# Average Case and Distributional Analysis of Dual-Pivot Quicksort

Sebastian Wild* Markus E. Nebel*,† Ralph Neininger‡

September 25, 2014

In 2009, Oracle replaced the long-serving sorting algorithm in its Java 7 runtime library by a new dual-pivot Quicksort variant due to Vladimir Yaroslavskiy. The decision was based on the strikingly good performance of Yaroslavskiy's implementation in running time experiments. At that time, no precise investigations of the algorithm were available to explain its superior performance — on the contrary: Previous theoretical studies of other dual-pivot Quicksort variants even discouraged the use of two pivots. Only in 2012, two of the authors gave an average case analysis of a simplified version of Yaroslavskiy's algorithm, proving that savings in the number of comparisons are possible. However, Yaroslavskiy's algorithm needs more swaps, which renders the analysis inconclusive.

To force the issue, we herein extend our analysis to the fully detailed style of Knuth: We determine the exact number of executed Java Bytecode instructions. Surprisingly, Yaroslavskiy's algorithm needs sightly *more* Bytecode instructions than a simple implementation of classic Quicksort — contradicting observed running times. Like in Oracle's library implementation we incorporate the use of Insertionsort on small subproblems and show that it indeed speeds up Yaroslavskiy's Quicksort in terms of Bytecodes; but even with optimal Insertionsort thresholds the new Quicksort variant needs slightly more Bytecode instructions on average.

Finally, we show that the (suitably normalized) costs of Yaroslavskiy's algorithm converge to a random variable whose distribution is characterized by a fixed-point equation. From that, we compute variances of costs and show that for large $n$, costs are concentrated around their mean.

## 1. Introduction

Quicksort is a divide and conquer sorting algorithm originally proposed by Hoare [1961a; 1961b]. The procedure starts by selecting an arbitrary element from the list to be sorted as *pivot*. Then, Quicksort *partitions* the elements into two groups: those smaller than the pivot and those larger than the pivot. After partitioning, we know the exact *rank* of the pivot element in the sorted list, so we can put it at its final landing position between the groups of smaller and larger elements. Afterwards, Quicksort proceeds by recursively sorting the two parts, until it reaches lists of length zero or one, which are already sorted by definition.

We will in the following always assume random access to the data, i. e., the elements are given as entries of an *array*. Then, the partitioning process can work *in place* by directly manipulating

*Computer Science Department, University of Kaiserslautern
†Department of Mathematics and Computer Science, University of Southern Denmark
‡Institute for Mathematics, J. W. Goethe University

the array. This makes Quicksort convenient to use and avoids the need for extra space (except for the recursion stack). Hoare's initial implementation [Hoare, 1961b] works in place and Sedgewick [1975] studies several variants thereof.

In the worst case, Quicksort has quadratic complexity, namely if in every partitioning step, the pivot is the smallest or largest element of the current subarray. However, this behavior occurs very infrequently, such that the expected complexity is $\Theta(n \log n)$. Hoare [1962] already gives a precise average case analysis of his algorithm, which is nowadays contained in most algorithms textbooks, [e. g. Cormen et al. 2009]. Sedgewick [1975; 1977] refines this analysis to count the exact number of executed primitive instructions of a low level implementation. This detailed breakdown reveals that Quicksort has the asymptotically fastest average running time on MIX among all the sorting algorithm studied by Knuth [1998].

Only considering average results can be misleading. To increase our confidence in a sorting method, we also require that it is *likely* to observe costs close to the expectation. The standard deviation of the Quicksort complexity grows linearly with $n$ [Hennequin, 1989; Knuth, 1998], which implies that the costs are concentrated around their mean for large $n$. Precise tail bounds which ensure tight concentration around the mean were derived by McDiarmid and Hayward [1996]; see also Fill and Janson [2002].

Much more information is available on the full distribution of the number of key comparisons. When suitably normalized, the number of comparisons converges in law [Regnier, 1989], with a certain unknown limit distribution. Hennequin [1989] computed its first cumulants and proved that it is not a normal distribution. The limiting distribution can be implicitly characterized by a stochastic fixed-point equation [Rösler, 1991] and it is known to have a smooth density [Fill and Janson, 2000; Tan and Hadjicostas, 1995].

Due to its efficiency in the average, Quicksort has been used as general purpose sorting method for decades, for example in the C/C++ standard library and the Java runtime library. As sorting is a widely used elementary task, even small speedups of such library implementations can be worthwhile. This caused a run on variations and modifications to the basic algorithm. One very successful optimization is based on the observation that Quicksort's performance on tiny subarrays is comparatively poor. Therefore, we should switch to some special purpose sorting method for these cases [Hoare, 1962]. Singleton [1969] proposed using Insertionsort for this task, which indeed "for small $n$ [...] is about the best sorting method known" according to Sedgewick [1975, p. 22]. He also gives a precise analysis of Quicksort where Insertionsort is used for subproblems of size less than $M$ [Sedgewick, 1977]. For his MIX implementation, the optimal choice is $M = 9$, which leads to a speedup of 14 % for $n = 10\,000$.

Another very successful optimization is to improve the choice of the pivot element by selecting the *median* of a small sample of the current subarray. This idea has been studied extensively [Chern and Hwang, 2001; Durand, 2003; Emden, 1970; Hennequin, 1989; Hoare, 1962; Martínez and Roura, 2001; Sedgewick, 1977; Singleton, 1969], and real world implementations make heavy use of it [Bentley and McIlroy, 1993].

Precise analysis of the impact of a modification often helped in understanding and assessing its usefulness, and in fact, many proposed variations turned out detrimental in the end (many examples are exposed by Sedgewick [1975]). Partitioning with more than one pivot used to be counted among those. Sedgewick [1975, p. 150ff] studies a dual-pivot Quicksort variant in detail, but finds that it uses more swaps and comparisons than classic Quicksort.[1] Later Hennequin [1991] considers the general case of partitioning into $s \geq 2$ partitions. For $s = 3$, his Quicksort uses asymptotically the same number of comparisons as classic Quicksort; for $s > 3$, he attests minor savings which, however, will not compensate for the much more complicated partitioning process in practice. These negative results may have discouraged further research along these lines in the following two decades.

[1] Interestingly, tiny changes make Sedgewick's dual-pivot Quicksort competitive w. r. t. the number of comparisons; in fact it even needs only $28/15 n \ln n + O(n)$ comparisons [Wild, 2012, Chapter 5], which is less than Yaroslavskiy's algorithm! Yet, the many swaps dominate overall performance.

In 2009, however, Vladimir Yaroslavskiy presented his new dual-pivot Quicksort variant at the Java core library mailing list.[2] After promising running time benchmarks, Oracle decided to use Yaroslavskiy's algorithm as default sorting method for arrays of primitive types[3] in the Java 7 runtime library, even though literature did not offer an explanation for the algorithm's good performance.

Only in 2012, Wild and Nebel [2012] made a first step towards closing this gap by giving exact expected numbers of swaps and comparisons for a simple version of Yaroslavskiy's algorithm. We will re-derive these results here as a special case.

The surprising finding is that Yaroslavskiy's algorithm uses only $1.9n\ln n + \mathrm{O}(n)$ comparisons on average — asymptotically 5 % less than the $2n\ln n + \mathrm{O}(n)$ comparisons needed by classic Quicksort.

The reason for the savings lies in the clever usage of stochastic dependencies: Yaroslavskiy's algorithm contains two opposite pairs of locations $\mathtt{C}_k$ (lines 11 and 15 of Algorithm 1) and $\mathtt{C}_g$ (lines 16 and 17) in the code where key comparisons are done: At $\mathtt{C}_k$, elements are first compared with the small pivot $p$ (in line 11) and then with the large pivot $q$ (in line 15) — if still needed, i. e., only if the element is larger than $p$. This means that we need only *one* comparison to identify a small element, whereas all other elements cost us a second comparisons. For $\mathtt{C}_g$ it is vice versa: We first compare with $q$, and thus large elements are cheap to identify there.

By the way partitioning is organized, it happens that elements which are initially to the *right* of the final position of $q$ are classified at $\mathtt{C}_g$; whereas elements to the left are classified at $\mathtt{C}_k$. This implies that the *number* of elements classified at $\mathtt{C}_g$ *co-varies* with the number of large elements: $\mathtt{C}_g$ is executed more often if there are more elements larger than $q$ (on average) and similarly, $\mathtt{C}_k$ is visited often if there are many small elements. Consequently, the probability that one comparison suffices to determine an element's target partition is strictly larger than 1/3 — which would be the probability if *all* elements are first compared to $p$ (or all first to $q$). The asymmetric treatment of elements is the novelty that makes Yaroslavskiy's algorithm superior to the dual-pivot partitioning schemes studied earlier.[4]

While the lower number of comparisons seems promising, Yaroslavskiy's dual-pivot Quicksort needs more swaps than classic Quicksort, so the high level analysis remains inconclusive. In this paper, we extend our analysis to detailed instruction counts, complementing previous work on classic Quicksort [Sedgewick, 1977]. However, instead of Knuth's slightly dated mythical machine MIX, we consider the Java Virtual Machine [Lindholm and Yellin, 1999] and count the number of executed Java Bytecode instructions. Wild [2012] gives similar results for Knuth's MMIX [Knuth, 2005], the successor of MIX.

The number of executed Bytecode instructions has been shown to resemble actual running time [Camesi et al., 2006], even though just-in-time compilation can have a tremendous influence [Wild et al., 2013] and some aspects of modern processor architectures are neglected.

Extending the results of Wild and Nebel [2012], the analysis in this paper includes sorting short subproblems with Insertionsort. Moreover, all previous results on Yaroslavskiy's algorithm only concern expected behavior. In this article, we show existence and give characterizations of limit distributions. A comforting result of these studies is that the standard deviation grows linearly for Yaroslavskiy's algorithm as well, which implies concentration around the mean.

This paper does not consider more refined ways to choose pivots, like selecting order statistics of a random sample. We decided to defer a detailed treatment of Yaroslavskiy's algorithm under this optimization to a separate article [Nebel and Wild, 2014].

The rest of this paper is organized as follows. Section 1.1 presents our object of study. In Section 2, we review basic notions used in the analysis later. We also define our input model and

[2] see e. g. the archive on `http://permalink.gmane.org/gmane.comp.java.openjdk.core-libs.devel/2628`

[3] Primitive types are all integer types as well as Boolean, character and floating point types. For arrays of objects, the library specification prescribes a stable sorting method, which Quicksort does not provide. Instead a variant of Mergesort is used, there.

[4] For details consider [Wild and Nebel, 2012] or the corresponding talk at `http://www.slideshare.net/sebawild/average-case-analysis-of-java-7s-dual-pivot-quicksort`.

**Algorithm 1** Yaroslavskiy's Dual-Pivot Quicksort with Insertionsort.

```
QUICKSORTYAROSLAVSKIY(A,left,right)
    // Sort A[left,...,right] (including end points).
 1  if right − left < M              // i. e. the subarray has n ≤ M elements
 2      INSERTIONSORT(A,left,right)
 3  else
 4      if A[left] > A[right]
 5          p := A[right];   q := A[left]
 6      else
 7          p := A[left];    q := A[right]
 8      end if
 9      ℓ := left + 1;   g := right − 1;   k := ℓ
10      while k ≤ g
11          if A[k] < p
12              Swap A[k] and A[ℓ]
13              ℓ := ℓ + 1
14          else
15              if A[k] ≥ q
16                  while A[g] > q and k < g do g := g − 1 end while
17                  if A[g] ≥ p
18                      Swap A[k] and A[g]
19                  else
20                      Swap A[k] and A[g];  Swap A[k] and A[ℓ]
21                      ℓ := ℓ + 1
22                  end if
23                  g := g − 1
24              end if
25          end if
26          k := k + 1
27      end while
28      ℓ := ℓ − 1;       g := g + 1
29      A[left] := A[ℓ];      A[ℓ] := p      // Swap pivots to final position
30      A[right] := A[g];     A[g] := q
31      QUICKSORTYAROSLAVSKIY(A, left , ℓ − 1)
32      QUICKSORTYAROSLAVSKIY(A, ℓ + 1, g − 1)
33      QUICKSORTYAROSLAVSKIY(A, g + 1, right)
34  end if
```

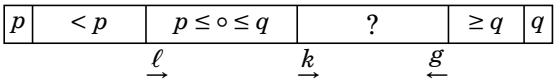
collect elementary properties of Yaroslavskiy's algorithm. In Section 3, we derive exact average costs in terms of comparisons, swaps and executed Bytecode instructions. These are used in Section 4 to identify a limiting distribution of normalized costs in all three measures, from which we obtain asymptotic variances. Finally, Section 5 summarizes our findings and puts them in context.

## 1.1. Yaroslavskiy's Algorithm

Yaroslavskiy's dual-pivot Quicksort is shown in Algorithm 1. The initial call to the procedure takes the form QUICKSORTYAROSLAVSKIY(A, 1, $n$), where A is an array containing the elements to be sorted and $n$ is its length. After selecting the outermost elements as pivots $p$ and $q$ such that $p \le q$, lines 9 – 30 of Algorithm 1 comprise the partitioning method. After that, all small elements, i. e., those smaller than $p$ (and $q$), form a contiguous region at the left end of the array, followed by $p$

and the medium elements. Finally $q$ separates the medium and large elements. After recursively sorting these three regions, the whole array is in order.

Yaroslavskiy's partitioning algorithm is an asymmetric generalization of Hoare's crossing pointers technique: The index pointers $k$ and $g$ start at the left and right ends, respectively, and are moved towards each other until they cross. Additionally, pointer $\ell$ marks the position of the rightmost small element, such that the array is kept invariably in the following form:




Our Algorithm 1 differs from Algorithm 3 of [Wild and Nebel, 2012] as follows:

- For lists of length less than $M$, we switch to INSERTIONSORT.[5] A possible implementation is given in Appendix B. The case $M = 1$ corresponds to not using Insertionsort at all.
- The swap of $\mathtt{A}[k]$ and $\mathtt{A}[g]$ has been moved behind the check $\mathtt{A}[g] \geq p$. Thereby, we never use array positions in a key comparison after we have overwritten their contents in one partitioning step; see Fact 2.3 below. (This is just to simplify discussions.)
- The comparison in line 15 has been made non-strict. For distinct elements this makes no difference, but it drastically improves performance in case of many equal keys [Wild, 2012, p. 54]. The reader might find it instructive to consider the behavior on an array with all elements equal.

Note that partitioning an array around two pivots is similar in nature to the *Dutch National Flag Problem (DNFP)* posed by Dijkstra [1976] as a programming exercise:

> *Given an array of $n$ red, white and blue pebbles, rearrange them by swaps, such that the colors form the Dutch national flag: red, white and blue in contiguous regions. Each pebble may be inspected only once and only a constant amount of extra storage may be used.*

Dijkstra assumes an operation "buck" that tells us an element's color in one shot, so any algorithm must use exactly $n$ buck-operations. Performance differences only concern the number of swaps needed.

Interestingly, Meyer [1978] gave an algorithm for the DNFP which is essentially equivalent to Yaroslavskiy's partitioning method. Indeed, it even outperforms the algorithm proposed by Dijkstra [McMaster, 1978]! Yet, the real advantage of Yaroslavskiy's partitioning scheme — the reduced expected number of key comparisons — is hidden by the atomic buck operation; its potential use in Quicksort went unnoticed.

# 2. Preliminaries

In this section, we recall elementary definitions and collect some notation and basic facts used throughout this paper.

By $\mathscr{H}_n := \sum_{i=1}^{n} 1/i$, we denote the $n$th *Harmonic Number*. We use $\delta_{ij}$ for the *Kronecker delta*, which is defined to be 1 if $i = j$ and 0 otherwise. We define $x\ln(x) = 0$ for $x = 0$, so that $x \mapsto x\ln(x)$ becomes a continuous function on $[0,\infty)$.

The *probability* of an event $E$ is denoted by $\mathbb{P}[E]$ and we write $\mathbb{1}_{\{E\}}$ for its *indicator random variable*, which is 1 if the event occurs and 0 otherwise. For a random variable $X$, let $\mathbb{E}[X]$, $\mathrm{Var}(X)$ and $\mathscr{L}(X)$ denote its *expectation*, *variance* and *distribution*, respectively. $X \overset{\mathscr{D}}{=} Y$ means that $X$ has the *same distribution* as $Y$.

[5] Note that even if Sedgewick [1977] proposes to use one final run of Insertionsort over the entire input array, modern cache hierarchies suggest to immediately sort small subarrays as done in our implementation.

By $\|X\|_p := \mathbb{E}[|X|^p]^{1/p}$, $1 \le p < \infty$, we denote the *$L_p$-norm* of random variable $X$. For random variables $X_1, X_2, \ldots$ and $X$, we say *$X_n$ converges in $L_p$ to $X$*

$$X_n \xrightarrow{L_p} X \qquad \text{iff} \qquad \lim_{n\to\infty} \|X_n - X\|_p = 0\,.$$

The *Bernoulli distribution* with parameter $p$ is written as $\mathrm{B}(p)$. Provided that $\sum_{r=1}^{b} p_r = 1$ and $b \ge 1$ is a fixed integer, we denote by $\mathrm{M}(n; p_1, \ldots, p_b)$ the *multinomial distribution* with $n$ trials and success probabilities $p_1, \ldots, p_b \in [0,1]$. For *random probabilities* $V = (V_1, \ldots, V_b)$, i. e., random variables $0 \le V_r \le 1$ $(r = 1, \ldots, b)$ with $\sum_{r=1}^{b} V_r = 1$ almost surely, we write $Y \stackrel{@}{=} \mathrm{M}(n; V_1, \ldots, V_b)$ to denote that $Y$ *conditional* on $V = v$ (i. e., conditional on $(V_1, \ldots, V_b) = (v_1, \ldots, v_b)$) is multinomially $\mathrm{M}(n; v_1, \ldots, v_b)$ distributed.

For $k, r, b \in \mathbb{N}$ satisfying $k \le r + b$, the *hypergeometric distribution* with $k$ trials from $r$ red and $b$ black balls is denoted by $\mathrm{HypG}(k, r, r+b)$. Given an urn with $r$ red and $b$ black balls, it is the distribution of the number of red balls drawn when drawing $k$ times without replacement. The mean and variance of a hypergeometrically $\mathrm{HypG}(k, r, r+b)$ distributed random variable $G$ are given by [Kendall, 1945, p. 127]

$$\mathbb{E}[G] = k \cdot \frac{r}{r+b}\,, \qquad \mathrm{Var}(G) = \frac{krb(r+b-k)}{(r+b)^2(r+b-1)}\,. \tag{2.1}$$

As for the multinomial distribution, given *random* parameters $K$, $R$ in $\{0, \ldots, n\}$ we use $Y \stackrel{@}{=} \mathrm{HypG}(K, R, n)$ to denote that $Y$ *conditional* on $(K, R) = (k, r)$ is hypergeometrically $\mathrm{HypG}(k, r, n)$ distributed.

## 2.1. Input Model

We assume the *random permutation model*: The keys to be sorted are the integers $1, \ldots, n$ and each permutation of $\{1, \ldots, n\}$ has equal probability $1/n!$ to become the input. Note that we implicitly exclude the case of equal keys by that.

As sorting is only concerned with the relative order of elements, not the key values themselves, we can equivalently assume keys to be i. i. d. real random variables from any (non-degenerate) continuous distribution. Equal keys do not occur almost surely and the *ranks* of the elements form in fact a random permutation of $\{1, \ldots, n\}$ again [see e. g. Mahmoud 2000]. For the analysis of Section 4, this alternative point of view will be helpful.

## 2.2. Basic Properties of Yaroslavskiy's Algorithm

As typical for divide and conquer algorithms, the analysis is based on setting up a recurrence relation for the costs. For such a recurrence to hold, it is vital that the costs for subproblems of size $k$ behave the same as the costs for dealing with an original random input of initial size $k$. For Quicksort, we require the following property:

**Property 2.1** (Randomness Preservation)**.**
If the whole input is a (uniformly chosen) random permutation of its elements, so are the subproblems Quicksort is recursively invoked on.

Hennequin [1989] showed that Property 2.1 is implied by the following property.

**Property 2.2** (Sufficient Condition for Randomness Preservation)**.**
Every key comparison involves a pivot element of the current partitioning step.

Now, it is easy to verify that Yaroslavskiy's algorithm fulfills Property 2.2 and hence Property 2.1.

Since Yaroslavskiy's algorithm is an in-place sorting method, it modifies the array `A` over time. This dynamic component makes discussions inconvenient. Fortunately, a sharp look at the algorithm reveals the following fact, allowing a more static point of view:

**Fact 2.3.** The array elements used in key comparisons have not been changed since the beginning of the current partitioning step. More precisely, if a key comparison involves an array element $\mathtt{A}[i]$, then there has not been a write access to $\mathtt{A}[i]$ in the current partitioning step. □

# 3. Average Case Analysis

Throughout Section 3, we assume that array A stores a random permutation of $\{1,\dots,n\}$.

