Theoretical Computer Science 412 (2011) 6537–6555

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

The analysis of Range Quickselect and related problems☆

Conrado Martínez [a,*], Alois Panholzer [b], Helmut Prodinger [c]

[a] Departament de Llenguatges i Sistemes Informàtics, Unversitat Politècnica de Catalunya, E-08034 Barcelona, Spain
[b] Institut für Diskrete Mathematik und Geometrie, Technische Universität Wien, Wiedner Hauptstr. 8-10/104, 1040 Wien, Austria
[c] Department of Mathematics, University of Stellenbosch, 7602 Stellenbosch, South Africa

ARTICLE INFO

Article history:
Received 4 February 2011
Received in revised form 2 May 2011
Accepted 22 June 2011
Communicated by W. Szpankowski

Keywords:
Quickselect
Hoare's Find
Moves
Range Quickselect
Binary search trees
Average-case analysis

ABSTRACT

Range Quickselect, a simple modification of the well-known Quickselect algorithm for selection, can be used to efficiently find an element with rank k in a given range $[i..j]$, out of n given elements. We study basic cost measures of Range Quickselect by computing exact and asymptotic results for the expected number of passes, comparisons and data moves during the execution of this algorithm.

The key element appearing in the analysis of Range Quickselect is a trivariate recurrence that we solve in full generality. The general solution of the recurrence proves to be very useful, as it allows us to tackle several related problems, besides the analysis that originally motivated us.

In particular, we have been able to carry out a precise analysis of the expected number of moves of the pth element when selecting the jth smallest element with standard Quickselect, where we are able to give both exact and asymptotic results.

Moreover, we can apply our general results to obtain exact and asymptotic results for several parameters in binary search trees, namely the expected number of common ancestors of the nodes with rank i and j, the expected size of the subtree rooted at the least common ancestor of the nodes with rank i and j, and the expected distance between the nodes of ranks i and j.

1. Introduction

Quickselect, also called Hoare's Find algorithm, is a very flexible and easy to implement recursive algorithm to find the element of given rank k (i.e., the kth smallest element) in a given data array $A[1..n]$ of length n. The Quickselect algorithm uses partitioning of the array into two subarrays around a pivot element, as in the popular Quicksort, also by Hoare [5,6].

The behavior of fundamental quantities like the number of comparisons between data elements and the number of passes (recursive calls of the algorithm) in Quickselect has been extensively studied; see, for instance [4,8,10,14] and references

☆ This work was supported by the Spanish–Austrian research agreement 'Acciones Integradas', grant ES 10/2008 and by the Spanish–South African research agreement 'Acciones Integradas', grant HS2008-0003. The first author was supported by the Spanish Min. of Science and Technology, project TIN2006-11345 (ALINEX). The second author was supported by the Austrian Science Foundation FWF, grant S9608. The third author was supported by the South African Science Foundation NRF, grant 2053748. This research was also supported by the Center for Mathematical Research (CRM), Bellaterra, Spain, while the first and the third authors held research visiting positions there.

* Corresponding author.

0304-3975 © 2011 Elsevier B.V. Open access under CC BY-NC-ND license.
doi:10.1016/j.tcs.2011.06.030

therein. These quantities have also been studied for many variants of the standard algorithm, for example, for the median-of-three partitioning scheme [9]. In the present work, we consider a variant of Quickselect, that we have dubbed Range Quickselect, which receives as input the data array and a *range* $[i..j]$. Its goal is to find an element whose rank falls in the given range. The analysis of Range Quickselect poses several quite natural questions related to the Quickselect algorithm that do not seem to have been treated up until now.

Range Quickselect (RQS, for short) is useful when we are not necessarily interested in an exact order statistic, but some order statistic within a range $[i..j]$ of ranks. For example, instead of finding the exact median we could be content with an element whose rank is, say, between $0.48n$ and $0.52n$. This "relaxation" of the Quickselect algorithm will lead, depending on the range $[i..j]$, to a reduction of the number of passes and of the number of comparisons between elements in the array during the execution, and thus will lead to a faster execution time. We compute the exact average number of passes and the exact average number of comparisons between elements when executing RQS and as a consequence we can give results quantifying the average amount of savings compared to standard Quickselect. In particular, given some measure of performance X, we compare the difference between $X_{n,i}$, the average value of X corresponding to Quickselect when given an input of size n and looking for the ith smallest element, and $X_{n,i-d,i+d}$, the average value of X corresponding to Range Quickselect when given an input of size n and looking for an element whose rank falls in the range $[i - d, i + d]$. The asymptotic behavior of that difference in terms of n and d provides a clear picture of the benefits of Range Quickselect and the trade-off between speed and accuracy.

