Median-of-k Jumplists and Dangling-Min BSTs*

Markus E. Nebel† Elisabeth Neumann‡ Sebastian Wild§

October 30, 2018

Abstract

We extend randomized jumplists introduced by Brönnimann, Cazals, and Durand [2] to choose jump-pointer targets as median of a small sample for better search costs, and present randomized algorithms with expected $O(\log n)$ time complexity that maintain the probability distribution of jump pointers upon insertions and deletions. We analyze the expected costs to search, insert and delete a random element, and we show that omitting jump pointers in small sublists hardly affects search costs, but significantly reduces the memory consumption.

We use a bijection between jumplists and "dangling-min BSTs", a variant of (fringe-balanced) binary search trees for the analysis. Despite their similarities, some standard analysis techniques for search trees fail for dangling-min trees (and hence for jumplists).

1 Introduction

Jumplists were introduced by Brönnimann, Cazals, and Durand [2] as a simple randomized comparison-based dictionary implementation. They allow iteration over the stored elements in *sorted* order and supports queries and updates in expected logarithmic time. The core is a sorted (singly-) linked listed augmented with *jump pointers*, i.e., shortcuts that speed up searches. Jump-pointers are required to be well-nested, i.e., they may not cross. This allows binary-search-like navigation. Fig. 1 shows an exemplary jumplist; a detailed definition is deferred to § 3.

If all jump pointers point to the middle of their sublist, we obtain perfect binary search, but we need a rule that is also efficiently maintainable upon insertions and deletions. Brönnimann, Cazals, and Durand [2] proposed a *randomized* solution: jump pointers invariably have a uniform distribution over their sublist, i.e., the first jump pointer equally likely points to any element and thereby divides the list in two parts, the next- and jump-sublists. Both follow the same rule recursively; since pointers may not cross, they can do so independently.

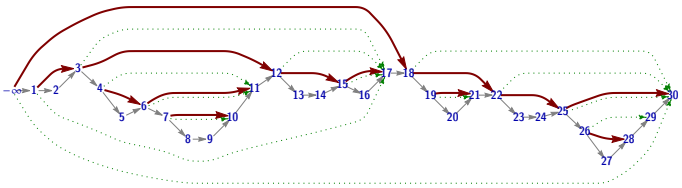


Figure 1: A jumplist on $n = 30$ keys (with $k = 1$ and $w = 2$). Gray arrows are backbone links, thick red arrows are jump pointers. Dotted green arrows delimit a node's conceptual sublist; (they are not stored).

In this article, we generalize jumplists to use a more balanced distribution: each jump pointer points to the *median of a small sample of k elements* of its sublist. (The original jumplists correspond to $k = 1$.) Building on the algorithms from [2] we present $O(\log n)$ expected-time insertion and deletion algorithms for median-of-k jumplists that maintain this more balanced distribution. Here n counts the number of keys currently stored. A larger k balances the structure more rigidly which improves searches, but makes the cleanup after updates more expensive. Our main contribution is an analysis of median-of-k jumplists that precisely quantifies the influence of k on searches, insertions and deletions.

We also introduce a novel search strategy (named *spine search*) that reduces the number of needed key comparisons significantly, and we suggest a further modification of jumplists: for sublists smaller than a threshold w, we omit the jump pointers altogether. This allows to trade space for time: elements in these small sublists do not have to store a jump pointer, but the corresponding subfile can only be searched sequentially. We show that this saves a constant fraction of the pointers while affecting expected search costs only by an additive constant.

Outline of the paper. In the remainder of the introduction we summarize related work. § 2 contains

*The last author is supported by the Natural Sciences and Engineering Research Council of Canada and the Canada Research Chairs Programme.

†Technische Fakultät, Universität Bielefeld, Germany.
nebel@techfak.uni-bielefeld.de

‡Carl-Friedrich-Gauß-Fakultät,
Technische Universität Braunschweig, Germany.
e.neumann@tu-braunschweig.de

§David R. Cheriton School of Computer Science, University of Waterloo, Canada. Email: wild@uwaterloo.ca

common notation and preliminaries used later. In § 3, we define jumplists. We present our spine search strategy in § 4. § 5 introduces the median-of-k extension, and § 6 describes the insertion and deletion algorithms. Our analysis is given in § 7, and we conclude the paper with a discussion of the results (§ 8). There is an extended online version of this paper available as arxiv preprint `1609.08513` (`https://arxiv.org/abs/1609.08513`) that contains detailed descriptions of all algorithms and some missing proofs.

1.1 Related Work. (Unbalanced) binary search trees (BSTs) perform close to optimal on average and with high probability when keys are inserted in random order [12, 13]. A standard approach is to enforce the average behavior through randomization. The most direct application of this paradigm is given by Martínez and Roura [14] who devised efficient randomized insert and delete operations that maintain the shape distribution of random insertions. The idea also works when duplicate keys are allowed [18].

Randomized BSTs store subtree sizes for maintaining the distribution. The *treaps* of Seidel and Aragon [22] instead store a random priority with each node. Treaps remain in random shape by enforcing a heap order w.r.t. the random priorities. Their performance characteristics are very similar to randomized BSTs.

Unless further memory is used, BSTs do not offer $O(1)$ time successor queries. Like jumplists, Pugh's skip lists [20] are augmented, sorted linked lists, so successors are found by following one pointer. Skip lists extend the list elements by towers of pointers of different heights, where each tower cell points to the successor among all element of at least this height. With geometrically distributed heights, operations run in $O(\log n)$ expected time with $O(n)$ extra pointers in expectation. The varying tower heights can be inconvenient; this originally motivated the introduction of jumplists. For skip lists, there is a direct and transparent bijection to BSTs [4]; this becomes more complicated for jumplists (see § 3).

The classic alternative to randomization are deterministically balanced BSTs [1]. Munro, Papadakis, and Sedgewick [15] transfer the height-balance rule of 2-3 trees to skip lists, and Elmasry [6] applied the weight-balancing criterion of $BB[\alpha]$ trees [17] to jumplists. Note that the latter achieves logarithmic update time only in an amortized sense.

A constant-factor speedup over BSTs is achieved with *fringe-balanced* BSTs. The name originates from *fringe analysis*, a technique used in their analysis [19].[1] In a fringe-balanced search tree, leaves *collect* keys in a buffer. Once a leaf holds k keys, it is *split:* the median of the k elements is used as the key of a new node; two new leaves holding the other elements form its subtrees. Many parameters like expected path length, height and profiles of fringe-balanced trees have been studied [5].

[1] The concept appears under a handful of other names in the (earlier) literature: *locally balanced search trees* [23], *diminished trees* [7], and *iR / SR trees* [10, 11].

2 Notation and Preliminaries

We first introduce some notation. We use Iverson's bracket $[stmt]$ to mean 1 if *stmt* is true and 0 otherwise. Falling resp. rising factorial powers are denoted by $x^{\underline{n}}$ and $x^{\overline{n}}$; for negative n holds $x^{\underline{n}} = 1/(x+1)^{\overline{n}}$ resp. $x^{\overline{n}} = 1/(x+1)^{\underline{n}}$. $\mathbb{P}[E]$ denotes the probability of event E and $\mathbb{E}[X]$ the expectation of random variable X. We write $X \stackrel{\mathcal{D}}{=} Y$ to denote equality in distribution.

For a self-contained presentation, we list here a few mathematical preliminaries used in the analysis later.

Beta distribution. The *beta distribution* has two parameters $\alpha, \beta \in \mathbb{R}_{>0}$ and is written as $\mathrm{Beta}(\alpha, \beta)$. If $X \stackrel{\mathcal{D}}{=} \mathrm{Beta}(\alpha, \beta)$, we have $X \in (0,1)$ and X has the density

$$f(x) \;=\; \frac{x^{\alpha-1}(1-x)^{\beta-1}}{\mathrm{B}(\alpha,\beta)}, \qquad x \in (0,1),$$

where $\mathrm{B}(\alpha,\beta) = \Gamma(\alpha)\Gamma(\beta)/\Gamma(\alpha+\beta)$ is the beta function.

The following lemma is helpful for computing expectations involving such beta distributed variables; it is a special case of [24, Lemma 2.30].

Lemma 2.1 ("Powers-to-Parameters"): *Let X_1 be a $\mathrm{Beta}(\alpha_1, \alpha_2)$ distributed random variable and write $X_2 = 1 - X_1$. Let further $m_1, m_2 \in \mathbb{Z}^d$ with $m_1, m_2 > -\alpha$ be given and abbreviate $A := \alpha_1 + \alpha_2$ and $M := m_1 + m_2$. Then for an arbitrary (real-valued, measurable) function f holds*

$$\mathbb{E}\big[X_1^{m_1} X_2^{m_2} \cdot f(X_1)\big] \;=\; \frac{\alpha_1^{\overline{m_1}}\alpha_2^{\overline{m_2}}}{A^{\overline{M}}} \cdot \mathbb{E}\big[f(\tilde{X}_1)\big]\,,$$

where $\tilde{X}_1$ is $\mathrm{Beta}(\alpha_1 + m_1, \alpha_2 + m_2)$ distributed. □

Beta-Binomial Distribution. The *beta-binomial distribution* is a discrete distribution with parameters $n \in \mathbb{N}_0$ and $\alpha, \beta \in \mathbb{R}_{>0}$. It is written as $\mathrm{BetaBin}(n, \alpha, \beta)$. If $I \stackrel{\mathcal{D}}{=} \mathrm{BetaBin}(n, \alpha, \beta)$, we have $I \in [0..n]$ and

$$\mathbb{P}[I = i] \;=\; \binom{n}{i} \frac{\mathrm{B}(\alpha + i, \beta + (n-i))}{\mathrm{B}(\alpha,\beta)}, \quad i \in \mathbb{Z}\,.$$

(Recall that $\binom{n}{i}$ is zero unless $i \in [0..n]$.) An alternative representation of the weights for $\alpha = t_1{+}1, \beta = t_2{+}1 \in \mathbb{N}$ with $k = t_1 + t_2 + 1$ is

$$\binom{n}{i} \frac{\mathrm{B}(\alpha + i, \beta + (n-i))}{\mathrm{B}(\alpha,\beta)} \;=\; \frac{\binom{i+t_1}{t_1}\binom{n-i+t_2}{t_2}}{\binom{n+k}{k}},$$

which yields a combinatorial interpretation.

