CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Modular smoothed analysis of median-of-three Quicksort
Authors	Hennessy, Aoife;Schellekens, Michel P.
Publication date	2014
Original Citation	HENNESSY, A. & SCHELLEKENS, M. P. 2014. Modular smoothed analysis of median-of-three Quicksort. Submitted to: Discrete Mathematics. [Preprint]
Type of publication	Article (preprint)
Link to publisher's version	http://www.sciencedirect.com/science/journal/0012365X
Download date	2026-11-06 21:26:23
Item downloaded from	https://hdl.handle.net/10468/1367



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Modular Smoothed Analysis of Median-of-three Quicksort[1]

Aoife Hennessy[2], M. Schellekens[3]

Abstract

Spielman's smoothed complexity - a hybrid between worst and average case complexity measures - relies on perturbations of input instances to determine where average-case behavior turns to worst-case. This approach simplifies the smoothed analysis and achieves greater precession in the expression of the smoothed complexity, where a recurrence equation is obtained as opposed to bounds. Moreover, the approach addresses, in this context, the formation of input instances–an open problem in smoothed complexity. In [23], we proposed a method supporting modular smoothed analysis and illustrated the method by determining the modular smoothed complexity of Quicksort. Here, we use the modular approach to calculate the median of three variant and compare these results with those in [23].

Keywords: Modular Smoothed analysis, $\mathcal{MOQA}$, Median-of-three Quicksort

1. Introduction

Smoothed Analysis is a framework for analyzing algorithms and heuristics, partially motivated by the observation that input parameters in practice often are subject to a small degree of random noise [17]. In smoothed

[1]This grant was supported by SFI grant SFI 07/IN.1/I977. The authors are grateful for fruitful discussions with D. Early that have improved the presentation of the paper and support by Ang Gao in preparing the diagrams.

[2]Centre for Efficiency Oriented Languages(CEOL), University College Cork, Ireland, a.hennessy@cs.ucc.ie

[3]Centre for Efficiency Oriented Languages(CEOL), University College Cork, Ireland, m.schellekens@cs.ucc.ie

analysis, one assumes that an input to an algorithm is subject to a slight random perturbation. The *smoothed measure* of an algorithm on an input instance is its expected performance over the perturbations of that instance. The *smoothed complexity* of an algorithm is the maximum smoothed measure over its inputs. The area has been widely studied, following up on the ground breaking work by Spielman and Teng on the smoothed analysis of the simplex method [16]. For an overview of the literature, we refer the reader to the survey paper [17].

In this paper we extend the modular approach of smoothed analysis that was presented in [23] to median-of-three Quicksort. Modularity is a property of systems (hardware or software), which reflects the extent to which it is decomposable into parts, from the properties of which one is able to predict the properties of the whole[12]. Modularity brings a strong advantage. The capacity to combine parts of code, where the complexity is simply the sum of the complexities of the parts, is a very helpful advantage in static analysis.

We obtain the following recurrence[4] for the modular smoothed complexity $f(n,k)$ of median-of-three Quicksort:

$$f(n,k) = (n-1) + 1 + \sum_{j=2}^{n-1} \beta_{n+1-j}^{n-1} f(j-1,k) + \sum_{j=2}^{n-1} \beta_j^{n-1} f(j-1,k) \qquad (1)$$

where β_j^{n-1} for $2 \le j \le n-2$ is

$$\frac{2!(j-1)}{k(n-2)\binom{n}{k}} \left(2\binom{n-4}{k-2} + 2\frac{(k+1)}{(k-1)}\binom{n-4}{k-3} + \frac{3(n-j)}{(k-1)}\binom{n-4}{k-4} \right),$$

[4]Where, as discussed in the paper, we work with a perturbation probability σ approximated by k/n, for a partial permutation of k elements out of n, in order to express the smoothed complexity as a recurrence.

and β_{n-1}^{n-1} is

$$\left\{k!\binom{n-3}{k}+(k-1)!\binom{n-3}{k-1}(2+k)\right.$$
$$\left.+\,2!(k-2)!\binom{n-3}{k-2}(2k-1)+3!(k-2)!\binom{n-3}{k-3}\right\}\frac{(n-k)!}{n!}. \tag{2}$$

As highlighted in [23] since the argument of [4] cannot rely on subproblems generated in recursive calls being again random permutations (for the case of partial permutations), the authors presented an alternative argument based on randomized incremental constructions [11]. This problem is overcome with our modular approach which allows us to form a recursive argument [23]. In [23] we illustrated our model with Quicksort. Here, we extend the modular approach to Quicksort's median-of-three variant, where as in [23] we will yield a traditional recurrence equation. We refer the reader to [23] for further details. We will contrast our results to our previous analysis of Quicksort. As in [23], in our model, the data will be represented as finite labelled partial orders, or LPOs, and random structures.

A *labelled partial order*, or LPO, is a triple $(A, \sqsubseteq, l)$, where A is a set, $\sqsubseteq$ is a partial order on A (that is, a binary relation which is reflexive, anti-symmetric, and transitive), and the labeling l is a bijection from A to some totally ordered set C which is increasing with respect to the order $\sqsubseteq$.

Given a finite partial order $(A, \sqsubseteq)$ and a totally ordered set C with $|C| = |A|$, the *random structure* $R_C(A, \sqsubseteq)$ is the set of all LPOs $(A, \sqsubseteq, l)$ with $l(A) = C$.

As the algorithms we consider are comparison-based, the choice of the set C is unimportant, and we will generally write a random structure as $R(A, \sqsubseteq)$, without the subscript[5].

The algorithms we consider have the property of "randomness preservation", which means that applying any operation to each LPO in a random structure results in an output isomorphic to one or more random structures, which is the key to systematic timing.

[5]More formally, we can consider the random structure to be the quotient of the set of all LPOs on the partial order $(A, \sqsubseteq)$ with respect to a natural isomorphism. See [18, 22] for a full discussion.

Formally: a *random bag* is a multiset[6] of random structures. We represent random bags using the multiplicity notation for multisets, so that, for each i, the random bag $\{(R_i, K_i)\}_{i=1,\ldots,n}$ contains K_i copies of the random structure R_i. A function which takes a random structure as argument is random bag preserving if its output is a random bag $\{(R_i, K_i)\}_{i=1,\ldots,n}$.

In this context, any sorting algorithm transforms the random structure with underlying discrete order of size n, i.e. a random bag containing a single random structure R_1 with multiplicity one, into the random bag containing a single random structure R_2 with underlying linear order with multiplicity $n!$. Here, R_1 comprises of exactly $n!$ labelings, representing the traditional $n!$ input lists used in the analysis of sorting algorithms. The random structure R_2 has exactly one labeling, representing the sorted list, of which $n!$ copies will be produced by any sorting algorithm.