## 3.1. The Dual-Pivot Quicksort Recurrence

In this section, we obtain a general solution to the recurrence relation corresponding to dual-pivot Quicksort. We denote by $\mathbb{E}[C_n]$ the *expected costs* — where different cost measures will be inserted later — of Yaroslavskiy's algorithm on a random permutation of $\{1,\dots,n\}$. $\mathbb{E}[C_n]$ decomposes as

$$\mathbb{E}[C_n] \;=\; \text{costs of first partitioning step} + \text{costs for subproblems}\,. \tag{3.1}$$

As Yaroslavskiy's algorithm satisfies Property 2.1, the costs for recursively sorting subarrays can be expressed in terms of $C$ with smaller arguments, leading to a recurrence relation. Every (sorted) pair of elements has the same probability $1/\binom{n}{2}$ of becoming pivots. Conditioning on the ranks of the pivots, this gives the following recursive form for the expected costs $\mathbb{E}[C_n]$ of Yaroslavskiy's algorithm on a random permutation of size $n$:

$$\mathbb{E}[C_n] = \begin{cases} \mathbb{E}[T_n] \,+\, 1/\binom{n}{2} \displaystyle\sum_{1\le p<q\le n} \big(\mathbb{E}[C_{p-1}] + \mathbb{E}[C_{q-p-1}] + \mathbb{E}[C_{n-q}]\big), & \text{for } n > M; \\ \mathbb{E}[C_n^{\mathrm{IS}}], & \text{for } n \le M, \end{cases} \tag{3.2}$$

where $C_n^{\mathrm{IS}}$ denotes the costs of InsertionSorting a random permutation of $\{1,\dots,n\}$ and $T_n$ is the cost contribution of the *first* partitioning step. This function $T_n$ quantifies the "toll" we have to pay for unfolding the recurrence once, therefore we will call $T_n$ the *toll function* of the recurrence. By adapting the toll function, we can use the same recurrence to describe different kinds of costs and we only need to derive a general solution to this single recurrence relation as provided by the following theorem:

**Theorem 3.1.** *Let $\mathbb{E}[C_n]$ be recursively defined by* (3.2)*. Then, $\mathbb{E}[C_n]$ satisfies*

$$\begin{aligned} \mathbb{E}[C_n] \;=\; & \frac{1}{\binom{n}{4}} \sum_{i=M+4}^{n} \binom{i}{4} \sum_{j=M+2}^{i-2} \Big( \mathbb{E}[T_{j+2}] - \tfrac{2j}{j+2}\,\mathbb{E}[T_{j+1}] + \frac{\binom{j}{2}}{\binom{j+2}{2}}\,\mathbb{E}[T_j] \Big) \\ & + \Big( \tfrac{n+1}{5} + \frac{\binom{M+3}{4} - \binom{M+4}{5}}{\binom{n}{4}} \Big) \mathbb{E}[C_{M+3}] \;-\; \tfrac{M-1}{M+3} \Big( \tfrac{n+1}{5} - \frac{\binom{M+4}{5}}{\binom{n}{4}} \Big) \mathbb{E}[C_{M+2}], \qquad \textit{for } n \ge M+3. \end{aligned} \tag{3.3}$$

*As an immediate consequence, $\mathbb{E}[C_n]$ — seen as a function of $\mathbb{E}[T_n]$ — is* linear *in $\mathbb{E}[T_n]$.*

The proof for Theorem 3.1 uses several layers of successive differences of $\mathbb{E}[C_n]$ to finally obtain a telescoping recurrence. Substituting back in then yields (3.3). The detailed computations are given in Appendix A. This general solution still involves non-trivial double sums. For the cost measures we are interested in, the following proposition gives an explicit solution for (3.2).

**Proposition 3.2.** *Let $\mathbb{E}[C_n]$ be recursively defined by* (3.2) *and let $\mathbb{E}[T_n] = an + b$ for $n \ge M+1$. Then, $\mathbb{E}[C_n]$ satisfies*

$$\begin{aligned} \mathbb{E}[C_n] \;=\; & \tfrac{6}{5}a(n+1)\big(\mathscr{H}_{n+1} - \mathscr{H}_{M+2}\big) \,+\, \tfrac{1}{5}(n+1)\big(\tfrac{19}{5}a + \tfrac{6(b-a)}{M+2}\big) \,+\, \tfrac{a-b}{2} \\ & + \tfrac{1}{5}(n+1) \sum_{k=0}^{M} \frac{3M-2k}{\binom{M+2}{3}}\,\mathbb{E}[C_k^{\mathrm{IS}}] \,+\, \frac{\binom{M+4}{5}}{\binom{n}{4}} R_M\,, \qquad \textit{for } n \ge M+3, \end{aligned} \tag{3.4}$$

*where*

$$R_M \;=\; \tfrac{6}{5}a + \tfrac{2(a-b)}{M+3} + \tfrac{5b-17a}{2(M+4)} - \tfrac{M-1}{M+4}\,\mathbb{E}[C_{M+3}] + \tfrac{M-1}{M+3}\,\mathbb{E}[C_{M+2}]\,.$$

*If* $\mathbb{E}[C_n^{\mathrm{IS}} = 0]$ *for all* $n$, $\mathbb{E}[C_n]$ *has the following asymptotic representation:*

$$\mathbb{E}[C_n] \;=\; \tfrac{6}{5}an\ln n + \big(\tfrac{19}{25}a + W\big)n + \tfrac{6}{5}a\ln n + \big(\tfrac{153}{50}a - \tfrac{1}{2}b + W\big) + \mathrm{O}\big(\tfrac{1}{n}\big), \qquad n \to \infty, \tag{3.5}$$

*where*

$$W \;=\; \tfrac{6}{5}\big(a\,\gamma + \tfrac{b-a}{M+2} - a\mathscr{H}_{M+2}\big)$$

*and* $\gamma \approx 0.57721$ *is the Euler-Mascheroni constant.*

*Moreover, if the toll function* $\mathbb{E}[T_n]$ *has essentially the form given above, but with* $\mathbb{E}[T_2] = 0 \neq 2a + b$, *we get an additional summand* $-\delta_{M1} \cdot \frac{1}{10}(2a+b)\cdot(n+1)$ *in* (3.4). *Equation* (3.5) *remains valid if we set* $W = \frac{6}{5}\big(a\,\gamma + \frac{b-a}{M+2} - a\mathscr{H}_{M+2} - \delta_{M1}(2a+b)/10\big)$.

The proof of Proposition 3.2 is basically "by computing", the details are again deferred to Appendix A.

*Remark:* For constant $M$, i. e., $M = \Theta(1)$ as $n \to \infty$, only the linear term of the expected costs is affected by $M$. This means that for the leading term of $\mathbb{E}[C_n]$, the "base case strategy" for solving small subproblems is totally irrelevant.

## 3.2. Basic Block Execution Frequencies

In this section, we compute for every single instruction of Yaroslavskiy's algorithm how often it is executed in expectation. Based on that, we can easily derive the expected number of key comparisons, swaps, but also more detailed measures, such as the expected number of executed Bytecode instructions. This is the kind of analysis Knuth popularized through his book series *The Art of Computer Programming* [Knuth, 1998]. A corresponding analysis of classic single pivot Quicksort was done by Sedgewick [1977]. Like the Quicksort variant discussed there, Algorithm 1 uses Insertionsort for sorting small subarrays. Our detailed implementation of Insertionsort and its analysis are given in Appendix B.

Consecutive lines of purely sequential[6] code always have the same execution frequencies; contracting maximal blocks of such code yields the control flow graph (CFG). Figure 1 shows the resulting CFG for Yaroslavskiy's algorithm. Simple flow conservation arguments (a. k. a. Kirchhoff's laws) allow to express execution frequencies of some blocks by the frequencies of others: The execution frequencies of the 20 basic blocks of Figure 1 only depend on the following nine frequencies: $A$, $B$, $R$, $F$, $C^{(1)}$, $C^{(3)}$, $C^{(4)}$, $S^{(1)}$ and $S^{(3)}$. The name $C^{(i)}$ indicates that this frequency counts executions of the $i$th location in the code of Algorithm 1, where a key comparison is done. Similarly, $S^{(i)}$ corresponds to the $i$th swap location.

The results are summarized in Tables 1 and 2 at the end of the section.

The expected execution frequencies allow a recursive representation of the following form, here using the example of $C^{(1)}$:

$$\mathbb{E}[C_n^{(1)}] \;=\; \begin{cases} \mathbb{E}[T_{C^{(1)}}(n)] \;+\; 1\big/\binom{n}{2} \sum\limits_{1 \le p < q \le n} \big(\mathbb{E}[C_{p-1}^{(1)}] + \mathbb{E}[C_{q-p-1}^{(1)}] + \mathbb{E}[C_{n-q}^{(1)}]\big), & \text{for } n > M; \\ 0, & \text{for } n \le M, \end{cases} \tag{3.6}$$

where $T_{C^{(1)}} = T_{C^{(1)}}(n)$ is the frequency specific toll function — namely the corresponding frequency during the first partitioning step only. For the other frequencies, we similarly denote by $T_A$, $T_F$, $T_{C^{(3)}}$, $T_{C^{(4)}}$, $T_{S^{(1)}}$ and $T_{S^{(3)}}$ the toll functions corresponding to $A$, $F$, $C^{(3)}$, $C^{(4)}$, $S^{(1)}$ and $S^{(3)}$, respectively.

[6] Purely sequential blocks contain neither (outgoing) jumps, nor targets for (incoming) jumps from other locations, except for the last and first instructions, respectively.

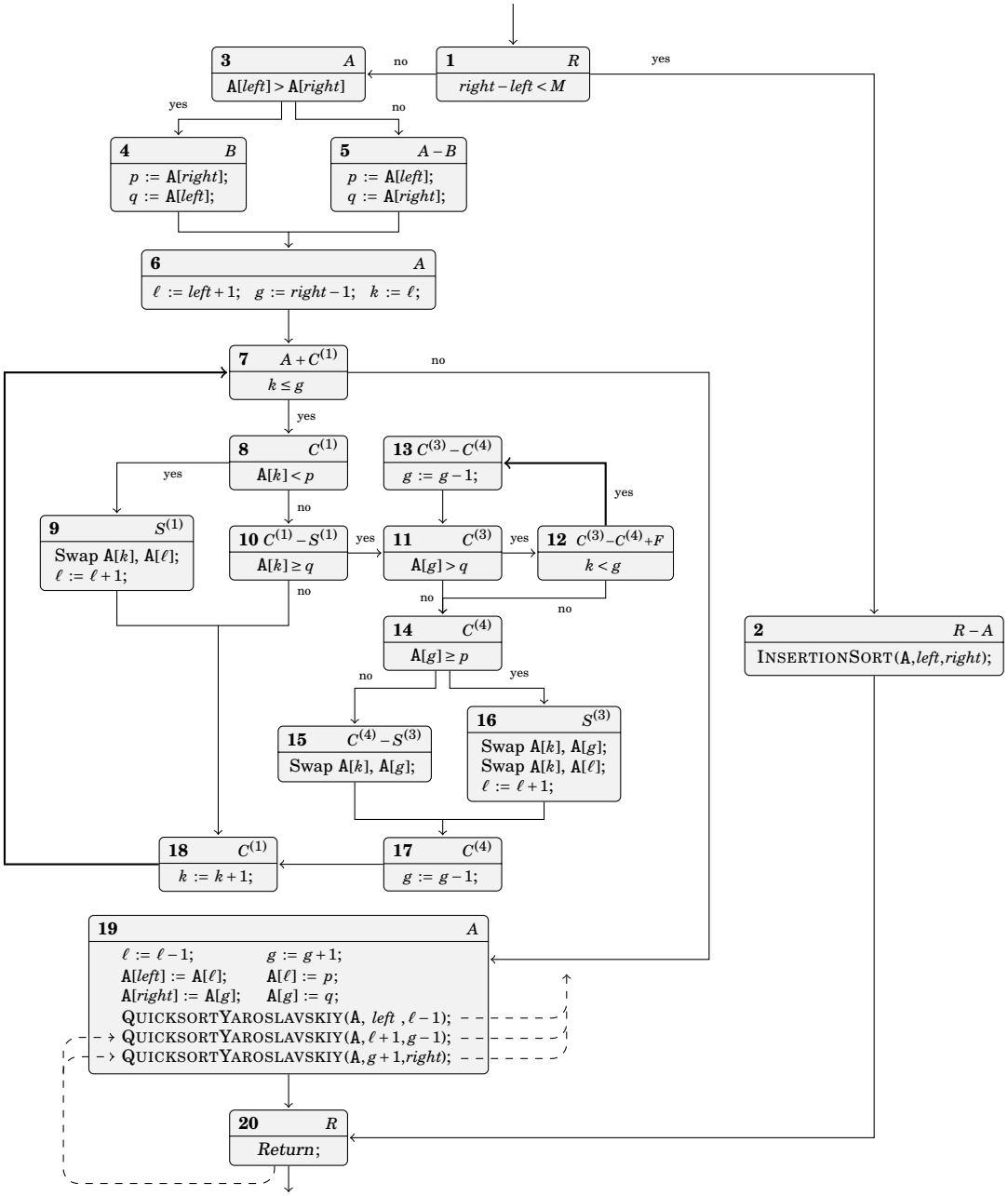



**Figure 1:** Control flow graph for Algorithm 1. The algorithm is decomposed into *basic blocks* of purely sequential code. Possible transitions from one block to another are indicated by arrows. Blocks with two outgoing arrows end with a conditional, the "yes" path is taken if the condition is fulfilled, otherwise the "no" transition is chosen. We refer to blocks using the number shown in the upper left corner. In the upper right corner, a block's symbolic *execution frequency* is given. For clarity of presentation, the recursive calls in block 19 are not explicitly shown, but only sketched by the dashed arrows. Block 2 calls InsertionSort which is given in Appendix B.

Later, in Section 4 where the distinction becomes important, we reserve $A$, $B$ etc. for the actual random variables and explicitly write $\mathbb{E}[A]$, $\mathbb{E}[B]$ etc. for their expectations.

The frequencies $F$, $C^{(1)}$, $C^{(3)}$, $C^{(4)}$, $S^{(1)}$ and $S^{(3)}$ correspond to basic blocks in the body of the main partitioning loop, i. e., blocks 8 – 18. All these blocks have in common that they are *not executed at all* during calls with $right - left \le 1$, i. e., when $n \le 2$: In that case we have $k > g$ directly after block 6 and hence immediately leave the partitioning loop from block 7 to block 19. Therefore, we have $T_F(2) = T_{C^{(1)}}(2) = T_{C^{(3)}}(2) = T_{C^{(4)}}(2) = T_{S^{(1)}}(2) = T_{S^{(3)}}(2) = 0$. In the subsequent sections, we will determine the toll functions for $n \ge 3$.

For that, we will first compute their values given *fixed* pivot ranks $P$ and $Q$, i. e., we determine their distribution *conditional* on $(P,Q) = (p,q)$. Here, we capitalized $P$ and $Q$ to emphasize the fact that the pivot ranks are themselves random variables. Then, we get the unconditional expected frequencies via the *law of total expectation*. Note that for permutations of $\{1,\dots,n\}$, ranks and values coincide and we will not always dwell on the difference to keep the presentation concise, but unless stated otherwise, $P$ and $Q$ refer to the ranks of the two pivots.

### 3.2.1. The Crossing Point Lemma

The following lemma is the key to the precise analysis of the execution frequencies that depend on how pointers $k$ and $g$ "cross". As the pointers are moved alternatingly towards each other, one of them will reach the crossing point *first* — waiting for the other to arrive.

**Lemma 3.3** (Crossing Point Lemma)**.** *Let* A *store a random permutation of* $\{1,\dots,n\}$ *with* $n \ge 2$*. Then, Algorithm 1 leaves the outer loop of the first partitioning step with*

$$k \;=\; q+\delta \;=\; g+1+\delta, \qquad \text{where } \delta = 0 \text{ or } \delta = 1. \tag{3.7}$$

*(More precisely,* (3.7) *holds for the valuations of $k$, $g$ and $q$ upon entrance of block 19).*
*Moreover, $\delta = 1$* iff *initially* $\mathtt{A}[q] > q$ *holds, where $q = \max\{\mathtt{A}[1], \mathtt{A}[n]\}$ is the large pivot.*

*Proof of Lemma 3.3:* Between two consecutive "$k \le g$"-checks in block 7, we move $k$ and $g$ towards each other by at most one position each; so we always have $k \le g+2$ and we exit the loop as soon as $k > g$ holds. Therefore, we always leave the loop with $k = g+1+\delta$ for some $\delta \in \{0,1\}$. In the end, $q$ is moved to position $g$ in block 19. Just above in the same block, $g$ has been incremented, so we have $g = q-1$ upon entrance of block 19.

For the "moreover" part, we show both implications separately. Assume first that $\delta = 1$, i. e., the loop is left with a difference of $\delta + 1 = 2$ between $k$ and $g$. This difference can only show up when both $k$ is incremented *and* $g$ is decremented in the last iteration. Hence, in this last iteration we must have gone from block 10 to 11 and accordingly $\mathtt{A}[k] \ge q$ must have held there — and by Fact 2.3 $\mathtt{A}[k]$ still holds its initial value.

In case $k < n$, even strict inequality $\mathtt{A}[k] > q$ holds since we then have $\mathtt{A}[k] \ne \mathtt{A}[n] = q$ by the assumption of distinct elements. Now assume towards a contradiction, $k = n$ holds in the last execution of block 10. Since $g$ is initialized in block 6 to $right - 1 = n-1$ and is only decremented in the loop, we have $g \le n-1$. But this is a contradiction to the loop condition "$k \le g$": $n = k \le g \le n-1$. So, $\mathtt{A}[k] > q$ holds for the last execution of block 10.

By assumption, $\delta = 1$, so $k = q+1$ upon termination of the loop. As $k$ has been incremented exactly once since the last test in block 10, we find $\mathtt{A}[q] > q$ there, as claimed.

Now, assume conversely that initially $\mathtt{A}[q] > q$ holds. As $g$ stops at $q-1$ and is decremented in block 17, we have $g = q$ for the last execution of block 11. Using the assumption yields $\mathtt{A}[g] = \mathtt{A}[q] > q$, since by Fact 2.3, $\mathtt{A}[q]$ still holds its initial value. Thus, we take the transition to block 12. Execution then proceeds with block 14, otherwise we would enter block 11 again, contradicting the assumption that we just finished its *last* execution. The transition from block 12 to 14 is only taken if $k \ge g = q$. With the following decrement of $g$ and increment of $k$, we leave the loop with $k \ge g+2$, so $\delta = 1$. □

**Corollary 3.4.** Let $\delta \in \{0,1\}$ be the random variable from Lemma 3.3.
It holds $\mathbb{E}[\delta] = \frac{1}{3}$ and $\mathbb{E}[\delta \mid (P,Q) = (p,q)] = \frac{n-q}{n-2}$.

*Proof:* We first compute the conditional expectation. As $\delta \in \{0,1\}$, we have $\mathbb{E}[\delta|P,Q] = \mathbb{P}[\delta = 1|P,Q]$, so it suffices to compute this probability. Now by Lemma 3.3, we have $\mathbb{P}[\delta = 1 \mid P,Q] = \mathbb{P}[\mathtt{A}[q] > q \mid (P,Q) = (p,q)]$. We do a case distinction.

- For $q < n$, $\mathtt{A}[q]$ is one of the non-pivot elements. (We have $1 \le p < q < n$.) Any of the $n-2$ non-pivot elements can take position $\mathtt{A}[q]$, and among those, $n-q$ elements are strictly greater than $q$. This gives a probability of $\frac{n-q}{n-2}$ for $\mathtt{A}[q] > q$.
- For $q = n$, $q$ is the maximum of all elements in the list, so we cannot possibly have $\mathtt{A}[q] > q$. This implies a probability of $0 = \frac{n-q}{n-2}$.

By the *law of total expectation*, the unconditional expectation is given by:

$$\begin{aligned}\mathbb{E}[\delta] &= \sum_{1\le p<q\le n} \mathbb{P}[(P,Q)=(p,q)] \cdot \mathbb{E}[\delta \mid (P,Q)=(p,q)] = 1/\tbinom{n}{2} \sum_{1\le p<q\le n} \tfrac{n-q}{n-2} \\ &= \frac{1}{\binom{n}{2}(n-2)} \Big( \sum_{1\le p<q\le n} n - \sum_{1\le p<q\le n} q \Big) = \frac{1}{\binom{n}{2}(n-2)} n\tbinom{n}{2} - \frac{\frac{2}{3}(n+1)}{n-2} = \frac{n-\frac{2}{3}(n+1)}{n-2} = \tfrac{1}{3}.\end{aligned}$$

□

The following expectations are used several times below, so we collect them here.