There is a second way to obtain beta-binomial distributed random variables: we first draw a random probability $D \stackrel{\mathcal{D}}{=} \mathrm{Beta}(\alpha, \beta)$ according to a beta distribution, and then use this as the success probability of a binomial distribution, i.e., $I \stackrel{\mathcal{D}}{=} \mathrm{Bin}(n; d)$ *conditional* on $D = d$. The beta-binomial distribution is thus also called a *mixed* binomial distribution, using a beta-distributed *mixer* D; this explains its name.

Since the binomial distribution is sharply concentrated, one can use Chernoff bounds on beta binomial variables after conditioning on the beta distributed success probability. That already implies that $\mathrm{BetaBin}(n, \alpha, \beta)/n$ converges to $\mathrm{Beta}(\alpha, \beta)$ (in a specific sense). We can obtain the stronger error bounds given in the following lemma by directly comparing the probability density functions.

Lemma 2.2 (Local limit law [24, Lem. 2.38]): *Let $(I^{(n)})_{n\in\mathbb{N}}$ be a sequence of random variables where $I^{(n)}$ is distributed like $\mathrm{BetaBin}(n, \alpha, \beta)$ for $\alpha, \beta \in \mathbb{N}_{\geq 1}$. Then for $n \to \infty$ we have uniformly for $z \in (0, 1)$ that*

$$(1) \quad n\mathbb{P}\big[J^{(n)}/n \in (z - \tfrac{1}{n}, z]\big] \;=\; f_B(z) \,\pm\, O(n^{-1}),$$

where $f_B(z) = z^{\alpha-1}(1-z)^{\beta-1}/\mathrm{B}(\alpha, \beta)$ is the density function of the beta distribution with parameters α and β. □

Since f_B is a polynomial in z, it is in particular bounded and Lipschitz continuous in the closed domain $z \in [0, 1]$. Hence, the local limit law also holds for the random variables $J^{(n)} = I^{(n-d)} + c$ for constants c and d. Further properties of the beta-binomial distribution are collected in [24, § 2.4.7].

We list the following expectations here for reference. The proofs are simple computations found in the extended online version.

Lemma 2.3: *Let $X \stackrel{\mathcal{D}}{=} \mathrm{Bin}(n, p)$ for $n \in \mathbb{N}_0$ and $p \in (0, 1]$. Then we have with $q = 1 - p$ that*

$$\mathbb{E}\big[X^{\underline{-1}}\big] \;=\; n^{\underline{-1}} \cdot p^{-1}(1 - q^{n+1}),$$
$$\mathbb{E}\big[X^{\underline{-2}}\big] \;\leq\; n^{\underline{-2}} \cdot p^{-2}.$$

Lemma 2.4: *For $D \stackrel{\mathcal{D}}{=} \mathrm{Beta}(t + 1, t + 1)$ we have (with $k = 2t + 1$)*

$$\mathbb{E}[\ln D] \;=\; H_t - H_k,$$
$$\mathbb{E}[D \ln D] \;=\; \frac{1}{2}\big(H_{t+1} - H_{k+1}\big).$$

Hölder continuity. A function $f : I \to \mathbb{R}$ defined on a bounded interval I is Hölder continuous with exponent $h \in (0, 1]$ when

$$\exists C\, \forall x, y \in I \;:\; \big|f(x) - f(y)\big| \;\leq\; C|x - y|^h.$$

Hölder continuity is a notion of smoothness that is stricter than (uniform) continuity, but slightly more liberal than Lipschitz continuity (which corresponds to $h = 1$). $f : [0, 1] \to \mathbb{R}$ with $f(z) = z \ln(1/z)$ is a stereotypical function that is Hölder continuous (for any $h \in (0, 1)$), but not Lipschitz.

For functions defined on a bounded domain, Lipschitz continuity implies Hölder continuity and Hölder continuity with exponent h implies Hölder continuity with exponent $h' < h$. Recall that a real-valued function is Lipschitz if its derivative is bounded.

2.1 The Distributional Master Theorem. To solve the recurrences in § 7, we use the "distributional master theorem" (DMT) [24, Thm. 2.76], reproduced below for convenience. It is based on Roura's continuous master theorem [21], but reformulated in terms of distributional recurrences in an attempt to give the technical conditions and occurring constants in Roura's original formulation a more intuitive, stochastic interpretation. We start with a bit of motivation for the latter.

The DMT is targeted at divide-and-conquer recurrences where the recursive parts have a *random* size. The average-case analyses of Quicksort and binary search trees are typical examples that lead to such recurrences. Because of the random subproblem sizes, a traditional recurrence for expected costs has to sum over all possible subproblem sizes, weighted appropriately. That way, the direct correspondence between the recurrence and the algorithmic process is lost, in particular the number of recursive applications is no longer directly visible.

An alternative that avoids this is a *distributional recurrence* that describes the full distribution of costs. The distribution for larger problem sizes is described by a "toll term" (for the divide and/or combine step) plus the contributions of recursive applications. Such a distributional formulation requires the toll costs and subproblem sizes to be stochastically independent of the recursive costs when conditioned on the subproblem sizes. In typical applications, this is fulfilled when the studied algorithm guarantees that the subproblems on which it calls itself recursively are of the same nature as the original problem. Such a form of randomness preservation is also required for the analysis using traditional recurrences. We can thus use the distributional language to describe costs directly mimicking the structure of our algorithms in this paper.

The DMT allows us to compute an asymptotic approximation of the expected costs directly from the distributional recurrence. Intuitively speaking, it is applicable whenever the *relative* subproblem sizes of recursive applications converge to a (non-degenerate) limit distribution as $n \to \infty$ (in a suitable sense; see

Equation (3) below). The local limit law provided by Lem. 2.2 gives exactly such a limit distribution.

Theorem 2.5 (DMT [24, Thm. 2.76]): *Let $(C_n)_{n\in\mathbb{N}_0}$ be a family of random variables that satisfies the distributional recurrence*

$$C_n \overset{\mathcal{D}}{=} T_n + \sum_{r=1}^{s} A_r^{(n)} \cdot C^{(r)}_{J_r^{(n)}}, \qquad (n \geq n_0), \tag{2}$$

where the families $(C_n^{(1)})_{n\in\mathbb{N}}, \ldots, (C_n^{(s)})_{n\in\mathbb{N}}$ are independent copies of $(C_n)_{n\in\mathbb{N}}$, which are also independent of $(J_1^{(n)}, \ldots, J_s^{(n)}) \in \{0, \ldots, n-1\}^s$, $(A_1^{(n)}, \ldots, A_s^{(n)}) \in \mathbb{R}^s_{\geq 0}$ and T_n. Define $Z_r^{(n)} = J_r^{(n)}/n$, $= 1, \ldots, s$, and assume that they fulfill uniformly for $z \in (0,1)$

$$n \cdot \mathbb{P}\big[Z_r^{(n)} \in (z - \tfrac{1}{n}, z]\big] \;=\; f_{Z_r^*}(z) \pm O(n^{-\delta}), \tag{3}$$

as $n \to \infty$ for a constant $\delta > 0$ and a Hölder-continuous function $f_{Z_r^*} : [0,1] \to \mathbb{R}$. Then $f_{Z_r^*}$ is the density of a random variable Z_r^* and $Z_r^{(n)} \xrightarrow{\mathcal{D}} Z_r^*$.

Let further

$$\mathbb{E}\big[A_r^{(n)} \;\big|\; Z_r^{(n)} \in (z - \tfrac{1}{n}, z]\big] \;=\; a_r(z) \pm O(n^{-\delta}), \tag{4}$$

as $n \to \infty$ for a function $a_r : [0,1] \to \mathbb{R}$ and require that $f_{Z_r^*}(z) \cdot a_r(z)$ is also Hölder continuous on $[0,1]$. Moreover, assume $\mathbb{E}[T_n] \sim K n^\alpha \log^\beta(n)$, as $n \to \infty$, for constants $K \neq 0$, $\alpha \geq 0$ and $\beta > -1$. Then, with $H = 1 - \sum_{r=1}^{s} \mathbb{E}[(Z_r^*)^\alpha a_r(Z_r^*)]$, we have the following cases.

