom{n}{2}\end{aligned}$$

Same technique of Lemma 14 yields $\binom{n-2(\rho+\alpha+1)}{2} \Big/ \binom{n}{2} \leq \left(1 - \frac{2(\rho+\alpha+1)}{n}\right)^2$.

Combining the previous inequalities together with $n \leq 3\rho+\alpha$ implies

$$\Pr[\delta(X_1)\geq 2 \wedge \delta(X_2)\geq 2] \leq \left(1 - \frac{2(\rho+\alpha+1)}{3\rho+\alpha}\right)^2\ .$$

Subtract α from the numerator and denominator. By Lemma 15, we have:

$$\Pr[\delta(X_1)\geq 2 \wedge \delta(X_2)\geq 2] \leq \left(1 - \frac{2\rho+\alpha+2}{3\rho}\right)^2 \leq \left(1 - \frac{2\rho}{3\rho}\right)^2 = \frac{1}{9}\ ,$$

as required. ◀

▶ **Theorem 21.** *For a subtree of size $n \leq \beta + \alpha$, we have that the expected number of non-full blocks $\mathbb{E}[E_n] \leq 27 \cdot \delta(n)$.*

Proof. We will prove the statement by induction on n. Observation 18, Lemma 19, and Lemma 20 provide the bases of the induction. The problem is divided into subproblems by using random variables $X_1, \ldots, X_{\delta(n)}$.

$$\mathbb{E}[E_n] = \mathbb{E}[E_{X_1}] + \mathbb{E}[E_{X_2}] + \cdots + \mathbb{E}[E_{X_{\delta(n)}}] = \delta(n)\mathbb{E}[E_{X_1}]$$

Note that $X_i \leq n - \alpha$, so the induction assumption is applicable to it. For $n \geq 4\rho + \alpha$, we will use the induction assumption and bases to obtain

$$\begin{aligned}
\mathbb{E}[E_{X_i}] &\leq \Pr[\delta(X_i) = 1] \cdot 1 + \Pr[\delta(X_i) = 2] \cdot 5 + \Pr[\delta(X_i) = 3] \cdot 10 + \sum_{k=4}^{\delta(n)} \Pr[\delta(X_i) = k] \cdot (27 \cdot k) \\
&< \Pr[\delta(X_i) = 1] \cdot 10 + \Pr[\delta(X_i) = 2] \cdot 10 + \Pr[\delta(X_i) = 3] \cdot 10 + \sum_{k=4}^{\delta(n)} \Pr[\delta(X_i) = k] \cdot (27 \cdot k) \\
&= 10 \Pr[1 \leq \delta(X_i) \leq 3] + 27 \sum_{k=4}^{\delta(n)} \Pr[\delta(X_i) = k] \cdot k \leq 10 + 27 \sum_{k=4}^{\delta(n)} \Pr[\delta(X_i) = k] \cdot k
\end{aligned}$$

We can rewrite $\sum_{k=4}^{\delta(n)} \Pr[\delta(X_i) = k] \cdot k$ to $3 \Pr[\delta(X_i) \geq 4] + \sum_{k=4}^{\delta(n)} \Pr[\delta(X_i) \geq k]$. Then by using Corollary 17, we get

$$\begin{aligned}
\mathbb{E}[E_{X_i}] &\leq 10 + 27(3 \Pr[\delta(X_i) \geq 4] + \sum_{k=4}^{\delta(n)} \Pr[\delta(X_i) \geq k]) \leq 10 + 27 \left(3e^{-2} + \sum_{k=4}^{\infty} e^{-(k-2)} \right) \\
&\leq 10 + 27 \left(\frac{3}{e^2} + \sum_{k=2}^{\infty} e^{-k} \right) = 10 + 27 \left(\frac{3}{e^2} + \frac{e^{-2}}{1 - e^{-1}} \right) \leq 10 + 0.6202 \times 27 < 27 \,,
\end{aligned}$$

as stated. ◀

B Additional Proofs

▶ **Lemma 22.** *Consider a subtree with $n > \gamma$ keys, where γ is equal to $\beta + \alpha = (\alpha + 1)\rho + \alpha$. We have:*

$$2\frac{\alpha}{\varepsilon}\binom{n-1}{\alpha-i} - 2\frac{\alpha}{\varepsilon}\binom{n-1-\gamma}{\alpha-i} \leq \binom{n}{\alpha+1-i} - \binom{n-\gamma}{\alpha+1-i} - \gamma\binom{n-1-\gamma}{\alpha-i} \,.$$

