Quicksort Is Optimal For Many Equal Keys*

Sebastian Wild†

Abstract

I prove that the average number of comparisons for median-of-k Quicksort (with fat-pivot a.k.a. three-way partitioning) is asymptotically only a constant α_k times worse than the lower bound for sorting random multisets of n elements with $\Omega(n^\varepsilon)$ duplicates of each value (for any $\varepsilon > 0$). The constant is $\alpha_k = \ln(2)/\big(H_{k+1} - H_{(k+1)/2}\big)$, which converges to 1 as $k \to \infty$, so median-of-k Quicksort is asymptotically optimal for inputs with many duplicates. This partially resolves a conjecture by Sedgewick and Bentley (1999, 2002) and constitutes the first progress on the analysis of Quicksort with equal elements since Sedgewick's 1977 article.

1 Introduction

Sorting is one of the basic algorithmic tasks that is used as a fundamental stepping stone for a multitude of other, more complex challenges. Quicksort is the method of choice for sorting in practice. Any undergraduate student learns as a justification for its quality that the average number of comparisons is $\Theta(n \log n)$ for a random permutation of n distinct elements, putting it into the same complexity class as, e.g., Mergesort. By choosing random pivots we can make this average the *expected* behavior regardless of the order of the input. Moreover, the hidden constants in $\Theta(n \log n)$ are actually small: Quicksort needs $2\ln(2) n \operatorname{ld}(n) \pm O(n)$ comparisons in expectation, i.e., asymptotically only a factor $2\ln(2) \approx 1.39$ more than the information-theoretic lower bound for any comparison-based sorting method.[1]

By choosing the pivot as median of k sample elements in each step, for k a fixed, odd integer, this constant can be reduced to $\alpha_k = \ln(2)/\big(H_{k+1} - H_{(k+1)/2}\big)$ [21], where $H_k = \sum_{i=1}^{k} 1/i$. Note that α_k converges to 1 as $k \to \infty$, so Quicksort's expected behavior is asymptotically optimal on random permutations.

These guarantees certainly justify the wide-spread use of Quicksort, but they only apply to inputs with n *distinct* elements. In many applications, duplicates (i.e., elements that are equal w.r.t. the order relation) appear naturally. The SQL clause `GROUP BY C`, for example, can be implemented by first sorting rows w.r.t. column `C` to speed up the computation of aggregating functions like `COUNT` or `SUM`. Many rows with equal `C`-entries are expected in such applications.

At the *KnuthFest* celebrating Donald Knuth's 1000000_2th birthday, Robert Sedgewick gave a talk titled *"Quicksort is optimal"* [24],[2] presenting a result that is at least very nearly so: Quicksort with *fat-pivot* (a.k.a. *three-way*) partitioning uses $2\ln 2 \approx 1.39$ times the number of comparisons needed by *any* comparison-based sorting method for *any* randomly ordered input, with or without equal elements. He closed with the conjecture that this factor can be made arbitrarily close to 1 by choosing the pivot as the median of k elements for sufficiently large k. This statement is referred to as the Sedgewick-Bentley conjecture.

The Sedgewick-Bentley conjecture is a natural generalization of the distinct-keys case (for which the statement is known to hold true), and sorting experts will not find the claim surprising, since it is the theoretical justification for a long-established best practice of general-purpose sorting: as observed many times in practice, Quicksort with the fat-pivot partitioning method by Bentley and McIlroy [3][3] and *ninther* (a.k.a. *pseudomedian of nine*) pivot sampling comes to within *ten percent* of the optimal expected comparison count for inputs with or without equal keys.

In this paper, I confirm the Sedgewick-Bentley conjecture for inputs with "many duplicates", i.e., where every key value occurs $\Omega(n^\varepsilon)$ times for an arbitrary

*An extended version of this paper with full proofs is available online: `arxiv.org/abs/1608.04906`.

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

[1] I write ld for $\log_2$, and ln for $\log_e$.

[2] Sedgewick presented the conjecture in a less widely-known talk already in 1999 [23]. Since they never published their results in an article, I briefly reproduce their arguments in Section 4.

[3] The implementation by Bentley and McIlroy [3] from 1993 is used in important programming libraries, e.g., the GNU C++ Standard Template Library (STL) and the Java runtime library (JRE). Since version 7, the JRE uses dual-pivot Quicksort instead, but if the two sampled pivots are found *equal*, it falls back to fat-pivot partitioning by Bentley and McIlroy.

constant $\varepsilon > 0$. To do so I show a bound on the constant of proportionality, namely the same α_k mentioned above. (Sedgewick and Bentley did not include a guess about the constant or the speed of convergence in their conjecture). While Quicksort can certainly be outperformed for particular types of inputs, the combination of simple, efficient code and almost universal proven optimality is unsurpassed; it is good news that the latter also includes inputs with equal keys.

Confirming the Sedgewick-Bentley conjecture is not a surprising outcome, but it has been surprisingly resistant to all attempts to formally prove it: no progress had been made in the nearly two decades since it was posed (see Section 5 for some presumable reasons for that), so restricting the problem might be sensible. I would like to remark that my restriction of many duplicates is a technical requirement of the *analysis*, but we have no reason to believe that Quicksort performs much different when it is violated. More importantly, it is *not* the purpose of this paper to suggest sorting inputs with many duplicates as a natural problem per se, for which tailored methods shall be developed. (One is tempted to choose a hashing-based approach when we *know* that the number u of different values is small, but my results show that precisely for such instances, Quicksort will be competitive to any tailored algorithm!) It must be seen as an idiosyncrasy of the present analysis one might try to overcome in the future.

Methods. For the analysis we will work in an alternative input model: instead of fixing the *exact* multiplicities of a multiset, we fix a *discrete probability distribution* over $[1..u]$ and draw n elements independently and identically distributed (i.i.d.) according to that distribution. We will see that going from fixed to expected multiplicities only increases the average sorting costs, so the i.i.d. model provides an upper bound.

The analysis of Quicksort on discrete i.i.d. inputs proceeds in two steps: First, I use that costs of (median-of-k) Quicksort correspond to costs of searching in a (k-fringe-balanced) binary search tree (BST). A concentration argument shows that for inputs with many duplicates, the search tree typically reaches a stationary state after a short prefix of the input, and is thus independent of the (larger) rest of the input.

The second step is to determine the expected search costs in a k-fringe-balanced BST built by repeatedly inserting i.i.d. elements until all u values appear in the tree; (I call the tree *saturated* then). I show that the search costs are asymptotically α_k times the *entropy* of the universe distribution. To do so I derive lower and upper bounds from the recurrence of costs (with the *vector* of probability weights as argument) using the aggregation property of the entropy function. To the best of my knowledge, the analysis of search costs in fringe-balanced trees with equal keys is novel, as well.

Most used techniques are elementary, but keeping the error terms under control (so that $\Omega(n^\varepsilon)$ duplicates suffice for any $\varepsilon > 0$) made some refinements of classical methods necessary that might be of independent interest.

Outline. We start with some notation and a collection of known results (Section 2), and introduce the input models (Section 3). Section 4 is devoted to related work on sorting in the presence of equal elements. We continue in Section 5 with a formal statement of the main result and a briefly discussion why some previous attempts did not succeed. In Sections 6 – 9, we study the cost of fat-pivot Quicksort on discrete i.i.d. inputs as outlined above. We establish a lower bound for the i.i.d. model in Section 10. Section 11 proves the main result by combining all pieces. The results are summarized and put into context in Section 12.

There is an extended version of this paper available online `arxiv.org/abs/1608.04906` that contains full proofs and more detailed comments.

2 Preliminaries

We start with some common notation and a precise description of fat-pivot Quicksort with median-of-k sampling, as well as the corresponding a search tree variant, the k-fringe-balanced trees. We collect further known results that will be used later on.

2.1 Notation. I write vectors in bold font $\boldsymbol{x} = (x_1, \dots, x_n)$ and always understand them as column vectors (even when written in a row in the text). By default, all operations on vectors are meant componentwise. By $\Sigma\boldsymbol{x}$, I denote $x_1 + \cdots + x_n$, the *total* of $\boldsymbol{x}$. The *profile* (or multiplicities vector) a multiset M over $[u] = \{1, \dots, u\}$ is a vector $\boldsymbol{x} = (x_1, \dots, x_u)$ where every values $v \in [u]$ occurs x_v times in M.

I use Landau-symbols (O, Ω etc.) in the formal sense of Flajolet and Sedgewick [7, Section A.2] and write $f(n) = g(n) \pm O(h(n))$ to mean $|f(n) - g(n)| = O(h(n))$. (I use $\pm$ to emphasize the fact that we specify $f(n)$ up to an additive error of $h(n)$ without restricting the sign of the error.)

For a random variable X, $\mathbb{E}[X]$ is its expectation and $\mathbb{P}[X = x]$ denotes the probability of the event $X = x$. For $\boldsymbol{q} \in [0,1]^u$ with $\Sigma\boldsymbol{q} = 1$, I write $U \stackrel{\mathcal{D}}{=} \mathcal{D}(\boldsymbol{q})$ to say that U is a random variable with $\mathbb{P}[U = i] = q_i$ for $i \in [u]$; generally, $\stackrel{\mathcal{D}}{=}$ denotes equality in distribution. $\mathrm{Mult}(n, \boldsymbol{q})$ is a *multinomial distributed* variable with n trials from $\mathcal{D}(\boldsymbol{q})$, and $\mathrm{Beta}(\alpha, \beta)$ is a *beta distributed* variable with parameters $\alpha, \beta > 0$. By $\mathcal{H}$ or $\mathcal{H}_{\mathrm{ld}}$ I denote the binary *Shannon entropy* $\mathcal{H}(\boldsymbol{q}) = \sum_{i=1}^{u} q_i \,\mathrm{ld}(1/q_i)$; likewise $\mathcal{H}_{\ln}$ is the base e entropy.

I say an event $E = E(n)$ occurs *with high probability (w.h.p.)* if for *any* constant c holds $\mathbb{P}[E(n)] = 1 \pm O(n^{-c})$ as $n \to \infty$. Note that other sources use w.h.p. if any such $c \in \mathbb{N}$ exists, which is a much weaker requirement. Similarly for $X_1, X_2, \ldots$ a sequence of real random variables, I say "$X_n = O(g(n))$ w.h.p." if for every constant c there is a constant d so that $\mathbb{P}[|X_n| \le d|g(n)|] = 1 \pm O(n^{-c})$ as $n \to \infty$.

2.2 Reference Version of Fat-Pivot Quicksort. By "fat-pivot" I mean a partitioning method that splits the input into three parts: elements (strictly) smaller than the pivot, elements equal to the pivot, and elements (strictly) larger than the pivot. Instead of only one pivot element in binary partitioning, we now obtain a "fat" pivot segment that separates the left and right subproblems. This idea is more commonly known as *three-way partitioning*, but I prefer the vivid term fat pivot to make the distinction between fat-pivot partitioning and partitioning around *two pivots* clear. The latter has become fashionable again in recent years [12, 27, 26]. Since all duplicates of the pivot are removed before recursive calls, the number of partitioning rounds is bounded by the number of different values in the input.