As computations proceed in our model, the partial orders underlying the random structures will become more refined (i.e. more order is introduced)[7]. For instance, any sorting algorithm will start its computation from a random structure with underlying order the discrete partial order. It will transform this into the sorted output, hence into the random structure with underlying partial order the linear partial order, a refinement of the discrete partial order.

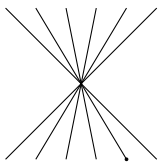
To set the stage for the analysis of median-of-three Quicksort, we briefly discuss the split operation, recursively called to order the data. In the traditional version of Quicksort, a fixed pivot is chosen. In the median-of-three variant, the pivot is chosen as the middle element in three randomly chosen elements. The split operation reorders a given list, placing all elements smaller than the pivot before the pivot (in the same order as encountered in the original list) and all elements greater than the pivot after the pivot (again in the same order as encountered in the original list). The usual representation of the output list is a list in which the pivot element sits in its "correct" position. I.e. if the pivot has relative rank i among the elements in the list, the pivot will be placed in the i-th position in the output list, all elements smaller than the pivot will occur to its left and all pivots greater than the pivot will occur to its right.

[6] I.e. a set which allows elements to occur with any integer multiplicity, rather than only zero or one.

[7] Formally: A poset $(X_1, \sqsubseteq_1)$ is a refinement of another poset $(X_2, \sqsubseteq_2)$ if for all $x, y \in X_1. x \sqsubseteq_2 y \Rightarrow x \sqsubseteq_1 y$.

The underlying partial order for which this output list is a labeling is the "star-shaped order" containing a central element (to store the pivot label), with $n-i-1$ elements placed above it (to store the labels greater than the pivot) and $n-i$ elements below it (to store the labels less than the pivot).

$n-i-1$ elements



i elements

Split is a random bag preserving operation. The random bag presentation of the outputs of split corresponds to representing the "order-information" gathered during the execution of the split operation. The elements placed by split to the left of the pivot, will be represented as "smaller" than the pivot, i.e. below the pivot as labels on the above Hasse diagram. The elements placed by split to the right of the pivot, will be represented as greater than the pivot, i.e. above the pivot as labels on the Hasse diagram.

As in [23] the model here can be viewed as a new step to explore the discrete case: the points in our space are the random structures (or more generally random bags). The perturbations happening throughout the computation, and acting on random structures, will become proportionally less in relation to the amount that our data is already sorted. The sorting computation gradually "refines" (= introduce more order on) the random bags after each operation. The perturbations will never affect already sorted elements (in this case the pivots placed in their correct position after each split) and ultimately, in the last step of the recursion, will not affect the sorted output. Formally, our perturbations on a given LPO can only act on "free pairs" of labels, i.e. labels that, when swapped result in a new LPO. For this to occur, the swapped labels need to respect the underlying partial order.

We note, in [23] sample-fair smoothed(SFS) was defined as

$$T_A^{SFS}(n,k) = max_{P\in\mathcal{P},P=\{P_1,\ldots,P_m\}}[T_A^S(P,k)]. \tag{3}$$

and it was shown that

$$T_A^{SFS}(n,k) = T_A^S(n,k). \tag{4}$$

where $\mathcal{P} = Partitions(\sum_n)$, $\sum_n$ being the collection of permutations of the first n integers and $P \in \mathcal{P}$ is denoted by $P = \{P_1, \ldots P_m\}$ and $i \leq m \leq n!$. We define the smoothed complexity of a partition $P, P = \{P_1, \ldots, P_m\}$ of $\sum_n$ as $T^S_A(P, k) = max_{i \in \{1,\ldots,m\}} \overline{T}_A(Pert_{k,n}(P_i))$, where $Pert_{k,n}(P_i)$ is the set of outputs from permutations of P_i after perturbation of k element of the permutation of size n. Smoothed complexity is the restriction of the fair smoothed complexity to a single partition P of $\sum_n$ consisting of all singletons, i.e. $P = \{\{s_1\}, \ldots, \{s_{n!}\}\}$, which we refer to as the *singleton base*.

As in [23] and given eq. 4 we will compute the smoothed complexity relying on a different base than the singleton base which we refer to as a *modular base*. Any random bag preserving operation has a modular base, consisting of the partition of it's input data that consists of inverse images of the random structures in its output random bag. For the split operation of median-of-three Quicksort, the modular base will consist exactly of partitions for which the elements consist of the lists which share the same pivot element.

2. Background

2.1. Compositionality of timing measures

Let $A; B$ represent the sequential execution of algorithm A followed by algorithm B, where A operates on the input multiset $\mathcal{I}$ and produces the output multiset $A(\mathcal{I})$.

The worst-case time satisfies the following compositionality inequality:

$$(*)\ T^W_{A;B}(\mathcal{I}) \leq T^W_A(\mathcal{I}) + T^W_B(A(\mathcal{I}))$$

The average-case time satisfies the following compositionality equality [18, 22].

$$(**)\ \overline{T}_{A;B}(\mathcal{I}) = \overline{T}_A(\mathcal{I}) + \overline{T}_B(A(\mathcal{I}))$$

For the average-case time measure, we focus on finite input sets. This corresponds to conventions of traditional average-case analysis. Indeed, for the discrete case, inputs are identified up to order isomorphism yielding a finite amount of "states" used during the analysis. For the case of lists of size n, the analysis is reduced to considering the $n!$ cases of input lists as opposed to the potentially infinite collection inputs. The argument, as usual, relies on the assumption that the algorithm under consideration runs in the same

time on lists that satisfy the same relative order between elements. This will be the case for the algorithms we consider.

We recall the proof from [18] (where the multiset $\mathcal{I}$ is finite):

$$\begin{aligned}\overline{T}_{A;B}(\mathcal{I}) &= \frac{\sum_{I\in\mathcal{I}} T_{A;B}(I)}{|\mathcal{I}|} \\ &= \frac{\sum_{I\in\mathcal{I}} T_A(I) + \sum_{J\in\mathcal{O}_A(\mathcal{I})} T_B(J)}{|\mathcal{I}|} \\ &= \overline{T}_A(\mathcal{I}) + \overline{T}_B(\mathcal{O}_A(\mathcal{I})),\end{aligned}$$

where the last equality follows from the fact that $|\mathcal{I}| = |\mathcal{O}_A(\mathcal{I})|$ (where the collections involved are multisets).

For randomness preserving algorithms it is possible to represent the multiset $\mathcal{O}_A(\mathcal{I})$ as a random bag. For such randomness preserving algorithms, the following compositionality theorem holds.