**Lemma 3.5.** $\mathbb{E}[P] = \frac{1}{3}(n+1)$ *and* $\mathbb{E}[Q] = \frac{2}{3}(n+1)$.

*Proof:* Conditioning on $(P,Q) = (p,q)$, we find

$$\mathbb{E}[Q] = \sum_{1\le p<q\le n} \frac{1}{\binom{n}{2}} \cdot q = \frac{1}{\binom{n}{2}} \sum_{q=2}^{n} q \sum_{p=1}^{q-1} 1 = \tfrac{2}{3}(n+1).$$

A similar calculation for $P$ proves the lemma. □

### 3.2.2. Frequency $A$

The frequency $A = A_n$ equals the number of partitioning steps or equivalently the number of (recursive) calls with $right - left \ge M$ when initially calling QUICKSORTYAROSLAVSKIY($\mathtt{A}, 1, n$) with a random permutation stored in $\mathtt{A}$. Therefore, the contribution $T_A$ of *one* partitioning step is $T_A(n) = 1$. By Proposition 3.2 with $T_n = 1$ and $C_n^{\mathrm{IS}} = 0$, we obtain the closed form

$$\mathbb{E}[A_n] = \tfrac{6}{5(M+2)}(n+1) - \tfrac{1}{2} + \tfrac{3}{10}\tbinom{M+1}{4}/\tbinom{n}{4}. \tag{3.8}$$

### 3.2.3. Frequency $R$

By $R = R_n$, we denote the number of calls to QUICKSORTYAROSLAVSKIY including those directly passing control to INSERTIONSORT for small subproblems. Every partitioning step entails three additional recursive calls on subarrays (see block 19). Moreover, we have one additional initial call to the procedure. Together, this implies

$$R_n = 3A_n + 1. \tag{3.9}$$

### 3.2.4. Frequency $B$

Frequency $B$ counts how often we execute block 4. This block is reached at most once per partitioning step, namely *iff* $\mathtt{A}[\mathit{left}] > \mathtt{A}[\mathit{right}]$. For random permutations, the probability for that is exactly $1/2$, so we find

$$\mathbb{E}[B_n] = \tfrac{1}{2}\mathbb{E}[A_n]. \tag{3.10}$$

### 3.2.5. Frequency $C^{(1)}$

$C^{(1)}(n)$ denotes the execution frequency of block 8 of Yaroslavskiy's algorithm. Block 8 is the first statement in the outer loop and the last block of this loop (block 18) is the only place where $k$ is incremented. Therefore, $T_{C^{(1)}}$ is the number of different values that $k$ attains during the first partitioning step. The following corollary quantifies this number as $T_{C^{(1)}} = Q - 2 + \delta$.

**Corollary 3.6.** Let us denote by $\mathcal{K}$ the set of values that pointer $k$ attains at block 8. Similarly, let $\mathcal{G}$ be the set of values of $g$ in block 11. We have

$$\begin{aligned} \mathcal{K} &= \{2, 3, \dots, Q - 1 + \delta\}, & |\mathcal{K}| &= Q - 2 + \delta, \\ \mathcal{G} &= \{n-1, n-2, \dots, Q+1, Q\}, & |\mathcal{G}| &= n - Q. \end{aligned}$$

*Proof:* By Lemma 3.3, we leave the outer loop with $k = Q + \delta$ and $g = Q - 1$. Since the last execution of block 8, $k$ has been incremented exactly once (in block 18), so the last value of $k$, namely $Q + \delta$, is not observed at block 8. Similarly, after the last execution of block 11, we always pass block 17 where $g$ is decremented. So the last value $Q - 1$ for $g$ is not attained in block 11. □

Continuing with frequency $C^{(1)}$, note that $Q$ and $\delta$ and hence $T_{C^{(1)}} = Q - 2 + \delta$ are random variables. By linearity of the expectation $\mathbb{E}[T_{C^{(1)}}] = \mathbb{E}[Q] - 2 + \mathbb{E}[\delta]$ holds, so with Lemma 3.5 and Corollary 3.4, we find

$$\mathbb{E}[T_{C^{(1)}}(n)] \;=\; \tfrac{1}{3}(n+1) \;-\; 2 \;+\; \tfrac{1}{3} \;=\; \tfrac{2}{3}n - 1\,. \tag{3.11}$$

### 3.2.6. Frequency $S^{(1)}$

Frequency $S^{(1)}$ corresponds to block 9. Block 9 is executed as often as block 8 is reached with $\mathtt{A}[k] < p$. This number depends on the input permutation: $T_{S^{(1)}}(n)$ is exactly the number of elements smaller than $p$ that happen to be located at positions in $\mathcal{K}$, the range that pointer $k$ scans. Denote this quantity by $s@\mathcal{K}$.

**Lemma 3.7.** *Conditional on the pivot ranks $P$ and $Q$, $s@\mathcal{K}$ is hypergeometrically* $\mathrm{HypG}(P-1, Q-2, n-2)$ *distributed.*

*Proof:* This is seen by considering the following (imaginary) generation process of the current input permutation: Assuming fixed pivots $(P, Q) = (p, q)$, we have to generate a random permutation of the remaining $n - 2$ elements $E := \{1, \dots, n\} \setminus \{p, q\}$. To do so, we first choose a random subset $S$ of the free positions $F := \{2, \dots, n-1\}$ with $|S| = p - 1$. Then we put a random permutation of $\{1, \dots, p-1\}$ into positions $S$ and a random permutation of $E \setminus \{1, \dots, p-1\}$ into positions $F \setminus S$. It is easily checked that this generates all permutations of $E$ with equal probability, if all choices are done uniformly.

Then by definition, $s@\mathcal{K} = |S \cap \mathcal{K}|$. This seemingly innocent equation hides a subtle intricacy not to be overlooked: $\mathcal{K} = \{2, \dots, q - 1 + \delta\}$ (Corollary 3.6) is itself a random variable which depends on the permutation via $\delta$. Luckily, the characterization of $\delta$ from Lemma 3.3 allows to resolve this inter-dependence. $\mathcal{K} = \{2, \dots, q\}$ if $\mathtt{A}[q] > q$ and $\mathcal{K} = \{2, \dots, q-1\}$ otherwise. Stated differently, we get the additional position $q$ in $\mathcal{K}$ *iff* the element at that position is large, which means position $q$ *never* contributes towards *small* elements at positions in $\mathcal{K}$. As a result, $s@\mathcal{K} = s@\mathcal{K}' = |S \cap \mathcal{K}'|$ for $\mathcal{K}' = \{2, \dots, q-1\}$, which is constant for fixed pivot values $p$ and $q$.

Drawing positions $S$ for small elements one by one is then equivalent to choosing $|S|$ balls out of an urn with $n - 2$ balls without replacement. If $|\mathcal{K}'|$ of the $n - 2$ balls are red, then $s@\mathcal{K}$ equals the number of red balls drawn, which is hypergeometrically

$$\mathrm{HypG}(|S|, |\mathcal{K}'|, n-2) \;=\; \mathrm{HypG}(p-1, q-2, n-2)$$

distributed by definition. □

The mean of hypergeometric distributions from (2.1) translates into the *conditional* expectation $\mathbb{E}[s@\mathcal{K} \,|\, P, Q] = (P-1)(Q-2)/(n-2)$. By the *law of total expectation*, we can compute the unconditional

expected value:

$$\mathbb{E}[T_{S^{(1)}}(n)] \;=\; \mathbb{E}[s@\mathscr{K}] \;=\; \mathbb{E}_{(P,Q)}\big[\mathbb{E}[s@\mathscr{K}\,|\,P,Q]\big] \;=\; 1\Big/\tbinom{n}{2} \sum_{1\le p<q\le n} \tfrac{(p-1)(q-2)}{n-2} \;=\; \tfrac{1}{4}n - \tfrac{5}{12}\,. \tag{3.12}$$

### 3.2.7. Frequency $C^{(3)}$

Block 11 — whose executions are counted in $C^{(3)}$ — compares $\mathtt{A}[g]$ to $q$. After every execution of block 11, pointer $g$ is decremented: depending on whether we leave the loop or not, either in block 13 or in block 17. Therefore, we execute block 11 for every value that $g$ attains at block 11, which by Corollary 3.6 amounts to $T_{C^{(3)}}(n) = |\mathscr{G}| = n - Q$. Using Lemma 3.5, we find

$$\mathbb{E}[T_{C^{(3)}}(n)] \;=\; \tfrac{1}{3}n - \tfrac{2}{3}\,. \tag{3.13}$$

### 3.2.8. Frequency $F$

Frequency $F$ counts how often we take the transition from block 12 to block 14. This transition is taken when we exit the inner loop of Yaroslavskiy's algorithm because the second part of its loop condition, "$k < g$", is violated, which means we had $k \ge g$.

After this has happened, we always execute blocks 17 and 18, where we decrement $g$ *and* increment $k$. Moreover by Lemma 3.3, $k$ is at most $g+2$ *after* the loop, and equality holds *iff* $\delta = 1$. So at block 12, we always have $k \le g$, which means the violation of the loop condition occurs for $k = g$ and can only happen in case $\delta = 1$.

We can also show that it *must* happen whenever $\delta = 1$: By Lemma 3.3, we have $\mathtt{A}[q] > q$, and $k = q+1 = g+2$ *after* the loop. Therefore, *during* the last iteration of the loop, $g = k = q$ and hence $\mathtt{A}[k] = \mathtt{A}[g] = \mathtt{A}[q] > q$ holds. As a consequence, execution always proceeds through blocks 8, 10 and 11 to block 12. There, "$k < g$" is not fulfilled, so we take the transition to block 14. Together, we obtain $T_F = \delta$ and Corollary 3.4 gives

$$\mathbb{E}[T_F(n)] \;=\; \tfrac{1}{3}\,. \tag{3.14}$$

### 3.2.9. Frequency $C^{(4)}$

Frequency $C^{(4)}$ corresponds to block 14, which compares $\mathtt{A}[g]$ to $p$. From the control flow graph, it is obvious that $C^{(4)}$ is the sum of the frequencies of the two incoming transitions, namely block 11 to 14 and block 12 to 14. The latter is exactly $F$.

For the former, recall from above that block 11 is executed once for all values $\mathscr{G} = \{n-1, n-2, \dots, Q\}$ that pointer $g$ attains there. The transition from block 11 to block 14 is taken *iff* $\mathtt{A}[g] \le q$. As $1 < g < n$ holds and all elements are distinct, $\mathtt{A}[g] = p$ cannot occur. Therefore, exactly the small and medium elements that are located at positions in $\mathscr{G}$ cause this transition; denote their number by $sm@\mathscr{G}$. A very similar argument as in the proof of Lemma 3.7 shows that conditional on $P$ and $Q$, $sm@\mathscr{G}$ is hypergeometrically $\mathrm{HypG}(Q-2, n-Q, n-2)$ distributed.

Adding both contributions yields $T_{C^{(4)}} = \delta + (sm@\mathscr{G})$. Using Corollary 3.4 and equation (2.1) shows

$$\mathbb{E}[T_{C^{(4)}}(n)] \;=\; \tfrac{1}{3} + 1\Big/\tbinom{n}{2} \sum_{1\le p<q\le n} \tfrac{(q-2)(n-q)}{n-2} \;=\; \tfrac{1}{6}n - \tfrac{1}{6}\,. \tag{3.15}$$

### 3.2.10. Frequency $S^{(3)}$

Frequency $S^{(3)}$ counts executions of block 16. Key to its analysis are the following two observations:

1. Block 16 and block 9 (with frequency $S^{(1)}$) are the only locations inside the loop where pointer $\ell$ is changed. Therefore, $T_{S^{(1)}} + T_{S^{(3)}} = |\mathscr{L}| - 1$, where $\mathscr{L}$ is the set of values pointer that $\ell$ attains inside the loop (minus one as we leave the loop after the last increment of $\ell$ without executing blocks 9 and 16 again).

| Toll function | $T_A$ | $T_F$ | $T_{C^{(1)}}$ | $T_{C^{(3)}}$ | $T_{C^{(4)}}$ | $T_{S^{(1)}}$ | $T_{S^{(3)}}$ |
|---|---|---|---|---|---|---|---|
| Expected value ($n \geq 3$) | 1 | $\frac{1}{3}$ | $\frac{2}{3}n-1$ | $\frac{1}{3}n-\frac{2}{3}$ | $\frac{1}{6}n-\frac{1}{6}$ | $\frac{1}{4}n-\frac{5}{12}$ | $\frac{1}{12}n-\frac{1}{4}$ |
| Special value for $n=2$ | no | 0 | 0 | 0 | 0 | 0 | 0 |

| Toll function | $T_{C^{(QS)}}$ | $T_{S^{(QS)}}$ | $T_{W^{(QS)}}$ | $T_{BC^{(QS)}}$ |
|---|---|---|---|---|
| Expected value ($n \geq 3$) | $\frac{19}{12}n-\frac{17}{12}$ | $\frac{1}{2}n+\frac{7}{6}$ | $\frac{11}{12}n+\frac{31}{12}$ | $\frac{217}{12}n+\frac{265}{4}$ |
| Special value for $n=2$ | 1 | 2 | 4 | $\frac{189}{2}$ |

**Table 1:** Expected values for the toll functions for execution frequencies that characterize the block execution frequencies of all blocks in Algorithm 1 (top) and the derived toll functions for the expected number of comparison, swaps, write accesses and Bytecode instructions during the first partitioning step (bottom).

| Frequency | $M=1$ (exact solution for $n \geq 4$) | $M \geq 2$ (asymptotic with error term $\mathrm{O}(\frac{1}{n^4})$) |
|---|---|---|
| $\mathbb{E}[A]$ | $\frac{2}{5}n-\frac{1}{10}$ | $\frac{6}{5(M+2)}(n+1)-\frac{1}{2}$ |
| $\mathbb{E}[B]$ | $\frac{1}{5}n-\frac{1}{20}$ | $\frac{3}{5(M+2)}(n+1)-\frac{1}{4}$ |
| $\mathbb{E}[R]$ | $\frac{6}{5}n+\frac{7}{10}$ | $\frac{18}{5(M+2)}(n+1)-\frac{1}{2}$ |
| $\mathbb{E}[F]$ | $\frac{1}{10}n-\frac{1}{15}$ | $\frac{2}{5(M+2)}(n+1)-\frac{1}{6}$ |
| $\mathbb{E}[C^{(1)}]$ | $\frac{4}{5}(n+1)\mathcal{H}_n-\frac{83}{50}n-\frac{2}{75}$ | $\frac{4}{5}(n+1)\big(\mathcal{H}_{n+1}-\mathcal{H}_{M+2}\big)+\big(\frac{38}{75}-\frac{2}{M+2}\big)(n+1)+\frac{5}{6}$ |
| $\mathbb{E}[C^{(3)}]$ | $\frac{2}{5}(n+1)\mathcal{H}_n-\frac{22}{25}n+\frac{1}{50}$ | $\frac{2}{5}(n+1)\big(\mathcal{H}_{n+1}-\mathcal{H}_{M+2}\big)+\big(\frac{19}{75}-\frac{6}{5(M+2)}\big)(n+1)+\frac{1}{2}$ |
| $\mathbb{E}[C^{(4)}]$ | $\frac{1}{5}(n+1)\mathcal{H}_n-\frac{39}{100}n-\frac{7}{300}$ | $\frac{1}{5}(n+1)\big(\mathcal{H}_{n+1}-\mathcal{H}_{M+2}\big)+\big(\frac{19}{150}-\frac{2}{5(M+2)}\big)(n+1)+\frac{1}{6}$ |
| $\mathbb{E}[S^{(1)}]$ | $\frac{3}{10}(n+1)\mathcal{H}_n-\frac{127}{200}n-\frac{1}{600}$ | $\frac{3}{10}(n+1)\big(\mathcal{H}_{n+1}-\mathcal{H}_{M+2}\big)+\big(\frac{19}{100}-\frac{4}{5(M+2)}\big)(n+1)+\frac{1}{3}$ |
| $\mathbb{E}[S^{(3)}]$ | $\frac{1}{10}(n+1)\mathcal{H}_n-\frac{49}{200}n+\frac{13}{600}$ | $\frac{1}{10}(n+1)\big(\mathcal{H}_{n+1}-\mathcal{H}_{M+2}\big)+\big(\frac{19}{300}-\frac{2}{5(M+2)}\big)(n+1)+\frac{1}{6}$ |
| $\mathbb{E}[T_n]=an+b$, $\mathbb{E}[T_2]=0$ | $\frac{6}{5}a(n+1)(\mathcal{H}_n-\mathcal{H}_{M+2})+\big(\frac{19}{25}a-\frac{6}{5}\frac{a-b}{M+2}-\delta_{M1}\frac{2a+b}{10}\big)(n+1)+\frac{a-b}{2}+\mathrm{O}\big(\frac{1}{n^4}\big)$ | |

**Table 2:** Expected execution frequencies characterizing all block execution frequencies of Figure 1. Those immediately follow from Proposition 3.2 and the toll functions of Table 1. For $M=1$, we give exact expectations (valid for $n \geq 4$), for $M \geq 2$ we confine ourselves to (extremely precise) asymptotics. Note that exact values can be computed using equation (3.4) if needed.

2. In block 19, we move the small pivot to $\mathtt{A}[\ell]$, so $\ell = P$ must hold there. Just above the swap, $\ell$ is decremented, so the last value of $\ell$ in the loop has been $P+1$. Moreover, $\ell$ is initialized to 2 (block 6), so $\mathscr{L} = \{2,\dots,P+1\}$.

Together, this implies $T_{S^{(3)}} = P - 1 - (s@\mathscr{K})$ and by (3.12) and Lemma 3.5:

$$\mathbb{E}[T_{S^{(3)}}(n)] \;=\; \mathbb{E}[P] - 1 - \mathbb{E}[s@\mathscr{K}] \;=\; \tfrac{1}{3}(n+1) - 1 - \big(\tfrac{1}{4}n - \tfrac{5}{12}\big) \;=\; \tfrac{1}{12}n - \tfrac{1}{4}\,. \tag{3.16}$$

## 3.3. Key Comparisons

**Theorem 3.8.** *In expectation, Yaroslavskiy's algorithm (Algorithm 1) uses*

$$\mathbb{E}[C_n] \;=\; \begin{cases} \tfrac{19}{10}(n+1)\big(\mathcal{H}_{n+1}-\mathcal{H}_{M+2}\big) \\ \quad + \big(\tfrac{124}{75}+\tfrac{3}{20}M-\tfrac{9}{5(M+2)}-\tfrac{12}{5(M+2)}\mathcal{H}_{M+1}\big)(n+1)+\tfrac{3}{2}+\mathrm{O}\big(\tfrac{1}{n^4}\big), & \text{for } M \geq 2; \\ \tfrac{19}{10}(n+1)\mathcal{H}_n - \tfrac{711}{200}n - \tfrac{31}{200}\,, & \text{for } M=1,\ n \geq 4, \end{cases} \tag{3.17}$$

*key comparisons to sort a random permutation of size n.*

*Proof:* Key comparisons in the partitioning loop happen in basic blocks 3, 8, 10, 11 and 14. Together this amounts to

$$C_n^{(QS)} \;=\; C_n^{(1)} + (C_n^{(1)} - S_n^{(1)}) + C_n^{(3)} + C_n^{(4)} + A_n \quad \text{and, in expectation,}$$

$$\mathbb{E}[C_n^{(QS)}] \;=\; \begin{cases} \frac{19}{10}(n+1)\big(\mathscr{H}_{n+1} - \mathscr{H}_{M+2}\big) + \big(\frac{361}{300} - \frac{18}{5(M+2)}\big)(n+1) + \frac{3}{2} + \mathrm{O}\big(\frac{1}{n^4}\big), & \text{for } M \geq 2; \\ \frac{19}{10}(n+1)\mathscr{H}_n - \frac{711}{200}n - \frac{31}{200}, & \text{for } M = 1,\, n \geq 4, \end{cases}$$

comparisons, where the second equation follows by summing the results from Table 2.