Algorithm 1. Fat-pivot median-of-k Quicksort.

```
QUICKSORT_k(L)
 1  if (L.length) ≤ k − 1
 2      return INSERTIONSORT(L)
 3  end if
 4  P := MEDIAN(L[1..k])
 5  L_1, L_2, L_3 := new List
 6  while ¬ L.empty
 7      U := L.removeFirstElement()
 8      case distinction on cmp(U, P)
 9          in case < do L_1.append(U)
10          in case = do L_2.append(U)
11          in case > do L_3.append(U)
12      end cases
13  end while
14  L_1 := QUICKSORT_k(L_1)
15  L_3 := QUICKSORT_k(L_3)
16  return L_1.append(L_2).append(L_3)
```

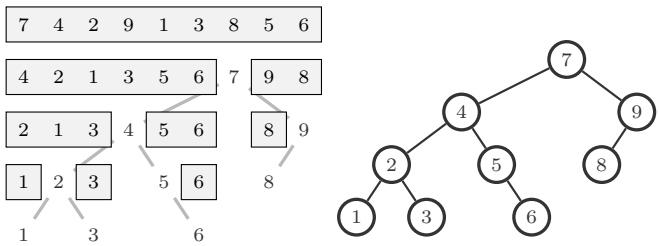
To simplify the presentation, I assume in this work that partitioning *retains the relative order* of elements that go to the same segment. A corresponding reference implementation operating on linked lists is given in Algorithm 1. It uses the median of k sample elements as pivot, where the sample size $k = 2t + 1$, $t \in \mathbb{N}_0$, is a tuning parameter of the algorithm. We consider k a fixed constant in the analysis.

I always mean Algorithm 1 when speaking of Quicksort in this paper. Moreover, by its costs I always mean the number of ternary key comparisons, i.e., the number of calls to *cmp*. Apart from selecting the median, one partitioning step in Algorithm 1 uses exactly n ternary comparisons.

Algorithm 1 serves as precise specification for the analysis but it is not the method of choice in practice. However, our results apply to practical methods, as well (see the discussion in Section 6.1).

2.3 Quicksort and Search Trees. It has become folklore that the comparison costs of (classic) Quicksort are the same as the internal path length of a BST, which equals the cost to construct the tree by successive insertions. Hibbard [9] first described this fact in 1962, right after Hoare's publication of Quicksort itself [10, 11]. Formally we associate a *recursion tree* to each execution of Quicksort: Each partitioning step contributes a node labeled with the used pivot value. Its left and right children are the recursion trees of the left and right recursive calls, respectively. Figure 1 shows an example.

Figure 1: Execution trace of Quicksort without pivot sampling and its recursion tree. An animated version of this example is available online: `youtu.be/yi6syj9nksk`.



Recursion trees obviously fulfill the search tree property; in fact, the recursion tree is *exactly* the binary search tree that results from successively inserting the input elements into an initially empty tree (in the order they appear in the original input) if Quicksort always uses the first element of the list as pivot and partitions so that the relative order of elements smaller resp. larger than the pivot is retained (as is done in Algorithm 1). Even the *same set* of comparisons is then used in both processes, albeit in a different order.

2.4 Fringe-Balanced Trees. The correspondence extends to median-of-k Quicksort with an appropriate *fringe-balancing rule* for BSTs. This is a little less widely known, but also well researched [6]. Upon constructing a k-fringe-balanced search tree, we *collect* up to $k - 1$ elements in a leaf. (This corresponds to truncating

the recursion in Quicksort for $n \leq k-1$ and leave small subproblems for Insertionsort.) Once a leaf has collected $k = 2t+1$ elements, it is *split:* Simulating the pivot selection process in Quicksort, we find the median from the k elements in the leaf and use it as the label of a new inner node. Its children are two new leaves with the elements that did not become pivots: the t smaller elements go to the left, the t larger to the right. Because of the dynamic process of splitting leaves, the correspondence is best shown in motion; the supplementary animation (`youtu.be/yi6syj9nksk`) includes fringe-balanced trees.

More formally, a k-fringe-balanced tree, for $k = 2t+1$ an odd integer, is a binary search tree whose leaves can store between 0 and $k-1$ elements. An empty fringe-balanced tree is represented as $\mathit{Leaf}()$, a single empty leaf. The k-fringe-balanced tree $\mathcal{T}$ corresponding to a sequence of elements $x_1, \ldots, x_n$ is obtained by successively inserting the elements into an initially empty tree using Algorithm 2; more explicitly, with $T_0 = \mathit{Leaf}()$ and $T_i = \textsc{Insert}_k(\mathcal{T}_{i-1}, x_i)$ for $1 \leq i \leq n$, we have $\mathcal{T} = \mathcal{T}_n$.

Algorithm 2. Insert into k-fringe-balanced tree.

$\textsc{Insert}_k(\mathcal{T}, x)$
1 **if** $\mathcal{T}$ is $\mathit{Leaf}(\boldsymbol{U})$
2 Append x to $\boldsymbol{U}$
3 **if** $|\boldsymbol{U}| \leq k-1$ **then return** $\mathit{Leaf}(\boldsymbol{U})$ **end if**
  // Else: Split the leaf
4 $P := \textsc{Median}(U_1, \ldots, U_k)$
5 $C_1, C_2 :=$ new empty list
6 **for each** U in $\boldsymbol{U}$
7 **case distinction** on $\mathit{cmp}(U, P)$
8 **in case** $U < P$ **do** append U to C_1
9 **in case** $U > P$ **do** append U to C_2
  // In case $U == P$, we drop U.
10 **end cases**
11 **end for**
12 **return** $\mathit{Inner}\big(P, \mathit{Leaf}(C_1), \mathit{Leaf}(C_2)\big)$
13 **else** $\mathcal{T}$ is $\mathit{Inner}(P, \mathcal{T}_1, \mathcal{T}_2)$
14 **case distinction** on $\mathit{cmp}(x, P)$
15 **in case** $x == P$ **do**
16 **return** $\mathcal{T}$ // tree unchanged
17 **in case** $x < P$ **do**
18 **return** $\mathit{Inner}\big(P, \textsc{Insert}_k(\mathcal{T}_1, x), \mathcal{T}_2\big)$
19 **in case** $x > P$ **do**
20 **return** $\mathit{Inner}\big(P, \mathcal{T}_1, \textsc{Insert}_k(\mathcal{T}_2, x)\big)$
21 **end cases**
22 **end if**

Growing trees with Algorithm 2 enforces a certain balance upon the lowest subtrees, i.e., at the *fringe* of the tree, hence the name *fringe-balanced*. Searching an element in a fringe-balanced tree works as in an ordinary BST, except for the leaves, where we sequentially search through the buffer. (Note that elements collected in leaves are not sorted.)

Duplicate Insertions. Since our analysis of Quicksort with equal keys builds on the precise performance of fringe-balanced trees, we put a particular emphasis is on the treatment of duplicate insertions in Algorithm 2. If a key x is already present during insertion and x appears as key of an *inner* node, the (repeated) insertion has no effect; but if x is found in a *leaf*, another copy of x is appended to the buffer of that leaf.

2.5 Tail Bounds.

We recall the classical Chernoff bound.

Lemma 2.1 (Chernoff Bound): *Let $X \stackrel{\mathcal{D}}{=} \mathrm{Bin}(n,p)$ for $n \in \mathbb{N}$ and $p \in (0,1)$ and let $\delta \geq 0$. Then*

$$\mathbb{P}\left[\left|\frac{X}{n} - p\right| \geq \delta\right] \;\leq\; 2\exp(-2\delta^2 n). \qquad \square$$

This bound appears, e.g., as Theorem 2.1 of McDiarmid [18]. We also use the following elementary observation; a proof appears in the extended version of this paper.

Lemma 2.2 (Far-End Left-Tail Bound): *Let $X^{(n)} \stackrel{\mathcal{D}}{=} \mathrm{Bin}(n, p^{(n)})$ be a sequence of random variables and $k \in \mathbb{N}$ a constant, where $p^{(n)}$ satisfies $p^{(n)} = \omega\big(\frac{\log n}{n}\big)$ as $n \to \infty$ and is bounded away from 1, i.e., there is a constant $\varepsilon > 0$ so that $p^{(n)} \leq 1-\varepsilon$ for all n. Then $\mathbb{P}[X^{(n)} \leq k] = o(n^{-c})$ as $n \to \infty$ for any constant c.*

2.6 Logarithmic Tree Height w.h.p..

It is a folklore theorem that randomly grown search trees have logarithmic height with high probability. For the present work, the following result is sufficient.

Proposition 2.3 (Height-Degeneracy): *For any fixed (odd) k holds: the probability that a k-fringe-balanced search tree built from n randomly ordered elements (with or without duplicates) has height $> 13 \ln n$ is in $O(1/n^2)$ as $n \to \infty$.* $\square$

The extended version of this paper gives a formal proof of Proposition 2.3 using good-split-bad-split indicators and Lemma 2.1. Curiously there is a technical pitfall in this proof that has not been rigorously addressed before (to my knowledge).

2.7 Properties of the Entropy Function.

This section collects a few useful properties of the entropy function. Proofs are given in the extended version.

Lemma 2.4 (Properties of Entropy): *Let $\mathcal{H}_{\ln} : [0,1]^u \to \mathbb{R}_{\geq 0}$ with $\mathcal{H}_{\ln}(\boldsymbol{x}) = \sum_{i=1}^{u} x_i \ln(1/x_i)$ be the base e entropy function.*

(a) $\mathcal{H}_{\ln}(\boldsymbol{x}) = \ln(2)\mathcal{H}_{\mathrm{ld}}(\boldsymbol{x})$.

(b) *For all $\boldsymbol{x} \in [0,1]^u$ with $\Sigma\boldsymbol{x} = 1$ we have that $0 \leq \mathcal{H}_{\ln}(\boldsymbol{x}) \leq \ln(u)$.*

(c) *$\mathcal{H}_{\ln}$ is Hölder-continuous in $[0,1]^u$ for any exponent $h \in (0,1)$, i.e., there is a constant $C = C_h$ such that $|f(\boldsymbol{y}) - f(\boldsymbol{x})| \leq C_h u \cdot \|\boldsymbol{y} - \boldsymbol{x}\|_\infty^h$ for all $\boldsymbol{x}, \boldsymbol{y} \in [0,1]^u$. A possible choice for C_h is given by*

$$C_h \;=\; \left(\int_0^1 \left|\ln(t) + 1\right|^{\frac{1}{1-h}}\right)^{1-h} \tag{1}$$

For example, $h = 0.99$ yields $C_h \approx 37.61$. □

A beta distributed variable Π can be seen as a *random* probability. We can then ask for the *expected* entropy of a Bernoulli trial with probability Π. Using that

$$\int_0^1 z^{a-1}(1-z)^{b-1}\ln(z)\,dz \;=\; \mathrm{B}(a,b)\big(\psi(a) - \psi(a+b)\big), \quad (a,b>0), \tag{2}$$

(which is Equation (4.253-1), p. 540, of Gradshteyn and Ryzhik [8] for $r = 1$), we obtain the answer. (Full details are given in the extended version.)