Theorem 1. *(Compositionality Theorem, [18, 22]) Consider random bag preserving programs/operations P and Q, where we execute P on a random bag R, producing random bag R'.*

- *The average-case time of the sequential execution of P followed by Q is:*
$$\overline{T}_{P;Q}(R) = \overline{T}_P(R) + \overline{T}_Q(R').$$
- *Consider random bag $R = \{(R_1, K_1), \ldots, (R_n, K_n)\}$, then:*
$$\overline{T}_P(R) = \sum_{i=1}^{n} Prob_i \times \overline{T}_P(R_i)$$
where
$$Prob_i = Prob[F \in R_i] = \frac{K_i|R_i|}{\sum_{i=1}^{n} K_i|R_i|} = \frac{K_i|R_i|}{|R|}$$
where F is any labeling belonging to the random structure R_i.
- *For the particular case where $R = \{(R_1, K_1)\}$, the previous equality reduces to:*
$$\overline{T}_P(R) = \overline{T}_P(R_1).$$

The compositionality theorem will be used to derive the modular smoothed complexity of median-of-three Quicksort.

The compositionality theorem has been fruitfully applied to design new static average-case timing tools. We refer the interested reader to [18].

Here, we focus on exploring modular smoothed analysis, applied to the median-of-three Quicksort algorithm.

2.2. Smoothed complexity and the partial permutation model

Smoothed analysis considers inputs that are subject to some random permutation. The *smoothed measure* of an algorithm acting on a given input is the average running time of the algorithm over the perturbations of that instance, while the *smoothed complexity* of the algorithm is the worst smoothed measure of the algorithm on any input instance. The degree of perturbation is measured by a parameter σ. As σ becomes very small, the perturbations on the input become insignificant, and the smoothed complexity tends towards the worst-case running time. As σ becomes large, the perturbations become more significant than the original instance and the smoothed complexity tends towards the average-case running time.

In general, the smoothed complexity is a function of σ which interpolates between the worst case and average case running times. The dependance on σ gives a sense of how improbable an occurrence of the worst case input actually is. Formally, we have the following definition for smoothed complexity:

Definition 1. [16] Given a problem P with input domain $\mathcal{D} = \mathcal{U}_n \mathcal{D}_n$ where $\mathcal{D}_n$ represents all instances whose input size is n. Let $\mathcal{R} = \bigcup_{n,\sigma} \mathcal{R}_{n,\sigma}$ be a family of perturbations where $\mathcal{R}_{n,\sigma}$ defines for each $x \in \mathcal{D}_n$ a perturbation distribution of x with magnitude σ. Let A be an algorithm for solving P. Let $T_A(x)$ be the complexity for solving an instance $x \in \mathcal{D}_n$.[8]
The smoothed complexity: $T_A^S(n)$ of the algorithm A is defined by:

$$T_A^S(n, \sigma) = max_{x \in \mathcal{D}_n} \big(\mathbb{E}_{y \leftarrow \mathcal{R}_{n,\sigma}(x)} [T_A(y)] \big),$$

where $y \leftarrow \mathcal{R}_{n,\sigma}(x)$ means y is chosen according to distribution $\mathcal{R}_{n,\sigma}(x)$. The smoothed complexity is the worst of smoothed measures of A on inputs of

[8] In our context: The algorithms will be comparison based and $T_A(x)$ will be the running time of A input x, measured in the number of comparisons A will carry out when computing the output on input x.

size n of the expected value (average time) of algorithm A on the family of perturbations of x, namely the set $\mathcal{R}_{n,\sigma}(x)$ = smoothed complexity measure.

The method of partial permutations to study the smoothed complexity of discrete data was first proposed in [4], who defined it as follows:

Definition 2. [4] *Partial Permutations:* This model applies to problems defined on sequences. It is parameterized by a real parameter σ with $0 \leq \sigma \leq 1$ and is defined as follows. Given a sequence $s_1, s_2, \ldots s_n$ each element is selected (independently) with probability σ. Let k be the number of selected elements (on average $k = \sigma n$). Choose one of the $m!$ permutations of m elements (uniformly at random) and let it act on the selected elements.

Example 1. [4] For $\sigma = 1/2$ and $n = 7$, one might select $m = 3$ elements(namely s_2, s_3 and s_7) out of an input sequence $(s_1, \overline{s_2}, s_3, \overline{s_4}, s_5, s_7\overline{s_7})$. Applying the permutation (312) to the selected elements yields

$$(s_1, \overline{s_7}, s_3, \overline{s_4}, s_5, s_6\overline{s_4}).$$

As stated in the introduction, the natural model of permutation on a sequence for recursive algorithms applies partial permutations at each call of the recursive algorithm. The input partitions and their perturbations are tracked throughout the computation and the newly produced outputs will be perturbed yet again, and passed on to the next basic operation involved in the algorithm. For this reason, we modify the partial permutation model defined in [4] and define a new model. We refer to our model as the recursive partial permutation model, and define it as follows:

Definition 3. *Recursive Partial Permutations:* This model applies partial permutations at each successive call of the recursive algorithm.

In $\mathcal{MOQA}$ we typically assume that all data (LPO = labelled partial order) has been created from the atomic random structures $\mathcal{A}_n (n \geq 1)$ These random structures can be represented (after identification up to label isomorphism) as the collection of permutations of the first n integers, denoted by $\sum_n$ [9]

[9]Note: $\mathcal{MOQA}$ programs can operate on arbitrary random structures or random bags, provided certain rules are respected. We consider in first instance $\mathcal{A}_n = \sum_n$ in particular since we analyze quicksort whose inputs stem from $\sum_n$.

For the case of $\sum_n$ we can carry out the following simplification. Definition 2 can be simplified to a random selection of k elements among an input permutation of size n, where for sufficiently large n the number of selected elements k will be close to $n\sigma$, i.e. $\frac{n}{k} \approx \sigma$.

In essence, this simplification amounts to focusing on the expected outcome of selecting the elements with probability σ, which in case of an outcome of k elements is $\frac{k}{n}$, where the expected outcome is a selection of k elements (and other outcomes become negligible in chance.) The formalization can be based on a similar argument as is presented in [13] relying on Chernoff bounds. The motivation for relying in our arguments on this simplification is that considering σ to be of the form $\frac{k}{n}$ allows for the expression of the smoothed complexity via a recurrence equation in terms of n and k.

Taking account of the above, from here on we focus, for inputs of size n from $\sum_n$, on probabilities $\sigma = \frac{k}{n}(k \geq 0, k \leq n)$ and on partial permutations (perturbations of magnitude σ). Our definition of partial permutations now becomes:

Definition 4. A σ-partial permutation of s is a random sequence $s' = (s'_1, s'_2, \ldots s'_n)$ obtained from $s = (s_1, s_2, \ldots s_n)$ in two steps.

1. k elements of s are selected at random, where $k \geq 0, k \leq n$.
2. Choose one of the $k!$ permutations of these elements (uniformly at random)and rearrange then in that order, leaving the positions of all the other elements fixed.