For $M \geq 2$, we get additional comparisons from InsertionSort, see Appendix B for details:

$$\mathbb{E}[C_n^{(IS)}] \;=\; \mathbb{E}[E_n] + \mathbb{E}[D_n] \;=\; \big(\tfrac{3}{20}(M+3) + \tfrac{9}{5(M+2)} - \tfrac{12}{5(M+2)}\mathscr{H}_{M+1}\big)(n+1)\,.$$

Summing both contributions yields (3.17). □

## 3.4. Swaps & Write Accesses

**Theorem 3.9.** *In expectation, Yaroslavskiy's algorithm performs*

$$\mathbb{E}[S_n] \;=\; \begin{cases} \frac{3}{5}(n+1)\big(\mathscr{H}_{n+1} - \mathscr{H}_{M+2}\big) + \big(\frac{19}{50} + \frac{4}{5(M+2)}\big)(n+1) - \frac{1}{3} + \mathrm{O}\big(\frac{1}{n^4}\big), & \textit{for } M \geq 2; \\ \frac{3}{5}(n+1)\mathscr{H}_n - \frac{47}{100}n - \frac{61}{300}, & \textit{for } M = 1,\, n \geq 4, \end{cases} \tag{3.18}$$

swaps *in partitioning steps while sorting a random permutation of size n.*

*Including the ones done in* InsertionSort *on small subproblems, Yaroslavskiy's algorithm uses*

$$\mathbb{E}[W_n] \;=\; \begin{cases} \frac{11}{10}(n+1)\big(\mathscr{H}_{n+1} - \mathscr{H}_{M+2}\big) & \\ \quad + \big(\frac{86}{75} + \frac{3}{20}M + \frac{18}{5(M+1)} - \frac{26}{5(M+2)}\big)(n+1) - \frac{5}{6} + \mathrm{O}\big(\frac{1}{n^4}\big), & \textit{for } M \geq 2; \\ \frac{11}{10}(n+1)\mathscr{H}_n - \frac{139}{200}n - \frac{257}{600}, & \textit{for } M = 1,\, n \geq 4, \end{cases} \tag{3.19}$$

write accesses *to the array to sort a random permutation.*

*Proof:* We find swaps in the partitioning loop of Yaroslavskiy's algorithm in basic blocks 9, 15, 16 and 19, where blocks 16 and 19 each contain two swaps.[7] Hence, the total number of swaps during all partitioning steps is given by

$$S^{(QS)} \;=\; S^{(1)} + (C^{(4)} - S^{(3)}) + 2S^{(3)} + 2A\,.$$

Now, (3.18) follows by inserting the terms from Table 2.

A clever implementation realizes the two consecutive swaps in block 16 with only three write operations, see for example Appendix C. This yields an overall number of

$$W_n^{(QS)} \;=\; 2S_n^{(1)} + 2(C_n^{(4)} - S_n^{(3)}) + 3S_n^{(3)} + 4A_n \quad \text{and, in expectation,}$$

$$\mathbb{E}[W_n^{(QS)}] \;=\; \begin{cases} \frac{11}{10}(n+1)\big(\mathscr{H}_{n+1} - \mathscr{H}_{M+2}\big) + \big(\frac{209}{300} + \frac{2}{M+2}\big)(n+1) - \frac{5}{6} + \mathrm{O}\big(\frac{1}{n^4}\big), & \text{for } M \geq 2; \\ \frac{11}{10}(n+1)\mathscr{H}_n - \frac{139}{200}n - \frac{257}{600}, & \text{for } M = 1,\, n \geq 4, \end{cases}$$

write operations during the partitioning steps. The contribution from InsertionSort is (cf. Appendix B):

$$\mathbb{E}[W_n^{(IS)}] \;=\; \mathbb{E}[E_n] + (\mathbb{E}[G_n] - \mathbb{E}[I_n]) \;=\; \big(\tfrac{3}{20}(M+3) + \tfrac{18}{5(M+1)} - \tfrac{36}{5(M+2)}\big)(n+1)\,.$$

Adding both together we obtain (3.19). □

[7] Note that Wild and Nebel [2012] included an additional contribution of $B$ for swapping the pivots if they are out of order. However, this swap can be done with local variables only, we do not need to write the swapped values back to the array. Therefore, this swap is *not* counted in this paper.

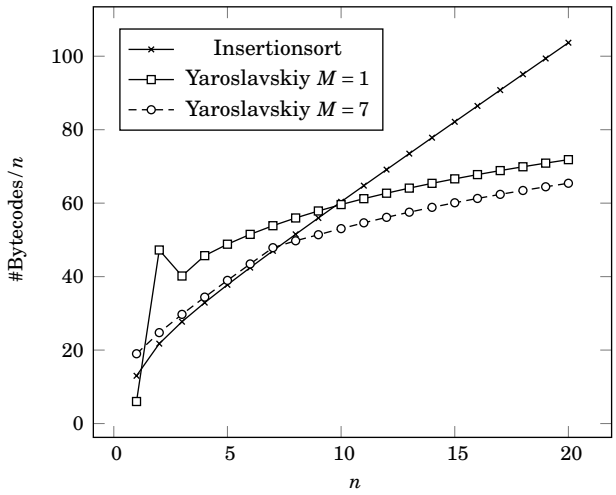



**Figure 2:** The expected number of executed Bytecodes for INSERTIONSORT and Yaroslavskiy's algorithm with different choices for $M$. The numbers of Bytecodes shown are normalized by $n$, i. e., we show the number of executed Bytecode instructions per element to be sorted. The data was obtained by naïvly evaluating the recurrence.

## 3.5. Executed Java Bytecode Instructions

**Theorem 3.10.** *In expectation, the Java implementation of Yaroslavskiy's algorithm given in Appendix C executes*

$$\mathbb{E}[BC_n] \;=\; \begin{cases} \frac{217}{10}(n+1)\big(\mathscr{H}_{n+1}-\mathscr{H}_{M+2}\big) \\ \quad +\big(\frac{4259}{150}+\frac{51}{20}M+\frac{72}{M+1}-\frac{317}{5(M+2)}-\frac{48}{5(M+2)}\mathscr{H}_{M+1}\big)(n+1)-\frac{181}{12}+\mathrm{O}\big(\frac{1}{n^4}\big), & M\geq 2; \\ \frac{217}{10}(n+1)\mathscr{H}_n-\frac{1993}{200}n-\frac{2009}{600}, & n\geq 4, M=1, \end{cases} \tag{3.20}$$

*Java Bytecode instructions to sort a random permutation of size $n$.*

*Proof:* By counting the number of Bytecode instructions in each basic block and multiplying it with this block's frequency, we obtain:

$$BC_n \;=\; 71A-1B+6R+15C^{(1)}+10C^{(3)}+11C^{(4)}+9S^{(1)}+8S^{(3)}+3F+4D+17E+20G-7I\,.$$

For details, see Appendix C. Inserting the expectations from Table 2 results in (3.20).

The Bytecode count for $M=1$ corresponds to entirely removing INSERTIONSORT from the code. INSERTIONSORT would execute 13 Bytecodes even on empty and one-element lists until it finds out that the list is already sorted. These obviously superfluous instructions are removed for $M=1$. □

### 3.6. Optimal Choice for $M$

The overall linear term for the expected number of comparisons is

$$\left(\tfrac{124}{75} + \tfrac{3}{20}M - \tfrac{37}{10(M+2)} - \tfrac{62+19M}{10(M+2)}\mathcal{H}_{M+1}\right)(n+1)\,.$$

This coefficient has a proper minimum at $M = 5$ with value $-3.62024\ldots$. Compared with $-711/200 = -3.555$ for $M = 1$ this a minor improvement though.

For the number of write operations, the linear term is

$$\left(\tfrac{86}{75} + \tfrac{3}{20}M - \tfrac{26}{5(M+2)} + \tfrac{18}{5(M+1)} - \tfrac{11}{10}\mathcal{H}_{M+2}\right)(n+1)\,.$$

The minimum $-1.098\overline{3}$ of this coefficient is also located at $M = 5$. The corresponding coefficient for $M = 1$ is $-0.695$. This improvement is more satisfying than the one for comparisons.

The linear term for the number of executed Bytecodes of Yaroslavskiy's algorithm with $M \geq 2$ attains its minimum $-16.0887\ldots$ at $M = 7$. This is a significant reduction over $-9.965\ldots$, the linear term without INSERTIONSORTing. Figure 2 shows the resulting expected number of Bytecodes for small lists. For $n \leq 20$, using INSERTIONSORT results in an improvement of over 10 %. For $n = 100$ we save 6.3 %, for $n = 1000$ it is 4.2 % and for $n = 10\,000$, we still execute 3.1 % less Bytecode instructions than the basic version of Yaroslavskiy's algorithm.

It is interesting to see that both elementary operations favor $M = 5$, but the overall Bytecode count is minimized for "much" larger $M = 7$. This shows that focusing on elementary operations can skew the view of an algorithm's performance. Only explicitly taking the overhead of partitioning into account reveals that INSERTIONSORT is significantly faster on small subproblems.

**Remark** The actual Java 7 runtime library implementation uses $M = 46$, which seems far from optimal at first sight. Note however that the implementation uses the more elaborate pivot selection scheme *tertiles of five* [Wild et al., 2013], which implies additional constant overhead per partitioning step.

## 4. Distribution of Costs

In this section we study the asymptotic *distributions* of our cost measures. We derive limit laws after normalization and identify the order of variances and covariances. In particular, we find that all costs are asymptotically concentrated around their mean.

As we confine ourselves to asymptotic statements of first order (leading terms in the expansions of variances and covariances), it turns out that the choice of $M$ does not affect the results of this section: All results hold for any (constant) $M$ (see [Neininger, 2001, proof of Corollary 5.5] for similar universal behavior of standard Quicksort). Appendix E shows that the asymptotic results are good approximations for practical input sizes $n$. A general survey on distributional analysis of various sorting algorithms covering many classical results is found in [Mahmoud, 2000].

### 4.1. The Contraction Method

Our tool to identify asymptotic variances, correlations and limit laws is the *contraction method*, which is applicable to many divide-and-conquer algorithms. Roughly speaking, the idea is to appropriately normalize a recurrence equation for the *distribution* of costs such that we can hope for convergence to a limit distribution. If we then replace all terms that depend on $n$ by their limits for $n \to \infty$, we obtain a *map* within the space of probability distributions that approximates the recurrence.

Next a (complete) metric between probability distributions is chosen such that this map becomes a contraction; then the *Banach fixed-point theorem* implies the existence of a unique fixed point for this map. This fixed point is the candidate for the limit distribution of the normalized costs and the

underlying contraction property is then exploited to also show convergence of the normalized costs towards the fixed point. This convergence is shown within the same complete metric. If the metric is sufficiently strong, it may imply more than convergence in law; in our case we additionally obtain convergence of the first two moments, i. e., convergence of mean and variance. This enables us to compute asymptotics for the variance of the cost as well. Note that a fixed-point representation for a limit distribution is implicit, but it is suitable to compute moments of the limit distribution and to identify further properties such as the existence of a (Lebesgue) density.

For the reader's convenience we formulate a general convergence theorem from the contraction method that is used repeatedly below and sufficient for our purpose. Let $(X_n)_{n\geq 0}$ denote a sequence of centered and square integrable random variables either in $\mathbb{R}$ or $\mathbb{R}^2$ whose distributions satisfy the recurrence

$$X_n \overset{\mathscr{D}}{=} \sum_{r=1}^{K} A_r^{(n)} X_{I_r^{(n)}}^{(r)} + b^{(n)}, \qquad n \geq n_0, \tag{4.1}$$

where the random variables $(A_1^{(n)},\dots,A_K^{(n)},b^{(n)},I^{(n)})$ and $(X_n^{(1)})_{n\geq 0},\dots,(X_n^{(K)})_{n\geq 0}$ are independent, and $X_i^{(r)}$ is distributed as $X_i$ for all $r=1,\dots,K$ and $i\geq 0$. Furthermore, $I^{(n)}=(I_1^{(n)},\dots,I_K^{(n)})$ is a vector of random integers in $\{0,\dots,n-1\}$ and $K$ and $n_0$ are fixed integers.

The coefficients $A_r^{(n)}$ and $b^{(n)}$ are real random variables in the univariate case, respectively random $2\times 2$ matrices and a 2-dimensional random vector in the bivariate case. We assume also that the coefficients are square integrable and that the following conditions hold:

(A) $(A_1^{(n)},\dots,A_K^{(n)},b^{(n)}) \xrightarrow{\ell_2} (A_1,\dots,A_k,b)$,

(B) $\sum_{r=1}^{K} \mathbb{E}\left[\|A_r^t A_r\|_{\mathrm{op}}\right] < 1$,

(C) $\sum_{r=1}^{K} \mathbb{E}\left[\mathbf{1}_{\{I_r^{(n)}\leq \ell\}} \|(A_r^{(n)})^t A_r^{(n)}\|_{\mathrm{op}}\right] \to 0$ as $n\to\infty$ for all constants $\ell\geq 0$.

Here $\|A\|_{\mathrm{op}} := \sup_{\|x\|=1}\|Ax\|$ denotes the *operator norm* of a matrix and $A^t$ the transposed matrix. Note that in the univariate case we just have $\|A_r^t A_r\|_{\mathrm{op}} = A_r^2$. In (A) we denote by $\xrightarrow{\ell_2}$ convergence in the Wasserstein-metric of order 2 which here is equivalent to the existence of vectors $(\tilde{A}_1^{(n)},\dots,\tilde{A}_K^{(n)},\tilde{b}^{(n)})$ with the distribution of $(A_1^{(n)},\dots,A_K^{(n)},b^{(n)})$ such that we have the $L_2$ convergence

$$(\tilde{A}_1^{(n)},\dots,\tilde{A}_K^{(n)},\tilde{b}^{(n)}) \xrightarrow{L_2} (A_1,\dots,A_k,b)\,.$$

Note in particular that $A_1,\dots,A_k,b$ are square integrable as well. Then we consider distributions of $X$ such that

$$X \overset{\mathscr{D}}{=} \sum_{r=1}^{K} A_r X^{(r)} + b\,, \tag{4.2}$$

where $(A_1,\dots,A_K,b)$, $X^{(1)},\dots,X^{(K)}$ are independent and $X^{(r)}$ are distributed as $X$ for $r=1,\dots,K$. The following two results from the contraction method are used:

(I) Under (B), among all centered, square integrable distributions there is a unique solution $\mathscr{L}(X)$ to (4.2).

(II) Assuming (A), (B) and (C), the sequence $(X_n)_{n\geq 0}$ converges in distribution to the solution $\mathscr{L}(X)$ from (I). The convergence holds as well for the second (mixed) moments of $X_n$.

These results are given by Rösler [2001, Theorem 3] for the univariate case and Neininger [2001, Theorem 4.1] for the multivariate case.

## 4.2. Distributional Analysis of Yaroslavskiy's Algorithm

We come back to Yaroslavskiy's algorithm (Algorithm 1). To apply the contraction method, we have to characterize the full distribution of costs $T_n$ in one partitioning step and then formulate a *distributional recurrence* for the resulting distribution of costs $C_n$ for the complete sorting. To obtain a contracting mapping, we rewrite the derived recurrence for $C_n$ in terms of suitably normalized costs $C_n^*$; here it will suffice to subtract the expected values computed in the last section and then to divide by $n$.

For the distributional analysis, it proves more convenient to consider i. i. d. uniformly on $[0,1]$ distributed random variables $U_1,\dots,U_n$ as input. Note that $U_1,\dots,U_n$ are thus pairwise different almost surely. As the actual element values do not matter, this input model is the same as considering a random permutation.

Yaroslavskiy's algorithm chooses $U_1$ and $U_n$ as pivot elements. Denote by $D=(D_1,D_2,D_3)$ the *spacings* induced by $U_1$ and $U_n$ on the interval $[0,1]$; formally we have

$$(D_1,D_2,D_3) \;=\; (U_{(1)}, U_{(2)}-U_{(1)}, 1-U_{(2)}),$$

for $U_{(1)} := \min\{U_1,U_n\}$ and $U_{(2)} := \max\{U_1,U_n\}$. It is well-known that $D$ is uniformly distributed in the standard 2-simplex [David and Nagaraja, 2003, p. 133f], i. e., $(D_1,D_2)$ has density

$$f_D(x_1,x_2) \;=\; \begin{cases} 2, & \text{for } x_1,x_2\ge 0 \,\wedge\, x_1+x_2\le 1;\\ 0, & \text{otherwise,}\end{cases}$$

and $D_3 = 1-D_1-D_2$ is fully determined by $(D_1,D_2)$. Hence, for a measurable function $g:[0,1]^3\to\mathbb{R}$ such that $g(D_1,D_2,D_3)$ is integrable, we have that

$$\mathbb{E}[g(D_1,D_2,D_3)] \;=\; 2\int_0^1\int_0^{1-x_1} g(x_1,x_2,1-x_1-x_2)\,dx_2\,dx_1\,. \tag{4.3}$$

Further we denote the sizes of the three subproblems generated in the first partitioning phase by $I^{(n)}=(I_1^{(n)},I_2^{(n)},I_3^{(n)})$. Then we have $I_1^{(n)}+I_2^{(n)}+I_3^{(n)}=n-2$. Moreover, the spacings $D_1$, $D_2$ and $D_3$ are exactly the probabilities for an element $U_i$ $(1<i<n)$ to be small, medium or large, respectively. As all these elements are independent, the vector $I^{(n)}$, conditional on $D$, has a multinomial distribution:

$$I^{(n)} \;\overset{\mathcal{D}}{=}\; \mathrm{M}(n-2;D_1,D_2,D_3).$$

We will use the short notation $I^{(n)}=I=(I_1,I_2,I_3)$ when the dependence on $n$ is obvious. From the strong law of large numbers and dominated convergence we have in particular for $r\in\{1,2,3\}$

$$\frac{I_r^{(n)}}{n} \;\xrightarrow{L_p}\; D_r \qquad (n\to\infty),\quad 1\le p<\infty\,. \tag{4.4}$$

The advantage of this random model is that we can *decouple* values from ranks of pivots. With $D_1=U_{(1)}$ and $D_1+D_2=U_{(2)}$, we choose the *values* of the two pivots; however, the ranks $P$ and $Q$ are not yet fixed. Therefore given fixed pivot values, we can still *independently* draw non-pivot elements (with probabilities $D_1$, $D_2$ and $D_3$ to become small, medium and large, resp.), without having to fuzz with a priori restrictions on the overall number of small, medium and large elements. This makes it much easier to compute cost contributions uniformly in pivot values than in pivot ranks. If we operate on random permutations of $\{1,\dots,n\}$, values and ranks coincide, so fixing pivot values there implies strict bounds on the number of small, medium and large elements.

### 4.2.1. Distribution of Toll Functions

In Section 3.2, we determined for each basic block of Yaroslavskiy's algorithm, how often it is executed in one partitioning step. There, we only used the expected values in the end, but we

| Quantity | | | Distribution given $I = (I_1, I_2, I_3)$ | |
|---|---|---|---|---|
| $P$ | = | $I_1 + 1$ | | |
| $Q$ | = | $n - I_3$ | | |
| $\delta$ | = | $\mathbb{1}_{\{\mathtt{A}[Q] > Q\}}$ | $\stackrel{\mathscr{D}}{=}$ | $\mathrm{B}\left(\frac{I_3}{n-2}\right)$ |
| $T_A$ | = | 1 | | |
| $T_B$ | = | $\mathbb{1}_{\{\mathtt{A}[left] > \mathtt{A}[right]\}}$ | $\stackrel{\mathscr{D}}{=}$ | $\mathrm{B}\left(\frac{1}{2}\right)$ |
| $T_F$ | = | $\delta$ | $\stackrel{\mathscr{D}}{=}$ | $\mathrm{B}\left(\frac{I_3}{n-2}\right)$ |
| $T_{C^{(1)}}$ | = | $Q - 2 + \delta$ | $\stackrel{\mathscr{D}}{=}$ | $I_1 + I_2 + \mathrm{B}\left(\frac{I_3}{n-2}\right)$ |
| $T_{C^{(3)}}$ | = | $n - Q$ | $\stackrel{\mathscr{D}}{=}$ | $I_3$ |
| $T_{C^{(4)}}$ | = | $\delta + (sm@\mathscr{G})$ | $\stackrel{\mathscr{D}}{=}$ | $\mathrm{B}\left(\frac{I_3}{n-2}\right) + \mathrm{HypG}(I_1 + I_2, I_3, n-2)$ |
| $T_{S^{(1)}}$ | = | $s@\mathscr{K}$ | $\stackrel{\mathscr{D}}{=}$ | $\mathrm{HypG}(I_1, I_1 + I_2, n-2)$ |
| $T_{S^{(3)}}$ | = | $P - 1 - (s@\mathscr{K})$ | $\stackrel{\mathscr{D}}{=}$ | $I_1 - \mathrm{HypG}(I_1, I_1 + I_2, n-2)$ |

**Table 3:** Exact distributions of the toll functions introduced in Section 3.2 or equivalently the distributions of block execution frequencies in the first partitioning step.

already characterized the full distributions in passing. They are summarized in Table 3 for reference.

Most of those distributions are in fact *mixed* distributions, i. e., their parameters depend on the random variable $I = (I_1, I_2, I_3)$, namely the sizes of the subproblems for recursive calls. For example, we find that $\delta$ conditional on the event $(I_1, I_2, I_3) = (i_1, i_2, i_3)$ is Bernoulli $\mathrm{B}(i_3/(n-2))$ distributed, which we briefly write as $\delta \stackrel{\mathscr{D}}{=} \mathrm{B}(I_3/(n-2))$. Note that since $I$ has itself a mixed distribution — namely conditional on $D$ — we actually have three layers of random variables: spacings, subproblem sizes and toll functions. The key technical lemmas for dealing with these three-layered distributions are given in Section 4.3.