The description of the algorithm and the analysis of the expected behavior of its fundamental performance characteristics form the core of Section 2.

The analysis of Range Quickselect involves the solution of trivariate recurrences which we have been able to solve in full generality. The result (Theorem 2) that we obtain in Section 2.3 turns out be very useful in the analysis of other interesting parameters, including the number of moves of a particular element during the execution of the standard Quickselect algorithm and the total number of moves made during the execution of Range Quickselect. In particular, we give exact results for the average number of moves of the element with rank p made while selecting the jth smallest element out of n, and also for the average total number of moves during the execution of the Range Quickselect algorithm, when finding an element with rank $k \in [i..j]$ out of n (Section 3).

These parameters give a further insight into the functionality of the Quickselect algorithms and moreover, since moves of elements correspond to variable assignments in the algorithm, these quantities appear when measuring the total cost of the Quickselect algorithms.

We also want to mention here two recent related studies, one about the number of moves of particular elements in the Quicksort sorting algorithm [18] and the other on the total number of moves in Quickselect, but for a randomly chosen rank [13].

The close connection between Quickselect and random binary search trees surfaces also in this paper, like in many previous works of the area (see, for instance [17]). We establish in Section 4 the relation between Range Quickselect and several parameters in random binary search trees that involve two given nodes. We study the average number of common ancestors of the nodes with ranks i and j, the average size of the subtree rooted at the least common ancestor of the nodes with ranks i and j, and the average distance (number of edges) from the node of rank i to the node of rank j. Despite these results can be obtained (and have been obtained) by other means, we show that all of them follow from direct application of Theorem 2. This is a further example of the generality and usefulness of this tool, which qualifies as one of the important contributions of this paper.

We shall insist here that in this paper we restrict our analysis to the expected value of the quantities considered. In all cases, we shall consider that the input is an array of n distinct elements, the $n!$ possible orderings taken equally likely. This assumption is standard in the probabilistic analysis of comparison-based sorting and selection algorithms (see, for instance, [11]). Furthermore, the assumption that the input is a random permutation can be removed if we consider that the pivot of each recursive stage is picked uniformly at random among the elements of the current subarray. Indeed, whatever the initial permutation is, if we pick pivots at random then the probability that we choose the kth smallest element out of N is $1/N$ for all k, $1 \leq k \leq N$. When we assume that the source of randomness comes from the algorithm itself, expectations are with respect the random choices made by the algorithm, not by assuming any particular distribution on the inputs. Both approaches yield the same results, but we will talk in terms of the random permutation model for the rest of the article.

It is also worth mentioning that, apart from the study of the number of moves of a particular element in Quickselect where dependences between the quantities appearing in the recursive description occur (see Section 3), our analysis could, at least in principle, be extended to higher moments, most notably to the second moment and thus to the variance, although the computational effort would be considerable (see, for instance [8]).

We conclude this section with a few remarks concerning notations used in this paper. We use Iverson's bracket notation $[\![Q]\!]$ for a statement Q: $[\![Q]\!] = 1$ if Q is true and $[\![Q]\!] = 0$ otherwise [3]. The harmonic numbers are always denoted by $H_n := \sum_{k=1}^{n} \frac{1}{k}$, for a positive integer n. Moreover, the random variable $\mathbb{1}_E$ always denotes the indicator function of the event E, which gives the value 1 when E occurs and gives the value 0 otherwise. Throughout this paper we use for all quantities considered a calligraphic letter as $\mathcal{P}$, $\mathcal{C}$, etc. to denote random variables, whereas the corresponding ordinary letters denote their expectations, e.g., $P = \mathbb{E}(\mathcal{P})$.

2. Range Quickselect

2.1. The algorithm

We begin with a description of the standard Quickselect algorithm for selection. The call QuickSelect(A, j, l, r) will find the $(j - l + 1)$st smallest element amongst all elements in the array $A[l..r]$, with $1 \leq l \leq j \leq r \leq n$. In full rigor, the algorithm will return an element x of $A[l..r]$ such that there are at least $j - l + 1$ elements in the subarray which are less or equal to x. To have a neat definition of *rank*, we shall assume that the n given elements are distinct. This will simplify the discussion about the algorithms and their correctness along the paper,[1] and it is also essential for our analysis, as we have already pointed out in the introduction.