We now adapt our notation in definition 1 according to our new definition above:

For $\sigma = \frac{k}{n}$, $\mathcal{R}_{n,\sigma}$ can be denoted as $\mathcal{R}_{k,n}$, the collection of partial permutations of size n that permute k out of n elements and leave the others fixed. If $s \in \sum_n$ and $t \in \mathcal{R}_{k,n}(\mathcal{R}_{k,n}$ will also be denoted as $\sum_{k,n})$ then $t \circ s$ denotes the effect of carrying out the partial permutations t on the permutation s.

The average time $\overline{T}$ of an algorithm A on an input collection $\mathcal{I} \subset \mathcal{D}_n$ (in our case $\mathcal{D}_n = \sum_n$) is

$$\overline{T}_A(\mathcal{I}) = \frac{\sum_{i \in \mathcal{I}} T_A(i)}{|\mathcal{I}|}.$$

$$\overline{T}_A(n) = \overline{T}_A(\mathcal{D}_n) = \frac{\sum_{s \in \sum_n} T_A(s)}{n!} \tag{5}$$

The definition of the smoothed complexity now simplifies:

$$T_A^S(n,k) = max_{s \in \sum_n}(\overline{T}_A(Pert_{k,n}(\mathcal{A}))) \tag{6}$$

where $Pert_{k,n}(\mathcal{A}) = \{t \circ s | s \in \mathcal{A}, t \in \sum_{k,n}\}$ is a multiset.

Lemma 2. $Pert_{k,n}(\sum_n) = \left\{ \left(\sum_n, \binom{n}{k} k!n! \right) \right\}$

Proof. The proof is left as an exercise. □

The result shows that the traditional smoothed measure is sample-fair in the above sense. The singleton base has been shown to suffice for a fair representation of all smoothed measures. In other words, the collection of all partitions can be replaced by the singleton base in practice.

As in [23], the notion of a base will play a central role in our approach to modular smoothed complexity. The modular smoothed complexity has been formulated to reflect "modular-fairness", where perturbations are systematically applied to all inputs of each basic operation of an algorithm, rather than to the algorithm's original inputs only.

In the following we will illustrate how the choice of a different base can be used to define the notion of a modular smoothed complexity $T_A^{MS}(n,k)$.

Intuitively, the modular smoothed complexity is defined similarly to the traditional smoothed complexity, for the case of recursive algorithms, with the distinction:

- the definition uses an alternative partition, referred to as the *modular base*.

We will formalize the approach below.

3. Modular smoothed analysis of Median-of-three Quicksort

In [23] we analysed the smoothed complexity of Quicksort, where the pivot chosen was the first element of the permutation/list. However, in [14], a median-of-three modification of Quicksort is analysed. It was first suggested by Hoare [5], however Sedgewick shows in [14] the percentage saving using a median-of-three modification. In [14], the first, middle and last elements were chosen as the three elements, and the median of those three elements as the pivot. We differ from this for reasons of randomness preservation. We

look at median-of-three Quicksort with all possible choices of the three pivot elements. We note that as with Quicksort, partial permutations on individual inputs do not have the property that recursive calls result in subproblems that are random permutations. Thus, again for this reason we focus on identifying collections of inputs on which partial permutations lead to random subproblems on which split can be called. For Quicksort, we considered partitions of inputs $\sum_n$, $P_1, \ldots P_l = \sum_n \forall i, j \quad i \neq j \implies P_i \cap P_j = \phi$. We referred to $\{P_1, \ldots P_l\}$ as the base of the operation. For Quicksort, the base of the split is the partition

$$(\sum_{i=1}^{n})_{i=\{1,\ldots n\}} = \{s \in \sum_n | s_1 = i\}.$$

We will consider a similar base for median-of-three Quicksort, however we note that the set of possible pivots is restricted for median-of-three Quicksort, since for a permutation of the first n integers a pivot of 1 or n is not possible. In order to preserve randomness properties we must consider partitions on the set of permutations arising from k-partial perturbations on permutations choosing all possible pivot selections. We note that the base sets that are used for Quicksort and median-of-three Quicksort are not necessarily unique. We introduce the notation for the base of the split of median-of-three Quicksort:

We consider partitions of inputs $\sum_n$, over all possible pivot selections. We denote $\sum_n^{(i,j,k)}$, as the set of all permutations, where (i, j, k) are the i^{th}, j^{th} and k^{th} positions of the elements $(1 \leq i < j < k \leq n)$ which form the three elements from which the median is chosen as the pivot. We partition the multiset of all possible choices of (i, j, k) into subsets.

Definition 5. $Pert_{k,n}^{m}(\sum_n^{(i,j,k)}), 2 \leq m \leq n-1$ is the multiset of permutations with pivot m arising after k-partial permutations on the set $\sum_n^{(i,j,k)}$.

Example 2. For $n = 5$ and $k = 2$ we have the following random bags under the Split operation:

$$Split : Pert_{2,5}^{2}(\sum_5^{(i,j,k)}) \longrightarrow \{(R_2, 960), (R_3, 240), (R_4, 80)\}$$
$$Split : Pert_{2,5}^{3}(\sum_5^{(i,j,k)}) \longrightarrow \{(R_2, 160), (R_3, 1920), (R_4, 160)\}$$
$$Split : Pert_{2,5}^{4}(\sum_5^{(i,j,k)}) \longrightarrow \{(R_2, 80), (R_3, 240), (R_4, 960)\}$$

We will show that split is a random bag preserving operation on each base element (ie $Pert^m_{k,n}(\sum_n^{(i,j,k)})$). As $Pert^m_{k,n}(\sum_n^{(i,j,k)})$ strictly speaking yields a set of perturbations that is not immediately representable as a random bag, we abuse our terminology. Instead we will relax the terminology as follows: An operation A is "random bag inducing" on a collection of inputs I when the output multiset yields a random bag R, $A : I \to R$. We now present the following theorem:

Theorem 3. *Split is random bag inducing on each*

$$Pert^{n-1}_{k,n}(\sum\nolimits_n^{(i,j,k)}).$$

More precisely

$$Split : Pert^{n-1}_{k,n}(\sum\nolimits_n^{(i,j,k)}) \longrightarrow \{(R(P[1,n{-}2]), K_2^{n-1}), \ldots (R(P[n{-}2,1]), K_{n-1}^{n-1})\}$$

for $n \geq 5, k \geq 4$, where K_j^{n-1} for $2 \leq j \leq n-2$ and for $j = n-1$ are calculated by

$$\binom{n}{3}\frac{3!(n-3)!2!(k-2)!}{(n-j)!(j-2)!}\Bigg\{2(k-1)\binom{n-4}{k-2} + 2(k+1)\binom{n-4}{k-3} + 3(n-j)\binom{n-4}{k-4}\Bigg\} \quad (7)$$

and K_{n-1}^{n-1} by

$$\binom{n}{3}3!\Bigg\{k!\binom{n-3}{k} + (k-1)!\binom{n-3}{k-1}(2+k) + 2!(k-2)!\binom{n-3}{k-2}(2k-1) + 3!(k-2)!\binom{n-3}{k-3}\Bigg\} \quad (8)$$

Proof. See appendix A for our counting argument and the calculation of the multiplicities.