### 4.2.2. Distributional Recurrence

Denote by $T_n$ the (random) costs of the first partitioning step of Yaroslavskiy's algorithm. By Property 2.1, subproblems generated in the first partitioning phase are, conditional on their sizes, again uniformly random permutations and independent of each other. Hence, we obtain the distributional recurrence for the (random) total costs $C_n$:

$$C_n \stackrel{\mathscr{D}}{=} C'_{I_1} + C''_{I_2} + C'''_{I_3} + T_n, \qquad (n \geq 3), \tag{4.5}$$

where $(I_1, I_2, I_3, T_n)$, $(C'_j)_{j\geq 0}$, $(C''_j)_{j\geq 0}$, $(C'''_j)_{j\geq 0}$ are independent and $C'_j$, $C''_j$, $C'''_j$ are identically distributed as $C_j$ for $j \geq 0$. By Theorems 3.8, 3.9 and 3.10, we know the expected costs $\mathbb{E}[C_n]$, so with $C^*_0 := 0$ and

$$C^*_n := \frac{C_n - \mathbb{E}[C_n]}{n}, \qquad \text{for } n \geq 1, \tag{4.6}$$

we have a sequence $(C^*_n)_{n\geq 0}$ of centered, square integrable random variables. Using (4.5) we find, cf. [Hwang and Neininger, 2002, eq. (27), (28)], that $(C^*_n)_{n\geq 0}$ satisfies (4.1) with

$$A^{(n)}_r = \frac{I_r}{n}, \qquad b^{(n)} = \frac{1}{n}\Big(T_n - \mathbb{E}[C_n] + \sum_{r=1}^{3} \mathbb{E}[C_{I_r} \mid I_r]\Big), \tag{4.7}$$

so we can apply the framework of the contraction method. It remains to check the conditions (A), (B) and (C) to prove that $C^*_n$ indeed converges to a limit law; the detailed computations are given in Appendix D. The key results needed therein are presented in the following section as technical lemmas.

## 4.3. Asymptotics of Mixed Distributions

The following convergence results for mixed distributions are essential for proving condition (A).

**Lemma 4.1.** *Let $(V_1,\dots,V_b)$ be a vector of random probabilities, i. e., $0 \le V_r \le 1$ for all $r = 1,\dots,b$ and $\sum_{r=1}^{b} V_r = 1$ almost surely. Let*

$$(L_1,\dots,L_b) \;:=\; (L_1^{(n)},\dots,L_b^{(n)}) \;\stackrel{\mathscr{D}}{=}\; \mathrm{M}(n;V_1,\dots,V_b), \tag{4.8}$$

*be mixed multinomially distributed. Furthermore for $J_1, J_2 \subset \{1,\dots,b\}$ let*

$$Z_n \;\stackrel{\mathscr{D}}{=}\; \mathrm{HypG}\Big(\sum_{j\in J_1} L_j,\; \sum_{j\in J_2} L_j,\; n\Big) \tag{4.9}$$

*be mixed hypergeometrically distributed. Then we have the $L_2$-convergence, as $n \to \infty$,*

$$\frac{Z_n}{n} \;\xrightarrow{L_2}\; \Big(\sum_{j\in J_1} V_j\Big)\cdot\Big(\sum_{j\in J_2} V_j\Big). \tag{4.10}$$

The proof exploits that the binomial and the hypergeometric distributions are both strongly concentrated around their means. The full-detail computations to lift this to conditional expectations are given in Appendix D.

**Lemma 4.2.** *For $L_1,\dots,L_b$ from Lemma 4.1, we have for $1 \le i \le b$ the $L_2$-convergence*

$$\tfrac{L_i}{n}\ln\big(\tfrac{L_i}{n}\big) \;\xrightarrow{L_2}\; V_i\ln(V_i), \qquad \text{as } n\to\infty. \tag{4.11}$$

(Recall that we set $x\ln(x) := 0$ for $x = 0$.)
The proof is directly obtained by combining the *law of large numbers* with the *dominated convergence theorem*; see Appendix D for details.

## 4.4. Key Comparisons

We have the following asymptotic results on the variance and distribution of the number of key comparisons of Yaroslavskiy's algorithm:

**Theorem 4.3.** *For the number $C_n$ of key comparisons used by Yaroslavskiy's Quicksort when operating on a uniformly at random distributed permutation we have*

$$\frac{C_n - \mathbb{E}[C_n]}{n} \;\to\; C^*, \qquad (n\to\infty), \tag{4.12}$$

*where the convergence is in distribution and with second moments. The distribution of $C^*$ is determined as the unique fixed point, subject to $\mathbb{E}[X] = 0$ and $\mathbb{E}[X^2] < \infty$, of*

$$X \;\stackrel{\mathscr{D}}{=}\; 1 + (D_1+D_2)(D_2+2D_3) + \sum_{j=1}^{3}\big(D_jX^{(j)} + \tfrac{19}{10}D_j\ln D_j\big), \tag{4.13}$$

*where $(D_1,D_2,D_3)$, $X^{(1)}$, $X^{(2)}$ and $X^{(3)}$ are independent and $X^{(j)}$ has the same distribution as $X$ for $j \in \{1,2,3\}$. Moreover, we have, as $n\to\infty$,*

$$\mathrm{Var}(C_n) \;\sim\; \sigma_C^2 n^2 \qquad \text{with} \qquad \sigma_C^2 \;=\; \tfrac{2231}{360} - \tfrac{361}{600}\pi^2 = 0.25901\ldots \tag{4.14}$$

For the proof, we apply the contraction method to $X_n = C_n^*$ as defined by (4.6), where the toll function $T_C(n)$ is used. Details on checking conditions (A), (B) and (C), as well as the derivation of the resulting fixed-point equation for $C^*$ and the computation of the variance are given in Appendix D.

## 4.5. Swaps

For the number of swaps in Yaroslavskiy's algorithm we have the following asymptotic behavior of variance and distribution.

**Theorem 4.4.** *For the number $S_n$ of swaps used by Yaroslavskiy's algorithm when operating on a random permutation we have*

$$\frac{S_n - \mathbb{E}[S_n]}{n} \;\to\; S^*, \qquad (n \to \infty), \tag{4.15}$$

*where the convergence is in distribution and with second moments. The distribution of $S^*$ is determined as the unique fixed point, subject to $\mathbb{E}[X] = 0$ and $\mathbb{E}[X^2] < \infty$, of*

$$X \;\stackrel{\mathcal{D}}{=}\; D_1 + (D_1 + D_2)D_3 + \sum_{j=1}^{3} \left(D_j X^{(j)} + \tfrac{3}{5} D_j \ln D_j\right), \tag{4.16}$$

*where $(D_1, D_2, D_3)$, $X^{(1)}$, $X^{(2)}$ and $X^{(3)}$ are independent and $X^{(j)}$ has the same distribution as $X$ for $j \in \{1,2,3\}$. Moreover, we have, as $n \to \infty$,*

$$\operatorname{Var}(S_n) \;\sim\; \sigma_S^2 n^2, \qquad \text{with} \qquad \sigma_S^2 \;=\; \tfrac{7}{10} - \tfrac{3}{50}\pi^2 \;=\; 0.10782\ldots \tag{4.17}$$

The proof is similar to the one for Theorem 4.3, details are given in Appendix D.

## 4.6. Executed Bytecode Instructions

For the number of executed Java Bytecode instructions in Yaroslavskiy's algorithm we have the following asymptotic variance and distribution.

**Theorem 4.5.** *For the number $BC_n$ of executed Java Bytecodes used by Yaroslavskiy's algorithm when sorting a random permutation, we have*

$$\frac{BC_n - \mathbb{E}[BC_n]}{n} \;\to\; BC^*, \qquad (n \to \infty), \tag{4.18}$$

*where the convergence is in distribution and with second moments. The distribution of $BC^*$ is determined as the unique fixed point, subject to $\mathbb{E}[X] = 0$ and $\mathbb{E}[X^2] < \infty$, of*

$$X \;\stackrel{\mathcal{D}}{=}\; 24 + (D_3 - 9)D_2 - 2D_3(5D_3 + 2) + \sum_{j=1}^{3} \left(D_j X^{(j)} + \tfrac{217}{10} D_j \ln D_j\right), \tag{4.19}$$

*where $(D_1, D_2, D_3)$, $X^{(1)}$, $X^{(2)}$ and $X^{(3)}$ are independent and $X^{(j)}$ has the same distribution as $X$ for $j \in \{1,2,3\}$. Moreover, we have, as $n \to \infty$,*

$$\operatorname{Var}(BC_n) \;\sim\; \sigma_{BC}^2 n^2, \qquad \text{with} \qquad \sigma_{BC}^2 \;=\; \tfrac{1469983}{1800} - \tfrac{47089}{600}\pi^2 \;=\; 42.0742\ldots \tag{4.20}$$

Again, the proof is similar to the one for Theorem 4.3 and details are given in Appendix D.

## 4.7. Covariance of Comparisons and Swaps

In this section we study the asymptotic *covariance* $\operatorname{Cov}(C_n, S_n)$ between the number of key comparisons $C_n$ and the number of swaps $S_n$ in Yaroslavskiy's algorithm.

**Theorem 4.6.** *For the number $C_n$ of key comparisons and the number $S_n$ of swaps used by Yaroslavskiy's algorithm on a random permutation, we have for $n \to \infty$*

$$\operatorname{Cov}(C_n, S_n) \;\sim\; \sigma_{C,S}\, n^2 \qquad \text{with} \qquad \sigma_{C,S} \;=\; \tfrac{28}{15} - \tfrac{19}{100}\pi^2 \;=\; -0.00855817\ldots \tag{4.21}$$

*The correlation coefficient of $C_n$ and $S_n$ consequently is*

$$\frac{\operatorname{Cov}(C_n, S_n)}{\sqrt{\operatorname{Var}(C_n)}\sqrt{\operatorname{Var}(S_n)}} \;\sim\; \rho \;\approx\; -0.0512112\ldots$$

For the proof of Theorem 4.6, we consider the bivariate random variables $Y_n := \binom{C_n}{S_n}$ and show that their normalized versions $Y_n^* := \frac{1}{n}\big(Y_n - \mathbb{E}[Y_n]\big)$ converge to the (bivariate) distribution $\mathscr{L}(\Lambda_1, \Lambda_2)$, which is, as before, characterized as the unique fixed point of a distributional equation. The covariance of $C_n$ and $S_n$ is then obtained by multiplying both components of $Y_n^*$, which converges to $\mathbb{E}[\Lambda_1 \cdot \Lambda_2]$. Full computations are found in Appendix D.

Note that Theorem 4.6 and its proof directly imply the asymptotic variance and limit distribution of *all* linear combinations $\alpha C_n + \beta S_n$, for $\alpha, \beta \in \mathbb{R}$, which are, for $\alpha, \beta > 0$, natural cost measures when weighting key comparisons against swaps. The reason is that in the proof of Theorem 4.6 we show the bivariate limit law

$$\left(\frac{C_n - \mathbb{E}[C_n]}{n}, \frac{S_n - \mathbb{E}[S_n]}{n}\right) \;\to\; (\Lambda_1, \Lambda_2),$$

which holds in distribution and with second mixed moments. Hence, the *continuous mapping theorem* implies, as $n \to \infty$,

$$\frac{\alpha C_n + \beta S_n \;-\; (\alpha\,\mathbb{E}[C_n] + \beta\,\mathbb{E}[S_n])}{n} \;\to\; \alpha\Lambda_1 + \beta\Lambda_2,$$

in distribution and with second moments. Thus, we obtain, as $n \to \infty$,

$$\begin{aligned} \operatorname{Var}(\alpha C_n + \beta S_n) &= \alpha^2 \operatorname{Var}(C_n) + \beta^2 \operatorname{Var}(S_n) + 2\alpha\beta \operatorname{Cov}(C_n, S_n) \\ &\sim (\alpha^2\sigma_C^2 + \beta^2\sigma_S^2 + 2\alpha\beta\sigma_{C,S})n^2. \end{aligned}$$

Note that by this approach also the covariances between all the single contributions from Table 3 that contribute with linear order in the first partitioning step to the number of executed Java Bytecodes used by Yaroslavskiy's algorithm can be identified asymptotically in first order.

# 5. Conclusion

In this paper, we conducted a fully detailed analysis of Yaroslavskiy's dual-pivot Quicksort — including the optimization of using Insertionsort on small subproblems — in the style of Knuth's book series *The Art of Computer Programming*. We give the exact expected number of executed Java Bytecode instructions for Yaroslavskiy's algorithm.[8] On top of the exact average case results, we establish existence and fixed-point characterizations of limiting distribution of normalized costs. From this, we compute moments of the limiting distributions, in particular the asymptotic variance of the number of executed Bytecodes. The mere fact that such a detailed average and distributional analysis is tractable, seems worth noting. For the reader's convenience, we summarize the main results of this paper in Table 4, where we also cite corresponding results on classic single-pivot Quicksort for comparison.

As observed by Wild and Nebel [2012], Yaroslavskiy's algorithm uses 5 % less key comparisons, but 80 % more swaps in the asymptotic average than classic Quicksort. Unless comparisons are very expensive, one should expect classic Quicksort to be more efficient in total. This intuition is confirmed by our detailed analysis: In the asymptotic average, the Java implementation of Yaroslavskiy's algorithm executes 20 % more Java Bytecode instructions than a corresponding implementation of classic Quicksort.

Strengthening confidence in expectations, we find that asymptotic standard deviations of all costs remain linear in $n$; by *Chebyshev's inequality*, this implies concentration around the mean. Whereas the number of comparisons in Yaroslavskiy's algorithm shows slightly less variance than for classic Quicksort, swaps exhibit converse behavior. In fact, the number of swaps in classic Quicksort is highly concentrated because it already achieves close to optimal average behavior: In

[8] Bytecode instructions serve merely as a sample of one possible detailed cost measure; implementations in different low level languages can easily be analyzed using the our block execution frequencies.

| Cost Measure | | error | Yaroslavskiy's Quicksort with $M = 7$ | Classic Quicksort with $M = 6$ |
|---|---|---|---|---|
| Comparisons | expectation | $O(\log n)$ | $1.9n \ln n - 2.49976n$ | $2n \ln n - 2.3045n$ * |
| | std. dev. | $o(n)$ | $0.50893n$ | $0.648278n$ † |
| Swaps (for $M = 1$) | expectation | $O(\log n)$ | $0.6n \ln n - 0.107004n$ | $0.\overline{3}n \ln n - 0.585373n$ * |
| | std. dev. | $o(n)$ | $0.328365n$ | $0.0237251n$ † |
| Writes Accesses | expectation | $O(\log n)$ | $1.1n \ln n - 0.408039n$ | $0.\overline{6}n \ln n + 0.316953n$ * |
| Executed Bytecodes | expectation | $O(\log n)$ | $21.7n \ln n - 3.56319n$ | $18n \ln n + 6.21488n$ ‡ |
| | std. dev. | $o(n)$ | $6.48646n$ | $3.52723n$ § |
| Correlation Coefficient for Comparisons and Swaps | | $o(1)$ | $-0.0512112$ | $-0.86404$ $ |

* see [Sedgewick, 1977, p. 334].
† see [Hennequin, 1989, p. 330].
‡ see [Wild, 2012, p. 123].
§ as in [Neininger, 2001, p. 515] for MIX, but with Bytecode costs of [Wild, 2012].
$ see [Neininger, 2001, Table 1].

**Table 4:** Summary of the results of this paper and comparison with corresponding results for classic Quicksort. $M$ is chosen such that the number of executed Bytecodes in minimized. (For swaps, results for $M = 1$ are given as Insertionsort is not swap-based.) Some asymptotic approximations have been weakened for conciseness of presentation, see the main text for full precision. The asymptotics use the well-known expansion of Harmonic Numbers [e. g. Graham et al. 1994, eq. (6.66)]

the partitioning step of classic Quicksort, every swap puts *both* elements into the correct partition and we never revoke a placement during one partitioning step. In contrast, in Yaroslavskiy's algorithm every swap puts only one element into its final location (for the current partitioning step); the other element might have to be moved a second time later.

Another facet of this difference is revealed by considering the correlation coefficient between swaps and comparisons. In classic Quicksort, swaps and comparisons are almost perfectly *negatively* correlated. A "good" run w. r. t. comparisons needs balanced partitioning, but the more balanced partitioning becomes, the higher is the potential for misplaced elements that need to be moved. In Yaroslavskiy's partitioning method, such a clear dependency does not exist for several reasons. First of all, even if pivots have extreme ranks, sometimes many swaps are done; e. g. if $p$ and $q$ are the two largest elements, *all* elements are swapped in our implementation. Secondly, for some pivot ranks, comparisons and swaps behave covariantly: For example if $p$ and $q$ are the two smallest elements, no swap is done and every element's partition is found with one comparison only. In the end, the number of comparisons and swaps is almost uncorrelated in Yaroslavskiy's algorithm.

The asymptotic standard deviation of the total number of executed Bytecode instructions is about twice as large in Yaroslavskiy's algorithm as in classic Quicksort. This might be a consequence of the higher variability in the number of swaps just described.

Concerning practical performance, asymptotic behavior is not the full story. Often, inputs in practice are of moderate size and only the massive number of calls to a procedure makes it a bottleneck of overall execution. Then, lower order terms are not negligible. For Quicksort, this means in particular that constant overhead per partitioning step has to be taken into account. For tiny $n$, this overhead turns out to be so large, that it pays to switch to a simpler sorting method instead of Quicksort. We showed that using INSERTIONSORT for subproblems of size at most $M$ speeds up Yaroslavskiy's algorithm significantly for moderate $n$. The optimal choice for $M$ w. r. t. the number of executed Bytecodes is $M = 7$.

Combining the results for INSERTIONSORT from Appendix B and a corresponding Bytecode count analysis of a Java implementation of classic Quicksort [Wild, 2012], we can compare classic

Quicksort and Yaroslavskiy's algorithm exactly. As striking result we observe that in expectation, Yaroslavskiy's algorithm needs more Java Bytecodes than classic Quicksort *for all* $n$. Thus, the efficiency of classic Quicksort in terms of executed Bytecodes is not just an effect of asymptotic approximations, it holds for realistic input sizes, as well.

These findings clearly contradict corresponding running time experiments [Wild, 2012, Chapter 8], where Yaroslavskiy's algorithm was significantly faster across implementations and programming environments. One might object that the poor performance of Yaroslavskiy's algorithm is a peculiarity of counting Bytecode instructions. Wild [2012, Section 7.1] also gives implementations and analyses thereof in MMIX, the new version of Knuth's imaginary processor architecture. Every MMIX instruction has well-defined costs, chosen to closely resemble actual execution time on a simple processor. The results show the same trend: Classic Quicksort is more efficient. Together with the Bytecode results of this paper, we see strong evidence for the following conjecture:

**Conjecture 5.1.** *The efficiency of Yaroslavskiy's algorithm in practice is caused by advanced features of modern processors. In models that assign constant cost contributions to single instructions — i. e., locality of memory accesses and instruction pipelining are ignored — classic Quicksort is more efficient.*

It will be the subject of future investigations[9] to identify the true reason of the success of Yaroslavskiy's dual-pivot Quicksort.

[9] Indeed, progress has been made since this article was submitted. Kushagra et al. [2014] analyzed Yaroslavskiy's algorithm in the *external memory model* and show that it needs significantly less I/Os than classic Quicksort. Their results indicate that with modern memory hierarchies, using even more pivots in Quicksort might be beneficial, since intuitively speaking, more work is done in one "scan" of the input.

# APPENDIX

## A. Solving the Dual-Pivot Quicksort Recurrence

The proof presented in the following is basically a generalization of the derivation given by Sedgewick [1975, p. 156ff]. Hennequin [1991] gives an alternative approach based on generating functions that is much more general. Even though the authors consider Hennequin's method elegant, we prefer the elementary proof, as it allows a self-contained presentation.