Lemma 2.5 (Expected Entropy Beta Variables): *Let $t \in \mathbb{N}_0$. For $\Pi \stackrel{\mathcal{D}}{=} \mathrm{Beta}(t+1, t+1)$ and $k = 2t+1$ we have*

$$\mathbb{E}\big[\mathcal{H}_{\ln}(\Pi, 1-\Pi)\big] \;=\; H_{k+1} - H_{t+1}.$$

3 Input Models

This section formally defines the input models and some related notation.

3.1 Multiset Model. In the *random multiset permutation model* (a.k.a. *exact-profile model*), we have parameters $u \in \mathbb{N}$, the *universe size,* and $\boldsymbol{x} \in \mathbb{N}^u$, the fixed *profile*. An input under this model always has size $n = \Sigma\boldsymbol{x}$, and is given by a uniformly chosen random permutation of

$$\underbrace{1,\ldots,1}_{x_1 \text{ copies}}, \underbrace{2,\ldots,2}_{x_2 \text{ copies}}, \ldots, \underbrace{u,\ldots,u}_{x_u \text{ copies}},$$

i.e., the multiset with x_v copies of the number v for $v = 1,\ldots,u$. I write $C_{\boldsymbol{x}}^{(k)}$ for the (random) number of (ternary) comparisons used by Algorithm 1 to sort a random multiset permutation with profile $\boldsymbol{x}$; I will in the following use $C_{\boldsymbol{x}}$ (omitting k) for conciseness.

3.2 I.I.D. Model. In the *(discrete) i.i.d. model* (a.k.a. *probability model* [14] or *expected-profile model* [26]) with parameters $u \in \mathbb{N}$ and $\boldsymbol{q} \in (0,1)^u$ with $\Sigma\boldsymbol{q} = 1$, an input of size n consists of n i.i.d. (independent and identically distributed) random variables $U_1,\ldots,U_n$ with $U_i \stackrel{\mathcal{D}}{=} \mathcal{D}(\boldsymbol{q})$ for $i = 1,\ldots,n$. The domain $[u]$ is called the *universe,* and $\boldsymbol{q}$ the (probability vector of the) *universe distribution*.

I denote by X_v, for $v \in [u]$, the number of elements U_i that have value v; the vector $\boldsymbol{X} = (X_1,\ldots,X_u)$ of all these *multiplicities* is called the *profile* of the input $\boldsymbol{U} = (U_1,\ldots,U_n)$. Clearly, $\boldsymbol{X}$ has a multinomial distribution, $\boldsymbol{X} \stackrel{\mathcal{D}}{=} \mathrm{Mult}(n,\boldsymbol{q})$, with mean $\mathbb{E}[\boldsymbol{X}] = n\boldsymbol{q}$.

By $C_{n,\boldsymbol{q}} = C_{n,\boldsymbol{q}}^{(k)}$, I denote the number of (ternary) comparisons used by Algorithm 1 to sort n i.i.d. $\mathcal{D}(\boldsymbol{q})$ elements (again usually omitting the dependence on k).

3.3 Relation of the Two Models. The two input models—random multiset permutations with profile $\boldsymbol{x}$ resp. $n = \Sigma\boldsymbol{x}$ i.i.d. elements with distribution $\boldsymbol{q} = \boldsymbol{x}/n$—are closely related. Both can be described using an urn with (initially) n balls, where for every $v \in [u]$ exactly x_v balls bear the label v. The multiset model corresponds to drawing n balls from this urn without replacement (ultimately emptying the urn completely), so that the profile of the input is always $\boldsymbol{x}$. The i.i.d. model corresponds to drawing n balls from the the urn *with* replacement; this leaves the urn unchanged and the n draws are mutually independent.

In our reference Quicksort (Algorithm 1), we use the first k elements, i.e., the first k balls drawn from the urn, to determine the pivot: it is the median of this sample. By that, we try to estimate the median of all n balls (initially) in the urn, so as to obtain a balanced split. In the multiset model, the k elements are drawn *without* replacement, whereas in the i.i.d. model, they are drawn *with* replacement from the urn. The algorithm is of course the same in both cases (and indeed chooses sample elements *without* replacement), but the mutual independence of elements in the i.i.d. model implies that the sampling process is (in this latter case) equivalent to sampling *with* replacement.

Now it is well-known that sampling without replacement yields a *better* (i.e., less variable) estimate of the true median than sampling with replacement, and thus leads to more favorable splits in Quicksort! The above reasoning applies for recursive calls, as well, so that on average the multiset model is no more costly for Quicksort than the i.i.d. model: $\mathbb{E}[C_{\boldsymbol{x}}] \leq \mathbb{E}[C_{n,\boldsymbol{x}/n}]$. We will thus focus on analyzing the i.i.d. model.[4]

[4] The full version of this paper will discuss the multiset model and the above reasoning in more detail.

4 Previous Work

We briefly discuss previous results relevant for this paper; the extended version contains a more comprehensive summary of related work.

In the *multiset sorting problem*, we are given a permutation of a *fixed* multiset, where value $v \in [u]$ appears $x_v \in \mathbb{N}$ times, for a total of $n = \Sigma\boldsymbol{x}$ elements. Neither u nor $\boldsymbol{x}$ are known to the algorithm. We consider only comparison-based sorting; unless stated otherwise, all results concern the number of *ternary* comparisons, i.e., one comparison has as result $<$, $=$ or $>$.

4.1 Multiset Sorting. Multiset sorting attracted considerable attention in the literature. Munro and Raman [20] prove a lower bound of $n \operatorname{ld} n - \sum_{i=1}^{u} x_i \operatorname{ld} x_i - n \operatorname{ld} e \pm O(\log n)$ (ternary) comparisons, which can equivalently be written in terms of the entropy as $\mathcal{H}(\boldsymbol{x}/n)n - n \operatorname{ld} e \pm O(\log n)$. We reproduce their main argument in Section 10 and extend it to i.i.d. inputs.

Algorithms to optimally sort multisets in the worst case up to an $O(n)$ error term are known [19, 20]. None of these methods made it into practical library implementations since they incur significant overhead w.r.t. existing sorting methods when there are not many equal keys in the input.

4.2 Quicksort on Multiset Permutations. Building on Burge's analysis of BSTs [4], Sedgewick analyzed several Quicksort variants on random permutations of multisets in his 1977 article [22]. For fat-pivot Quicksort without sampling, he found the exact average-case result (when every permutation of the multiset is equally likely): $2\mathcal{H}_Q(\boldsymbol{x}) + n - u$ ternary comparisons, where $\mathcal{H}_Q(\boldsymbol{x}) = \sum_{1 \le i < j \le u} x_i x_j / (x_i + \cdots + x_j)$ is the so-called "Quicksort entropy".

Two decades later, Sedgewick and Bentley [23, 24] combined this exact, but somewhat unwieldy result with the bound $\mathcal{H}_Q(\boldsymbol{q}) \le \mathcal{H}(\boldsymbol{q}) \ln 2$ and concluded that with at most $\big(2\ln(2)\mathcal{H}(\boldsymbol{x}/n) + 1\big)n$ ternary comparisons on average, fat-pivot Quicksort is asymptotically optimal in the average case for sorting a random permutation of any fixed multiset—up to the constant factor $2 \ln 2$. The bound $\mathcal{H}_Q(\boldsymbol{q}) \le \mathcal{H}(\boldsymbol{q}) \ln 2$ was noted in a seemingly unrelated context by Allen and Munro [1, Theorem 3.2] that appeared just one year after Sedgewick's Quicksort paper [22]. Allen and Munro studied the move-to-root heuristic for self-organizing BSTs, which they found to have the *same* search costs in the long run as a BST built by inserting elements drawn i.i.d. according to the access distribution until saturation. We will consider this connection between Quicksort and search trees in detail in Section 6.

Katajainen and Pasanen considered Quicksort-based approaches for multiset sorting. Their analysis only shows that we use at most $\mathcal{H}(\boldsymbol{x}/n)n + (\frac{2}{\operatorname{ld} e} - 1)n \ln n \pm O(n)$ comparisons, which is *not* entropy-optimal.

4.3 Fringe-Balanced Trees. Many parameters like path length, height and profiles of fringe-balanced trees have been studied when the trees are built from a random permutation of n distinct elements, see, e.g., Drmota [6]. The case of equal elements has not been considered except for the unbalanced case $k = 1$, i.e., ordinary BSTs; see Kemp [14], Archibald and Clément [2].

5 New Results

We now state the main claim of this paper. We include n as sub- or superscript whenever the dependence on the input size is important.

Theorem 5.1 (Main Result): *Let $(u_n)_{n\in\mathbb{N}}$ be a sequence of integers and $(\boldsymbol{q}^{(n)})_{n\in\mathbb{N}}$ be a sequence of vectors with $\boldsymbol{q}^{(n)} \in (0,1)^{u_n}$ and $\Sigma\boldsymbol{q}^{(n)} = 1$ for all $n \in \mathbb{N}$. Assume further that $(\boldsymbol{q}^{(n)})$ has "many duplicates", i.e., there is a constant $\varepsilon > 0$ so that $\min_{v\in[u_n]} q_v^{(n)} n = \Omega(n^\varepsilon)$ as $n \to \infty$. Abbreviate the corresponding (binary) entropy by $\mathcal{H}_n = \mathcal{H}(\boldsymbol{q}^{(n)})$.*

The number $C_{n,\boldsymbol{q}^{(n)}}$ of (ternary) comparisons used by median-of-k Quicksort with fat-pivot partitioning to sort n elements drawn i.i.d. according to $\mathcal{D}(\boldsymbol{q}^{(n)})$ fulfills

$$\mathbb{E}[C_{n,\boldsymbol{q}^{(n)}}] \;=\; \alpha_k \mathcal{H}_n n \;\pm\; O\Big(\big(\mathcal{H}_n^{1-\delta} + 1\big)n\Big),$$

as $n \to \infty$ for any constant $\delta < \frac{2}{k+5}$. This number is asymptotically optimal up to the factor α_k.

Previous Approaches And Why They Fail for $k > 1$. Sedgewick's analysis [22] is based on explicitly solving the recurrence for the expected number of comparisons. Since it has a *vector*, namely the profile $\boldsymbol{x}$, as parameter, tricky differencing operations are required to obtain a telescoping recurrence. They rely on symmetries that are only present for the most basic version of Quicksort: it has now been 40 years since Sedgewick's article appeared, and not the slightest generalization of the analysis to, say, median-of-3 Quicksort, has been found.