1. *If $H > 0$, then $\mathbb{E}[C_n] \sim \dfrac{\mathbb{E}[T_n]}{H}$.*
2. *If $H = 0$, then $\mathbb{E}[C_n] \sim \dfrac{\mathbb{E}[T_n] \ln n}{\tilde{H}}$ with $\tilde{H} = -(\beta+1) \displaystyle\sum_{r=1}^{s} \mathbb{E}[(Z_r^*)^\alpha a_r(Z_r^*) \ln(Z_r^*)]$.*
3. *If $H < 0$, then $\mathbb{E}[C_n] = O(n^c)$ for the $c \in \mathbb{R}$ with $\displaystyle\sum_{r=1}^{s} \mathbb{E}[(Z_r^*)^c a_r(Z_r^*)] = 1$.* □

3 Jumplists

We now present our (consolidated) definition of jumplists; some details differ from the original [2]; we discuss those in the extended online version.

Jumplists consist of *nodes*, where each node v stores a successor pointer ($v.next$) and a key ($v.key$). The nodes are connected using the next pointers to form a singly-linked list, the *backbone* of the jumplist, so that the key fields are sorted ascendingly.[2] It is convenient to add a "dummy" header node v_0 whose key field is ignored; ($v_0.key = -\infty$). If $x_1 < \cdots < x_n$ are the keys stored in the jumplist, we have the $n+1$ nodes $v_0, v_1, \ldots, v_n$ with $v_i.key = x_i$ and $v_{i-1}.next = v_i$ for $i = 1, \ldots, n$. A jumplist on n keys will always have $m = n+1$ nodes; we use n and m in this meaning throughout the paper.

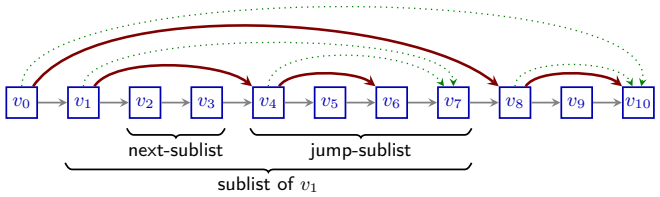
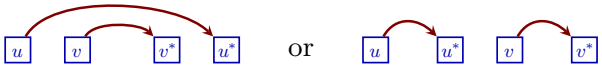


Figure 2: Illustration of the sublist definitions. The sublist of node v_1 contains $m(v_1) = 7$ nodes and stores the 6 keys $v_2.key, \ldots, v_7.key$. The sizes of the next- and jump-sublist are $J_1(v_1) = 2$ and $J_2(v_1) = 4$, respectively.

Jump Pointers. Jump pointers always point forward in the list, and we require the following two conditions. (1) *Non-degeneracy:* Any node may be the target of at most one jump pointer, and jump pointers never point to the direct successor. (2) *Well-nestedness:* Let $v \neq u$ be nodes with $v.key < u.key$, and let v^* resp. u^* be the nodes their jump pointers point to. (Note that $v^* \neq u^*$ by the first property). Then these nodes must appear in one of the following orders in the backbone: $u \ldots v \ldots v^* \ldots u^*$ or $v \ldots v^* \ldots u \ldots u^*$:



The second case allows $v^* = u$. Visually speaking, jump pointers may not cross.

Sublists. The *sublist of node v* starts at v (inclusive) and ends just before the first node targeted by a jump pointer originating before v – or extends to the end of the list if no overarching pointer exists. As for the overall jumplist, v acts as dummy header to its sublist: $v.key$ is *not* considered as part of v's sublist. We write $m(v)$ for the number of nodes in v's sublist. The next- and jump-sublists of v, denoted by $\mathcal{J}_1 = \mathcal{J}_1(v)$ resp. $\mathcal{J}_2 = \mathcal{J}_2(v)$, are the sublists of $v.next$ resp. $v.jump$. We use $J_r = J_r(v)$ for the number of nodes in $\mathcal{J}_r(v)$, $r \in \{1, 2\}$. Fig. 2 exemplifies the definitions. We include an imaginary "end pointer" in the figures, drawn as dotted green line, that connects a jump node with the last node in that node's sublist.

Node Types. Nodes in our jumplists come in two flavors: *plain nodes* only have next and key fields; *jump nodes* additionally store a *jump pointer*, $v.jump$, and their next-sublist size, $v.nsize = J_1$. The node types are

[2] We assume the keys stored in a jumplist are distinct. The insert procedures will prevent duplicate insertions.

determined by the following rule, where $w \geq 2$, the *leaf size,* is a parameter: If $m(v) \leq w$, then v (and all nodes in its sublist) are plain nodes. Otherwise v is a jump node, and we apply the rule recursively to $\mathcal{J}_1(v)$ and $\mathcal{J}_2(v)$. Fig. 1 shows a larger example.

Randomized Jumplists. The following probability distribution over all (legal) jump-pointer configurations invariantly holds in randomized jumplists. It is defined recursively: $v_0.jump$ is drawn *uniformly* from all $m-2$ feasible targets; (v_0 and v_1 are not allowed). Conditional on the choice of $v_0.jump$, the same property is required independently for $\mathcal{J}_1(v_0)$ and $\mathcal{J}_2(v_0)$. The probability $p(\mathcal{J})$ of a particular (legal) pointer configuration $\mathcal{J}$ is

$$p(\mathcal{J}) \quad = \quad \begin{cases} 1, & m \leq w; \\ \dfrac{1}{m-2} \cdot p(\mathcal{J}_1)\, p(\mathcal{J}_2), & m > w, \end{cases}$$

which is reminiscent of the probability of a given shape for a random BST, except for the offset -2 (see [12, ex. 6.2.2–5] or [3, Eq. (5.1)]).

3.1 Dangling-Min BSTs.

There is an intimate relation between jumplists and search trees, but the slight offset above complicates the matter.[3] Indeed, (random) jumplists are isomorphic to a rather peculiar variant of (random) BSTs (where random means "generated by insertions in random order"): the *dangling-min BSTs* (with leaf size $w \geq 2$). Such a tree is defined for a sequence of (distinct) keys $x_1, \ldots, x_n$ as follows. If $n \leq w-1$, it is a leaf with the keys in sorted order. Otherwise, its *root* node contains *two* keys: the smallest key, $\min\{x_1, \ldots, x_n\}$, as its *dangling min,* and the first key of the sequence after the min has been removed as *root key* (i.e., the root key is x_1, unless x_1 is the min; then it is x_2). The left resp. right subtrees of the root are the dangling-min BSTs for the keys smaller resp. larger than the root key in the remaining sequence (without root key and min, and preserving relative order). Dangling-min BSTs make the recursive decomposition in jumplists explicit, which helps for both designing algorithms and analyzing their performance.

We can transform a jumplist to a dangling-min BST (and vice versa): If $m \leq w$, v_0 is a plain node and the dangling-min BST is a leaf containing all $m-1 \leq w-1$ keys; (recall that a jumplist with m nodes stores $n = m-1$ keys). Otherwise, v_0 is a jump node; with x_1 the key in $v_0.next$ and x_j the key in $v_0.jump$, the root of the dangling-min BST has root key x_j and dangling min x_1. Next- resp. jump-sublist are recursively transformed into left and right subtree. Fig. 3 shows the jumplist corresponding to the given tree; Fig. 4 gives a larger example.

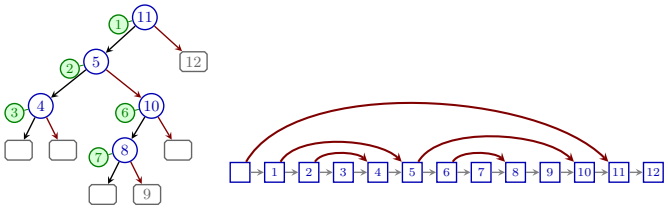


Figure 3: The dangling-min BST with $w = 2$ for the sequence $11, 2, 5, 3, 1, 4, 10, 8, 7, 9, 6, 12$, and the jumplist it corresponds to.

It is easy to see inductively that the dangling-min BST built from a randomized jumplist has the same distribution as if directly constructed for a random permutation of $\{1, \ldots, n\}$. We can therefore focus on analyzing the latter.

4 Spine Search

Searching a key x in a jumplist is straightforward: We start at the header. We stop when the key in the current node v is larger or equal to x. Otherwise we follow either the jump pointer – if the key in $v.jump$ is not larger than x – or the next-pointer. We call this strategy the classic search in the sequel.[4]

However, there is an alternative search strategy not considered in [2] and [6], which performs better! Consider searching key 8 in the jumplist from Fig. 1. A classic search in this list inspects keys $18, 1, 3, 12, 4, 6, 11, 7, 10, 8$ in the given order; a total of 10 key comparisons. Every step in the search that follows the next-pointer needs two comparisons.

Now do the search for 8 in the dangling-min BST from Fig. 4, as if it was a regular BST (ignoring the subtree minima and stopping at the leaves). While doing so, we compare with keys $18, 3, 12, 6, 11, 10$. All these steps need only one key comparison even though mostly the same keys are visited as above. However, our search is not yet finished; the reached leaf contains only 9, and we would (erroneously!) announce that 8 is not in the dictionary. Instead we have to return to the *last node we entered through a right-child pointer* and inspect all the dangling mins along the "left spine" of the corresponding subtree. In our example, we return to 11 and make comparisons with 7 and 8, terminating

[3] The complication is inherent to the feature of jumplists that every key has at most *one* jump pointer. Skip lists, for example, can be transformed into BSTs directly [4].

[4] Brönnimann, Cazals, and Durand [2] also studied the symmetric alternative—compare first to $v.next$ and then with $v.jump$ (if needed)—and found that it needs more comparisons on average.