After executing this algorithm, it holds that $A[j]$ stores the element of the desired rank $j-l+1$ in $A[l..r]$; in particular, the initial call QuickSelect$(A, j, 1, n)$ will bring the jth smallest element of $A[1..n]$ to $A[j]$. Moreover, the algorithm rearranges the contents of the array in such a way that it holds that $A[m] \leq A[j]$, for all $l \leq m < j$, and $A[j] \leq A[m]$, for all $j < m \leq r$.

If $r \leq l$, the subarray contains at most one element, and the problem is trivially solved, since $A[l]$ must contain the sought element. When $l < r$, we perform a partitioning phase, in which one of the elements in the array, say $A[l]$, is chosen as a *pivot* element. By comparing this pivot element v with all remaining elements in the array and interchanging elements, the pivot element will be brought to its correct position in the array, say $A[k]$, such that all elements in the array $A[l..k-1]$ are smaller than or equal to $v = A[k]$ and all elements in the array $A[k+1..r]$ are larger than or equal to v. The partitioning algorithm is given in full detail in Section 3.1, when we analyze the number of moves carried out by Quickselect and Range Quickselect. For the time being, it is enough to note that the partitioning algorithm will make exactly $n-1 = r-l$ comparisons between the pivot and the remaining elements in the (sub)array of size n; moreover, if the subarray contains a random permutation of n elements, the two subarrays that we obtain after partitioning are random permutations too.

After the partitioning phase, three cases can occur: (1) if $j = k$ we know then that $v = A[k] = A[j]$ is the $(j - l + 1)$st smallest element in $A[l..r]$ and the algorithm terminates, (2) if $j < k$ we know that the required element is contained in the left subarray and we proceed by searching for the $(j - l + 1)$st smallest element in the array $A[l..k - 1]$ with a recursive call of Quickselect, and (3) if $j > k$ we know that the required element is contained in the right subarray and we proceed by searching for the $(j-k)$th smallest element in the array $A[k+1..r]$, again with a recursive call of Quickselect. The algorithm is detailed as Algorithm 1.

Algorithm 1 The Quickselect algorithm

Require: array $A[l..r]$, integer j with $l \leq j \leq r$
Ensure: Returns j, $A[j]$ contains the $(j - l + 1)$st smallest element in the array $A[l..r]$

```
procedure QUICKSELECT(A, j, l, r)
    if r ≤ l then return l
    end if
    PARTITION(A, l, r, k)
                                ▷ ∀m : (l ≤ m < k) ⇒ A[m] ≤ A[k], and ∀m : (k < m ≤ r) ⇒ A[k] ≤ A[m]
    if j < k then return QUICKSELECT(A, j, l, k − 1)
    else if j > k then return QUICKSELECT(A, j, k + 1, r)
    else return k
    end if
end procedure
```

Two simple modifications of the Quickselect algorithm allow us to solve the problem of range selection. Range Quickselect is given the array A, the lower and upper indices l and r that delimit the subarray that contains the elements of interest, and the values i and j that specify a range of ranks. The call RQS(A, i, j, l, r) returns a value k such that the element at $A[k]$ has a rank between $i - l + 1$ and $j - l + 1$ amongst all elements in the array $A[l..r]$, for $1 \leq l \leq i \leq j \leq r \leq n$. A call to RQS$(A, i, j, 1, n)$ returns a value k such that $A[k]$ has a rank $k \in [i..j]$ among the elements in $A[1..n]$. Like in Quickselect, it also holds that $A[m] \leq A[k]$, for all $l \leq m < k$, and that $A[k] \leq A[m]$, for all $k < m \leq r$.

Compared to the standard Quickselect algorithm we need only to make the following two modifications. First, we stop if $j - i \geq r - l$, since the subarray contains elements whose ranks are between i and j and any of them will do.[2] The other modification comes after the partitioning phase, that is, after the pivot element v is brought to its correct position $A[k]$ in the array, with all elements in the array $A[l..k-1]$ smaller than or equal to $v = A[k]$ and all elements in the array $A[k+1..r]$ larger than or equal to v. We have three cases: (1) if $i \leq k \leq j$ the pivot has a rank in the range $[i - l + 1..j - l + 1]$ and

[1] Both standard Quickselect and Range Quickselect work correctly in the presence of repeated elements; they return an element such that there are at least some number, say k, of elements smaller or equal to it.