Following our approach outlined in the introduction, we can alternatively compute the expected costs to search each element of the multiset in a BST built by inserting the same elements in random order. The random number of comparisons can thus be written as the scalar product $\boldsymbol{\Gamma}^T\boldsymbol{x}$, where $\boldsymbol{\Gamma}$ is the node-depth vector of the BST (cf. Section 6). For an ordinary BST, once an element is present in the tree, any further insertions of the *same* value are without effect; so we obtain the same tree no

matter how many duplicates of this element later follow. This means that the resulting tree has *exactly the same shape* as when we insert elements drawn i.i.d. according to $\mathcal{D}(\boldsymbol{q})$ with $\boldsymbol{q} = \boldsymbol{x}/n$ until saturation.

For median-of-k Quicksort we obtain k-fringe-balanced trees, and now a certain number of duplicate insertions *do* affect the shape of the tree. As the multiset model corresponds to drawing elements without replacement, the probabilities for the values *change* after each insertion. Analyzing the search cost then essentially reduces to solving the vector-recurrence for Quicksort with pivot sampling that has resisted all attempts for 40 years. One might conclude that the case $k = 1$ can be explicitly solved precisely because we were effectively working in the i.i.d. model instead of the multiset model.

My Assumption. The only hope I see to make progress for $k > 1$ is thus to cling to the i.i.d. model even though it is not equivalent to the multiset model anymore. We thereby retain independent insertions and the analysis of search costs in fringe-balanced trees becomes conceivable. However, we face a new problem: How often we search each value v is now a random variable X_v and the random number of comparisons is $\boldsymbol{\Gamma}^T\boldsymbol{X}$, the *product* of two random variables. Since $\boldsymbol{\Gamma}$ is the node-depth vector of a tree built by inserting a multiset with profile $\boldsymbol{X}$ (in random order), the two random variables are *not independent*.

My assumption—$\Omega(n^\varepsilon)$ expected occurrences of each value—is a simple sufficient condition to circumvent this complication (see Section 8).

6 Quicksort and Search Trees with Duplicates

The correspondence discussed in Sections 2.3/2.4 extends to inputs with equal keys if we consider a *weighted* path length in the tree, where the weight of each node is the multiplicity X_v of its key value v. This is best seen in a concrete example.

Example. Let the universe size be $u = 5$ with profile $\boldsymbol{X} = (X_1, \dots, X_5)$. Assume that $\boldsymbol{X} \geq k$ so that each of the five values is used as a pivot in *exactly* one partitioning step, and the leaves in the final tree will be empty. Assume we obtain the recursion tree shown in Figure 2.

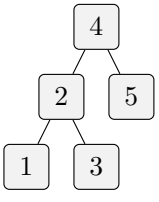


Figure 2: Exemplary recursion tree for $u = 5$. Each node represents a partitioning step, with the given pivot value. Empty leaves are not shown. The node-depths vector for this tree is $\boldsymbol{\Gamma} = (3, 2, 3, 1, 2)$.

The traditional recurrence for Quicksort sums up the costs of all partitioning steps. Each such uses one comparison per element, plus the comparisons for selecting the pivot, say, at most $c \cdot k$; ignoring the latter, we essentially sum up the subproblem sizes of all recursive calls.

For our recursion tree this yields

$$\begin{cases} X_1 + X_2 + X_3 + X_4 + X_5 \pm c\cdot k \\ X_1 + X_2 + X_3 \pm c\cdot k \\ X_1 \pm c\cdot k \\ X_3 \pm c\cdot k \\ X_5 \pm c\cdot k \end{cases}$$

comparisons, (each row corresponding to one partitioning step). For example, the step with pivot 2 gets as input all elements smaller than 4, i.e., $X_1 + X_2 + X_3$ many, so the number of comparisons used in this step is $X_1 + X_2 + X_3 \pm c \cdot k$ (for some constant c depending on the median-selection algorithm).

The key observation is that we can also read the result *column-wise,* aligning the rows by key values: up to the given error terms, we find that *sorting costs are the cost of searching each input element in the (final) recursion tree!* In vector form, we can write the search costs as $\boldsymbol{\Gamma}^T\boldsymbol{X}$, where $\boldsymbol{\Gamma}$ is the *node-depths vector* of the recursion tree, i.e., the vector of depths of nodes sorted by their keys. In the example $\boldsymbol{\Gamma} = (3, 2, 3, 1, 2)$.

6.1 Recursion Trees.

We formalize the result of the above example in two lemmas. The first one formally captures the correspondence of Quicksort and search trees.

Lemma 6.1 (Recursion Trees): *For any input $\boldsymbol{U} = (U_1, \dots, U_n)$ (with or without duplicates) the following processes execute the same set of (ternary) key comparisons and produce the same tree shape:*

(1) sorting $\boldsymbol{U}$ with Algorithm 1 and storing the recursion tree, ignoring any calls to Insertionsort,

(2) inserting $\boldsymbol{U}$ successively into an initially empty k-fringe-balanced tree using Algorithm 2.

Proof: We prove the equivalence by induction on n. If $n \leq k - 1$, Quicksort stops (it passes control directly to Insertionsort which we ignore), so the recursion tree consists of one leaf only. Likewise in the search tree, all elements are gathered in the single leaf and no comparisons happen. So assume the claim holds for inputs with less than k elements.

If now $n \geq k$, Quicksort chooses a pivot P from the first k elements, compares all elements to P, and divides the input into segments $\boldsymbol{U}^{(1)}$ and $\boldsymbol{U}^{(2)}$ containing the elements strictly smaller resp. strictly larger than P. All duplicates of P are put in place and vanish from the recursion tree.

Now consider what happens upon inserting the kth element in the search tree. This is the first time a leaf is split and the key for the inner node (the root of the tree) is chosen from the first k inserted elements overall. We thus choose same value P that was chosen as pivot in Quicksort. The other elements from the leaf are compared to P and inserted into one of the two new leaves (unless they are duplicates of P). Any later insertions must start at the root, so each of these elements are also compared to P before the insertion continues in one of the subtrees, or stops if a duplicate of P is found.

So we execute the same set of comparisons in both processes at the root of the tree. Towards applying the inductive hypothesis for recursive calls resp. subtrees, we note that the relative order of elements is retained in both processes, so the elements inserted in the left/right child of the root are exactly $\boldsymbol{U}^{(1)}$ resp. $\boldsymbol{U}^{(2)}$ in both cases. The claim thus follows by induction. □

The proof relies on the fact that Algorithm 1 retains the relative order of elements, but practical non-stable, in-place implementations (e.g., those in [3, 25]) do not fulfill this requirement. However, a weaker version of Lemma 6.1 remains valid for any fat-pivot partitioning method: the two processes always have the same *distribution* of the number of comparisons and tree shapes over *randomly ordered* inputs. Such a distributional version of Lemma 6.1 is sufficient for all results in this paper, so our results apply to practical implementations, as well.

6.2 Search Costs. The second lemma relates the costs to build a search tree to the cost of searching in the final tree.

Lemma 6.2 (Search Costs): *Let $\boldsymbol{U} = (U_1, \ldots, U_n)$ be an input (with or without duplicates), let $\mathcal{T}$ be built from $\boldsymbol{U}$ by successive insertions using Algorithm 2 and let $B_{\boldsymbol{U}}$ be the number of comparisons done in this process. Assume there are I inner nodes in $\mathcal{T}$. Searching each element of $\boldsymbol{U}$ in $\mathcal{T}$ (ignoring the sequential search in leaves) uses $B_{\boldsymbol{U}} \pm c \cdot Ik$ comparisons.*

Proof: The elements in the leaves of $\mathcal{T}$ require the exact same comparisons for insert and search to find the path to their leaf. (We ignore the sequential search within the leaf). So consider the key $v \in [u]$ of an inner node. The first occurrences of v in $\boldsymbol{U}$ are simply added to the leaf that later becomes the inner node with label v. Each of these entails one comparison more when searching for it in $\mathcal{T}$ as we paid when inserting it (namely the last comparison with v that identifies them as equal). However, there are be at most k such elements before the leaf overflows, and for the remaining duplicate insertions of v, we *do* the last comparison (same as in the search). So searching pays up to $I \cdot k$ comparisons more. On the other hand, whenever a leaf overflows, we need a certain number of comparisons to select the median, say at most $c \cdot k$ for some constant c. So insertion pays up to $I \cdot ck$ comparisons more than insertion. □

In general, I can be as large as $n/2$, but it is certainly bounded by the number u of distinct keys in the input. Together this yields the formal generalization of our example.

Corollary 6.3 (Quicksort and Search Costs): *Let $\boldsymbol{U} = (U_1, \ldots, U_n)$ be an input over universe $[u]$. Let $\mathcal{T}$ be built from $\boldsymbol{U}$ by successive insertions using Algorithm 2 and denote by $\boldsymbol{\Gamma}$ its node-depth vector. The cost of Algorithm 1 on $\boldsymbol{U}$ (ignoring Insertionsort) is $\boldsymbol{\Gamma}^T \boldsymbol{X} \pm cuk$ for $\boldsymbol{X}$ the profile of $\boldsymbol{U}$.* □

For a special class of such trees, we will be able to determine the search costs: the *saturated fringe-balanced trees*. They are the subject of the next section.

7 Saturated Fringe-Balanced Trees

Consider a k-fringe-balanced tree $\mathcal{T}$ built by successively inserting elements drawn i.i.d. $\mathcal{D}(\boldsymbol{q})$ (for a fixed universe distribution $\boldsymbol{q}$) into an initially empty tree. How does this tree evolve if we continue inserting indefinitely? Since the universe $[u]$ is finite and duplicate insertions do not alter $\mathcal{T}$, the process reaches a stationary state almost surely. We call the trees corresponding to such stationary states *saturated trees* (w.r.t. the given, fixed universe). The expected search costs in saturated trees will play a key role in the analysis of Quicksort.

We start by developing a stochastic description of the shape of a random saturated tree $\mathcal{T}$ and set up a recurrence equation for the expected search costs.

7.1 Stochastic Description. Let $\boldsymbol{q} \in (0,1)^u$ with $\Sigma \boldsymbol{q} = 1$ be given. The distribution function of the universe distribution $U \stackrel{\mathcal{D}}{=} \mathcal{D}(\boldsymbol{q})$ is $F_U(v) = \mathbb{P}[U \le v] = \sum_{i=1}^{\lfloor v \rfloor} q_i$ for $v \in [0, u+1)$, and I denote its (generalized) inverse by $F_U^{-1} : (0,1) \to [1..u]$ with $F_U^{-1}(x) = \inf\{v \in [1..u] : F_U(v) \ge x\}$.