Two basic identities involving binomials and Harmonic Numbers are used several times below, so we collect them here. They are found as equations (6.70) and (5.10) in [Graham et al., 1994].

$$\sum_{0\le k<n} \binom{k}{m}\mathcal{H}_k \;=\; \binom{n}{m+1}\Big(\mathcal{H}_n - \tfrac{1}{m+1}\Big), \qquad \text{integer } m \ge 0, \tag{A.1}$$

$$\sum_{0\le k\le n} \binom{k}{m} \;=\; \binom{n+1}{m+1}, \qquad \text{integers } n, m \ge 0. \tag{A.2}$$

*Proof of Theorem 3.1:* The first step is to use symmetries of the sum in (3.2).

$$\begin{aligned}\sum_{1\le p<q\le n} \big(C_{p-1} + C_{q-p-1} + C_{n-q}\big) \;&=\; \sum_{p=1}^{n-1}(n-p)C_{p-1} \;+\; \sum_{k=0}^{n-2}(n-1-k)C_k \;+\; \sum_{q=2}^{n}(q-1)C_{n-q}\\ &=\; 3\sum_{k=0}^{n-2}(n-k-1)C_k\,.\end{aligned}$$

So, our recurrence to solve is

$$C_n \;=\; T_n + \tfrac{6}{n(n-1)}\sum_{k=0}^{n-2}(n-k-1)C_k, \qquad \text{for } n > M. \tag{A.3}$$

We first consider $D_n := \binom{n+1}{2}C_{n+1} - \binom{n}{2}C_n$ to get rid of the factor in front of the sum:

$$\begin{aligned}D_n \;&=\; \overbrace{\tbinom{n+1}{2}T_{n+1} - \tbinom{n}{2}T_n}^{d(n):=} \qquad (n \ge M+2)\\ &\quad + \tfrac{(n+1)n}{2}\tfrac{6}{(n+1)n}\sum_{k=0}^{n-1}(n-k)C_k \;-\; \tfrac{n(n-1)}{2}\tfrac{6}{n(n-1)}\sum_{k=0}^{n-2}(n-k-1)C_k\\ &=\; d(n) \;+\; 3\sum_{k=0}^{n-1}C_k\,.\end{aligned}$$

The remaining full history recurrence is eliminated by taking ordinary differences

$$E_n \;:=\; D_{n+1} - D_n \;=\; d(n+1) - d(n) + 3C_n\,. \qquad (n \ge M+2)$$

Towards a telescoping recurrence, we consider $F_n := C_n - \frac{n-4}{n}\cdot C_{n-1}$, and compute

$$\begin{aligned}F_{n+2} - F_{n+1} \;&=\; C_{n+2} - \tfrac{n-2}{n+2}C_{n+1} - \big(C_{n+1} - \tfrac{n-3}{n+1}C_n\big)\\ &=\; C_{n+2} - \tfrac{2n}{n+2}C_{n+1} + \tfrac{n-3}{n+1}C_n\,.\end{aligned} \tag{A.4}$$

The expression on the right hand side in itself is not helpful. However, by expanding the definition of $E_n$, we find

$$\begin{aligned}(E_n - 3C_n)\,/\,\tbinom{n+2}{2} \;&=\; (D_{n+1} - D_n - 3C_n)\,/\,\tbinom{n+2}{2}\\ &=\; \Big(\tbinom{n+2}{2}C_{n+2} - \tbinom{n+1}{2}C_{n+1} - \Big(\tbinom{n+1}{2}C_{n+1} - \tbinom{n}{2}C_n\Big) - 3C_n\Big)\Big/\tbinom{n+2}{2}\\ &=\; C_{n+2} - \tfrac{2n}{n+2}C_{n+1} + \tfrac{\frac{1}{2}n(n-1)-3}{\frac{1}{2}(n+2)(n-1)}C_n\\ &=\; C_{n+2} - \tfrac{2n}{n+2}C_{n+1} + \tfrac{\frac{1}{2}(n-3)(n+2)}{\frac{1}{2}(n+2)(n-1)}C_n\\ &=\; C_{n+2} - \tfrac{2n}{n+2}C_{n+1} + \tfrac{n-3}{n+1}C_n\,.\end{aligned} \tag{A.5}$$

Equating (A.4) and (A.5) yields

$$F_{n+2} - F_{n+1} = (E_n - 3C_n) \,/\, \tbinom{n+2}{2} = \underbrace{\big(d(n+1) - d(n)\big) \,/\, \tbinom{n+2}{2}}_{f(n):=} . \quad (n \geq M+2)$$

This last equation is now amenable to simple iteration:

$$F_n = \underbrace{\sum_{i=M+4}^{n} f(i-2) + F_{M+3}}_{g(n)} . \qquad (n \geq M+4)$$

Plugging in the definition of $F_n = C_n - \frac{n-4}{n} \cdot C_{n-1}$ yields

$$C_n = \tfrac{n-4}{n} \cdot C_{n-1} + g(n) . \qquad (n \geq M+4) \tag{A.6}$$

Multiplying (A.6) by $\binom{n}{4}$ and using $\binom{n}{4} \cdot \frac{n-4}{n} = \binom{n-1}{4}$ gives a telescoping recurrence:

$$\begin{aligned} G_n := \tbinom{n}{4} C_n &= G_{n-1} + \tbinom{n}{4} g(n) \\ &= \sum_{i=M+4}^{n} \tbinom{i}{4} g(i) + G_{M+3} \\ &= \sum_{i=1}^{n} \tbinom{i}{4} g(i) + G_{M+3} - \sum_{i=1}^{M+3} \tbinom{i}{4} \Big( \overbrace{\sum_{j=M+4}^{i} f(j-2)}^{=0} + F_{M+3} \Big) \\ &= \sum_{i=1}^{n} \tbinom{i}{4} g(i) + G_{M+3} - \tbinom{M+4}{5} F_{M+3} , \end{aligned} \tag{A.7}$$

where the last equation uses (A.2). Applying definitions, we find

$$\begin{aligned} \sum_{i=1}^{n} \tbinom{i}{4} g(i) &= \sum_{i=1}^{n} \tbinom{i}{4} \Big( F_{M+3} + \sum_{j=M+2}^{i-2} \frac{d(j+1) - d(j)}{\binom{j+2}{2}} \Big) \\ &= \tbinom{n+1}{5} F_{M+3} + \sum_{i=M+4}^{n} \tbinom{i}{4} \sum_{j=M+2}^{i-2} \Big( T_{j+2} - \tfrac{2j}{j+2} T_{j+1} + \frac{\binom{j}{2}}{\binom{j+2}{2}} T_j \Big) . \end{aligned} \tag{A.8}$$

Using (A.8) in (A.7), we finally arrive at the explicit formula for $C_n$ valid for $n \geq M+3$:

$$\begin{aligned} C_n = {} & \frac{\binom{n+1}{5}}{\binom{n}{4}} F_{M+3} + \frac{1}{\binom{n}{4}} \sum_{i=M+4}^{n} \tbinom{i}{4} \sum_{j=M+2}^{i-2} \Big( T_{j+2} - \tfrac{2j}{j+2} T_{j+1} + \frac{\binom{j}{2}}{\binom{j+2}{2}} T_j \Big) \\ & + \frac{G_{M+3} - \binom{M+4}{5} F_{M+3}}{\binom{n}{4}} . \end{aligned}$$

Expanding $F$ and $G$ according to their definition gives

$$\begin{aligned} = {} & \frac{1}{\binom{n}{4}} \sum_{i=M+4}^{n} \tbinom{i}{4} \sum_{j=M+2}^{i-2} \Big( T_{j+2} - \tfrac{2j}{j+2} T_{j+1} + \frac{\binom{j}{2}}{\binom{j+2}{2}} T_j \Big) \\ & + \Big( \tfrac{n+1}{5} + \frac{\binom{M+3}{4} - \binom{M+4}{5}}{\binom{n}{4}} \Big) C_{M+3} - \tfrac{M-1}{M+3} \Big( \tfrac{n+1}{5} - \frac{\binom{M+4}{5}}{\binom{n}{4}} \Big) C_{M+2} , \end{aligned}$$

which concludes the proof. □

*Proof of Proposition 3.2:* Of course, we start with the closed form (3.3) from Theorem 3.1, which consists of the double sum and two terms involving "base cases" $C_{M+2}$ and $C_{M+3}$.

$$C_n \;=\; \frac{1}{\binom{n}{4}} \sum_{i=M+4}^{n} \binom{i}{4} \sum_{j=M+2}^{i-2} \overbrace{\Big(T_{j+2} - \tfrac{2j}{j+2} T_{j+1} + \frac{\binom{j}{2}}{\binom{j+2}{2}} T_j\Big)}^{\eta_j} \\ + \underbrace{\Big(\tfrac{n+1}{5} + \frac{\binom{M+3}{4} - \binom{M+4}{5}}{\binom{n}{4}}\Big) C_{M+3} \;-\; \tfrac{M-1}{M+3}\Big(\tfrac{n+1}{5} - \frac{\binom{M+4}{5}}{\binom{n}{4}}\Big) C_{M+2}}_{\rho}$$

We first focus on the sums. Assuming the even more general form

$$T_n \;=\; a\,n + b + \tfrac{c_1}{n-1} + \tfrac{c_2}{n} + \tfrac{c_3}{n(n-1)}\,,$$

partial fraction decomposition of the innermost term yields

$$\eta_j \;:=\; \big(T_{j+2} - \tfrac{2j}{j+2}\,T_{j+1} + \binom{j}{2}\big/\binom{j+2}{2} T_j\big) \;=\; \tfrac{8a-2b}{j+2} - \tfrac{2a-2b}{j+1}\,.$$

Note that contributions from $\frac{c_1}{n-1}$, $\frac{c_2}{n}$ and $\frac{c_3}{n(n-1)}$ cancel out. This allows to write the inner sum in terms of Harmonic Numbers:

$$\begin{aligned} \sum_{j=M+2}^{i-2} \eta_j \;&=\; (8a-2b)(\mathcal{H}_i - \mathcal{H}_{M+3}) - (2a-2b)(\mathcal{H}_{i-1} - \mathcal{H}_{M+2}) \\ &=\; 6a(\mathcal{H}_i - \mathcal{H}_{M+3}) + (2a-2b)(\tfrac{1}{i} - \tfrac{1}{M+3})\,. \end{aligned} \tag{A.9}$$

(The second equation uses the basic fact $\mathcal{H}_{k-1} = \mathcal{H}_k - \frac{1}{k}$.)
Using (A.1), (A.2) and the absorption property of binomials $\binom{n}{k} = \binom{n-1}{k-1}\frac{n}{k}$, one obtains

$$\begin{aligned} \frac{1}{\binom{n}{4}} \sum_{i=M+4}^{n} \binom{i}{4} \sum_{j=M+2}^{i-2} \eta_j \;&=\; \frac{6a}{\binom{n}{4}}\Big(\binom{n+1}{5}(\mathcal{H}_{n+1} - \tfrac{1}{5}) - \binom{M+4}{5}(\mathcal{H}_{M+4} - \tfrac{1}{5})\Big) \\ &\quad + \frac{2a-2b}{\binom{n}{4}} \sum_{i=M+3}^{n-1} \Big(\tfrac{1}{4}\binom{i-3}{3} - \tfrac{1}{M+3}\binom{i}{4}\Big) \\ &\quad - \frac{6a}{\binom{n}{4}} \mathcal{H}_{M+3}\Big(\binom{n+1}{5} - \binom{M+4}{5}\Big) \\ &=\; \tfrac{6}{5}a(n+1)(\mathcal{H}_{n+1} - \tfrac{1}{5}) - 6a\frac{\binom{M+4}{5}}{\binom{n}{4}}(\mathcal{H}_{M+4} - \tfrac{1}{5}) \\ &\quad + \frac{2a-2b}{\binom{n}{4}}\Big(\tfrac{1}{4}\big(\binom{n}{4} - \binom{M+3}{4}\big) - \tfrac{1}{M+3}\big(\binom{n+1}{5} - \binom{M+4}{5}\big)\Big) \\ &\quad - 6a\,\mathcal{H}_{M+3}\Big(\tfrac{n+1}{5} - \binom{M+4}{5}\big/\binom{n}{4}\Big) \\ &=\; \tfrac{6}{5}a(n+1)\mathcal{H}_{n+1} - \tfrac{n+1}{5}\big(6a\mathcal{H}_{M+3} + \tfrac{2a-2b}{M+3} + \tfrac{6}{5}a\big) + \tfrac{a-b}{2} \\ &\quad + \frac{\binom{M+4}{5}}{\binom{n}{4}}\big(\tfrac{6}{5}a - 6a(\mathcal{H}_{M+4} - \mathcal{H}_{M+3}) - \tfrac{a-b}{2}\tfrac{5}{M+4} + \tfrac{2a-2b}{M+3}\big) \\ &=\; \tfrac{6}{5}a(n+1)\mathcal{H}_{n+1} - \tfrac{n+1}{5}\big(6a\mathcal{H}_{M+3} + \tfrac{2a-2b}{M+3} + \tfrac{6}{5}a\big) \\ &\quad + \tfrac{a-b}{2} + \frac{\binom{M+4}{5}}{\binom{n}{4}}\big(\tfrac{6}{5}a + \tfrac{2a-2b}{M+3} + \tfrac{5b-17a}{2(M+4)}\big)\,. \end{aligned} \tag{A.10}$$

It remains to consider the second and third summands of (3.3)

$$\rho \;:=\; \Big(\tfrac{n+1}{5} + \frac{\binom{M+3}{4} - \binom{M+4}{5}}{\binom{n}{4}}\Big) C_{M+3} \;-\; \tfrac{M-1}{M+3}\Big(\tfrac{n+1}{5} - \frac{\binom{M+4}{5}}{\binom{n}{4}}\Big) C_{M+2}\,.$$

We start by applying definition (3.2) twice and using $C_n = C_n^{\mathrm{IS}}$ for $n \le M$ to expand $C_{M+3}$

$$\begin{aligned} C_{M+3} &= T_{M+3} + \frac{3}{\binom{M+3}{2}} \sum_{k=0}^{M+1} (M+2-k) C_k \\ &= T_{M+3} + \frac{3}{\binom{M+3}{2}} \Big( T_{M+1} + \frac{3}{\binom{M+1}{2}} \sum_{k=0}^{M-1} (M-k) C_k^{\mathrm{IS}} \Big) + \frac{3}{\binom{M+3}{2}} \sum_{k=0}^{M} (M+2-k) C_k^{\mathrm{IS}} \\ &= T_{M+3} + \frac{3}{\binom{M+3}{2}} T_{M+1} + \frac{3}{\binom{M+3}{2}} \sum_{k=0}^{M} \Big( (M+2-k) + \frac{3(M-k)}{\binom{M+1}{2}} \Big) C_k^{\mathrm{IS}} \end{aligned} \tag{A.11}$$

and $C_{M+2}$

$$C_{M+2} = T_{M+2} + \frac{3}{\binom{M+2}{2}} \sum_{k=0}^{M} (M+1-k) C_k^{\mathrm{IS}} . \tag{A.12}$$

Equations (A.11) and (A.12) are now inserted into the second and third summands of (3.3). With $T_n = an + b$ for $n \ge M+1$, this yields

$$\begin{aligned} \rho &= \tfrac{n+1}{5} \Big( (M+3)a + b + \frac{3}{\binom{M+3}{2}} \big( (M+1)a + b \big) - \tfrac{M-1}{M+3} \big( (M+2)a + b \big) \Big) \\ &\quad + \tfrac{n+1}{5} \sum_{k=0}^{M} \Big( \frac{3(M+2-k)}{\binom{M+3}{2}} + \frac{9(M-k)}{\binom{M+1}{2}\binom{M+3}{2}} - \tfrac{M-1}{M+3} \frac{3(M+1-k)}{\binom{M+2}{2}} \Big) C_k^{\mathrm{IS}} \\ &\quad + \frac{\binom{M+4}{5}}{\binom{n}{4}} \Big( (\tfrac{5}{M+4} - 1) C_{M+3} + \tfrac{M-1}{M+3} C_{M+2} \Big) \\ &= \tfrac{n+1}{5} \Big( 5a + \tfrac{6(b-a)}{M+2} + \tfrac{2(4a-b)}{M+3} \Big) + \tfrac{n+1}{5} \sum_{k=0}^{M} \frac{3M-2k}{\binom{M+2}{3}} C_k^{\mathrm{IS}} \\ &\quad + \frac{\binom{M+4}{5}}{\binom{n}{4}} \big( \tfrac{M-1}{M+3} C_{M+2} - \tfrac{M-1}{M+4} C_{M+3} \big) \end{aligned} \tag{A.13}$$

Adding (A.10) and (A.13) finally yields the claimed representation

$$\begin{aligned} C_n &= \tfrac{6}{5} a (n+1) \mathscr{H}_{n+1} + \tfrac{n+1}{5} \big( \tfrac{19}{5} a + \tfrac{6(b-a)}{M+2} - 6a \mathscr{H}_{M+2} \big) + \tfrac{a-b}{2} + \tfrac{n+1}{5} \sum_{k=0}^{M} \frac{3M-2k}{\binom{M+2}{3}} C_k^{\mathrm{IS}} \\ &\quad + \frac{\binom{M+4}{5}}{\binom{n}{4}} \big( \tfrac{6}{5} a + \tfrac{2(a-b)}{M+3} + \tfrac{5b-17a}{2(M+4)} - \tfrac{M-1}{M+4} C_{M+3} + \tfrac{M-1}{M+3} C_{M+2} \big) . \end{aligned}$$

For the asymptotic representation (3.5) of $C_n$, the penultimate summand is 0 because of the assumption $C_n^{\mathrm{IS}} = 0$. The last summand is in $\Theta(n^{-4})$ and therefore vanishes in the $\mathrm{O}(\frac{1}{n})$ term (we assume $M = \Theta(1)$ as $n \to \infty$ to be constant). Now, replacing $\mathscr{H}_n$ by its well-known asymptotic estimate

$$\mathscr{H}_n = \ln(n) + \gamma + \tfrac{1}{2} n + \mathrm{O}\big(\tfrac{1}{n^2}\big) \qquad \text{[Graham et al., 1994, eq. (6.66)]}$$

and expanding terms in (3.4) directly yields (3.5).

Finally, the case $T_2 = 0 \ne 2a + b$ affects the derivation only at a single point: As $M \ge 1$, the only occurring toll function that can ever equal $T_2$ is $T_{M+1}$, which occurs only in $C_{M+3}$, see (A.11). In $\rho$, we multiply $C_{M+3}$ by $\big(\frac{n+1}{5} + \big(\binom{M+3}{4} - \binom{M+4}{5}\big) / \binom{n}{4}\big) = \frac{1}{5}(n+1) + \mathrm{O}(n^{-4})$. Consequently, we have to subtract

$$\delta_{M1} \Big( \tfrac{1}{5}(n+1) \cdot \frac{3}{\binom{M+3}{2}} (2a+b) + \mathrm{O}(n^{-4}) \Big) = \delta_{M1} \tfrac{2a+b}{10} (n+1) + \mathrm{O}(n^{-4}) .$$

(The second equation follows by setting $M = 1$.)
This concludes the proof of Proposition 3.2. □

**Algorithm 2** Insertionsort as given and analyzed by Knuth [1998].

INSERTIONSORT(A,*left*,*right*)

```
1  for i = left + 1, ..., right
2      j := i − 1;    v := A[i]
3      while j ≥ left ∧ v < A[j]
4          A[j + 1] := A[j];    j := j − 1
5      end while
6      A[j + 1] := v
7  end for
```

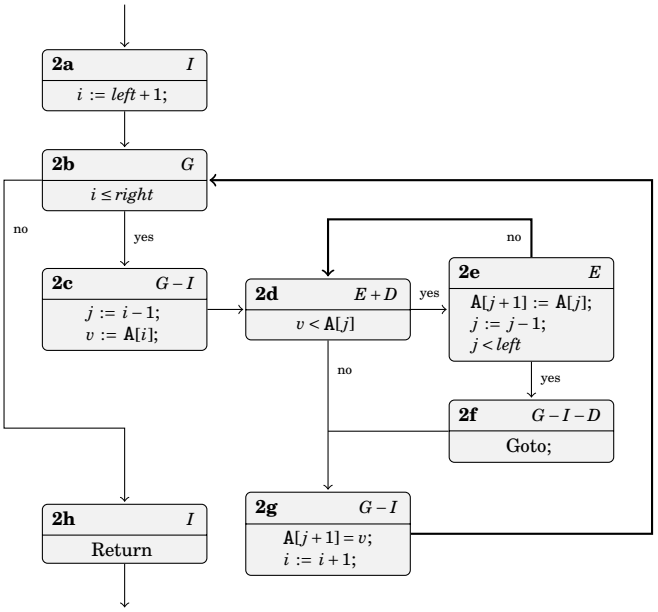



**Figure 3:** Control flow graph of our Java implementation of INSERTIONSORT (Algorithm 2). The block names "2a" – "2h" indicate that these blocks replace block 2 in Figure 1; this figure provides a "close-up view" of block 2. Blocks 2f and 2h contain control flow statements needed in Java Bytecode, which would normally be represented by arrows only. They are shown to remind us of their cost contributions.

# B. Insertionsort

In this section, we consider in some detail the INSERTIONSORT procedure used for sorting small subproblems. Insertionsort is a primitive sorting algorithm with quadratic running time in both worst and average case. On very small arrays however, it is extremely efficient, which makes it a good choice for our purpose.

Our implementation of INSERTIONSORT is given as Algorithm 2 and its control flow graph is shown in Figure 3. Algorithm 2 is based on the implementation by Knuth [1998, Program S]. Knuth assumes $n \geq 2$ in his code and analysis, but our Quicksort implementation also calls INSERTIONSORT on subproblems of size 0 or 1. Therefore, Algorithm 2 starts with an index comparison "$i \leq right$" to handle these cases.