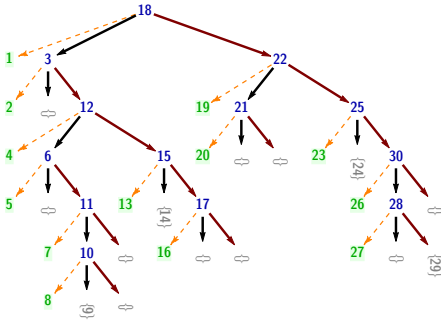


Figure 4: The dangling-min BST for the jumplist from Fig. 1. Black arrows are left child pointers, red arrows are right child pointers, and dotted yellow arrows indicate the dangling min. Gray nodes are leaves that contain between 0 and $w - 1 = 1$ keys.

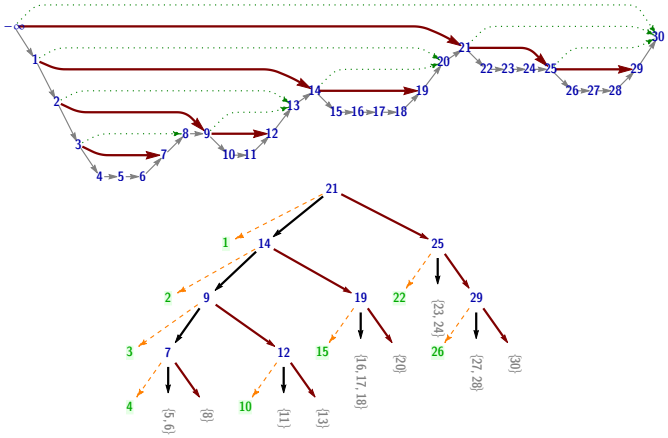


Figure 5: A typical median-of-three ($k = 3$, $w = 4$) jumplist on $n = 30$ keys and its corresponding fringe-balanced dangling-min BST.

successfully. We call this search strategy *spine search.* In our example, it needed 2 comparisons less than the classic search.

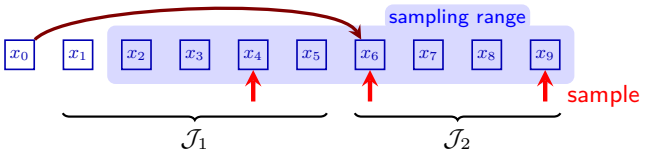
Spine search only compares x with the dangling-mins for nodes on the *left spine above the leaf,* whereas the classic strategy does so for *every* node we leave through the left-child edge. Our modification is correct because when going to the right child we know that all keys left to v are smaller than x and thus x cannot be any of the dangling minima we skipped. The extended online version gives detailed pseudocode.

The left spine is always a subset of the nodes where we took a left child edge, so spine search never needs more comparisons than the classic strategy. It seems reasonable that spine search should need roughly as many key comparisons as the search in a BST since most left spines are short. Indeed, we prove in § 7 that the linear search along the left spine is only a lower order term when averaging over all possible unsuccessful searches—spine search needs $\sim 2\ln(n)$ comparisons, compared to $\sim 3\ln(n)$ for the classic search strategy.

5 Median-of-k Jumplists

The search costs in BSTs can be improved by using medians of a small sample as subtree roots; the idea is called fringe-balancing in that context (§ 1.1) and corresponds to the median-of-k rule for Quicksort [9, 5, 26]. Applied to our trees, we obtain *k-fringe-balanced dangling-min BSTs:* if $n \geq w$, we choose the root key as the median of the first k keys in the sequence after removing the min (and otherwise proceed as before). Here $k = 2t + 1$ is a fixed odd integer and we require $w \geq k + 1$.

Similarly, we define a *randomized median-of-k jumplist* by choosing the jump target as the median of k elements. The situation is illustrated below for $k = 3$ and $m = 10$; to have x_6 as the median of 3 elements from the sample range, we must select $t = 1$ further elements from $\{x_2, \ldots, x_5\}$ and $t = 1$ further elements from $\{x_7, \ldots, x_9\}$.



The number of such samples is $\binom{J_1-1}{t}\binom{J_2-1}{t}$, which we have to divide by the total number of possible samples, $\binom{m-2}{k}$. The probability of a (legal) jump pointer configuration $\mathcal{J}$ thus is

$$p(\mathcal{J}) = \begin{cases} 1 & m \leq w; \\ \dfrac{\binom{J_1-1}{t}\binom{J_2-1}{t}}{\binom{m-2}{k}} \cdot p(\mathcal{J}_1)\, p(\mathcal{J}_2), & m > w. \end{cases}$$

This puts more probability weight on balanced configurations, and hence improves the expected search costs. Fig. 5 shows a typical median-of-3 jumplist and its fringe-balanced dangling-min tree.[5]

Distribution of subproblem sizes. For our analysis, an alternative description of the distribution of the subproblem sizes is more convenient. Note that both J_1 and J_2 are always at least $t + 1$: the sublists must contain t other sampled nodes plus their header. If we denote by $I_r = J_r - t - 1$, $r \in \{1, 2\}$,

[5] A possible generalization could use asymmetric sampling with (t_1, t_2) and $k = t_1 + t_2 + 1$, where we select the $(t_1 + 1)$st smallest instead of the median. Then, we have $\binom{J_1-1}{t_1}$ and $\binom{J_2-1}{t_2}$ in Equation (5). For the present work, we will however stick to the case $t_1 = t_2 = t$.

we find that I_r has a beta-binomial distribution (§ 2), $I_r \stackrel{\mathcal{D}}{=} \text{BetaBin}(m-2-k, t+1, t+1)$. This implies that with $D \stackrel{\mathcal{D}}{=} \text{Beta}(t+1, t+1)$, we have the mixed distribution $I_r \stackrel{\mathcal{D}}{=} \text{Bin}(m-2-k, D)$ conditional on D.[6]

6 Insert and Delete

We briefly sketch the update operations for randomized median-of-k jumplists; the extended online version describes them in more detail. The common theme is that we first modify the jumplist blindly and afterwards "repair" the distribution by rebuilding one suitably chosen sublist randomly from scratch. For example upon insertion, the new node has a certain chance to be the target of the first jump pointer. We flip a coin to decide whether this should happen; if so, we rebuild the entire structure and are done. Otherwise, we recursively repair a sublist.

Rebalance. As in [2], we use a procedure Rebalance($\mathcal{J}$) that (re)assigns jump pointers from scratch. It only uses the backbone, existing jump pointers are ignored. A careful recursive implementation of Rebalance rebuilds a sublist of m nodes in time $\Theta(m)$.

Insert. Insertion in jumplists consists of the three phases found in many dictionaries: (unsuccessful) search, local insertion, and cleanup. Unless x is already present, the search ends at the node with the largest key (strictly) smaller than x. There we insert a new node with key x into the backbone.

It does not have a jump pointer yet, and it is a new potential jump target for all the nodes whose sublist contains the new node. Procedure RestoreAfterInsert rectifies this as follows. Let m be the total number of nodes after the insertion, i.e., including the new node. If $m \le w$, no cleanup is necessary; if $m = w+1$, we draw the jump pointer for v_0 and are done. Otherwise, we first restore the pointer distribution of v_0. Due to the insertion of a new node, the sample range now contains an additional node u. (u is not necessarily the newly inserted node; if the new key is the first or second smallest in $\mathcal{J}$, u is the former second node of $\mathcal{J}$).

If we, conceptually, drew $v_0.jump$ anew, there are two possibilities: either u is part of the sample, namely with probability $p = \frac{k}{m-2}$, or u is not part of it. In the first case, we rebalance all of $\mathcal{J}$. In the second case, conditional on the event that u is *not* in the sample, the current jump pointer of v_0 already has the correct distribution: the median of a random sample not containing u. We thus rebalance $\mathcal{J}$ with probability p, where we draw the jump pointer of v_0 conditional on u being part of the sample. Otherwise we continue recursively in the uniquely determined sublist that contains the inserted node. Fig. 6 summarizes RestoreAfterInsert graphically.

Delete. We now sketch the procedure RestoreAfterDelete, which is similar to RestoreAfterInsert. Let m be the number of nodes after deletion, and let u be the deleted node. First assume that $u \neq v_0$. Assume $m > w$, i.e., v_0 is a jump node whose sublist contained u. If the sample drawn to choose $v_0.jump$ did *not* contain u, the deletion of u does not affect $v_0.jump$, and we recursively clean up the sublist that formerly contained u. If u was part of the sample, we have to rebalance $\mathcal{J}$; the probability for that is

$$p = \begin{cases} 1, & \text{if } u = v_0.jump; \\ \frac{t}{J_1 - 1}, & \text{if } u \text{ was in } \mathcal{J}_1; \\ \frac{t}{J_2 - 1}, & \text{if } u \text{ was in } \mathcal{J}_2. \end{cases}$$

(We define $\frac{0}{0} := 1$ in case $t = J_1 - 1 = 0$.) When the deleted node is $u = v_0$, the new header v_1 can inherit v_0's jump pointer and we have the same situation as if v_1 had been deleted. We have to rebalance with probability $p = \frac{t}{J_1 - 1}$, otherwise we continue the cleanup in the next-sublist.

Cost Measure. Insertion and deletion consist of a search and RestoreAfterInsert/-Delete. The latter procedures retrace (a prefix of) the search path to the element and rebuild at most one sublist using Rebalance. So apart from the search costs (which we analyze separately), the dominating cost is the number of *"rebalanced elements":* the size of the sublists on which Rebalance is called. We will use this as our measure of costs.