Let P be the label of the root of $\mathcal{T}$; the distribution of P is a key ingredient to (recursively) describe saturated trees. When the first leaf overflows, P is chosen as the median of the first $k = 2t+1$ inserted values, which are i.i.d. $\mathcal{D}(\boldsymbol{q})$ distributed, so it is given by

$$P \stackrel{\mathcal{D}}{=} f_U^{-1}(\Pi), \tag{3}$$

where Π has a Beta$(t+1, t+1)$ distribution, i.e., it has density $f_\Pi(z) = z^t(1-z)^t/\mathrm{B}(t+1, t+1)$. $\mathrm{B}(a,b) = \Gamma(a)\Gamma(b)/\Gamma(a+b)$ is the *beta function*. This

Figure 3: Illustration of pivot sampling in the i.i.d. model. Π is the x-coordinate of a point uniformly chosen in the gray area (the area under the curve), and P is the index of the interval this point lies in.

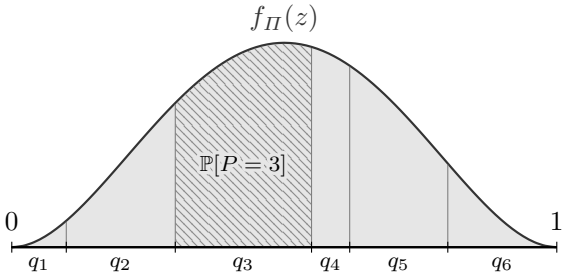
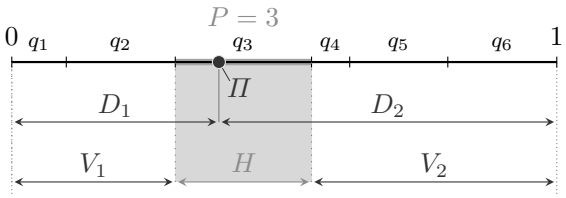


Figure 4: Relation of the different quantities in the stochastic description.



is the generalized *inversion method of random sampling*, see Devroye [5, Sec. V.3.4], illustrated in our Figure 3, which is based on the fact that $\mathrm{Beta}(t+1, t+1)$ is the distribution of the median of $2t+1$ i.i.d. uniformly in $(0,1)$ distributed random variables (see, e.g., [5, Sec. I.4.3]). For convenient notation, we write $\boldsymbol{D} = (D_1, D_2) = (\Pi, 1-\Pi)$ for the induced *spacings*; see Figure 4.

We further denote by V_1 and V_2 the probability that a random element $U \stackrel{\mathcal{D}}{=} \mathcal{D}(\boldsymbol{q})$ belongs to the left resp. right subtree of the root, and by $H = \mathbb{P}[U = P]$ the probability to "hit" the root's value. These quantities are fully determined by P (see also Figure 4):

$$V_1 = q_1 + \cdots + q_{P-1}, \tag{4.1}$$
$$V_2 = q_{P+1} + \cdots + q_u, \tag{4.2}$$
$$H = q_P. \tag{4.3}$$

(In the boundary case $P = 1$, we have $V_1 = 0$, and similarly $V_2 = 0$ for $P = u$.) Finally, we denote by $\boldsymbol{Z_1}$ and $\boldsymbol{Z_2}$ the "zoomed-in" universe distributions in the left resp. right subtree:

$$\boldsymbol{Z_1} = \left(\frac{q_1}{V_1}, \ldots, \frac{q_{P-1}}{V_1}\right), \tag{5.1}$$
$$\boldsymbol{Z_2} = \left(\frac{q_{P+1}}{V_2}, \ldots, \frac{q_u}{V_2}\right). \tag{5.2}$$

$\boldsymbol{Z_1}$ is not well-defined for $P = 1$; we set it to the *empty* vector $\boldsymbol{Z_1} = ()$ in this case. Similarly $\boldsymbol{Z_2} = ()$ for $P = u$.

7.2 Search Costs. Let $\mathcal{T}$ be a random k-fringe-balanced tree resulting from inserting i.i.d. $\mathcal{D}(\boldsymbol{q})$ elements until saturation. Each value $v \in [u]$ then appears as the key of one inner node of $\mathcal{T}$; let Γ_v denote its depth, i.e., the (random) number of nodes on the path (including endpoints) from the root to the node containing v in the (random) tree $\mathcal{T}$. The vector $\boldsymbol{\Gamma} = (\Gamma_1, \ldots, \Gamma_u)$ is called the (random) *node-depths vector* of $\mathcal{T}$ (see Figure 2 (page 7) for an example). Finally, we write $A_{\boldsymbol{q}} = \boldsymbol{\Gamma}^T \boldsymbol{q}$. This is the average depth of a node drawn according to $\mathcal{D}(\boldsymbol{q})$ in the (random) tree $\mathcal{T}$; note that we *average* the costs over the searched *key*, but consider the *tree* fixed; so $A_{\boldsymbol{q}}$ is a random variable since $\mathcal{T}$ remains random: the (weighted) average node depth in a random saturated k-fringe-balanced tree.

The expected node depth, or equivalently, the expected search cost in the tree, can be described recursively: The root contributes one comparison to any searched element, and with probability H the search stops there. Otherwise, the sought element is in the left or right subtree with probabilities V_1 resp. V_2, and the expected search costs in the subtrees are given recursively by $A_{\boldsymbol{Z_1}}$ and $A_{\boldsymbol{Z_2}}$. With the notation from above, this yields a *distributional recurrence* for $A_{\boldsymbol{q}}$:

$$A_{\boldsymbol{q}} \stackrel{\mathcal{D}}{=} 1 + V_1 A^{(1)}_{\boldsymbol{Z_1}} + V_2 A^{(2)}_{\boldsymbol{Z_2}}, \qquad (u \geq 1), \tag{6.1}$$
$$A_{()} = 0, \tag{6.2}$$

where $(A^{(1)}_{\boldsymbol{q}})$ and $(A^{(2)}_{\boldsymbol{q}})$ are independent copies of $(A_{\boldsymbol{q}})$, which are also independent of $(V_1, V_2, \boldsymbol{Z_1}, \boldsymbol{Z_2})$.

8 Quicksort With Many Duplicates

We consider the following restricted i.i.d. model.

Definition 8.1 (Many Duplicates): *Let $(\boldsymbol{q}^{(n)})_{n\in\mathbb{N}}$ be a sequence of stochastic vectors, where $\boldsymbol{q}^{(n)}$ has u_n entries, i.e., $\boldsymbol{q}^{(n)} \in (0,1)^{u_n}$ and $\Sigma\boldsymbol{q}^{(n)} = 1$, for all $n \in \mathbb{N}$. An input of size $n \in \mathbb{N}$ under the i.i.d. model for $(\boldsymbol{q}^{(n)})_{n\in\mathbb{N}}$ consists of the n i.i.d. $\mathcal{D}(\boldsymbol{q}^{(n)})$ distributed random variables.*

$(\boldsymbol{q}^{(n)})_{n\in\mathbb{N}}$ is said to have many duplicates if there is a constant $\varepsilon > 0$ so that $\mu_n = \Omega(n^{-1+\varepsilon})$ as $n \to \infty$ where $\mu_n := \min_r q^{(n)}_r$ is the smallest probability.

This condition ensures that every value occurs $\Omega(n^\varepsilon)$ times in expectation. With many duplicates, we expect few *degenerate* inputs in the following sense.

Definition 8.2 (Profile-Degenerate Inputs): *Let $\nu \in [0,1)$ and $k \in \mathbb{N}$. An input vector $\boldsymbol{U} = (U_1, \ldots, U_n) \in [u]^n$ of size n is called (ν, k)-profile-degenerate if not all u elements of the universe appear at least k times in the first $n_T = \lceil n^\nu \rceil$ elements $U_1, \ldots, U_{n_T}$ of $\boldsymbol{U}$. If the parameters are clear from the context or are not important, we call $\boldsymbol{U}$ simply profile-degenerate.*

For non-degenerate inputs, the recursion tree will depend only on the first n_T elements, and the profile of the remaining $n-n_T$ elements is *independent of this tree.* By choosing n_T large enough to have non-degenerate inputs w.h.p., but small enough to keep the contribution of the first n_T elements to the search costs negligible, we obtain the following theorem.

Theorem 8.3 (Separation Theorem): *Consider median-of-k Quicksort with fat-pivot partitioning under a discrete i.i.d. model with many duplicates. The expected number of comparisons fulfills*

$$\mathbb{E}[C_{n,\boldsymbol{q}^{(n)}}] \;=\; \mathbb{E}[A_{\boldsymbol{q}^{(n)}}]\cdot n \;\pm\; O(n^{1-\varepsilon}), \tag{7}$$

for a constant $\varepsilon > 0$ as $n \to \infty$; more precisely, we need $\varepsilon \in (0, \tilde{\varepsilon})$ when $\mu_n = \min_r q_r^{(n)} = \Omega(n^{-1+\tilde{\varepsilon}})$.

Proof: By Corollary 6.3 we can study search costs in fringe-balanced trees; the main challenge is that this tree is built form the same input that is used for searching. In other words, $\boldsymbol{\Gamma}$ and $\boldsymbol{X}$ are not independent; for non-degenerate inputs, they are however almost so.

We start noting the following basic fact that we will use many times: In any discrete i.i.d. model, $u_n \le \frac{1}{\mu_n}$ for all n. In particular, an i.i.d. model with many duplicates has $u_n = O(n^{1-\varepsilon})$.

Probability of degenerate profiles. Since no element of the universe is very unlikely, we can bound the probability of degenerate inputs.

Lemma 8.4 (Non-Degenerate w.h.p.): *Assume an i.i.d. model with $\mu_n = \min_v q_v^{(n)} = \Omega(n^{-\rho})$ for $\rho \in [0,1)$ and let $k \in \mathbb{N}$ and $\rho < \nu < 1$. Then the probability of an input of size n to be (ν,k)-profile-degenerate is in $o(n^{-c})$ as $n \to \infty$ for any constant c.*

The lemma follows from a standard application of the union bound and tail inequalities; in this case Lemma 2.2 yields a stronger results than classical Chernoff bounds. The detailed proof is in the extended version.

In Theorem 8.3, we assume an i.i.d. model with many duplicates, i.e., $\mu_n = \Omega(n^{-1+\tilde{\varepsilon}})$ with $\tilde{\varepsilon} \in (0,1]$, and an $\varepsilon \in (0,\tilde{\varepsilon})$ is given. We set

$$\nu \;:=\; \frac{(1-\tilde{\varepsilon})+(1-\varepsilon)}{2} \;=\; 1-\frac{\tilde{\varepsilon}+\varepsilon}{2} \;\in\; (1-\tilde{\varepsilon}, 1-\varepsilon).$$

Then, by Lemma 8.4, an input of size n is (ν,k)-degenerate with probability in $o(n^{-c})$ for all c. This also means that the overall cost contribution of degenerate inputs to expected costs is in $o(n^{-c})$ for all c, and hence covered by the error term in Equation (7), since costs for any input are at most quadratic in n.

We will thus, for the remainder of this proof, assume that the input is not (ν,k)-degenerate, i.e., each of the values of the universe appears at least k times among the first $n_T = \lceil n^\nu \rceil$ elements.

Independence of Profiles and Trees. The shape of the recursion tree is determined by at most $u\cdot k$ elements: we have at most u partitioning rounds since each of the u elements of the universe becomes a pivot in at most one partitioning step, and each partitioning step inspects k elements for choosing its pivot.