[2] Equivalently, we stop if $i \leq l$ and $r \leq j$.

we can return k and terminate the algorithm, (2) if $j < k$ we know that each element of interest is contained in the left subarray and we continue with the selection of an element with a rank between $i-l+1$ and $j-l+1$ in the array $A[l..k-1]$ by making a recursive call of Range Quickselect on $A[i..k-1]$, and (3) if $i > k$ we know that each element of interest is contained in the right subarray and we recursively proceed looking for an element with a rank between $i-k$ and $j-k$ in the array $A[k+1..r]$. An implementation of this algorithm is given as Algorithm 2.

Algorithm 2 The Range Quickselect algorithm

Require: Array $A[l..r]$, integers i and j with $l \le i \le j \le r$
Ensure: Returns k, with $i \le k \le j$, $A[k]$ has rank between $i-l+1$ and $j-l+1$ in the array $A[l..r]$

```
procedure RQS(A, i, j, l, r)
    if r − l ≤ j − i then return l
    end if
    PARTITION(A, l, r, k)
                                  ▷ ∀m : (l ≤ m < k) ⇒ A[m] ≤ A[k], and ∀m : (k < m ≤ r) ⇒ A[k] ≤ A[m]
    if j < k then return RQS(A, i, j, l, k − 1)
    else if i > k then return RQS(A, i, j, k + 1, r)
    else return k
    end if
end procedure
```

2.2. The number of passes

We start our analysis of Range Quickselect with the average behavior of the random variable $\mathcal{P}_{n,i,j}$ which counts the number of passes, i.e., (recursive) calls, of the algorithm RQS until an element with a rank between i and j is found in an array $A[1..n]$. Here, and for the rest of the paper, as we have already discussed in the introduction, we assume that the array contains a random permutation of n distinct elements. We also assume that we choose the first element of the current subarray as the pivot of each recursive stage.

Theorem 1. *The expected number of passes $P_{n,i,j} = \mathbb{E}\left(\mathcal{P}_{n,i,j}\right)$ of the algorithm Range Quickselect until an element with a rank between i and j is found in an array of n elements is*

$$\begin{aligned} P_{n,i,j} &= H_j + H_{n-i+1} - 2H_{j-i+1} + 1, \quad \text{for } 1 \le i \le j \le n \\ &= \log j + \log(n-i+1) - 2\log(j-i+1) + \mathcal{O}(1). \end{aligned}$$

The asymptotic estimate given holds uniformly for $1 \le i \le j \le n$ and $n \to \infty$. When $i = j$ the formula yields the well-known average number of passes of Quickselect (see, for instance, [17]):

$$P_{n,j,j} = H_j + H_{n-j+1} - 1, \quad \text{for } 1 \le j \le n.$$

In order to show this theorem we start with a recursive description of $\mathcal{P}_{n,i,j}$. Since we assume that the input is a random permutation of size n we get that the probability that the pivot element $v = A[1]$ is the kth smallest element in the array is $1/n$ for all k, $1 \le k \le n$. After the partitioning phase the left subarray $A[1..k-1]$ and the right subarray $A[k+1..n]$ contain random permutations of lengths $k-1$ and $n-k$, respectively. If $i \le k \le j$ the algorithm terminates and we only have to count the original call to RQS. If $k < i$ we proceed with a recursive call of RQS for the right subarray and if $k > j$ we proceed with a recursive call of RQS for the left subarray. In these latter cases we have to add the number of calls of RQS occurring therein to the original call.

These considerations immediately lead to the following proposition (see for instance [4,20] and references therein for background on distributional recurrences like the one below).