□

Example 3. For $n = 5, k = 4$ we have:

$$\begin{aligned} K_j^4 &= \frac{(10)3!(2)!2!(2)!}{(5-j)!(j-2)!}\left(2(5)\binom{1}{1} + (5-j)\{3\binom{1}{0}\}\right) \\ &= \frac{480}{(5-j)!(j-2)!}(10 + 3(5-j)), \quad 2 \leq j \leq 3, \end{aligned}$$

and

$$\begin{aligned} K_4^4 &= (10)3!\left\{2!(2)!\binom{2}{2}(7) + 3!(1)!\binom{2}{1}(2)\}\right\} \\ &= 60(28+24) \end{aligned}$$

Thus split on $Pert_{4,5}^4(\sum_5^{(i,j,k)})$ gives

$$\{(R(P[1,3]), 1520), (R(P[2,2]), 3840), (R(P[3,1]), 3120)\}.$$

Now that we have established the effect on our base elements of split, the central operation in median-of-three Quicksort, we take a similarly approach to that in [23] where we followed the compositional analysis of Quicksort presented in [21]. We have the following lemma from [21](pg 27)

Lemma 4. *When X ranges over $\sum_n$, the multiset of restrictions of Split(X) to I^{Lower} is a Random Bag*

$$\{R(\Delta_{j-1}), (j-1)!\} = \{\sum\nolimits_{j-1}, (j-1)!\}$$

by a labeling isomorphism. Similarly for I^{Upper}:

$$\{(R(\Delta_{n-j}), (j-1)!)\} = \left\{(\sum_{n-j}, (j-1)!)\right\}.$$

As with $\mathcal{MOQA}$ Quicksort, we again take the maximum over the averages in this approach giving

$$T_{QS_3}^{MS}(n,k) = max_{m\in\{2,\ldots n-1\}} T_{QS_3}\big(Pert_{k,n}^m(\sum\nolimits_n^{(i,j,k)})\big) \tag{9}$$

where

$$\begin{aligned} T_{QS}\big(Pert^m_{k,n}(\sum_n^{(i,j,k)})\big) &= \overline{T}_{Split}\big(Pert^m_{k,n}(\sum_n^{(i,j,k)})\big) \\ &+ T_{QS_3}(\{(R(\Delta_0),N_i),\ldots(R(\Delta_{n-1}),N^i_n)\}) \\ &+ T_{QS_3}(\{(R(\Delta_n),N_i),\ldots(R(\Delta_0),N^i_1)\}) \end{aligned}$$

where $N^i_j = K^i_j L_j$ and $L_j = (n-j)!$ Recall from Theorem 25 in [21]. $R = \{(R_1,K_1),\ldots(R_p,K_p)\}$ is a random bag. We have

$$\overline{T}_p(R) = \sum_{i=1}^{p} prob_i \overline{T}_p(R_i) \tag{10}$$

where

$$\begin{aligned} Prob_i &= Prob[F \in R_i] \\ &= \frac{K_i|R_i|}{|R|} \end{aligned}$$

Split takes $n-1$ comparisons on every permutations of size n, so:

$$\overline{T}_{Split}\big(Pert_{k,n}(\sum_n^{(i,j,k)})\big) = n-1 \tag{11}$$

Lemma 5.

$$max_{m\in\{2,\ldots n-1\}} T_{QS_3}\big(Pert^m_{k,n}(\sum_n^{(i,j,k)})\big) = T_{QS_3}\big(Pert^{n-1}_{k,n}(\sum_n^{(i,j,k)})\big)$$

Proof. The proof is left as an exercise □

Thus $T^{MS}_{QS_3}(n,k)$

$$\begin{aligned} max_{m\in\{2,\ldots n-1\}} \Big[&(n-1) + 1^{10} \\ &+ \sum_{j=2}^{n-1} \beta^m_{n+1-j} T^{MS}_{QS_3}\big(Pert^m_{k,n}(\sum_n^{(i,j,k)})\big) + \sum_{j=2}^{n-1} \beta^m_j T^{MS}_{QS_3}\big(Pert^m_{k,n}(\sum_n^{(i,j,k)})\big)\Big], \end{aligned} \tag{12}$$

simplifies to

$$(n-1)+1 \\ +\sum_{j=2}^{n-1}\beta_{n+1-j}^{n-1}T_{QS_3}^{MS}\big(Pert_{k,n}^{n-1}(\sum_{n}^{(i,j,k)})\big)+\sum_{j=1}^{n}\beta_{j}^{n-1}T_{QS_3}^{MS}\big(Pert_{k,n}^{n-1}(\sum_{n}^{(i,j,k)})\big). \tag{13}$$

Let $f(n,k)=T_{QS_3}^{MS}(n,k)$. We now have the following equation

$$f(n,k)=\left[(n-1)+1+\sum_{j=2}^{n-1}\beta_{n+1-j}^{n-1}f(j-1,k)+\sum_{j=2}^{n-1}\beta_{j}^{n-1}f(j-1,k)\right] \tag{14}$$

Definition 6. The recurrence equation for median-of-three quicksort is:

$$f(n,k)=(n-1)+1+\sum_{j=2}^{n-1}\beta_{n+1-j}^{n-1}f(j-1,k)+\sum_{j=2}^{n-1}\beta_{j}^{n-1}f(j-1,k) \tag{15}$$

where

$$\beta_{n-1}^{i}=\frac{N_{n-1}^{i}|\sum_{j-1}|}{|Pert_{k,n}^{n-1}(\sum_{n}^{(i,j,k)})|},$$

is the probability that a labeling belongs to $R(\Delta_{j-1})$ in the random bag $\{(R(\Delta_2),N_2^{n-1}),\ldots(R(\Delta_{n-1}),N_{n-1}^{n-1})\}$.

$$\beta_{j}^{n-1}=\frac{2!(j-1)}{k(n-2)\binom{n}{k}}\left(2\binom{n-4}{k-2}+2\frac{(k+1)}{(k-1)}\binom{n-4}{k-3}+\frac{3(n-j)}{(k-1)}\binom{n-4}{k-4}\right), \tag{16}$$

for $2\leq j\leq n-2$ and β_{n-1}^{n-1} is

$$\left\{k!\binom{n-3}{k}+(k-1)!\binom{n-3}{k-1}(2+k) \right. \\ \left. +2!(k-2)!\binom{n-3}{k-2}(2k-1)+3!(k-2)!\binom{n-3}{k-3}\right\}\frac{(n-k)!}{n!} \tag{17}$$