7 Analysis

We now turn to the analysis of the expected behavior of median-of-k jumplists with leaf size w. (The expectation is always over the random choices of the jump pointers.) We summarize our results in the theorem below. Its proof is spread over the following subsections.

Theorem 7.1:
Consider randomized median-of-k jumplists with leaf size w on n keys, where k and w are fixed constants. Abbreviate by $H(k) = H_{k+1} - H_{(k+1)/2}$ for H_n the harmonic numbers. Then the following holds:

(a) *The expected number of key comparisons in a* ***spine search*** *is asymptotic to $1/H(k) \cdot \ln n$, as $n \to \infty$, when each position is equally likely to be requested.*
(b) *The expected number of rebalanced elements in the* ***cleanup after insertion*** *is asymptotic to*

[6] The symmetry in the sublist sizes, $J_1 \stackrel{\mathcal{D}}{=} J_2$, is a major convenience of our definition of jumplists as opposed to the original one.

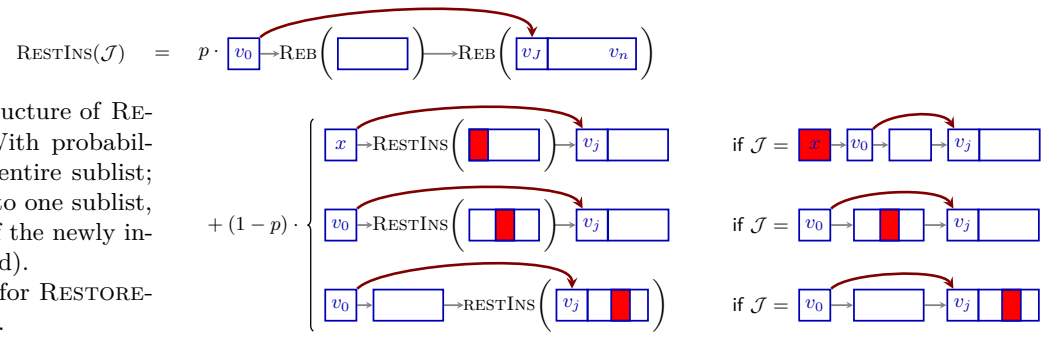


Figure 6: Recursion structure of RestoreAfterInsert. With probability p, we rebalance the entire sublist; otherwise, we recurse into one sublist, depending on the rank of the newly inserted node (shown in red).
The recursion structure for RestoreAfterDelete is similar.

$k/H(k) \cdot \ln n$, *as* $n \to \infty$, *when each of the* $n+1$ *possible gaps is equally likely.*

(c) *The expected number of rebalanced elements in the* ***cleanup after deletion*** *is asymptotic to* $k/H(k) \cdot \ln n$, *as* $n \to \infty$, *when each key is equally likely to be deleted.*

(d) *The expected number of additional machine words per key required to store the jumplist is asymptotically at most* $1 + \frac{2}{(w+1)H(k)}$ *as* $n \to \infty$.

7.1 Search Costs. Let P_n be the (random) total number of comparisons to search all numbers $x \in \{0.5, 1.5, \ldots, n+0.5\}$ (searching each gap once) in $\mathcal{J}_n$ the randomized jumplist on $\{1, \ldots, n\}$, using SpineSearch. The corresponding quantity in BSTs is called external path length, and we will use this term for P_n, as well. The quotient P_n/n describes the average costs of one call to SpineSearch when all $n+1$ gaps are equally likely to be requested. P_n is random w.r.t. to the locations of the jump pointers in $\mathcal{J}_n$. To set up a recurrence for P_n, the perspective of random dangling-min BSTs is most convenient, since SpineSearch follows the tree structure. We describe recurrences here in terms of the distributions of families of random variables.

$$P_n \stackrel{\mathcal{D}}{=} \begin{cases} (n+1) + (S_n + L_n + 1) + S_{J_1} \\ \qquad + P_{J_1} + P'_{J_2}, & n \geq w, \\ \frac{(n+1)(n+2)}{2}, & n < w, \end{cases}$$

$$S_n \stackrel{\mathcal{D}}{=} \begin{cases} 1 + S_{J_1}, & n \geq w, \\ 0, & n < w, \end{cases}$$

$$L_n \stackrel{\mathcal{D}}{=} \begin{cases} L_{J_1}, & n \geq w, \\ n, & n < w, \end{cases}$$

The terms P_{J_1} and P'_{J_2} on the right-hand side denote members of independent copies of the family of random variables $(P_n)_{n \in \mathbb{N}_0}$, which are also independent of $J_r = J_r^{(n)}$, $r \in \{1, 2\}$. (We omitted the superscripts above for readability.) Here $J_r = I_r + t$, $r \in \{1, 2\}$, $I_1 \stackrel{\mathcal{D}}{=} \mathrm{BetaBin}(n-1-k; t+1, t+1)$ and $J_2 = n-1-k-J_1$. (We use n here instead of m in § 5; hence the slightly different parameters.)

The terms in the expression for P_n are the comparisons with (1) the root key, (2) the dangling min of the root, (3) the comparisons done in the left subtree while searching the leftmost gap (which does not exist in the subtrees any more!), and (4) the external path lengths of the subtrees. Two additional quantities are used to express these: L_n is the number of keys in the leftmost leaf; by definition we have $0 \leq \mathbb{E}[L_n] \leq w - 1 = O(1)$. S_n is the number of internal nodes on the "left spine" of the tree, an essential parameter for the linear-search part of SpineSearch. S_n is also the depth of the internal node with the smallest root key (ignoring dangling mins). For ordinary BSTs, S_n is essentially the number of left-to-right minima, which is a well-understood parameter; for (fringe-balanced) dangling-min BSTs, such a simple correspondence does not seem to hold.

We point out that the distribution of P_n has a subtle complication, namely that even conditional on (J_1, J_2), the quantities S_n, S_{J_1} and P_{J_1} are *not* independent: all consider the *same* left subtree! For example, we always have $S_{J_1} = S_n - 1$ (for $n \geq w$). We will only compute the expected value here, so by linearity, these dependencies can be ignored.

We will derive an asymptotic approximation using Thm. 2.5, the distributional master theorem (DMT).

Remark 7.1: For ordinary BSTs, the expectation of above quantities is known precisely, and some generalizations for fringe-balanced trees are possible by solving an Euler differential equation for the generating function. Unlike there, for dangling-min BSTs the resulting differential equation is *not* an Euler equation. The case $t = 0$ could be solved since the differential equation has order one [2], but there is little hope to obtain a solution for the generating function for $t \geq 1$.

Lemma 7.2: $\mathbb{E}[S_n] \sim \dfrac{1}{H_k - H_t} \ln n.$

Proof: We apply Thm. 2.5 to the distributional recurrence $S_n \stackrel{\mathcal{D}}{=} S_{J_1}+1$. It has the form of (2) with (matching the notation of Thm. 2.5) $C_n = S_n$. We have $s = 1$ recursive term with size J_1 plus a "toll term" $T_n = 1$. The latter has the asymptotic form $\mathbb{E}[T_n] = 1 \sim 1 \cdot n^0 \lg^0 n$ as $n \to \infty$, i.e., $K = 1$, $\alpha = 0$, $\beta = 0$. Moreover, there is no "coefficient" in from of the recursive term, so $A_1 = 1$.

We next check the conditions. The independence assumptions are trivially fulfilled here, in particular because T_n is a fixed constant. We next consider (3). Recall that $J_1 \stackrel{\mathcal{D}}{=} \mathrm{BetaBin}(n-1-k; t+1, t+1) + t$. By Lem. 2.2 and the remark below it, $Z_1^{(n)} = J_1^{(n)}/n$ fulfills

$$n\mathbb{P}\big[Z_1^{(n)} \in (z - \tfrac{1}{n}, z]\big] \;=\; f_{Z_1^*} \pm O(n^{-1}),$$

for $f_{Z_1^*} : [0,1] \to \mathbb{R}$ with $f_{Z_1^*}(z) = z^t(1-z)^t/\mathrm{B}(t+1, t+1)$. This function is a polynomial in z, so it has bounded derivative (on the compact domain $[0,1]$) and is hence Lipschitz continuous (and thus Hölder continuous). So (3) is satisfied with $\delta = 1$. The limiting relative subproblem size Z_1^* has a $\mathrm{Beta}(t+1, t+1)$ distribution.

For the second condition, (4), we find that $\mathbb{E}\big[A_r^{(n)} \;\big|\; Z_r^{(n)} \in (z - \frac{1}{n}, z]\big] = 1$ since A_1 is constant. So this condition is trivially satisfied with $a_1(z) = 1$ (which is a Hölder-continuous function). We have now established that we can apply the DMT to our recurrence.