Also, for each of the u values in the universe, at most the first k occurrences in the input, reading from left to right, can influence the tree: if a value $v \in [u]$ is already contained in an inner node, all further duplicates of v are ignored. Otherwise, all occurrences of v must appear in a single leaf, which can hold up to $k-1$ values, so there are never more than $k-1$ copies of v in the tree. The leaf will overflow at the latest upon inserting the kth occurrence of v, and then a new internal node with pivot v is created.

In a non-degenerate input, the first k duplicates appear among the first n_T elements $U_1,\dots,U_{n_T}$ of the input, so all pivots are chosen based on these elements only. Moreover, after these n_T insertions, all u values appear as labels of inner nodes.

We denote by $\boldsymbol{\Gamma} = \boldsymbol{\Gamma}(\boldsymbol{q}^{(n)})$ the node-depths vector for the final recursion tree and by $\tilde{\boldsymbol{X}}$ the profile of $U_{n_T+1},\dots,U_n$. Since they are derived from disjoint ranges of the i.i.d. input, $\boldsymbol{\Gamma}$ and $\tilde{\boldsymbol{X}}$ are stochastically independent.

Overall Result. Recall that $u_n = O(n^{1-\tilde{\varepsilon}})$ and $n_T \sim n^\nu$ with $1-\tilde{\varepsilon} < \nu < 1-\varepsilon$. By Proposition 2.3, our trees have height $> 13\ln n$ with probability in $O(n^{-2})$, i.e., for most inputs, $\Gamma_v = O(\log n)$ for all v.

We therefore further split the set of non-profile-degenerate inputs into *"height-degenerate"* ones where the height is $> 13\ln n$ and all other ones. This gives the following stochastic representation conditional on the input being not (ν,k)-profile-degenerate.

$$\begin{aligned}
C_{n,\boldsymbol{q}^{(n)}} &= \boldsymbol{\Gamma}^T\boldsymbol{X} \;\pm\; O(u)\\
&= \boldsymbol{\Gamma}^T\tilde{\boldsymbol{X}} \;\pm\; n_T\|\boldsymbol{\Gamma}\|_\infty \;\pm\; O(u)\\
&\overset{\mathcal{D}}{=} \boldsymbol{\Gamma}^T\hat{\boldsymbol{X}} \;\pm\; 2n_T\|\boldsymbol{\Gamma}\|_\infty \;\pm\; O(u)\\
&= \boldsymbol{\Gamma}^T\hat{\boldsymbol{X}} \;\pm\; O\Big(n_T\big(\mathbb{1}_{\{\text{not height-deg.}\}}\cdot\log n\\
&\qquad + \mathbb{1}_{\{\text{height-deg.}\}}\cdot u\big)\Big)\\
&= \boldsymbol{\Gamma}^T\hat{\boldsymbol{X}} \;\pm\; O\Big(n^\nu\big(\mathbb{1}_{\{\text{not height-deg.}\}}\cdot\log n\\
&\qquad + \mathbb{1}_{\{\text{height-deg.}\}}\cdot n^{1-\tilde{\varepsilon}}\big)\Big),
\end{aligned}$$

where $\hat{\boldsymbol{X}}$ is independent of $\boldsymbol{\Gamma}$ and $\hat{\boldsymbol{X}} \overset{\mathcal{D}}{=} \mathrm{Mult}(n,\boldsymbol{q}^{(n)})$.

We thus find that

$$C_{n,\boldsymbol{q}^{(n)}} \stackrel{\mathcal{D}}{=} \boldsymbol{\Gamma}^T \hat{\boldsymbol{X}} \pm O(n^{1-\varepsilon}), \quad \text{(input neither profile- nor height-degenerate).} \tag{8}$$

Taking expectations over all non-degenerate inputs in Equation (8), exploiting independence, and inserting $\mathbb{P}[\text{height-deg.}] = O(1/n^2)$ (Proposition 2.3) yields

$$\begin{aligned} \mathbb{E}[C_{n,\boldsymbol{q}^{(n)}}] &= \mathbb{E}\big[\boldsymbol{\Gamma}^T \hat{\boldsymbol{X}}\big] \pm O\big(n^{1-\varepsilon}\big) \\ &= \mathbb{E}[\boldsymbol{\Gamma}]^T \cdot \mathbb{E}[\hat{\boldsymbol{X}}] \pm O\big(n^{1-\varepsilon}\big) \\ &= \underbrace{\big(\mathbb{E}[\boldsymbol{\Gamma}]^T \cdot \boldsymbol{q}^{(n)}\big)}_{\mathbb{E}[A_{\boldsymbol{q}^{(n)}}]} \cdot n \pm O(n^{1-\varepsilon}), \end{aligned} \tag{9}$$

with $A_{\boldsymbol{q}^{(n)}}$ as given in Equation (6) on page 9. As argued above, the contribution of profile-degenerate inputs is in $o(n^{-c})$ for any c and thus covered by $O(n^{1-\varepsilon})$, so Equation (9) holds also for the unconditional expectation. This concludes the proof of Theorem 8.3. □

9 Expected Search Costs in Saturated Trees

The remaining step of the analysis consists in computing $\mathbb{E}[A_{\boldsymbol{q}}]$, the expected search costs in saturated k-fringe-balanced trees built from i.i.d. $\mathcal{D}(\boldsymbol{q})$ elements.

Since the recurrence equation for the expected search costs (Equation (6) on page 9) seems hard to solve, we try to relate it to a quantity that we already know: the *entropy of the universe distribution.* The entropy function satisfies an *aggregation property:* intuitively speaking, we do not change the entropy of the final outcomes if we *delay* some decisions in a random experiment by first deciding among whole groups of outcomes and then continuing within the chosen group.

The aggregation property can nicely be formalized in terms of trees, see Lemma 6.2.2E of Knuth [15, p. 444], and we can use it in a search tree to express the entropy of node access probabilities as the entropy of the split corresponding to the root plus the entropies of its subtrees, weighted by their respective total probabilities. More specifically in our setting, we have $\boldsymbol{q} \in [0,1]^u$ with $\Sigma\boldsymbol{q} = 1$ and we condition on the value $P \in [u]$ in the root. Using the notation from Section 7.1 we find that

$$\mathcal{H}(\boldsymbol{q}) = \mathcal{H}(V_1, H, V_2) + \sum_{r=1}^{2} V_r \mathcal{H}(\boldsymbol{Z_r}). \tag{10}$$

(By linearity, $\mathcal{H}$ can be w.r.t. any base; we will use it with $\mathcal{H}_{\ln}$ below.)

If we likewise condition on the root value P in the recursive description of the search costs, we find for the expected search costs that

$$\mathbb{E}[A_{\boldsymbol{q}}] = 1 + \sum_{r=1}^{2} V_r \mathbb{E}[A_{\boldsymbol{Z_r}}]. \tag{11}$$

$\mathcal{H}(\boldsymbol{q})$ and $\mathbb{E}[A_{\boldsymbol{q}}]$ fulfill the *same form* of recurrence, only with a different toll function: 1 instead of $\mathcal{H}(V_1, H, V_2)$! We thus try to relate these two toll functions by obtaining bounds on $\mathcal{H}(V_1, H, V_2)$, and then extend this relation inductively to $\mathcal{H}(\boldsymbol{q})$ and $\mathbb{E}[A_{\boldsymbol{q}}]$. The technical difficulties in doing so are that $\mathcal{H}(V_1, H, V_2)$ is very sensitive to $\boldsymbol{q}$, so we have to do a case distinction to obtain bounds on it. We hence cannot give matching upper and lower bounds for $\mathcal{H}(V_1, H, V_2)$, which necessitates the introduction of second order terms to account for the slack (cf. the constant d below). Separately deriving upper and lower bounds on $\mathbb{E}[A_{\boldsymbol{q}}]$ from that in terms of $\mathcal{H}(\boldsymbol{q})$, and making them match asymptotically in the leading term, we obtain the following result.

Theorem 9.1 (Expected Search Costs): *Let a sequence of universe distributions $(\boldsymbol{q}^{(n)})_{n\in\mathbb{N}}$ be given for which $\mathcal{H}_n := \mathcal{H}_{\mathrm{ld}}(\boldsymbol{q}^{(n)}) \to \infty$ as $n \to \infty$. The expected search costs of a saturated k-fringe-balanced tree (with $k = 2t+1$) built from i.i.d. $\mathcal{D}(\boldsymbol{q}^{(n)})$ keys is given by*

$$\mathbb{E}[A_{\boldsymbol{q}^{(n)}}] = \alpha_k \mathcal{H}_{\mathrm{ld}}(\boldsymbol{q}^{(n)}) \pm O\Big(\mathcal{H}_n^{\frac{t+2}{t+3}} \log\big(\mathcal{H}_n\big)\Big), \quad (n \to \infty).$$

Proof: We start with the upper bound. We actually derive a whole class of upper bounds characterized by a parameter ε.

Lemma 9.2 (Upper Bound): *Let $\mathbb{E}[A_{\boldsymbol{q}}]$ satisfy Equation (11), and let $\varepsilon \in (0,1)$ be given. Define*

$$\begin{aligned} c = c_\varepsilon &= \frac{1}{\tilde{H} - 4\varepsilon\tilde{h}}, \\ d = d_\varepsilon &= \frac{(t+1)\,\mathrm{B}(t+1, t+1)}{\varepsilon^{t+2}(1-\varepsilon)^t}, \\ \text{where } \tilde{H} &= H_{k+1} - H_{t+1} \\ \text{and } \tilde{h} &= H_k - H_t\,. \end{aligned}$$

If $c \geq 0$, we have that $\mathbb{E}[A_{\boldsymbol{q}}] \leq c \cdot \mathcal{H}_{\ln}(\boldsymbol{q}) + d$ for all stochastic vectors $\boldsymbol{q}$.

Proof: Let ε with $c = c_\varepsilon \geq 0$ be given. Note that $d = d_\varepsilon \geq 0$ holds for all ε. The proof is by induction on u, the size of the universe. If $u = 0$, i.e., $\boldsymbol{q} = ()$, we have $\mathbb{E}[A_{\boldsymbol{q}}] = 0$, see Equation (6.2). Since $d \geq 0$ and here $\mathcal{H}_{\ln}(\boldsymbol{q}) = 0$, the claim holds.