Proposition 1. *The random variable $\mathcal{P}_{n,i,j}$ satisfies the following distributional recurrence:*

$$\mathcal{P}_{n,i,j} \overset{(d)}{=} 1 + \mathbb{1}_{\mathcal{U}_n < i} \cdot \mathcal{P}^{(1)}_{n-\mathcal{U}_n, i-\mathcal{U}_n, j-\mathcal{U}_n} + \mathbb{1}_{\mathcal{U}_n > j} \cdot \mathcal{P}^{(2)}_{\mathcal{U}_n - 1, i, j}, \quad \text{for } 1 \le i \le j \le n,$$

and $\mathcal{P}_{n,i,j} = 0$, if $i < 1$ or $j < i$ or $j > n$, where the rank $\mathcal{U}_n$ of the pivot element is uniformly distributed on $\{1, 2, \ldots, n\}$ and it is independent of $(\mathcal{P}_{n,i,j})_{n,i,j}$, $(\mathcal{P}^{(1)}_{n,i,j})_{n,i,j}$ and $(\mathcal{P}^{(2)}_{n,i,j})_{n,i,j}$; furthermore $(\mathcal{P}^{(1)}_{n,i,j})_{n,i,j}$ and $(\mathcal{P}^{(2)}_{n,i,j})_{n,i,j}$ are independent copies of $(\mathcal{P}_{n,i,j})_{n,i,j}$.

Proposition 1 immediately leads to the following recurrence for the expectation $P_{n,i,j}$ of the number of passes:

$$P_{n,i,j} = 1 + \frac{1}{n}\sum_{k=1}^{i-1} P_{n-k,i-k,j-k} + \frac{1}{n}\sum_{k=j+1}^{n} P_{k-1,i,j}, \quad \text{for } 1 \le i \le j \le n, \tag{1}$$

and $P_{n,i,j} = 0$, if $i < 1$ or $j < i$ or $j > n$.

It is not difficult to show by induction that the closed form for $P_{n,i,j}$ given in Theorem 1 is indeed the solution of the recurrence above. This recurrence and others that we will find later can in principle be solved using more or less standard techniques in an ad hoc fashion; however, the details of the derivation are already cumbersome for (1) and they get even worse when we have to deal with more complicated recurrences.

Therefore, we will take a detour in the next section, where we will investigate the general solution of trivariate recurrences whose shape is that of (1), but with a generic non-recursive cost $T_{n,i,j}$. With this systematic and general approach the solution of (1) will be then a by-product of the main result in the next subsection (Theorem 2). We will need only to set $T_{n,i,j} = 1$ and apply the theorem.

The rewards of this general analysis will be manifest soon afterwards, when we use Theorem 2 to obtain the expected number of comparisons of Range Quickselect (Section 2.4), later in Section 3 when we analyze the number of moves of particular elements made by Quickselect and the total number of moves made by Range Quickselect, and finally, in Section 4 when we investigate several parameters of random binary search trees.

2.3. Solving a trivariate recurrence

We consider the following recurrence for numbers $X_{n,i,j}$, which appears in our studies of the Quickselect and Range Quickselect algorithms, and later for binary search trees:

$$X_{n,i,j} = \frac{1}{n}\sum_{k=1}^{i-1} X_{n-k,i-k,j-k} + \frac{1}{n}\sum_{k=j+1}^{n} X_{k-1,i,j} + T_{n,i,j}, \quad \text{for } 1 \le i \le j \le n. \tag{2}$$

Furthermore we define $X_{n,i,j} = 0$, if $i < 1$ or $j < i$ or $n < j$. For the "toll function" $T_{n,i,j}$ we also define $T_{n,i,j} = 0$, if $i < 1$ or $j < i$ or $n < j$. We remark that (2) is a generalization of the "ordinary Quickselect recurrence" which appears when studying the moments of the number of comparisons and passes of Quickselect to select the jth smallest element in an array of size n. Indeed, the ordinary Quickselect recurrence is the special instance of (2) where $i = j$. The ordinary Quickselect recurrence was first studied by Knuth [10]; an exact solution for arbitrary toll functions has been given by Kuba in [12].

To treat recurrence (2) we introduce the following trivariate generating functions:

$$X(z,u_1,u_2) := \sum_{i\ge1}\sum_{j\ge i}\sum_{n\ge j} X_{n,i,j} z^n u_1^i u_2^j,$$
$$T(z,u_1,u_2) := \sum_{i\ge1}\sum_{j\ge i}\sum_{n\ge j} T_{n,i,j} z^n u_1^i u_2^j.$$

Multiplying (2) by $nz^{n-1}u_1^i u_2^j$ and summing up for all values $1 \le i \le j \le n$ leads, after straightforward computations, to the following differential equation for the generating function $X(z,u_1,u_2)$:

$$\frac{\partial}{\partial z}X(z,u_1,u_2) = \left(\frac{1}{1-z} + \frac{u_1u_2}{1-zu_1u_2}\right)X(z,u_1,u_2) + \frac{\partial}{\partial z}T(z,u_1,u_2),$$

with initial condition $X(0,u_1,u_2) = 0$.