Proof. The recurrence formula for median-of-three Quicksort is calculated similarly to that for Quicksort which we saw in the last section. However, we make a cost allowance for comparing the three elements from which we choose our pivot. Firstly, let us calculate $|Pert_{k,n}^{n-1}(\sum_n^{(i,j,k)})|$

Definition 7.

$$|Pert_{k,n}^{n-1}(\sum_{n}^{(i,j,k)})| = \binom{n}{3}\frac{3!(n-2)!n!}{(n-k)!} \tag{18}$$

Proof. The total number of permutations after k-partial permutations on permutations of length n is

$$n!\binom{n}{k}k!.$$

The proportion of total permutations with the $(n-1)^{th}$ pivot can be calculated as

$$\frac{3!(n-3)!(n-2)}{n!}.$$

Thus, the total permutations after k partial permutations with pivot $(n-1)$, over all possible pivot choices is

$$\begin{aligned} |Pert_{k,n}^{n-1}(\sum_{n}^{(i,j,k)})| &= \binom{n}{3}\frac{3!(n-3)!(n-2)}{n!}n!\binom{n}{k}k! \\ &= \binom{n}{3}\frac{3!(n-2)!n!}{(n-k)!} \end{aligned}$$

□

Secondly, the calculations for the beta values are shown below:
Calculating β_j^{n-1} for $2 \leq j \leq n-2$

$$\begin{aligned} \beta_j^{n-1} &= \frac{N_j^{n-1}|\sum_{j-1}|}{|Pert_{k,n}(\sum_n^{(i,j,k)})|} \\ &= \frac{2!(j-1)}{k(n-2)\binom{n}{k}}\left(2\binom{n-4}{k-2} + 2\frac{(k+1)}{(k-1)}\binom{n-4}{k-3} + (n-j)\frac{3}{(k-1)}\binom{n-4}{k-4}\right) \end{aligned}$$

For β_{n-1}^{n-1} we have the following:

$$\begin{aligned}\beta_{n-1}^{n-1} &= \frac{N_{n-1}^{n-1}|\sum_{j-1}|}{|Pert_{k,n}(\sum_n^{(i,j,k)})|} \\ &= \left(k!\binom{n-3}{k} + (k-1)!\binom{n-3}{k-1}(2+k) + \right. \\ &\quad \left. 2!(k-2)!\binom{n-3}{k-2}(2k-1) + 3!(k-2)!\binom{n-3}{k-3}\right)\frac{(n-k)!}{n!}\end{aligned}$$

□

The modular smoothed complexity of Median-of-three Quicksort, $f(n,k)$, is summarised below,

$$f(n,k) = (n-1) + 1 + \sum_{j=2}^{n-1} \beta_{n+1-j}^{n-1} f(j-1,k) + \sum_{j=2}^{n-1} \beta_j^{n-1} f(j-1,k) \quad (19)$$

where β_j^{n-1} for $2 \leq j \leq n-2$ is

$$\frac{2!(j-1)}{k(n-2)\binom{n}{k}}\left(2\binom{n-4}{k-2} + 2\frac{(k+1)}{(k-1)}\binom{n-4}{k-3} + \frac{3(n-j)}{(k-1)}\binom{n-4}{k-4}\right),$$

and β_{n-1}^{n-1} is

$$\left\{k!\binom{n-3}{k} + (k-1)!\binom{n-3}{k-1}(2+k) + 2!(k-2)!\binom{n-3}{k-2}(2k-1)\right.$$
$$\left. +3!(k-2)!\binom{n-3}{k-3}\right\}\frac{(n-k)!}{n!}.$$

4. Results and conclusion

Here we have presented a closed form equation for measuring the smoothed complexity of median-of-three Quicksort. From this equation we can find

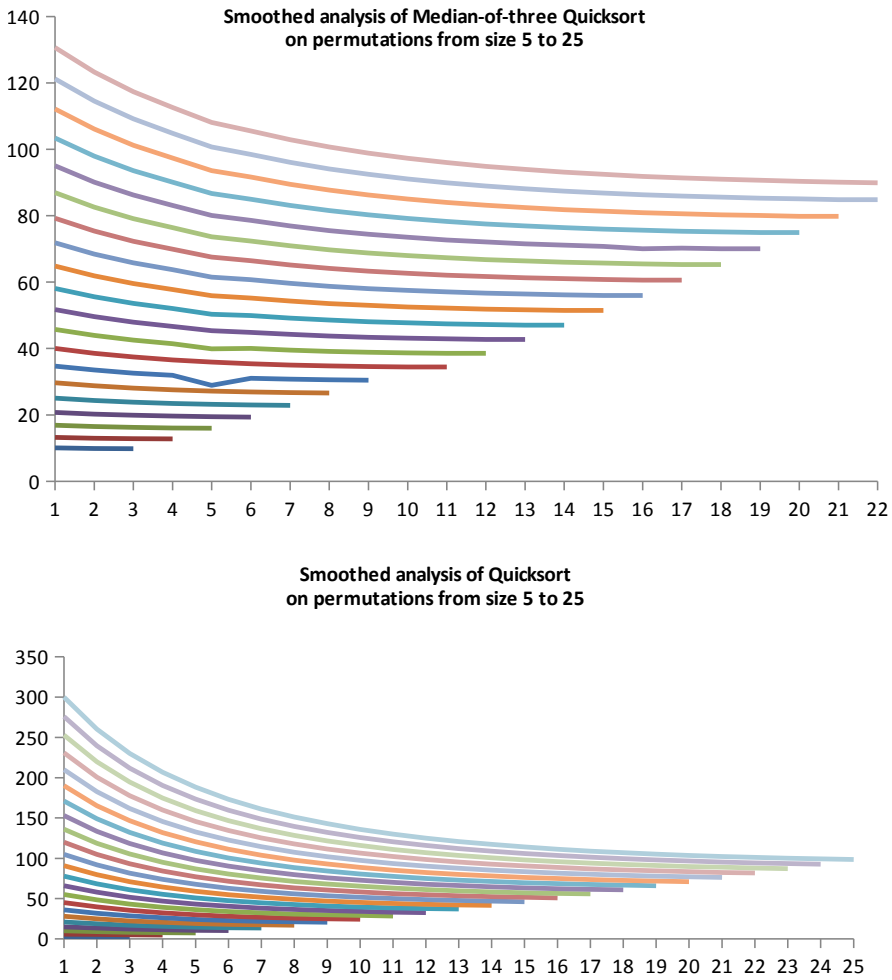


Figure 1: The modular smoothed complexity of Quicksort v Median-of-three for the first 25 values of n