Now assume that $u \geq 1$ and the claim holds for all (strictly) smaller universe sizes. We start by taking expectations in Equation (6) and conditioning on the pivot value P:

$$\begin{aligned} \mathbb{E}[A_{\boldsymbol{q}}] &= 1 + \mathbb{E}_P\left[\sum_{r=1}^{2} \mathbb{E}[V_r A_{\boldsymbol{Z_r}} \mid P]\right] \\ &= 1 + \mathbb{E}_P\left[\sum_{r=1}^{2} V_r\, \mathbb{E}[A_{\boldsymbol{Z_r}} \mid P]\right] \end{aligned}$$

using the inductive hypothesis

$$\begin{aligned} &\leq 1 + \mathbb{E}_P\left[\sum_{r=1}^{2} V_r\,(c\mathcal{H}_{\ln}(\boldsymbol{Z_r}) + d)\right] \\ &= 1 + c\cdot\mathbb{E}\left[\sum_{r=1}^{2} V_r\,\mathcal{H}_{\ln}(\boldsymbol{Z_r})\right] + d\cdot\mathbb{E}[\Sigma\boldsymbol{V}] \\ &\underset{(10)}{=} c\cdot\mathcal{H}_{\ln}(\boldsymbol{q}) \\ &\quad + \underbrace{1 - c\cdot\mathbb{E}\big[\mathcal{H}_{\ln}(\boldsymbol{V},H)\big] + d\cdot\mathbb{E}[\Sigma\boldsymbol{V}]}_{\varpi} \end{aligned} \tag{12}$$

It remains to show that $\varpi \leq d$. We consider two cases depending on the maximal probability $\lambda = \max_{1\leq v\leq u} q_v$.

1. Case $\lambda < \varepsilon$:
   In this case, all individual probabilities are smaller than ε, so it is plausible that we can bound the expected entropy of partitioning $\mathbb{E}[\mathcal{H}_{\ln}(\boldsymbol{V},H)]$ from below. The subuniverse probabilities V_r are quite close to the continuous spacings D_r: by definition (see also Figure 4 on page 9) we have in interval-arithmetic notation

   $$V_r = D_r + (-2\varepsilon, 0), \qquad (r = 1, 2). \tag{13}$$

   For the expected partitioning entropy, this means

   $$\begin{aligned} \mathbb{E}[\mathcal{H}_{\ln}(\boldsymbol{V},H)] &\geq \sum_{r=1}^{2}\mathbb{E}\big[V_r\ln(1/V_r)\big] \end{aligned}$$

   using Equation (13) and $x\log(1/x) \geq (x-\varepsilon)\log(1/(x+\varepsilon'))$ for $\varepsilon,\varepsilon' \geq 0$ and $x \in [0,1]$, this is

   $$\begin{aligned} &\geq \sum_{r=1}^{2}\mathbb{E}\big[(D_r - 2\varepsilon)\ln(1/D_r)\big] \\ &= \mathbb{E}\big[\mathcal{H}_{\ln}(\boldsymbol{D})\big] + 2\varepsilon\sum_{r=1}^{2}\mathbb{E}\big[\ln(D_r)\big] \\ &\underset{\text{Lemma 2.5, (2)}}{=} \tilde{H} + 2\varepsilon\sum_{r=1}^{2}\big(\psi(t+1) - \psi(k+1)\big) \\ &= \tilde{H} - 4\varepsilon(H_k - H_t) \\ &= \tilde{H} - 4\varepsilon\tilde{h} \\ &= 1/c\,. \end{aligned}$$

   Hence c satisfies $c \geq 1/\mathbb{E}[\mathcal{H}_{\ln}(\boldsymbol{V},H)] \geq 0$, which implies

   $$\begin{aligned} \varpi &\leq 1 - \frac{1}{\mathbb{E}[\mathcal{H}_{\ln}(\boldsymbol{V},H)]}\cdot\mathbb{E}\big[\mathcal{H}_{\ln}(\boldsymbol{V},H)\big] \\ &\quad + d\cdot\underbrace{\mathbb{E}[\Sigma\boldsymbol{V}]}_{\leq 1} \\ &\leq d. \end{aligned}$$

   The inductive step is proven in this case.

2. Case $\lambda \geq \varepsilon$:
   In the second case, there is a *likely* value $v \in [u]$ with $q_v \geq \varepsilon$. We will show a lower bound for having this value as label of the root.

   We have $\Pi \stackrel{\mathcal{D}}{=} \mathrm{Beta}(t+1, t+1)$ and hence $f_\Pi(z) = \frac{z^t(1-z)^t}{\mathrm{B}(t+1,t+1)}$. Recall that $P = f_U^{-1}(\Pi)$ (Figure 3), so we can bound the probability to draw v from below by the smallest value of the integral over any ε-wide strip of the density:

   $$\begin{aligned} \mathbb{P}[P=v] &\geq \min_{0\leq\zeta\leq 1-\varepsilon}\int_{z=\zeta}^{\zeta+\varepsilon} f_{\Pi_r}(z)\,dz \\ &= \min_{0\leq\zeta\leq 1-\varepsilon}\int_{z=\zeta}^{\zeta+\varepsilon}\frac{z^t(1-z)^t}{\mathrm{B}(t+1,t+1)}\,dz \\ &= \int_{z=0}^{\varepsilon}\frac{z^t(1-z)^t}{\mathrm{B}(t+1,t+1)}\,dz \\ &\geq \int_{0}^{\varepsilon}\frac{z^t(1-\varepsilon)^t}{\mathrm{B}(t+1,t+1)}\,dz \\ &= \frac{\varepsilon^{t+1}(1-\varepsilon)^t}{(t+1)\,\mathrm{B}(t+1,t+1)}\,. \end{aligned} \tag{14}$$

   For the expected hitting probability, we thus have for any $\boldsymbol{q}$ with a $q_v \geq \varepsilon$ that

   $$\begin{aligned} \mathbb{E}[H] &\geq q_v\cdot\mathbb{P}[P=v] \\ &\underset{(14)}{\geq} \frac{\varepsilon^{t+2}(1-\varepsilon)^t}{(t+1)\,\mathrm{B}(t+1,t+1)} \\ &= 1/d, \end{aligned} \tag{15}$$

   so $d \geq 1/\mathbb{E}[H]$. This implies

   $$\begin{aligned} \varpi - d &\leq 1 - c\cdot\mathbb{E}\big[\mathcal{H}_{\ln}(\boldsymbol{V},H)\big] + d\cdot\mathbb{E}[\Sigma\boldsymbol{V}] - d \\ &\leq 1 - (1 - \mathbb{E}[\Sigma\boldsymbol{V}])\cdot\frac{1}{\mathbb{E}[H]} \\ &= 0. \end{aligned}$$

   This concludes the inductive step in case 2.

The inductive step is thus done in both cases, and the claim holds for all stochastic vectors $\boldsymbol{q}$ by induction. □

The lower bound on $\mathbb{E}[A_{\boldsymbol{q}}]$ uses basically the same techniques; only a few details differ.

Lemma 9.3 (Lower Bound): *Let $\mathbb{E}[A_{\boldsymbol{q}}]$ satisfy Equation (11), and let $\varepsilon \in (0, 1/e)$ be given. Define*

$$\begin{aligned} c = c_\varepsilon &= \frac{1}{\tilde{H} + 4\varepsilon + \varepsilon\ln(1/\varepsilon)}\,, \\ d = d_\varepsilon &= \big(c_\varepsilon\ln(3) - 1\big)\frac{(t+1)\,\mathrm{B}(t+1,t+1)}{\varepsilon^{t+2}(1-\varepsilon)^t}\,, \\ \text{where}\quad \tilde{H} &= H_{k+1} - H_{t+1}\,. \end{aligned}$$

If $d \geq 0$, we have that $\mathbb{E}[A_{\boldsymbol{q}}] \geq c\cdot\mathcal{H}_{\ln}(\boldsymbol{q}) - d$ all stochastic vectors $\boldsymbol{q}$.

The proof is similar to the one for the upper bound and omitted; it appears in full in the extended version of this paper. We observe that c_ε converges to $1/\tilde{H}$ for both bounds as $\varepsilon \to 0$, so there is hope to show $\mathbb{E}[A_{\boldsymbol{q}}] \sim \mathcal{H}_{\ln}(\boldsymbol{q})/\tilde{H}$. We have to be precise about the limiting process and error terms, though.

Let $(\boldsymbol{q}^{(i)})_{i\in\mathbb{N}}$ be a sequence of universe distributions for which $\mathcal{H}_i := \mathcal{H}_{\ln}(\boldsymbol{q}^{(i)}) \to \infty$ as $i \to \infty$. Now, consider c_ε and d_ε from Lemma 9.2. As functions in ε, they satisfy for $\varepsilon \to 0$

$$c_\varepsilon \;=\; \frac{1}{\tilde{H}} \pm O(\varepsilon)\,, \qquad d_\varepsilon \;=\; O(\varepsilon^{-t-2}) \tag{16}$$

Since the bounds above hold simultaneously for all feasible values of ε, we can let ε depend on $\mathcal{H}_i$. If we set

$$\varepsilon \;=\; \varepsilon_i \;=\; {\mathcal{H}_i}^{-\frac{1}{t+3}} \tag{17}$$

we have $\varepsilon_i \to 0$ as $i \to \infty$ and so $c_{\varepsilon_i} > 0$ for large enough i. Then we have by Lemma 9.2

$$\mathbb{E}[A_{\boldsymbol{q}^{(i)}}] \;\le\; c_{\varepsilon_i}\mathcal{H}_i + d_{\varepsilon_i} \underset{(16)}{=} \frac{\mathcal{H}_i}{\tilde{H}} \pm O\Big({\mathcal{H}_i}^{\frac{t+2}{t+3}}\Big) \tag{18}$$

as $i \to \infty$.

Now consider the lower bound. For c_ε and d_ε from Lemma 9.3 we similarly find as $\varepsilon \to 0$ that

$$c_\varepsilon \;=\; \frac{1}{\tilde{H}} \pm O(\varepsilon \log \varepsilon)\,, \quad d_\varepsilon \;=\; O(\varepsilon^{-t-2}). \tag{19}$$

With the same ε_i as above (Equation (17)) we have $d_{\varepsilon_i} \ge 0$ for large enough i, so by Lemma 9.3 it holds that

$$\mathbb{E}[A_{\boldsymbol{q}^{(i)}}] \;\ge\; \frac{\mathcal{H}_i}{\tilde{H}} \pm O\Big({\mathcal{H}_i}^{\frac{t+2}{t+3}} \log \mathcal{H}_i\Big), \qquad (i \to \infty). \tag{20}$$

Together with Equation (18), this concludes the proof of Theorem 9.1. □

10 Lower Bound For I.I.D. Sorting

We follow the elegant argument of Munro and Raman [20] for multiset sorting to obtain a lower bound for the discrete i.i.d. model. Since profiles are concentrated around the mean and the entropy function is smooth, we obtain essentially the result of their Theorem 4, but with a weaker error term.

Theorem 10.1 (Lower Bound): *Let $u = O(n^\nu)$ for a constant $\nu \in [0,1)$ and $\boldsymbol{q} \in (0,1)^u$ with $\Sigma\boldsymbol{q} = 1$. For any constant $\varepsilon > 0$, $\mathcal{H}_{\mathrm{ld}}(\boldsymbol{q})n - n/\ln(2) \pm o\big(n^{(1+\nu)/2+\varepsilon}\big)$ ternary comparisons are necessary in expectation as $n \to \infty$ to sort n i.i.d. $\mathcal{D}(\boldsymbol{q})$ elements by any comparison-based algorithm.*

The proof is omitted; it is given in the extended version of this paper.