Figure 3 lists the execution frequencies of all basic blocks. The names are chosen to match the corresponding notation of Sedgewick [1977] and denote the *total* execution frequencies across all

| **Block** | 1 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|
| **# Bytecodes** | 5 | 7 | 8 | 9 | 10 | 3 | 7 | 12 | 3 | 5 |
| **Block** | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | |
| **# Bytecodes** | 3 | 2 | 5 | 6 | 14 | 5 | 2 | 42 | 1 | |
| **Block** | 2a | 2b | 2c | 2d | 2e | 2f | 2g | 2h | | |
| **# Bytecodes** | 8 | 3 | 8 | 5 | 12 | 1 | 8 | 2 | | |

**Table 5:** Bytecode counts for the basic blocks of Figures 1 and 3.

invocations of INSERTIONSORT on small subproblems caused by one initial call to QUICKSORTYAROSLAVSKIY. Let us define $\tilde{I}$, $\tilde{G}$, $\tilde{D}$ and $\tilde{E}$ to denote the frequencies when we use INSERTIONSORT *in isolation* for sorting a random permutation. These frequencies are analyzed by Knuth [1998, p. 82] for $n \geq 2$. As mentioned above, our implementation has to work for $n \geq 0$, so our analysis must take the special cases $n \in \{0,1\}$ into account. We find

$$\tilde{I}(n) \;=\; 1, \qquad \tilde{G}(n) \;=\; n+\delta_{n0}, \qquad \tilde{D}(n) \;=\; n-\mathscr{H}_n \qquad \text{and} \qquad \tilde{E}(n) \;=\; \tbinom{n}{2}/2\,.$$

We can compute $I$, $G$, $D$ and $E$ by inserting $\tilde{I}$, $\tilde{G}$, $\tilde{D}$ and $\tilde{E}$ for $C^{\text{IS}}$ in the solution provided by Proposition 3.2 on page 7.

$$\begin{aligned} I(n) \;&=\; \tfrac{1}{5}(n+1)\frac{1}{\binom{M+2}{3}}\sum_{k=0}^{M}(3M-2k)\tilde{I}(k) \;=\; \tfrac{12}{5(M+2)}(n+1), \\ G(n) \;&=\; \big(1+\tfrac{18}{5(M+1)}-\tfrac{6}{M+2}\big)(n+1), \\ D(n) \;&=\; \big(1+\tfrac{3}{5(M+2)}-\tfrac{12}{5(M+2)}\mathscr{H}_{M+1}\big)(n+1), \\ E(n) \;&=\; \big(\tfrac{3}{20}M+\tfrac{6}{5(M+2)}-\tfrac{11}{20}\big)(n+1)\,. \end{aligned} \tag{B.1}$$

Using these frequencies, we can easily express the expected number of key comparisons, write accesses and executed Bytecode instructions: The only place where key comparisons occur, is in block 2d, so $C^{(IS)} = D + E$. Write accesses to array A happen in blocks 2e and 2g, giving $W^{(IS)} = E + (G - I)$. The number of Bytecodes is given in the next section.

# C. Low-Level Implementations and Instruction Counts

Figure 4 shows the Java implementation of Yaroslavskiy's algorithm whose Bytecode counts are studied in this paper. The partitioning loop is taken from the original sources of the Java 7 Runtime Environment library (see for example `http://www.docjar.com/html/api/java/util/DualPivotQuicksort.java.html`).

The Java code has been compiled using Oracle's Java Compiler (javac version 1.7.0_17). The resulting Java Bytecode was decomposed into the basic blocks of Figures 1 and 3. Then, for each block the number of Bytecode instructions was counted, the result is given in Table 5. We have automated this process as part of our tool MaLiJAn (Maximum Likelihood Java Analyzer), which provides a means of automating empirical studies of algorithms based on their control flow graphs [Laube and Nebel, 2010; Wild et al., 2013].

By multiplying the Bytecodes per block with the block's frequency, we get the overall number of executed Bytecodes. For Yaroslavskiy's Quicksort, we get

$$\begin{aligned} BC^{(QS)} \;&=\; 5R+7A+8B+9(A-B)+10A+3(A+C^{(1)})+7C^{(1)}+12S^{(1)}+3(C^{(1)}-S^{(1)}) \\ &\quad +5C^{(3)}+3(C^{(3)}-C^{(4)}+F)+2(C^{(3)}-C^{(4)})+5C^{(4)}+6(C^{(4)}-S^{(3)})+14S^{(3)} \\ &\quad +5C^{(4)}+2C^{(1)}+42A+1R \\ &=\; 71A-1B+6R+15C^{(1)}+10C^{(3)}+11C^{(4)}+9S^{(1)}+8S^{(3)}+3F\,. \end{aligned} \tag{C.1}$$

```
1  void quicksortYaroslavskiy(int[] A, int left, int right) {
2     if (right - left < M) {
3        insertionsort(A, left, right);
4     } else {
5        final int p, q;
6        if (A[left] > A[right]) {
7           p = A[right]; q = A[left];
8        } else {
9           p = A[left]; q = A[right];
10       }
11       int l = left + 1, g = right - 1, k = l;
12       while( k <= g ) {
13          final int ak = A[k];
14          if (ak < p) {
15             A[k] = A[l]; A[l] = ak; ++l;
16          } else if (ak >= q) {
17             while (A[g] > q && k < g)
18                --g;
19             if (A[g] < p) {
20                A[k] = A[l]; A[l] = A[g]; ++l;
21             } else {
22                A[k] = A[g];
23             }
24             A[g] = ak; --g;
25          }
26          ++k;
27       }
28       --l; ++g;
29       A[left] = A[l]; A[l] = p; A[right] = A[g]; A[g] = q;
30       quicksortYaroslavskiy(A, left, l - 1);
31       quicksortYaroslavskiy(A, g + 1, right);
32       quicksortYaroslavskiy(A, l + 1, g - 1);
33    }
34 }

36 void insertionsort(int[] A, int left, int right) {
37    for (int i = left + 1; i <= right; ++i) {
38       int j = i-1; final int v = A[i];
39       while (v < A[j]) {
40          A[j+1] = A[j]; --j;
41          if (j < left) break;
42       }
43       A[j+1] = v;
44    }
45 }
```

**Figure 4:** Java implementation of Yaroslavskiy's algorithm.

Additionally, we have for INSERTIONSORT

$$\begin{aligned} BC^{(IS)} &= 8I + 3G + 8(G-I) + 5(E+D) + 12E + 1(G-I-D) + 8(G-I) + 2I \\ &= 4D + 17E + 20G - 7I\,. \end{aligned} \tag{C.2}$$

Note that Wild [2012] investigated a different (more naïve) Java implementation of Yaroslavskiy's algorithm and hence reports different Bytecode counts.

# D. Details on the Distributional Analysis

In this appendix, we give details on the application of the contraction method to the distributions of costs in Yaroslavskiy's algorithm and we prove the main technical lemmas from Section 4.3.

## D.1. Proof of Theorem 4.3

We consider the first partitioning step of Yaroslavskiy's algorithm and denote by $T_C(n)$ the number of key comparisons of the first partitioning phase. By Property 2.1, subproblems generated in the first partitioning phase are, conditional on their sizes, again uniformly random permutations and independent of each other. Hence, we obtain the distributional recurrence

$$C_n \overset{\mathscr{D}}{=} C'_{I_1} + C''_{I_2} + C'''_{I_3} + T_C(n) \qquad (n \geq 3), \tag{D.1}$$

where $(I_1, I_2, I_3, T_C(n))$, $(C'_j)_{j\geq 0}$, $(C''_j)_{j\geq 0}$, $(C'''_j)_{j\geq 0}$ are independent and $C'_j$, $C''_j$, $C'''_j$ are identically distributed as $C_j$ for $j \geq 0$. Note that equation (D.1) is simply obtained from the generic distributional recurrence (4.5) upon inserting the toll function $T_C(n)$.

As in Section 4.2.2, we now define the normalized number of comparisons $C^*_n$ as

$$C^*_0 := 0 \qquad \text{and} \qquad C^*_n := \frac{C_n - \mathbb{E}[C_n]}{n}, \qquad (n \geq 1) \tag{D.2}$$

Note that $\mathbb{E}[C^*_n] = 0$ and the $\operatorname{Var}(C^*_n) < \infty$, i. e., $(C^*_n)_{n\geq 0}$ is a sequence of centered, square integrable random variables. Using (D.1) we find, cf. [Hwang and Neininger, 2002, eq. (27), (28)], that $(C^*_n)_{n\geq 0}$ satisfies (4.1) with

$$A^{(n)}_r = \frac{I_r}{n}, \qquad\qquad b^{(n)} = \frac{1}{n}\Big(T_C(n) - \mathbb{E}[C_n] + \sum_{r=1}^{3} \mathbb{E}[C_{I_r} \,|\, I_r]\Big).$$

We apply the framework of the contraction method outlined in Section 4.1. To check condition (A) note that from (4.4), we have $A^{(n)}_r \to D_r$ in $L_2$ for $r = 1,2,3$ as $n \to \infty$.

To identify the $L_2$-limit of $b^{(n)}$ we look at the summands $T_C(n)/n$ and $\big(-\mathbb{E}[C_n] + \sum_{r=1}^{3} \mathbb{E}[C_{I_r} \,|\, I_r]\big)/n$ separately. By Theorem 3.8, the expectation has the form $\mathbb{E}[C_n] = 19/10\, n \ln n + cn + \mathrm{O}(\log n)$ for some constant $c \in \mathbb{R}$, which implies $\mathbb{E}[C_{I_r} \,|\, I_r] = 19/10\, I_r \ln I_r + cI_r + o(n)$ since we have $o(I_r) = o(n)$. Plugging in these expansions, using $I_1 + I_2 + I_3 = n - 2$ and rearranging terms gives the asymptotic identity, as $n \to \infty$,

$$\begin{aligned} \frac{1}{n}\Big(-\mathbb{E}[C_n] + \sum_{r=1}^{3} \mathbb{E}[C_{I_r} \,|\, I_r]\Big) &= \sum_{r=1}^{3} \tfrac{19}{10} \tfrac{I_r}{n} \ln \tfrac{I_r}{n} \;-\; \tfrac{2}{n} \tfrac{19}{10} \tfrac{n \ln n}{n} \;-\; \tfrac{2c}{n} \;+\; o(1) \\ &= \sum_{r=1}^{3} \tfrac{19}{10} \tfrac{I_r}{n} \ln \tfrac{I_r}{n} \;+\; o(1)\,. \end{aligned}$$

Hence, Lemma 4.2 implies

$$\frac{1}{n}\Big(-\mathbb{E}[C_n] + \sum_{r=1}^{3} \mathbb{E}[C_{I_r} \,|\, I_r]\Big) \xrightarrow{L_2} \tfrac{19}{10} \sum_{j=1}^{3} D_j \ln D_j \qquad (n \to \infty)\,. \tag{D.3}$$

For the limit behavior of $T_C(n)/n$ we use the distributions listed in Table 3 and find

$$\begin{aligned} T_C(n) &= T_{C^{(1)}}(n) + \big(T_{C^{(1)}}(n) - T_{S^{(1)}}(n)\big) + T_{C^{(3)}}(n) + T_{C^{(4)}}(n) + T_A(n) \\ &\overset{\mathscr{D}}{=} n - 1 + I_1 + I_2 + \mathrm{HypG}(I_1+I_2, I_3, n-2) - \mathrm{HypG}(I_1, I_1+I_2, n-2) + 3\mathrm{B}(\tfrac{I_3}{n-2}). \end{aligned}$$

Using Lemma 4.1 and (4.4), we find for the normalized number of comparisons:

$$\begin{aligned} \frac{T_C(n)}{n} &\overset{\mathscr{D}}{=} \frac{n-2}{n} + \frac{I_1}{n} + \frac{I_2}{n} + \frac{\mathrm{HypG}(I_1+I_2, I_3, n-2)}{n-2} \cdot \frac{n-2}{n} \\ &\qquad - \frac{\mathrm{HypG}(I_1, I_1+I_2, n-2)}{n-2} \cdot \frac{n-2}{n} + 3\frac{\mathrm{B}\big(\frac{I_3}{n-2}\big)}{n} \\ &\xrightarrow{L_2} 1 + D_1 + D_2 + (D_1+D_2)D_3 - D_1(D_1+D_2) + 0 \\ &= 1 + (D_1+D_2)(1+D_3-D_1) \\ &= 1 + (D_1+D_2)(D_2+2D_3). \end{aligned}$$

Altogether, we obtain that condition (A) holds with

$$(A_1, A_2, A_3, b) = \Big(D_1, D_2, D_3,\, 1 + (D_1+D_2)(D_2+2D_3) + \tfrac{19}{10}\sum_{j=1}^{3} D_j \ln D_j\Big).$$

Concerning condition (B), note that $D_1$, $D_2$ and $D_3$ are identically distributed with density $x \mapsto 2(1-x)$ for $0 \le x \le 1$. This implies

$$\sum_{r=1}^{3} \mathbb{E}[D_r^2] = 3\mathbb{E}[D_1^2] = 3\int_0^1 x^2\, 2(1-x)\, dx = \tfrac{1}{2} < 1. \tag{D.4}$$

Moreover, condition (C) is fulfilled since

$$\sum_{r=1}^{K} \mathbb{E}\big[\mathbf{1}_{\{I_r^{(n)} \le \ell\}} \cdot \|(A_r^{(n)})^t A_r^{(n)}\|_{\mathrm{op}}\big] \le \sum_{r=1}^{K} \mathbb{P}(I_r^{(n)} \le \ell) \to 0, \qquad (n \to \infty), \quad \text{for any fixed } \ell \in \mathbb{N}.$$

Now the conclusions (I) and (II) give the claims $C_n^* \to C^*$ in distribution with the characterization of the distribution of $C^*$. For the asymptotic of the variance note that convergence of the second moment $\mathbb{E}[(C_n^*)^2] \to \mathbb{E}[(C^*)^2]$ and the normalization (D.2) imply

$$\mathrm{Var}(C_n) \sim \sigma_C^2 n^2 \qquad \text{with} \qquad \sigma_C^2 = \mathbb{E}[(C^*)^2].$$

To identify $\sigma_C^2$, let $C^{*,(1)}, C^{*,(2)}$ and $C^{*,(3)}$ be independent copies of $C^*$ also independent of $(D_1, D_2, D_3)$. We abbreviate $\tau := 1 + (D_1+D_2)(D_2+2D_3) + \frac{19}{10}\sum_{r=1}^{3} D_r \ln D_r$. Taking squares and expectations in (4.13) and noting that $\mathbb{E}[C^*] = \mathbb{E}[C^{*,(r)}] = 0$, we find

$$\begin{aligned} \mathbb{E}[(C^*)^2] &= \mathbb{E}\Big[\Big(1 + (D_1+D_2)(D_2+2D_3) + \tfrac{19}{10}\sum_{r=1}^{3} D_r \ln D_r + \sum_{r=1}^{3} D_r C^{*,(r)}\Big)^2\Big] \\ &= \mathbb{E}[\tau^2] + 2\mathbb{E}\Big[\tau \sum_{r=1}^{3} D_r C^{*,(r)}\Big] + \mathbb{E}\Big[\Big(\sum_{r=1}^{3} D_r C^{*,(r)}\Big)^2\Big] \\ &= \mathbb{E}[\tau^2] + 2\sum_{r=1}^{3} \mathbb{E}[\tau D_r]\mathbb{E}[C^*] + \sum_{r=1}^{3} \mathbb{E}[D_r^2]\mathbb{E}[(C^*)^2] + \sum_{\substack{1 \le r,s \le 3 \\ r \ne s}} \mathbb{E}[D_r D_s]\mathbb{E}[C^*]^2 \\ &= \mathbb{E}[\tau^2] + \mathbb{E}[(C^*)^2]\sum_{r=1}^{3} \mathbb{E}[D_r^2] \\ &= \mathbb{E}[\tau^2] + \tfrac{1}{2}\mathbb{E}[(C^*)^2], \end{aligned}$$

where for the last equality (D.4) was used. Solving for $\mathbb{E}[(C^*)^2]$ implies

$$\mathbb{E}[(C^*)^2] = 2\mathbb{E}\Big[\Big(1+(D_1+D_2)(D_2+2D_3)+\tfrac{19}{10}\sum_{j=1}^{3} D_j \ln D_j\Big)^2\Big].$$

Now, the integral representation (4.3) and the use of a computer algebra system yields the expression for $\sigma_C^2$. □

## D.2. Proof of Theorem 4.4

The proof can be done similarly to the one for Theorem 4.3. We have the recurrence

$$S_n \overset{\mathscr{D}}{=} S'_{I_1} + S''_{I_2} + S'''_{I_3} + T_S(n), \qquad (n \geq 3), \tag{D.5}$$

with conditions on independence and distributions as in (D.1), where $T_S(n)$ is the number of swaps in the first partitioning step of the algorithm. We set $S_0^* := 0$ and

$$S_n^* := \frac{S_n - \mathbb{E}[S_n]}{n}, \qquad (n \geq 1). \tag{D.6}$$

Hence, $(S_n^*)_{n\geq 0}$ is a sequence of centered, square integrable random variables satisfying (4.1) with

$$A_r^{(n)} = \frac{I_r}{n}, \qquad b^{(n)} = \frac{1}{n}\Big(T_S(n) - \mathbb{E}[S_n] + \sum_{r=1}^{3} \mathbb{E}[S_{I_r} \,|\, I_r]\Big),$$

where by Theorem 3.9, we know $\mathbb{E}[S_n] = \frac{3}{5} n \ln(n) + c' n + \mathrm{O}(\log n)$ for a constant $c' \in \mathbb{R}$. Analogously to (D.3) we obtain

$$\frac{1}{n}\Big(-\mathbb{E}[S_n] + \sum_{r=1}^{3} \mathbb{E}[S_{I_r} \,|\, I_r]\Big) \;\to\; \tfrac{3}{5}\sum_{j=1}^{3} D_j \ln D_j, \qquad (n\to\infty), \quad \text{in } L_2\,. \tag{D.7}$$

It remains to study the asymptotic behavior of $T_S(n)/n$. Again profiting from the spadework of Section 3.2, we find the exact distribution of the number of swaps:

$$\begin{aligned} T_S(n) &= T_{S^{(1)}}(n) + \big(T_{C^{(4)}}(n) - T_{S^{(3)}}(n)\big) + 2T_{S^{(3)}}(n) + 2T_A(n) \;=\; I_1 + sm@\mathscr{G} + \delta + 2 \\ &\overset{\mathscr{D}}{=} I_1 + \mathrm{HypG}(I_1+I_2, I_3, n-2) + \mathrm{B}\big(\tfrac{I_3}{n-2}\big) + 2\,. \end{aligned}$$

By Lemma 4.1 and (4.4), we find

$$\frac{T_S(n)}{n} \;\xrightarrow{L_2}\; D_1 + (D_1+D_2)D_3\,.$$

The conditions (A), (B) and (C) are now checked as in the proof of Theorem 4.3. The assertions of Theorem 4.4 follow from (I) and (II), the identification of $\sigma_S^2$ is done as that of $\sigma_C^2$ in Theorem 4.3. □

## D.3. Proof of Theorem 4.5

The proof can be done very similarly as for Theorems 4.3 and 4.4. We only present the key points where changes are needed. For the distributional recurrence, we here have the toll function

$$\begin{aligned} T_{BC}(n) \;=\; & 15T_{C^{(1)}}(n) + 10T_{C^{(3)}}(n) + 11T_{C^{(4)}}(n) + 9T_{S^{(1)}}(n) + 8T_{S^{(3)}}(n) \\ & + 71T_A(n) - 1T_B(n) + 6T_R(n) + 3T_F(n)\,. \end{aligned}$$

All contributions from the second line are bounded by $\mathrm{O}(1)$ (see Table 3). Therefore they vanish in the limit of $T_{BC}(n)/n$:

$$\begin{aligned} \frac{T_{BC}(n)}{n} &\;\xrightarrow{L_2}\; 15(D_1+D_2) + 10D_3 + 11(D_1+D_2)D_3 + 9D_1(D_1+D_2) + 8\big(D_1 - D_1(D_1+D_2)\big) \\ &= 24 + (D_3-9)D_2 - 2D_3(5D_3+2)\,. \end{aligned}$$

The rest of the proof is carried out along the same lines as for the proofs above. □

## D.4. Proof of Theorem 4.6

We define the column vector $Y_n := \binom{C_n}{S_n}$. Then from (D.1) and (D.5), we obtain

$$Y_n \overset{\mathscr{D}}{=} Y'_{I_1} + Y''_{I_2} + Y'''_{I_3} + \begin{pmatrix} T_C(n) \\ T_S(n) \end{pmatrix},$$

with conditions on independence and distributions as in (D.1). We set $Y_0^* := \binom{0}{0}$ and

$$Y_n^* := \frac{1}{n}\big(Y_n - \mathbb{E}[Y_n]\big). \tag{D.8}$$

Hence, $(Y_n^*)_{n\geq 0}$ is a sequence of centered, square integrable, bivariate random variables satisfying (4.1) with

$$A_r^{(n)} = \frac{1}{n}\begin{bmatrix} I_r & 0 \\ 0 & I_r \end{bmatrix}, \qquad b^{(n)} = \frac{1}{n}\begin{pmatrix} T_C(n) - \mathbb{E}[C_n] + \sum_{r=1}^3 \mathbb{E}[C_{I_r} \mid I_r] \\ T_S(n) - \mathbb{E}[S_n] + \sum_{r=1}^3 \mathbb{E}[S_{I_r} \mid I_r] \end{pmatrix}.$$

Using the asymptotic behavior from the proofs of Theorems 4.3 and 4.4 we obtain that condition (A) holds with

$$A_r = \begin{bmatrix} D_r & 0 \\ 0 & D_r \end{bmatrix}, \qquad b = \begin{pmatrix} 1 + (D_1 + D_2)(D_2 + 2D_3) + \frac{19}{10}\sum_{j=1}^3 D_j \ln D_j \\ D_1 + (D_1 + D_2)D_3 \quad + \frac{3}{5}\sum_{j=1}^3 D_j \ln D_j \end{pmatrix}. \tag{D.9}$$

Condition (B) is satisfied as

$$\sum_{r=1}^3 \mathbb{E}\big[\|A_r^t A_r\|_{\mathrm{op}}\big] = \sum_{r=1}^3 \mathbb{E}[D_r^2] = \tfrac{1}{2} < 1.$$

Condition (C) is checked similarly as in the proof of Theorem 4.3.