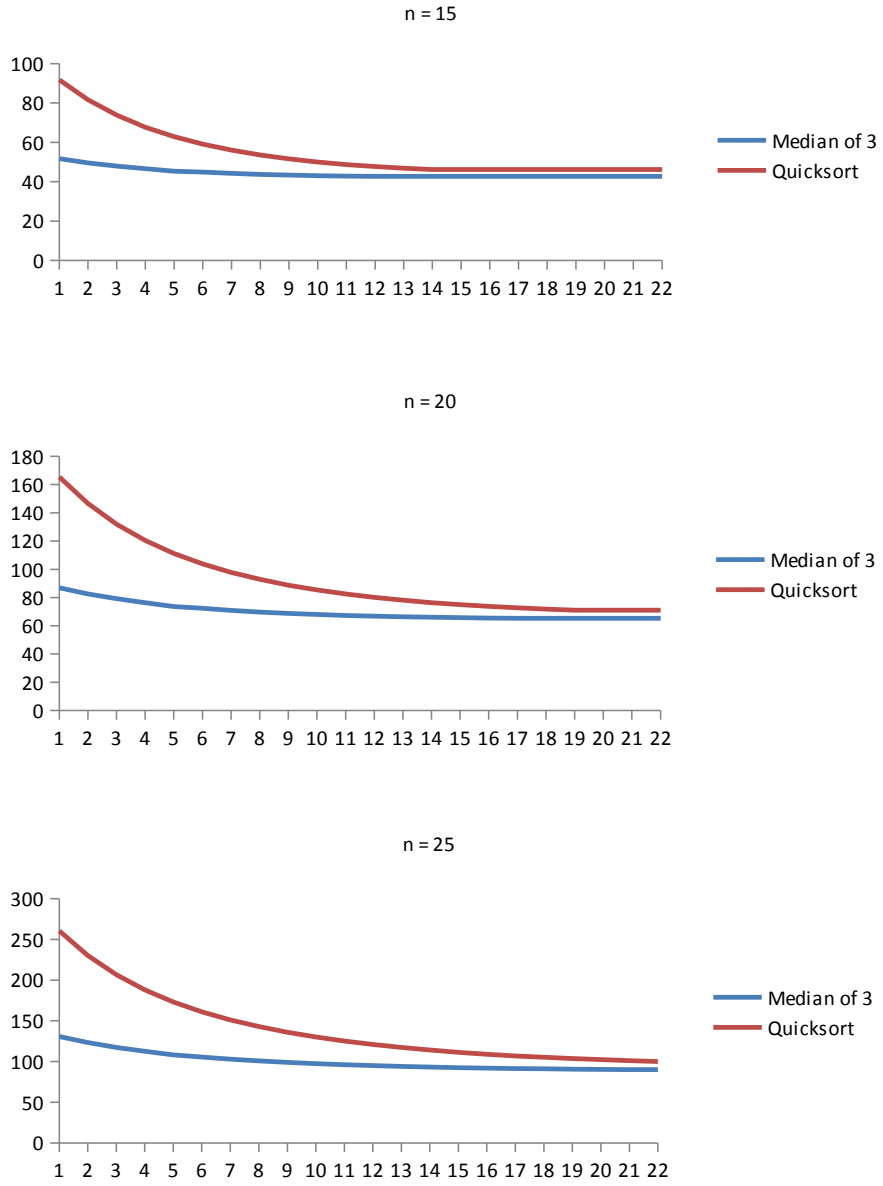


Figure 2: The modular smoothed complexity of Quicksort v Median-of-three, for $n = 15, 20$ and 25 for increasing k

more precise bounds for the timing of such randomness preserving algorithms. Figure 1 shows the timing for $n = 15, 20$ and 25 for Quicksort versus Median-of-three Quicksort. Figure 2 shows the modular smoothed complexity of Quicksort and the modular smoothed complexity of median-of-three Quicksort for permutations of length $1 \leq n \leq 25$ for increasing values of the k. The saving using a median-of-three modification is clearly seen with the recursive permutation model having a greater impact on the unmodified Quicksort. With the closed from equations for both, we can quantify this impact for the various perturbations of k.

Appendix A.

Here, we calculate the input and output permutations of the random bag with the pivot $n-1$ before perturbation. We split this calculation into two parts, the first part calculates the permutations with pivot j where $2 \leq j \leq n-2$ and the second part calculates the permutations with pivot $n-1$.

1. To calculate the permutations arising after perturbation we need to split the permutations into groups. After perturbation we want to count the permutations arising with pivot j.

   The total number of permutations with pivot $(n-1)$ is

   $$3!(n-3)!\binom{1}{1}\binom{1}{1}(n-2).$$

   We split into groups tracking the position of element j in the count. This grouping enables us to count the sum of the output permutations with pivot j after perturbation. The groups are formed as follows:
   (a) Input permutations with three pivot elements n, $n-1$ and x where $x < j$. There are $\binom{j-1}{1}\binom{1}{1}\binom{1}{1}3!(n-3)!$ such permutations.
   (b) Input permutations with three pivot elements n, $n-1$ and j. There are $\binom{1}{1}\binom{1}{1}\binom{1}{1}3!(n-3)!$ such permutations.
   (c) Input permutations with three pivot elements n, $n-1$ and x where $x > j$. There are $\binom{n-j-2}{1}\binom{1}{1}\binom{1}{1}3!(n-3)!$ such permutations.

   We count the number of permutations arising with pivot j from each of the groups above. We do this by counting the total possible ways the pivot j can be the median of the three elements, resulting in the equation below:
   after some simplification and rearranging we have M_j for $n \geq 5, k \geq 4$,

   $$3!(n-3)!(j-1)2!(k-2)!\left(2(k-1)\binom{n-4}{k-2}+2(k+1)\binom{n-4}{k-3}+3(n-j)\binom{n-4}{k-4}\right), \tag{A.1}$$

   We now calculate the multiplicities of the j^{th} random structure, where $2 \leq j \leq n-2$, in the random bag with pivot $(n-1)$ to be

   $$K_j^{n-1} = \frac{M_j}{(n-j)!(j-1)!}$$

Thus, for $n \geq 5, k \geq 4$, $2 \leq j \leq n-2$ we have K_j^{n-1} as

$$\frac{3!(n-3)!2!(k-2)!}{(n-j)!(j-2)!}\left(2(k-1)\binom{n-4}{k-2}+2(k+1)\binom{n-4}{k-3}+3(n-j)\binom{n-4}{k-4}\right) \tag{A.2}$$