To obtain the asymptotic approximation for $\mathbb{E}[S_n]$, we consider $H = 1 - \mathbb{E}[(Z_1^*)^0] = 0$, so Case 2 applies: $\mathbb{E}[S_n] \sim \tilde{H}^{-1} \cdot \mathbb{E}[T_n] \ln n = \tilde{H}^{-1} \cdot \ln n$ for the constant $\tilde{H} = -\sum_{r=1}^{s} \mathbb{E}[\ln(Z_r^*)]$. (Note that this constant only involved the limiting relative subproblem size Z_r^*, not the relative subproblem size $Z_1^{(n)}$ for a fixed n.) The expectation in $\tilde{H}$ is exactly the first part of Lem. 2.4, so we find $\tilde{H} = H_k - H_t$. Now the claim follows by inserting above. □

Remark 7.2 (Spine lengths): Lem. 7.2 implies that the expected left spine of the root is logarithmic – as one might expect in a random BST; indeed, the expected left spine lengths of the root in a random BST and a dangling-min BST differ only in lower order terms. Note that the former is exactly H_n and the proof is elementary: The left spine length in a BST is the number of left-to-right minima in the insertion order. For dangling-min BSTs, no such simple argument is available.

With these preparations, we can prove the main statement about search costs.

Proof of Thm. 7.1−(a): We again use the distributional master theorem (DMT); this time on the recurrence $P_n \stackrel{\mathcal{D}}{=} (n+1) + (S_n + L_n + 1) + S_{J_1} + P_{J_1} + P'_{J_2}$. The recurrence is more involved than the one for S_n that we just solved, but the distribution of subproblem sizes are the same, and we again have no coefficient in front of the recursive terms. Therefore, a large part of the argument can be copied from the proof of Lem. 7.2.

We here have $C_n = P_n$, there are $s = 2$ recursive terms and $T_n = (n+1)+(S_n+L_n+1)+S_{J_1}$. By Lem. 7.2, all but the first summand in $\mathbb{E}[T_n]$ are actually in $O(\log n)$, so from the initially complicated toll function, only $\mathbb{E}[T_n] \sim n$ remains in the leading term as $n \to \infty$. We thus have $K = 1$, $\alpha = 1$, $\beta = 0$.

The coefficients $A_r = 1$ for $r \in \{1, 2\}$, so (4) again holds trivially with $a_r(z) = 1$. As in the proof of Lem. 7.2, $Z_1^* \stackrel{\mathcal{D}}{=} Z_2^* \stackrel{\mathcal{D}}{=} \mathrm{Beta}(t+1, t+1)$ holds and condition (3) holds with the same $f_{Z_1^*}$. We find again $H = 0$ (since $Z_1^* + Z_2^* = 1$), so Case 2 applies. The constant $\tilde{H}$ this time involves the second part of Lem. 2.4: $\tilde{H} = -\sum_{r=1}^{s} \mathbb{E}[D_r \ln(D_r)] = H_{k+1} - H_{t+1}$. So we have $\mathbb{E}[P_n] \sim \frac{1}{H_{k+1} - H_{t+1}} n \ln n$ and dividing by $n+1$ yields the claim. □

7.2 Insertion Costs. The steps taken by RestoreAfterInsert depend on the position of the newly inserted element; we denote here by R the rank of the gap the new element is inserted into. When the current sublist has m nodes, we have $R \in [0..m]$. Similar as for searches, we consider the average costs of insertion when all possible gaps are equally likely to be requested.

Unlike for searches, the distribution of R' in subproblems is *not* uniform even when R is: a close inspection of RestoreAfterInsert reveals that (a) recursive calls in the jump-sublist always have $R' \geq 1$, and (b) $R = 0$ and $R = 1$ yield $R' = 0$ in the recursive call in the next-sublist; in fact, once $R = 0$ holds, we get this rank in all later recursive calls. We can therefore handle this by splitting the cases $R = 0$ and $R \geq 1$; Also note that for the topmost call to RestoreAfterInsert, $R = 0$ is not possible, since no insertion before the header with dummy-key $-\infty$ is possible. This means that initially $R \stackrel{\mathcal{D}}{=} \mathcal{U}[1..m]$ holds. Recall that a jumplist on m nodes stores only $n = m - 1$ keys, so that there are only $n + 1 = m$ possible gaps. We obtain the following distributional recurrence for B_m^{ins}, the random number of rebalanced elements during insertion into the Rth gap in a randomized median-of-k jumplist with m nodes. (Note that m is here the number of nodes in the jumplist *before* the insertion.)

$$B_m^{\mathrm{ins}} \stackrel{\mathcal{D}}{=} \begin{cases} F \cdot (m+1) & \\ \quad + (1-F)\Big(\mathbb{1}_{\{R=1\}} B_{J_1}^{\mathrm{ins0}} & \\ \quad\quad + \mathbb{1}_{\{2 \leq R \leq J_1+1\}} B_{J_1}^{\mathrm{ins}} & m > w, \\ \quad\quad + \mathbb{1}_{\{R \geq J_1+2\}} B_{J_2}^{\mathrm{ins}}\Big), & \\ & \\ [m = w] \cdot (m+1), & m \leq w, \end{cases}$$

$$B_m^{\text{ins0}} \overset{\mathcal{D}}{=} \begin{cases} F\cdot(m+1) \;+\; (1-F)B_{J_1}^{\text{ins0}}, & m > w, \\ [m=w]\cdot(m+1), & m \le w, \end{cases}$$

$$\text{where} \quad R \overset{\mathcal{D}}{=} \mathcal{U}[1..m], \quad F \overset{\mathcal{D}}{=} \mathrm{B}\Big(\frac{k}{m-1}\Big),$$

All B_m terms on the right-hand side denote independent copies of the family of random variables and R and F are independent of B_m and (J_1, J_2). Here $J_r = I_r + t + 1$, $r \in \{1,2\}$, $J_1 \overset{\mathcal{D}}{=} \text{BetaBin}(m-2-k; t+1, t+1)$ and $J_2 = m-2-k-J_1$ (as in § 5).

Lemma 7.3: $\mathbb{E}[B_m^{\text{ins0}}] \sim \dfrac{k}{H_k - H_t} \ln m.$

Proof: We use once more the distributional master theorem. As before, $Z_1^* \overset{\mathcal{D}}{=} \text{Beta}(t+1, t+1)$ and the condition (3) is satisfied by Lem. 2.2. We have $\mathbb{E}[T_n] = \mathbb{E}[F(n+1)] \sim k = \Theta(1)$. Unlike before, we here have a non-constant coefficient $A_1^{(n)} = 1 - F$ in front of the recursive term, but since $\mathbb{E}[1-F] = 1 \pm O(n^{-1})$, (4) is again fulfilled with $a_1(z) = 1$. As in the proof of Lem. 7.2, we find $H = 0$ (Case 2) and with the claim follows from $\tilde{H} = -\mathbb{E}[\ln D_1] = H_k - H_t$ (Lem. 2.4). □

Proof of Thm. 7.1−(b): Towards applying the DMT on $C_n = B_n^{\text{ins}}$, we compute

$$\begin{aligned} \mathbb{E}[T_n] &= \mathbb{E}\Big[F(n+1) + (1-F)\mathbb{1}_{\{R=1\}} B_n^{\text{ins0}}\Big] \\ &= \frac{k(n+1)}{n-1} + \frac{n-1-k}{n-1}\cdot\frac{1}{n}\cdot\mathbb{E}[B_n^{\text{ins0}}] \\ &\underset{\text{Lem. 7.3}}{=} k \pm O(n^{-1}\log n). \end{aligned}$$

As usual, we have $Z_r^* \overset{\mathcal{D}}{=} \text{Beta}(t+1, t+1)$, $r \in \{1,2\}$, and (3) is fulfilled by Lem. 2.2. For the coefficients of the recursive terms holds

$$\begin{aligned} &\mathbb{E}\big[A_1^{(n)} \;\big|\; Z_1^{(n)} \in (z - \tfrac{1}{n}, n]\big] \\ &= \mathbb{P}\Big[2 \le R \le J_1 + 1 \;\Big|\; Z_1^{(n)} \in (z - \tfrac{1}{n}, n]\Big] \\ &= \mathbb{P}\Big[\tfrac{J_1}{n} \;\Big|\; Z_2^{(n)} \in (z - \tfrac{1}{n}, n]\Big] \\ &= \mathbb{P}\big[Z_1^{(n)} \;\big|\; Z_1^{(n)} \in (z - \tfrac{1}{n}, n]\big] \\ &= z \pm O(n^{-1}), \end{aligned}$$

and similarly

$$\begin{aligned} &\mathbb{E}\big[A_2^{(n)} \;\big|\; Z_2^{(n)} \in (z - \tfrac{1}{n}, n]\big] \\ &= \mathbb{P}\big[R \ge J_1 + 2 \;\big|\; Z_2^{(n)} \in (z - \tfrac{1}{n}, n]\big] \\ &= z \pm O(n^{-1}), \end{aligned}$$

so that (4) holds with $a_1(z) = a_2(z) = z$, and we can apply the DMT. Since $H = 1 - \sum_{r=1}^{2} \mathbb{E}\big[(Z_r^*)^0 \times a_r(Z_r^*)\big] = 1 - \sum_{r=1}^{2} \mathbb{E}[Z_r^*] = 0$, we again have Case 2 and find $\tilde{H} = -\sum_{r=1}^{2} \mathbb{E}[D_r \ln D_r] = H_{k+1} - H_{t+1}$ with Lem. 2.4. This proves the claim. □

7.3 Deletion Costs. As for insertion, we analyze the size of the sublist B_m^{del} that is rebuilt using REBALANCE when the rank of the deleted element is chosen uniformly. Initially, we have $R \overset{\mathcal{D}}{=} \mathcal{U}[2..m]$ since the dummy key $-\infty$ in the header cannot be deleted. In recursive calls, also $R = 1$ is possible, and we remain in this case for good whenever we enter it once. We can thus characterize the deletion costs using the two quantities B_m^{del} and B_m^{del1}. As for insertion, m is the "old" size of the jumplist, i.e., the number of nodes *before* the deletion.