Hence from (I) we obtain the existence of a centered, square integrable, bivariate distribution $\mathscr{L}(\Lambda_1, \Lambda_2)$ that solves the bivariate fixed-point equation (4.2) with the choices for $A_r$ and $b$ given in (D.9). Furthermore (II) implies that the sequence $(Y_n^*)$ defined in (D.8) converges in distribution and with mixed second moments towards $(\Lambda_1, \Lambda_2)$. This implies in particular, as $n \to \infty$,

$$\mathbb{E}\left[\frac{C_n - \mathbb{E}[C_n]}{n} \cdot \frac{S_n - \mathbb{E}[S_n]}{n}\right] \to \mathbb{E}[\Lambda_1 \cdot \Lambda_2].$$

Hence, we obtain

$$\mathrm{Cov}(C_n, S_n) \sim \mathbb{E}[\Lambda_1 \Lambda_2] n^2.$$

The value $\mathbb{E}[\Lambda_1\Lambda_2]$ is obtained from the fixed-point equation (4.2) with the choices for $A_r$ and $b$ given in (D.9) by multiplying the components on left and right hand side, taking expectations and solving for $\mathbb{E}[\Lambda_1\Lambda_2]$. The integral representation (4.3) then leads to the expression given in (4.21). □

## D.5. Proof of Lemma 4.1

We denote by $\mathrm{Bin}(n,p)$ the binomial distribution with $n$ trials and success probability $p$.

The hypergeometric distribution $\mathrm{HypG}(k, r, r+b)$ has mean and variance given in (2.1). In particular for sequences $(\alpha_n)_{n\geq 1}$, $(\beta_n)_{n\geq 1}$ with $\alpha_n = \alpha n + r_n$ and $\beta_n = \beta n + s_n$ with $\alpha, \beta \in (0,1)$ and $|r_n|, |s_n| \leq n^{2/3}$, we obtain for hypergeometrically $\mathrm{HypG}(\alpha_n, \beta_n, n)$ distributed random variables $\Upsilon_n$

that $\mathrm{Var}(\Upsilon_n) \le n$ and moreover

$$\begin{aligned}\left\| \frac{\Upsilon_n}{n} - \alpha\beta \right\|_2 &\le \left\| \frac{\Upsilon_n}{n} - \frac{\mathbb{E}\Upsilon_n}{n} \right\|_2 + \left| \frac{\mathbb{E}\Upsilon_n}{n} - \alpha\beta \right| \\ &= \frac{\sqrt{\mathrm{Var}(\Upsilon_n)}}{n} + \left| \frac{\alpha_n\beta_n}{n^2} - \alpha\beta \right| \\ &\le n^{-1/2} + \frac{\alpha|s_n|}{n} + \frac{\beta|r_n|}{n} + \frac{|r_n s_n|}{n^2} \\ &\le n^{-1/2} + 2n^{-1/3} + n^{-2/3} \;\le\; 4n^{-1/3}. \end{aligned} \tag{D.10}$$

Now, we first condition on $V := (V_1, \dots, V_b) = (v_1, \dots, v_b) =: v$, where $v$ satisfies $v_i \in [0,1]$ and $\sum_{r=1}^b v_r = 1$. Conditionally on $V = v$, the random variables $\sum_{j \in J_i} L_j$ have a binomial $\mathrm{Bin}(n, w_i)$ distribution, where $w_i := \sum_{j \in J_i} v_j$ for $i = 1,2$. We denote

$$B_n^v \;:=\; \bigcap_{i=1}^2 \Big\{ \sum_{j \in J_i} L_j \in \Big[ n w_i - n^{2/3},\, n w_i + n^{2/3} \Big] \Big\}.$$

Then by Chernoff's bound [McDiarmid, 1998, p. 195], we obtain uniformly in $v$ that

$$\mathbb{P}(B_n^v) \;\ge\; 1 - 4\exp\big(-2n^{1/3}\big) \;\to\; 1 \qquad \text{as } n \to \infty. \tag{D.11}$$

We abbreviate $W_i := \sum_{j \in J_i} V_j$ for $i = 1,2$ and let $Z_n^v$ denote $Z_n$ conditional on $V = v$. By $\mathbb{P}_V$ we denote the distribution of $V$. Then, we obtain with (D.10)

$$\begin{aligned}\mathbb{E}\left[ \left| \frac{Z_n}{n} - W_1 W_2 \right|^2 \right] &= \int \mathbb{E}\left[ \left| \frac{Z_n^v}{n} - w_1 w_2 \right|^2 \right] d\,\mathbb{P}_V(v) \\ &\le \iint_{B_n^v} \left| \frac{Z_n^v}{n} - w_1 w_2 \right|^2 d\,\mathbb{P} \;+\; 4\exp\big(-2n^{1/3}\big) \cdot \max_{0 \le z \le n} \left| \tfrac{z}{n} - w_1 w_2 \right|^2 d\,\mathbb{P}_V(v) \\ &\le 4n^{-1/3} + 4\exp\big(-2n^{1/3}\big) \\ &\to 0, \qquad \text{as } n \to \infty, \end{aligned}$$

which concludes the proof. □

## D.6. Proof of Lemma 4.2

Let $i \in \{1, \dots, n\}$ be arbitrary. The strong law of large numbers implies that $L_i / n \to V_i$ *almost surely* as $n \to \infty$. The function $x \mapsto x\ln(x)$ is continuous on $[0,1]$ (with the convention $x\ln(x) = 0$ for $x = 0$). Hence, as $n \to \infty$,

$$\left| \tfrac{L_i}{n} \ln\big(\tfrac{L_i}{n}\big) - V_i \ln V_i \right|^2 \;\to\; 0 \qquad \text{almost surely}.$$

Since the non-positive function $x \mapsto x\ln(x)$, $x \in [0,1]$ is lower bounded (e. g. by $-1/e$) the square in the latter display is uniformly bounded (e. g. by $(2/e)^2$). Hence the *dominated convergence theorem* implies

$$\mathbb{E}\Big[ \Big| \tfrac{L_i}{n} \ln\big(\tfrac{L_i}{n}\big) - V_i \ln V_i \Big|^2 \Big] \;\to\; 0, \qquad \text{as } n \to \infty.$$

This concludes the proof. □

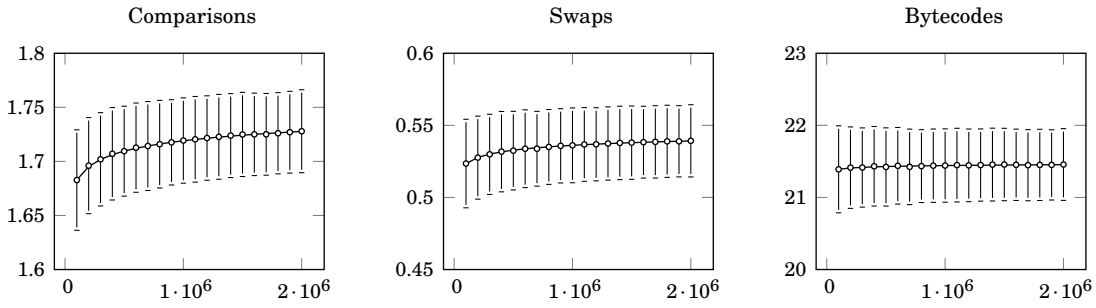



**Figure 5:** Comparison of sample means with the asymptotics computed in Section 3. The asymptotics are shown as solid line —×—, the sample means are indicated by circles ○. Additionally, the error bars show the sample standard deviation. On the $x$-axis, the input size $n$ is shown, the $y$-axis shows the corresponding counts normalized by $n \ln n$.

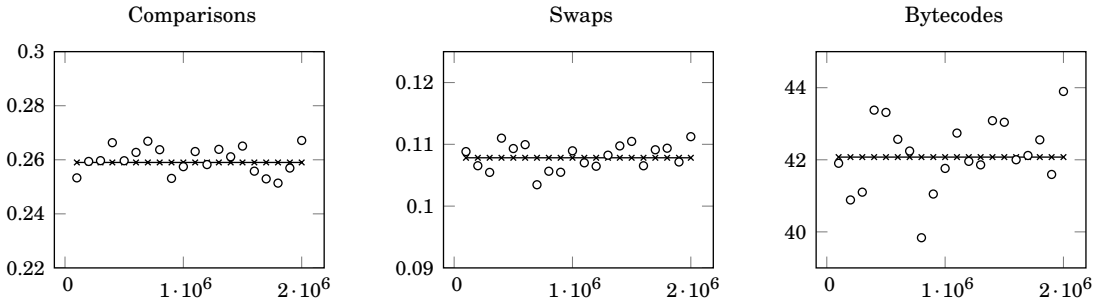



**Figure 6:** Comparison of sample variances with the asymptotics computed in Section 4. The asymptotics are shown as solid line —×—, the sample variances are indicated by circles ○. On the $x$-axis, the input size $n$ is shown, the $y$-axis shows the corresponding variances normalized by $n^2$.

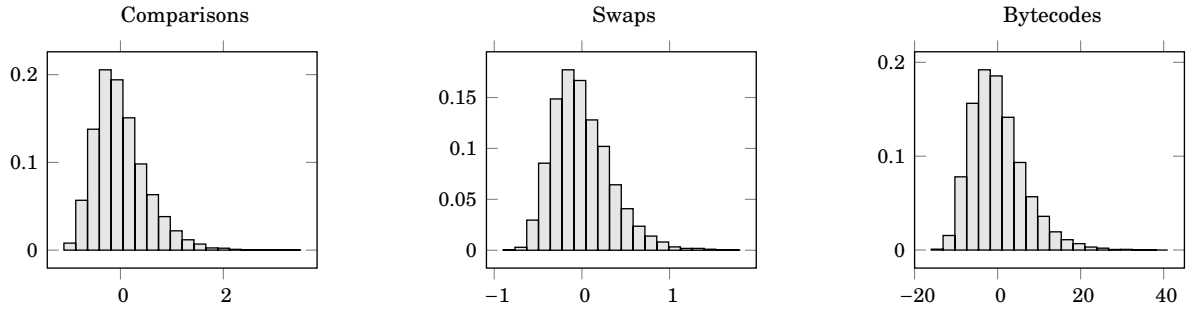



**Figure 7:** Histograms for the distributions of the normalized number of comparisons $C_n^*$, swaps $S_n^*$ and executed Bytecode instructions $BC_n^*$ from the sample of $10\,000$ random permutations of size $n = 10^6$.

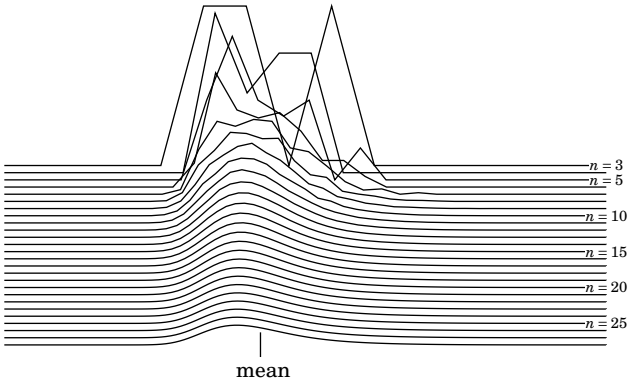



**Figure 8:** This figure shows the distributions of the normalized number of comparisons $C_n^*$ for small $n$. The distributions are computed by unfolding the distributional recurrence (D.1). This figure serves as visual evidence for the convergence of cost distributions to a limit law. It is heavily inspired by a corresponding figure for classic Quicksort [Sedgewick and Flajolet, 1996, Figure 1.3].

# E. Experimental Validation of Asymptotics

In this paper, we computed asymptotics for mean and variance of the costs of Yaroslavskiy's algorithm. Whereas the results for the mean are very precise and indeed can be made exact with some additional diligence, our contraction arguments only provide leading term asymptotics. In this section, we compare the asymptotic approximations with experimental sample means and variances.

We use the Java implementation given in Appendix C and run it on 10000 pseudo-randomly generated permutations of $\{1,\ldots,n\}$ for each of the 20 sizes in $\{10^5, 2\cdot 10^5, \ldots, 2\cdot 10^6\}$. Note that for sensible estimates of variances, much larger samples are needed than for means. The experiment itself is done using our tool MaLiJAn, which automatically counts the number of comparisons, swaps and Bytecode instructions [Wild et al., 2013].

Figure 5 shows the results for the expected costs and Figure 6 compares asymptotic and sampled variances. The histograms in Figure 7 give some impression how the limit laws will look like.

It is clearly visible in Figure 5 that for the given range of input sizes, the average costs computed in Section 3 are extremely precise. In fact, hardly any deviation between prediction and measurement is visible. The variances in Figure 6 show more erratic behavior. As variances are much harder to estimate than means, this does not come as a surprise. From the data we cannot tell whether the true variances show some oscillatory behavior (in lower order terms) or whether we observe sampling noise. Nevertheless, Figure 6 shows that for the given range of sizes, the asymptotic is a sensible approximation of the exact variance.

Figure 8 shows how fast the exact probability distribution of the normalized number of comparisons approaches a smooth limiting shape even for tiny $n$. This strengthens the above quantitative arguments that the limiting distributions computed in this paper are useful approximations of the true behavior of costs in Yaroslavskiy's algorithm.

# ACKNOWLEDGEMENTS

We would like to thank two anonymous reviewers for their helpful comments and suggestions.

# References


Jon L. Bentley and M. Douglas McIlroy. Engineering a sort function. *Software: Practice and Experience*, 23(11):1249–1265, 1993.

Andrea Camesi, Jarle Hulaas, and Walter Binder. Continuous Bytecode Instruction Counting for CPU Consumption Estimation. In *Third International Conference on the Quantitative Evaluation of Systems*, pages 19–30. IEEE, 2006. ISBN 0-7695-2665-9.

Hua-Huai Chern and Hsien-Kuei Hwang. Transitional behaviors of the average cost of quicksort with median-of-$(2t+1)$. *Algorithmica*, 29(1-2):44–69, 2001.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009. ISBN 978-0-262-03384-8.

Herbert A. David and Haikady N. Nagaraja. *Order Statistics (Wiley Series in Probability and Statistics)*. Wiley-Interscience, 3rd edition, 2003. ISBN 0-471-38926-9.

Edsger W. Dijkstra. *A Discipline of Programming*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1976. ISBN 013215871X.

Marianne Durand. Asymptotic analysis of an optimized quicksort algorithm. *Information Processing Letters*, 85(2):73–77, 2003.

Maarten H. van Emden. Increasing the efficiency of quicksort. *Communications of the ACM*, pages 563–567, 1970.

James Allen Fill and Svante Janson. Smoothness and decay properties of the limiting Quicksort density function. In *Mathematics and computer science (Versailles, 2000)*, Trends Math., pages 53–64. Birkhäuser, Basel, 2000.

James Allen Fill and Svante Janson. Quicksort asymptotics. *J. Algorithms*, 44(1):4–28, 2002.

Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete mathematics: a foundation for computer science*. Addison-Wesley, 1994. ISBN 978-0-20-155802-9.

Pascal Hennequin. Combinatorial analysis of Quicksort algorithm. *Informatique théorique et applications*, 23(3):317–333, 1989.

Pascal Hennequin. *Analyse en moyenne d'algorithmes : tri rapide et arbres de recherche*. PhD Thesis, Ecole Politechnique, Palaiseau, 1991.

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

C. A. R. Hoare. Algorithm 63: Partition. *Communications of the ACM*, 4(7):321, July 1961b.

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

Hsien-Kuei Hwang and Ralph Neininger. Phase change of limit laws in the quicksort recurrence under varying toll functions. *SIAM J. Comput.*, 31(6):1687–1722 (electronic), 2002.

Maurice G. Kendall. *The advanced theory of statistics*. Charles Griffin, London, 2nd edition, 1945.

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

Donald E. Knuth. *The Art of Computer Programming: Volume 1, Fascicle 1. MMIX, A RISC Computer for the New Millennium*. Addison-Wesley, 2005. ISBN 0-201-85392-2.

Shrinu Kushagra, Alejandro López-Ortiz, Aurick Qiao, and J. Ian Munro. Multi-Pivot Quicksort: Theory and Experiments. In *ALENEX 2014*, pages 47–60. SIAM, 2014.

Ulrich Laube and Markus E. Nebel. Maximum likelihood analysis of algorithms and data structures. *Theoretical Computer Science*, 411(1):188–212, January 2010.

Tim Lindholm and Frank Yellin. *Java Virtual Machine Specification*. Addison-Wesley Longman Publishing, Boston, MA, USA, 2nd edition, April 1999. ISBN 0-201-43294-3.

Hosam M. Mahmoud. *Sorting: A distribution theory*. John Wiley & Sons, Hoboken, NJ, USA, 2000. ISBN 1-118-03288-8.

Conrado Martínez and Salvador Roura. Optimal Sampling Strategies in Quicksort and Quickselect. *SIAM Journal on Computing*, 31(3):683, 2001.

Colin McDiarmid. Concentration. In *Probabilistic methods for algorithmic discrete mathematics*, pages 195–248. Springer, Berlin, 1998.

Colin McDiarmid and Ryan B. Hayward. Large deviations for Quicksort. *J. Algorithms*, 21(3): 476–507, 1996.

Colin L. McMaster. An analysis of algorithms for the Dutch National Flag Problem. *Communications of the ACM*, 21(10):842–846, October 1978.

S. J. Meyer. A failure of structured programming. Technical report, Zilog Corp., Cupertino, CA, USA, 1978.

Markus E. Nebel and Sebastian Wild. Pivot Sampling in Dual-Pivot Quicksort, 2014. URL `http://arxiv.org/abs/1403.6602`.

Ralph Neininger. On a multivariate contraction method for random recursive structures with applications to Quicksort. *Random Structures & Algorithms*, 19(3-4):498–524, 2001.

Mireille Regnier. A limiting distribution for Quicksort. *Informatique théorique et applications*, 23 (3):335–343, 1989.

Uwe Rösler. A limit theorem for "quicksort". *Informatique théorique et applications*, 25(1):85–100, 1991.

Uwe Rösler. On the analysis of stochastic divide and conquer algorithms. *Algorithmica*, 29(1): 238–261, 2001.

Robert Sedgewick. *Quicksort*. PhD Thesis, Stanford University, 1975.

Robert Sedgewick. The analysis of Quicksort programs. *Acta Inf.*, 7(4):327–355, 1977.

Robert Sedgewick and Philippe Flajolet. *An Introduction to the Analysis of Algorithms*. Addison-Wesley-Longman, 1996. ISBN 978-0-201-40009-0.

Richard C. Singleton. Algorithm 347: an efficient algorithm for sorting with minimal storage [M1]. *Communications of the ACM*, 12(3):185–186, March 1969.

Kok Hooi Tan and Petros Hadjicostas. Some properties of a limiting distribution in Quicksort. *Statistics & Probability Letters*, 25(1):87–94, October 1995.

Sebastian Wild. Java 7's Dual Pivot Quicksort. Master thesis, University of Kaiserslautern, 2012.

Sebastian Wild and Markus E. Nebel. Average Case Analysis of Java 7's Dual Pivot Quicksort. In Leah Epstein and Paolo Ferragina, editors, *ESA 2012*, volume 7501 of *LNCS*, pages 825–836. Springer, 2012.

Sebastian Wild, Markus E. Nebel, Raphael Reitzig, and Ulrich Laube. Engineering Java 7's Dual Pivot Quicksort Using MaLiJAn. In Peter Sanders and Norbert Zeh, editors, *ALENEX 2013*, pages 55–69. SIAM, 2013.