11 Proof of Main Result

With all these preparations done, the proof of our main result reduces to properly combining the ingredients developed above.

Proof of Theorem 5.1: Let the sequence $(\boldsymbol{q}^{(n)})_{n\in\mathbb{N}}$ be given. By assumption the model has many duplicates, i.e., $\mu_n = \Omega(n^{-1+\varepsilon})$. We thus obtain from Theorem 8.3 that

$$\mathbb{E}[C_{n,\boldsymbol{q}^{(n)}}] \;=\; \mathbb{E}[A_{\boldsymbol{q}^{(n)}}] \cdot n \pm O(n^{1-\delta'}), \qquad (n \to \infty),$$

for any $\delta' \in (0,\varepsilon)$. If $\mathcal{H}_n = \mathcal{H}(\boldsymbol{q}^{(n)}) \to \infty$ as $n \to \infty$, we can continue with Theorem 9.1 right away. To cover the case that $(\mathcal{H}_n)_{n\in\mathbb{N}}$ contains an infinite subsequence that is bounded, we add an error bound of $O(n)$; this dominates $\mathbb{E}[A_{\boldsymbol{q}^{(n)}}] \cdot n$ (making the claim is essentially vacuous) for such inputs. So in any case we have that

$$\begin{aligned}\mathbb{E}[C_{n,\boldsymbol{q}^{(n)}}] \;&=\; \alpha_k\mathcal{H}_n \cdot n \pm O(\mathcal{H}_n^{1-\delta}n + n + n^{1-\delta'}) \\ &=\; \alpha_k\mathcal{H}_n \cdot n \pm O(\mathcal{H}_n^{1-\delta}n + n)\end{aligned}$$

as $n \to \infty$ for any $\delta \in (0, \frac{1}{t+3}) = (0, \frac{2}{k+5})$. The optimality claim follows directly by comparing with the lower bound in Theorem 10.1. □

12 Conclusion

Computer scientists are so accustomed to the random-permutation model that the possibility of duplicate keys in sorting is easily overlooked. The following formulation (which is equivalent to random permutations) makes the presence of an underlying restriction more obvious: we sort n numbers drawn independently from the same *continuous* probability distribution. The assumption of i.i.d. samples is as natural as declaring all orderings equally likely, and certainly adequate when no specific knowledge is available; even more so for Quicksort where randomly shuffling the input prior to sorting is best practice. The use of a *continuous* distribution, though, is a restriction worth questioning; what happens if we use a discrete distribution instead?

The most striking difference certainly is that with a discrete distribution, we expect to see equal elements appear in the input. There are more differences, though. If the distribution is (absolutely) continuous (i.e., attains values in a real interval and has a continuous density), almost surely all elements are different and the ranks form a random permutation [16], no matter how the continuous distribution itself looks like. By contrast, different discrete universe distributions each yield a *different* input model; in particular the universe size u can have a great influence.

In this paper, I presented the first analysis of median-of-k Quicksort under the model of independently and

identically distributed numbers from a *discrete* universe distribution. I showed that it uses asymptotically a factor $\alpha_k = \ln(2)/\big(H_{k+1} - H_{(k+1)/2}\big)$ more comparisons than necessary in expectation. Analytical complications necessitated the restriction to the case where every element is expected to appear $\Omega(n^\varepsilon)$ times for some $\varepsilon > 0$, but I conjecture that the result generalizes beyond the limitations of the current techniques (see the comments below).

The very same statement—asymptotically a factor α_k over the lower bound—holds in the i.i.d. model with a continuous distribution, so, apart from the above restriction, we can say that median-of-k Quicksort is asymptotically optimal to within the constant factor α_k for *any* randomly ordered input. It is reassuring (and somewhat remarkable given the big differences in the derivations that lead to it) that we obtain the same constant in both cases: the relative benefit of pivot sampling is independent of the presence or absence of equal keys. Table 1 shows the first values of α_k; most savings happen for small k.

Table 1: A few values of α_k and the relative improvement over the case without sampling.

k	α_k	saving over $k = 1$
1	1.38629	—
3	1.18825	14.3 %
5	1.12402	18.9 %
7	1.09239	21.2 %
9	1.07359	22.6 %
∞	1	27.9 %

I surveyed previous results on multiset sorting, which uses a different input model. We cannot directly relate the result, but we have seen (in Section 3.3) that the i.i.d. model yields at least an upper bound. (A more fine-grained discussion of the relation of the two models is deferred to a full version of this article.)

Since $H_{k+1} - H_{(k+1)/2} \sim \ln(k+1) - \ln((k+1)/2) = \ln(2)$ as $k \to \infty$, we have $\alpha_k \to 1$, so we have confirmed the conjecture of Sedgewick and Bentley for inputs with many duplicates: by increasing k, we obtain sorting methods that are arbitrarily close to the asymptotically optimal number of comparisons for randomly ordered inputs, and in particular for random permutations of a multiset.

Extensions and Future Directions. The methods used in this paper can readily be generalized to analyze skewed sampling, i.e., when we pick an order statistic other than the median of the sample. It is known that the number of comparisons is minimal for the median [17], so this might not be very interesting in its own right, but it can be used to analyze *ninther* sampling and similar schemes [26, Sec. 6.4].

Another line of future research is to extend the present analysis to multiway Quicksort, e.g., the Yaroslavskiy-Bentley-Bloch dual-pivot Quicksort used in Java. The uniform case is known [26], but the techniques of the present paper do not carry over: the partitioning costs for such methods also depend on $\boldsymbol{q}$, which means that we do not get matching lower and upper bounds for $\mathbb{E}[A_{\boldsymbol{q}}]$ using the method of Section 9 any more.

Acknowledgments

I am very grateful for inspiring discussions with Conrado Martínez, Markus Nebel, Martin Aumüller and Martin Dietzfelbinger, which laid the ground for the present paper. Moreover I thank Sándor Fekete for many helpful comments that improved the presentation and framing of the results. Finally, I thank my anonymous referees for their scrutiny and thorough reviews; their comments helped to clarify the presentation significantly.

References

[1] B. Allen and I. Munro. Self-organizing binary search trees. *Journal of the ACM*, 25(4):526–535, October 1978. doi: 10.1145/322092.322094.

[2] M. Archibald and J. Clément. Average depth in a binary search tree with repeated keys. In *Colloquium on Mathematics and Computer Science*, volume 0, pages 309–320, 2006.

[3] J. L. Bentley and M. D. McIlroy. Engineering a sort function. *Software: Practice and Experience*, 23(11): 1249–1265, 1993.

[4] W. H. Burge. An analysis of binary search trees formed from sequences of nondistinct keys. *Journal of the ACM*, 23(3):451–454, July 1976. doi: 10.1145/321958.321965.

[5] L. Devroye. *Non-Uniform Random Variate Generation*. Springer New York, 1986. (available on author's website `http://luc.devroye.org/rnbookindex.html`).

[6] M. Drmota. *Random Trees*. Springer, 2009. ISBN 978-3-211-75355-2.

[7] P. Flajolet and R. Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009. ISBN 978-0-52-189806-5. (available on author's website: `http://algo.inria.fr/flajolet/Publications/book.pdf`).

[8] I. Gradshteyn and I. Ryzhik. *Table of Integrals, Series, and Products*. Academic Press, 7th edition, 2007. ISBN 978-0-12-373637-6.

[9] T. N. Hibbard. Some combinatorial properties of certain trees with applications to searching and sorting. *Journal of the ACM*, 9(1):13–28, January 1962. doi: 10.1145/321105.321108.

[10] C. A. R. Hoare. Algorithm 64: Quicksort. *Communications of the ACM*, 4(7):321, July 1961.

[11] C. A. R. Hoare. Quicksort. *The Computer Journal*, 5 (1):10–16, January 1962.

[12] Java Core Library Development Mailing List. Replacement of quicksort in java.util.arrays with new dual-pivot quicksort, 2009. URL `https://www.mail-archive.com/core-libs-dev@openjdk.java.net/msg02608.html`.

[13] J. Katajainen and T. Pasanen. Stable minimum space partitioning in linear time. *BIT*, 32(4):580–585, December 1992. ISSN 0006-3835. doi: 10.1007/BF01994842.

[14] R. Kemp. Binary search trees constructed from nondistinct keys with/without specified probabilities. *Theoretical Computer Science*, 156(1-2):39–70, March 1996. doi: 10.1016/0304-3975(95)00302-9.

[15] D. E. Knuth. *The Art Of Computer Programming: Searching and Sorting*. Addison Wesley, 2nd edition, 1998. ISBN 978-0-20-189685-5.

[16] H. M. Mahmoud. *Sorting: A distribution theory*. John Wiley & Sons, 2000. ISBN 1-118-03288-8.

[17] C. Martínez and S. Roura. Optimal sampling strategies in Quicksort and Quickselect. *SIAM Journal on Computing*, 31(3):683–705, 2001. doi: 10.1137/S0097539700382108.

[18] C. J. H. McDiarmid. Concentration. In M. Habib, C. McDiarmid, J. Ramirez-Alfonsin, and B. Reed, editors, *Probabilistic Methods for Algorithmic Discrete Mathematics*, pages 195–248. Springer, Berlin, 1998.

[19] I. Munro and P. M. Spira. Sorting and searching in multisets. *SIAM Journal on Computing*, 5(1):1–8, March 1976. doi: 10.1137/0205001.

[20] J. I. Munro and V. Raman. Sorting multisets and vectors in-place. In *Workshop on Algorithms and Data Structures (WADS)*, volume 519 of *LNCS*, pages 473–480, Berlin/Heidelberg, 1991. Springer. doi: 10.1007/BFb0028285.

[21] R. Sedgewick. The analysis of Quicksort programs. *Acta Informatica*, 7(4):327–355, 1977.

[22] R. Sedgewick. Quicksort with equal keys. *SIAM Journal on Computing*, 6(2):240–267, 1977.

[23] R. Sedgewick and J. Bentley. New research on theory and practice of sorting and searching (talk slides), 1999. URL `http://www.cs.princeton.edu/~rs/talks/Montreal.pdf`.

[24] R. Sedgewick and J. Bentley. Quicksort is optimal (talk slides), 2002. URL `http://www.cs.princeton.edu/~rs/talks/QuicksortIsOptimal.pdf`.

[25] R. Sedgewick and K. Wayne. *Algorithms*. Addison-Wesley, 4th edition, 2011. ISBN 978-0-32-157351-3.

[26] S. 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`. ISBN 978-3-00-054669-3.

[27] S. Wild and M. E. Nebel. Average case analysis of Java 7's dual pivot Quicksort. In L. Epstein and P. Ferragina, editors, *European Symposium on Algorithms (ESA)*, volume 7501 of *LNCS*, pages 825–836. Springer, 2012. URL `http://arxiv.org/abs/1310.7409`.