$$B_m^{\text{del}} \overset{\mathcal{D}}{=} \begin{cases} F\cdot(m-1) \\ \quad + (1-F)\Big(\mathbb{1}_{\{R=2\}} B_{J_1}^{\text{del1}} \\ \quad\quad + \mathbb{1}_{\{3 \le R \le J_1+1\}} B_{J_1}^{\text{del}} & m > w, \\ \quad\quad + \mathbb{1}_{\{R \ge J_1+3\}} B_{J_2}^{\text{del}}\Big), \\ [m=w]\cdot 1, & m \le w, \end{cases}$$

$$\text{where } R \overset{\mathcal{D}}{=} \mathcal{U}[2..m], \quad \text{and cond. on } (R, J_1, J_2)$$

$$F \overset{\mathcal{D}}{=} \begin{cases} \mathrm{B}\big(\frac{t}{J_1-1}\big), & R \le J_1 + 1; \\ 1, & R = J_1 + 2; \\ \mathrm{B}\big(\frac{t}{J_2-1}\big), & R \ge J_1 + 3, \end{cases}$$

$$B_m^{\text{del1}} \overset{\mathcal{D}}{=} \begin{cases} F_1\cdot(m-1) + (1-F_1)B_{J_1}^{\text{del1}}, & m > w, \\ [m=w]\cdot 1, & m \le w, \end{cases}$$

$$\text{where cond. on } J_1 \;\; F_1 \overset{\mathcal{D}}{=} \mathrm{B}\Big(\frac{t}{J_1 - 1}\Big).$$

As before, the B_m terms on the right are independent copies of the family of random variables and R and F/F_1 are independent of B_m and (J_1, J_2). We have $J_r = I_r + t + 1$, $r \in \{1,2\}$, $J_1 \overset{\mathcal{D}}{=} \text{BetaBin}(m-2-k; t+1, t+1)$ and $J_2 = m-2-k-J_1$. The (asymptotic) solution of these recurrences is similar to the case of insertion, but a few more complications arise.

Lemma 7.4: *For $t = 0$ we have $\mathbb{E}[B_m^{\text{del1}}] \le 1$.*
If $t \ge 1$, $\mathbb{E}[B_m^{\text{del1}}] \sim \dfrac{k}{H_k - H_t} \ln m.$

Proof: For $t = 0$, we have $F_1 = 0$ (almost surely) in each iteration, so the recurrence collapses to its initial condition, which is at most 1. In the following, we now consider $t \ge 1$. The proof will ultimately use the DMT on $C_n = B_n^{\text{del1}}$, but we need a few preliminary results to compute the toll function $\mathbb{E}[T_n] = \mathbb{E}[F_1(n-1)]$. We write the $a = b \pm d$ to mean $b - d \le a \le b + d$ here and throughout. With that notation, we give the following elementary approximation:

$$(5) \quad \forall t \in \mathbb{N}_{\ge 1}\, \forall n \ge 0 \;:\; \frac{t}{n+t} \;=\; t\underline{n^{-1}} \;\pm\; t(t-1)\underline{n^{-2}}.$$

Now, we compute the expectation of F_1 conditional on $I_1 = J_1 - t - 1$.

$$\mathbb{E}[F_1 \mid I_1] \;=\; \frac{t}{J_1 - 1} \;=\; \frac{t}{I_1 + t} \;\underset{(5)}{=}\; t \cdot I_1^{\underline{-1}} \pm t(t-1) \cdot I_1^{\underline{-2}}.$$

Next, we use the stochastic representation of beta-binomials (recall § 2); we take expectations over $I_1 \overset{\mathcal{D}}{=} \mathrm{Bin}(\eta, D_1)$ with $\eta = m - 2 - k$, but conditional on D_1. We write $D_2 = 1 - D_1$. Then it holds that

$$\mathbb{E}[F_1 \mid D_1] \;\underset{\text{Lem. 2.3}}{=}\; \frac{t}{\eta+1} D_1^{-1}(1 - D_2^{\eta+1}) \pm t(t-1) D_1^{-2} \eta^{\underline{-2}}.$$

Finally, we also compute the expectation w.r.t. $D_1 \overset{\mathcal{D}}{=} \mathrm{Beta}(t+1, t+1)$; note that for $t \geq 2$, $\mathbb{E}[D_1^{-2}]$ exists and has a finite value (independent of n); whereas for $t = 1$, the error term is zero. So we find in both cases with Lem. 2.1:

$$\begin{aligned}
\mathbb{E}[F_1] \;&=\; \frac{t}{\eta+1}\mathbb{E}[D_1^{-1}] - \frac{t}{\eta+1}\mathbb{E}[D_1^{-1} D_2^{\eta+1}] \pm O(\eta^{-2}) \\
&=\; \frac{t}{\eta+1}\frac{k}{t} - \frac{t}{\eta+1}\frac{(t+1)^{\overline{\eta+1}}}{t(k+1)^{\overline{\eta}}} \pm O(\eta^{-2}) \\
&=\; \frac{k}{\eta+1} - \frac{(t+1)(t+2)}{(\eta+1)(\eta+2)} \underbrace{\frac{(t+3)^{\overline{\eta-1}}}{(k+2)^{\overline{\eta-1}}}}_{<1} \pm O(\eta^{-2}) \\
&\underset{(6)}{=}\; \frac{k}{\eta+1} \pm O(\eta^{-2}).
\end{aligned}$$

With this we finally get $\mathbb{E}[T_n] = \mathbb{E}[F_1(n-1)] = k \pm O(n^{-1})$. $Z_1^* \overset{\mathcal{D}}{=} \mathrm{Beta}(t+1, t+1)$ and fulfills (3). For (4), we compute

$$\begin{aligned}
&\mathbb{E}\big[A_1^{(n)} \,\big|\, Z_1^{(n)} \in (z - \tfrac{1}{n}, z]\big] \\
&\quad= \mathbb{E}\big[1 - F_1 \,\big|\, Z_1^{(n)} \in (z - \tfrac{1}{n}, z]\big] \\
&\quad= 1 \pm O(n^{-1}).
\end{aligned}$$

So the DMT applies; we have $H = 0$, i.e., Case 2. The claim follows with $\tilde{H} = -\mathbb{E}[\ln Z_1^*] = H_k - H_t$. □

Proof of Thm. 7.1−(c): We start with computing the conditional expectation of F, the coin flip indicator.

$$\begin{aligned}
\mathbb{E}[F \mid J_1] \;&=\; \frac{J_1}{n-1}\frac{t}{J_1 - 1} + \frac{1}{n-1}1 + \frac{J_2 - 1}{n-1}\frac{t}{J_2 - 1} \\
&=\; \frac{2t+1}{n-1} + \frac{1}{n-1} \cdot \frac{t}{J_1 - 1}.
\end{aligned}$$

Hence

$$\begin{aligned}
\mathbb{E}[F] \;&\underset{(6)}{=}\; \frac{2t+1}{n-1} + \frac{1}{n-1} \cdot \frac{k}{\eta+1} \pm O(n^{-3}) \\
&=\; \frac{k}{n-1} \pm O(n^{-2}).
\end{aligned}$$

Towards applying the DMT on $C_n = B_n^{\mathrm{del}}$, we compute

$$\begin{aligned}
\mathbb{E}[T_n] \;&=\; \mathbb{E}\Big[F(n-1) + (1-F)\mathbb{1}_{\{R=2\}} B_n^{\mathrm{del}1}\Big] \\
&\underset{\text{Lem. 7.4}}{=}\; k \pm O(n^{-1} \log n).
\end{aligned}$$

We have $Z_r^* \overset{\mathcal{D}}{=} \mathrm{Beta}(t+1, t+1)$ and (3) is fulfilled. Similarly as in § 7.2, we find that (4) holds with $a_1(z) = a_2(z) = z$. Once more we have $H = 0$ and Case 2 applies, and the claim follows with $\tilde{H} = -\sum_{r=1}^{2} \mathbb{E}[D_r \ln D_r] = H_{k+1} - H_{t+1}$. □

7.4 Memory Requirements. We assume that a pointer requires one word of storage, and so does an integer that can take values in $[0..n+1]$. We do not count memory to store the keys since any (general-purpose) data structure has to store them. This means that a plain node requires 1 word of (additional) storage, and a jump node needs 3 additional words (two pointers and one integer). Let A_n denote the (random) number of jump nodes, excluding the dummy header, of a random median-of-k jumplist with leaf size w on n keys, then its additional memory requirement is $3(A_n + 1) + 1(n - A_n)$. It remains to show that A_n is asymptotically at most $1/\big((w+1)(H_{k+1} - H_{t+1})\big)n$.

A_n counts the internal nodes in a random fringe-balanced dangling-min BST over n keys; a distributional recurrence is thus easy to set up:

$$A_n \;\overset{\mathcal{D}}{=}\; \begin{cases} 1 + A_{J_1} + A_{J_2}, & n > w - 1, \\ 0, & n \leq w - 1. \end{cases}$$

Here again $J_r = I_r + t$, $r \in \{1, 2\}$, $J_1 \overset{\mathcal{D}}{=} \mathrm{BetaBin}(n - 1 - k; t+1, t+1)$ and $J_2 = n - 1 - k - J_1$. For A_n, the DMT only gives us $\mathbb{E}[A_n] = O(n)$ (Case 3). It is easy to see that $\mathbb{E}[A_n]$ is also $\Omega(n)$, but a precise leading-term seems very hard to obtain.