2. We now calculate the permutations after perturbation with pivot $n-1$. Thus, this group have the same pivot before and after perturbation. We have the following result:

$$\begin{aligned} M_{n-1} &= 3!(n-3)!(n-2)\Bigg\{k!\binom{n-3}{k}+(k-1)!\binom{n-3}{k-1}(2+k) \\ &\quad +2!(k-2)!\binom{n-3}{k-2}(2k-1)+3!(k-3)!\binom{n-3}{k-3}(k-2)\}\Bigg\} \end{aligned}$$

for $n \geq 5, k \geq 4$. We now calculate the multiplicities of the $(n-1)^{th}$ random structure,

$$K_{n-1}^{n-1} = \frac{M_{n-1}}{(n-j)!(j-1)!}$$

Thus, for $n \geq 5, k \geq 4$, we have K_{n-1}^{n-1} as

$$3!\Bigg\{k!\binom{n-3}{k}+(k-1)!\binom{n-3}{k-1}(2+k)+2!(k-2)!\binom{n-3}{k-2}(2k-1)+3!(k-2)!\binom{n-3}{k-3}\}\Bigg\} \tag{A.3}$$

References

[1] A.Aho, J.Hopcroft and J. Ullman, Data Structures and Algorithms, Addison-Wesley Series in Computer Science and Information Processing, Addison-Wesley, 1987.

[2] A. Gao. Modular Average Case Analysis: Language Implementation and Extension, PhD thesis, University College Cork, 2013.

[3] A. Gao, K. Rea, and M. Schellekens, Static Average Case Analysis Fork-Join Framework Programs Based On $\mathcal{MOQA}$ Method, 6th International Symposium on Parallel Computing in Electrical Engineering, accepted for publication, Luton, UK, April 2011.

[4] C. Banderier, R. Beier, and K. Mehlhorn. Smoothed analysis of three combinatorial problems. In the 28th International Symposium on Mathematical Foundations of Computer Science, pages 198207, 2003.

[5] Hoare, C.A.R. Partition: Algorithm 63; Quicksort: Algorithm 64 and Find: ALgorithm 65 *Comm. ACM 4*, 7 (July 1961), 321–322.

[6] S. Edelkamp, Weak-Heapsort, ein schnelles sortierverfahren, Diplomarbeit Universität Dortmund, 1996.

[7] D. Hickey, Distritrack: Automated Average-Case Analysis, in the proceedings of the Fourth International Conference on the Quantatative Evaluation of Systems (QEST 2007), 17-19 September 2007, Edinburgh, Scotland, UK.

[8] D. Hickey, D. Early and M. Schellekens, A Tool for Average-Case and Worst-Case Execution Time Analysis, in proceedings of the Worst-Case Execution Time Workshop, satelite event of the Euromicro conference on Real-Time Systems, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (publisher), Germany, 2008.

[9] D. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching, Addison-Wesley, 1998.

[10] M. Li, P. Vitanyi, *An introduction to Kolmogorov Complexity and its Applications*, Texts and Monographs in Computer Science, Springer Verlag, 1993.

[11] R. Motwani, P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

[12] T. Maibaum, Mathematical Foundations of Software Engineering: a roadmap, Proceedings of the Conference on The Future of Software Engineering, ICSE00, 161 - 172, 2000.

[13] M. Mitzenmacher, E. Upfal, Probability and Computing: Randomized Algorithms and Probabilistic Analysis, Cambridge University Press, Cambridge, 2005.

[14] R. Sedgewick, Implementing quicksort programs, Comm. ACM **21** (10), 847–857, 1978.

[15] R. Schaffer and R. Sedgewick, The Analysis of Heapsort, Journal of Algorithms **15**(1), 76–100, 1993.

[16] D. Spielman, S. Teng, Smoothed Analysis: Why The Simplex Algorithm Usually Takes Polynomial Time, Journal of the ACM, Vol 51 (3), pp. 385 - 463, 2004.

[17] D. Spielman, S. Teng, Smoothed Analysis of Algorithms and Heuristics, Foundations of Computational Mathematics Santander 2005, London Mathematical Society Lecture Note Series, no. 331, Cambridge University Press, 274 - 342, 2006.

[18] M. P. Schellekens,"A Modular Calculus for the Average Cost of Data Structuring", Springer book, published in August, 2008.

[19] M. P. Schellekens, $\mathcal{MOQA}$ Unlocking the potential of compositional average-case analysis, Journal of Logic and Algebraic Programming, Volume 79, Issue 1, January 2010, Pages 61-83.

[20] D. Spielman, Commentary on Smoothed Analysis of Three Combinatorial Problems, published electronically at http://www.cs.yale.edu/homes/spielman/SmoothedAnalysis, 2003.

[21] M. Schellekens, G. Bollella and D. Hickey. $\mathcal{MOQA}$ a Linearly-Compositional Programming Language for (semi-) automated Average-Case analysis, IEEE Real-Time Systems Symposium - WIP Session, 2004.

[22] M. Schellekens, $\mathcal{MOQA}$ Unlocking the potential of compositional average-case analysis, Journal of Logic and Algebraic Programming, Volume 79, Issue 1, January 2010, Pages 61-83.

[23] M. P. Schellekens, A. Hennessy and B. Shi, Modular Smoothed Analysis, Preprint.

[24] Diarmuid Early, Ang Gao and Michel Schellekens. "Frugal encoding in reversible $\mathcal{MOQA}$ a case study for Quicksort". 4th Workshop on Reversible Computation, Copenhagen, Denmark, 2012.

[25] Ang Gao, Aoife Hennessy, Michel Schellekens: "$\mathcal{MOQA}$ Min-Max heapify: A Randomness Preserving Algorithm". 10th International Conference Of Numerical Analysis And Applied Mathematics, Kos, Greece, 2012.

[26] Kopetz, H.; Fohler, G.; Grnsteidl, G. et al.: RealTime Systems Development: The Programming Model of MARS, in Proceedings of the International Symposium on Autonomous Decentralized Sys- tems, pp. 190 199, Kawasaki, Japan, March. 1993.

[27] Erik Yu-Shing Hu, Guillem Bernat, Andy Wellings, A Static Timing Analysis Environment Using Java Architecture for Safety Critical Real-Time Systems, Seventh IEEE International Workshop on Object-Oriented Real-Time Dependable Systems, p. 0077, 2002.

[28] D. Early, M. Schellekens, Running time of the Treapsort algorithm, Theoretical Computer Science, in press, accepted manuscript, available online from Springer Marh 25, 2013 at: http://www.sciencedirect.com/science/article/pii/S0304397513002132.

[29] D. Early, A Mathematical Analysis of the $\mathcal{MOQA}$ language, PhD thesis, University College Cork, 2010.

[30] D. Spielman, Smoothed Analysis Homepage, http://www.cs.yale.edu/homes/spielman/SmoothedAnalysis/.