Proof. The proof is by induction on n. For $n = \gamma + 1$ the statement is

$$\begin{aligned}
& 2\frac{\alpha}{\varepsilon}\binom{\gamma}{\alpha-i} \leq \binom{\gamma+1}{\alpha+1-i} \\
\iff \quad & 2\frac{\alpha}{\varepsilon}\frac{\gamma(\gamma-1)\ldots(\gamma-\alpha+i+1)}{(\alpha-i)!} \leq \frac{(\gamma+1)\gamma\ldots(\gamma-\alpha+i+1)}{(\alpha-i+1)(\alpha-i)!} \qquad (6) \\
\iff \quad & 2\frac{\alpha}{\varepsilon}(\alpha-i+1) \leq \gamma+1 \quad ,
\end{aligned}$$

This is true because $\gamma = (\alpha+1)\rho + \alpha = 108(\alpha+1)\alpha/\varepsilon + \alpha > 2(\alpha+1)\alpha/\varepsilon$.

Assume the lemma holds for each $\gamma < m \leq n$. We will prove the lemma for $n+1$, i.e. we will show:

$$2\frac{\alpha}{\varepsilon}\binom{n}{\alpha-i} - 2\frac{\alpha}{\varepsilon}\binom{n+1-1-\gamma}{\alpha-i} \leq \binom{n+1}{\alpha+1-i} - \binom{n+1-\gamma}{\alpha+1-i} - \gamma\binom{n+1-1-\gamma}{\alpha-i}$$

holds for all i. The inequality is true for $i = \alpha - 1$, since

$$\begin{aligned} 2\frac{\alpha}{\varepsilon}n - 2\frac{\alpha}{\varepsilon}(n-\gamma) &\leq \frac{n(n+1)}{2} - \frac{(n-\gamma)(n-\gamma+1)}{2} - \gamma(n-\gamma) \\ \iff \quad 2\frac{\alpha}{\varepsilon}\gamma &\leq \frac{2\gamma n - \gamma^2 + \gamma}{2} - \frac{2\gamma n - 2\gamma^2}{2} \\ \iff \quad 2\frac{\alpha}{\varepsilon}\gamma &\leq \frac{\gamma^2+\gamma}{2} \quad . \end{aligned}$$

Hence, it suffices to observe that

$$2\frac{\alpha}{\varepsilon}\gamma \leq \frac{\gamma^2+\gamma}{2} \quad \iff \quad 4\frac{\alpha}{\varepsilon} \leq \gamma + 1 = (\alpha+1)\rho + \alpha + 1 = (\alpha+1)108\alpha/\varepsilon + \alpha + 1$$

which is true.

For $i < \alpha - 1$, we will use the identity equation $\binom{m}{k} = \binom{m-1}{k} + \binom{m-1}{k-1}$ together with the induction assumption. We have

$$\begin{aligned} &2\frac{\alpha}{\varepsilon}\binom{n}{\alpha-i} - 2\frac{\alpha}{\varepsilon}\binom{n-\gamma}{\alpha-i} \\ &= 2\frac{\alpha}{\varepsilon}\binom{n-1}{\alpha-(i+1)} + 2\frac{\alpha}{\varepsilon}\binom{n-1}{\alpha-i} - 2\frac{\alpha}{\varepsilon}\binom{n-1-\gamma}{\alpha-(i+1)} - 2\frac{\alpha}{\varepsilon}\binom{n-1-\gamma}{\alpha-i} \\ &= \left[2\frac{\alpha}{\varepsilon}\binom{n-1}{\alpha-(i+1)} - 2\frac{\alpha}{\varepsilon}\binom{n-1-\gamma}{\alpha-(i+1)}\right] + \left[2\frac{\alpha}{\varepsilon}\binom{n-1}{\alpha-i} - 2\frac{\alpha}{\varepsilon}\binom{n-1-\gamma}{\alpha-i}\right] \\ &\leq \binom{n}{\alpha+1-(i+1)} - \binom{n-\gamma}{\alpha+1-(i+1)} - \gamma\binom{n-1-\gamma}{\alpha-(i+1)} \\ &+ \binom{n}{\alpha+1-i} - \binom{n-\gamma}{\alpha+1-i} - \gamma\binom{n-1-\gamma}{\alpha-i} \\ &= \binom{n+1}{\alpha+1-i} - \binom{n+1-\gamma}{\alpha+1-i} - \gamma\binom{n-\gamma}{\alpha-i} \quad . \end{aligned}$$

◀