Proof of Thm. 7.1−(d): The recurrence for A_n is very similar to that for the number of partitioning steps in median-of-k Quicksort with Insertionsort threshold $w - 1$; the only difference is that we there have $I_1 \overset{\mathcal{D}}{=} I_2 \overset{\mathcal{D}}{=} \mathrm{BetaBin}(n - k; t+1, t+1)$, i.e., with $n - k$ instead of $n - k - 1$. By monotonicity, $\mathbb{E}[A_n]$ is at most the number of partitioning steps in Quicksort since also the subproblems sizes are smaller. The number of partitioning steps in median-of-k Quicksort with Insertionsort threshold M is $1/\big((M+2)(H_{k+1} - H_{t+1})\big)n \pm O(1)$, see, e.g., [8, p. 327]. Setting $M = w - 1$ yields the claim. □

8 Conclusion

In this article, we presented median-of-k jumplists and analyzed their efficiency in terms of the expected number

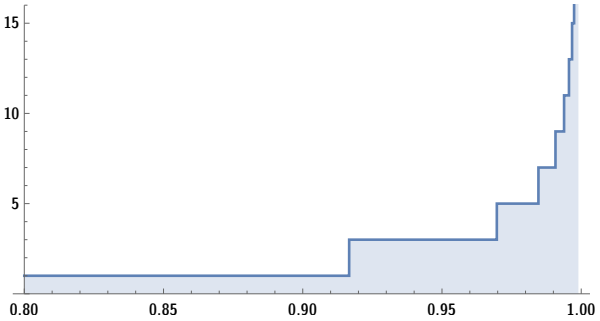


Figure 7: The k that minimizes the leading-term coefficient of total costs of insertion/deletion, if one comparison costs $\xi \in [0,1]$ and each rebalanced element costs $1-\xi$, i.e., $\arg\min_k \xi \cdot \frac{1}{H(t)} + (1-\xi) \cdot \frac{k}{H(t)}$ as a function of ξ.

of comparisons (for searches) and rebalanced elements (for updates). The precise analysis of insertion and deletion costs is also novel for the original version of jumplists ($k = 1$).

Our analysis shows that a search profits from sampling; in particular going from $k = 1$ to $k = 3$ entails significant savings: $\frac{12}{7} \ln n \approx 1.714 \ln n$ instead of $2 \ln n$ comparisons on average. As for median-of-k Quicksort, we see diminishing returns for much larger k. For jumplists, also the cleanup after insertions and deletions gets more expensive; the effort grows linearly with k. Very large k will thus be harmful.

The efficiency of insertion and deletion depends on both the time for search and the time for cleanup, so it is natural to ask for optimal k. Since the cost units are rather different (comparisons vs. rebalanced elements) we need a weighing factor. Depending on the relative weight $\xi \in [0,1]$ of comparisons, we can compute optimal k, see Fig. 7. In the realistic range, we should try $k = 1$, 3, or 5, unless we do many more searches than updates.

We conducted a small running time study based on a proof-of-concept implementation [25] in Java that confirms our analytical findings: Sampling leads to some savings for searches, but slows down insertions and deletions significantly. Comparing running times with that of Java's `TreeMap` (a red-black tree implementation) shows that our data structure is only partially competitive: for iterating over all elements, jumplists are about 50% faster, but searches are between 20% and 100% slower (depending on the choice for w) and for insertions/deletions `TreeMap`s are 5 to 10 times faster. However, `TreeMap`s use 4 additional words per key (without even storing subtree sizes needed for efficient rank-based access), whereas our jumplists never need more than $\sim 2.\overline{3}$ additional words per key and less than 1.04 with $w \approx 100$. For $n = 10^6$ keys, $w \approx 100$ did not affect searches much (+25%) but actually sped up insertions and deletions (roughly by a factor of 2!).

8.1 Future Work. Some interesting questions are left open. What is the optimal choice for w? Answering this question requires second-order terms of search, insertion and deletion costs; due to the underlying mathematical challenges it is unlikely that those can be computed exactly, but an upper bound using analysis results on Quicksort should be possible. Other future directions are the analysis of branch misses, in particular in the context of an asymmetric sampling strategy, and the design of a "bulk insert" algorithm that is faster than inserting elements subsequently, one at a time.

On modern computers the cache performance of data structures is important for their running time efficiency. Here, a larger fanout of nodes is beneficial since it reduces the expected number of I/Os. For jumplists this can be achieved by using more than one jump pointer in each node. The case of two jump pointers per node has been worked out in detail [16], but the general scheme invites further investigation.

References

[1] A. Andersson, R. Fagerberg, and K.S. Larsen. Balanced binary search trees. In D. Mehta and S. Sahni, editors, *Handbook of Data Structures and Applications*, chapter 10. CRC Press, 2005.

[2] Hervé Brönnimann, Frédéric Cazals, and Marianne Durand. Randomized jumplists: A jump-and-walk dictionary data structure. In *STACS 2003*, pages 283–294, 2003. doi:10.1007/3-540-36494-3_26.

[3] R. Casas, J. Díaz, and C. Martinez. Statistics on random trees. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 186–203. Springer, 1991. doi:10.1007/3-540-54233-7_134.

[4] Brian C. Dean and Zachary H. Jones. Exploring the duality between skip lists and binary search trees. In *Annual southeast regional conference*, pages 395–399. ACM Press, 2007. doi:10.1145/1233341.1233413.

[5] Michael Drmota. *Random Trees*. Springer, 2009.

[6] Amr Elmasry. Deterministic jumplists. *Nordic Journal of Computing*, 12(1):27–39, 2005.

[7] Daniel Hill Greene. *Labelled formal languages and their uses*. Ph.D. thesis, Stanford University, 1983.

[8] Pascal Hennequin. Combinatorial analysis of Quicksort algorithm. *RAIRO - Theoretical Informatics and Applications*, 23(3):317–333, 1989.

[9] Pascal Hennequin. *Analyse en moyenne d'algorithmes : tri rapide et arbres de recherche*. Thèse (Ph.D. Thesis), Ecole Politechnique, Palaiseau, 1991.

[10] Shou-Hsuan Stephen Huang and C. K. Wong. Binary search trees with limited rotation. *BIT*, (4):436–455, 1983. doi:10.1007/BF01933619.

[11] Shou-Hsuan Stephen Huang and C. K. Wong. Average number of rotations and access cost in iR-trees. *BIT*, 24(3):387–390, 1984. doi:10.1007/BF02136039.

[12] Donald E. Knuth. *The Art Of Computer Programming: Searching and Sorting*. Addison Wesley, 2nd edition, 1998.

[13] Hosam M. Mahmoud. *Evolution of Random Search Trees*. Wiley, 1992.

[14] Conrado Martínez and Salvador Roura. Randomized binary search trees. *J. ACM*, 45(2):288–323, 1998. doi:10.1145/274787.274812.

[15] J. Ian Munro, Thomas Papadakis, and Robert Sedgewick. Deterministic skip lists. In *ACM-SIAM Symposium on Discrete Algorithms*, SODA 1992, pages 367–375. SIAM, 1992.

[16] Elisabeth Neumann. *Randomized Jumplists With Several Jump Pointers*. Bachelor's thesis, 2015. URL: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:hbz:386-kluedo-41642.

[17] J. Nievergelt and E. M. Reingold. Binary search trees of bounded balance. *SIAM Journal on Computing*, 2(1):33–43, 1973. doi:10.1137/0202005.

[18] Tomi A. Pasanen. Random binary search tree with equal elements. *Theoretical Computer Science*, 411(43):3867–3872, 2010. doi:10.1016/j.tcs.2010.06.023.

[19] Patricio V Poblete and J. Ian Munro. The analysis of a fringe heuristic for binary search trees. *Journal of Algorithms*, 6(3):336–350, 1985. doi:10.1016/0196-6774(85)90003-3.

[20] William Pugh. Skip lists: A probabilistic alternative to balanced trees. *Communications of the ACM*, 33(6):668–676, 1990. doi:10.1145/78973.78977.

[21] Salvador Roura. Improved Master Theorems for Divide-and-Conquer Recurrences. *Journal of the ACM*, 48(2):170–205, 2001.

[22] R. Seidel and C. R. Aragon. Randomized search trees. *Algorithmica*, 16(4-5):464–497, 1996. URL: http://link.springer.com/10.1007/BF01940876, doi:10.1007/BF01940876.

[23] A. Walker and D. Wood. Locally balanced binary trees. *The Computer Journal*, 19(4):322–325, 1976. doi:10.1093/comjnl/19.4.322.

[24] Sebastian Wild. *Dual-Pivot Quicksort and Beyond: Analysis of Multiway Partitioning and Its Practical Potential*. Doktorarbeit (Ph.D. thesis), Technische Universität Kaiserslautern, 2016. URL: http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:hbz:386-kluedo-44682.

[25] Sebastian Wild. sebawild/jumplists: snapshot-for-paper. 2016. doi:10.5281/zenodo.155326.

[26] Sebastian Wild. Quicksort is optimal for many equal keys. In *Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 8–22. SIAM, 2018. doi:10.1137/1.9781611975062.2.