The solution of this first order linear differential equation, which can be obtained by standard techniques, is:

$$X(z,u_1,u_2) = \frac{1}{(1-z)(1-zu_1u_2)}\int_0^z (1-t)(1-u_1u_2t)\Big(\frac{\partial}{\partial t}T(t,u_1,u_2)\Big)dt. \tag{3}$$

The numbers $X_{n,i,j}$ can then be obtained by extracting coefficients from the solution (3). By taking into account that $T_{n,i,j} = [z^n u_1^i u_2^j]T(z,u_1,u_2) = 0$, if $i < 1$ or $j < i$ or $n < j$, we get then, for $1 \le i \le j \le n$:

$$\begin{aligned}
X_{n,i,j} &= [z^n u_1^i u_2^j]X(z,u_1,u_2)\\
&= \sum_{\ell=0}^{i}[z^{n-\ell}u_1^{i-\ell}u_2^{j-\ell}]\frac{1}{1-z}\int_0^z (1-t)(1-u_1u_2t)\Big(\frac{\partial}{\partial t}T(t,u_1,u_2)\Big)dt\\
&= \sum_{\ell=0}^{i}\sum_{k=j-\ell}^{n-\ell}[z^k u_1^{i-\ell}u_2^{j-\ell}]\int_0^z (1-t)(1-u_1u_2t)\Big(\frac{\partial}{\partial t}T(t,u_1,u_2)\Big)dt\\
&= \sum_{\ell=0}^{i}\sum_{k=j-\ell}^{n-\ell}\frac{1}{k}[z^{k-1}u_1^{i-\ell}u_2^{j-\ell}](1-z)(1-u_1u_2z)\frac{\partial}{\partial z}T(z,u_1,u_2)\\
&= \sum_{\ell=0}^{i}\sum_{k=j-\ell}^{n-\ell}\frac{1}{k}\Big(kT_{k,i-\ell,j-\ell} - (k-1)T_{k-1,i-\ell,j-\ell} - (k-1)T_{k-1,i-\ell-1,j-\ell-1} + (k-2)T_{k-2,i-\ell-1,j-\ell-1}\Big).
\end{aligned}$$

The expression can be simplified easily by straightforward manipulations, thus

$$\begin{aligned}
X_{n,i,j} &= \sum_{\ell=1}^{i}\sum_{k=j-i+\ell}^{n-i+\ell}\left[\frac{kT_{k,\ell,j-i+\ell}-(k-1)T_{k-1,\ell,j-i+\ell}}{k}-\frac{(k-1)T_{k-1,\ell-1,j-i+\ell-1}-(k-2)T_{k-2,\ell-1,j-i+\ell-1}}{k}\right]\\
&= \sum_{\ell=1}^{i}\sum_{k=j-i+\ell}^{n-i+\ell}\frac{kT_{k,\ell,j-i+\ell}-(k-1)T_{k-1,\ell,j-i+\ell}}{k}-\sum_{\ell=1}^{i-1}\sum_{k=j-i+\ell+1}^{n-i+\ell+1}\frac{(k-1)T_{k-1,\ell,j-i+\ell}-(k-2)T_{k-2,\ell,j-i+\ell}}{k}\\
&= \sum_{\ell=1}^{i}\sum_{k=j-i+\ell}^{n-i+\ell}\frac{kT_{k,\ell,j-i+\ell}-(k-1)T_{k-1,\ell,j-i+\ell}}{k}-\sum_{\ell=1}^{i-1}\sum_{k=j-i+\ell}^{n-i+\ell}\frac{kT_{k,\ell,j-i+\ell}-(k-1)T_{k-1,\ell,j-i+\ell}}{k+1}\\
&= \sum_{\ell=1}^{i-1}\sum_{k=j-i+\ell}^{n-i+\ell}\frac{kT_{k,\ell,j-i+\ell}-(k-1)T_{k-1,\ell,j-i+\ell}}{k(k+1)}+\sum_{k=j}^{n}\frac{kT_{k,i,j}-(k-1)T_{k-1,i,j}}{k}.
\end{aligned}$$

Further simplifications yield

$$\begin{aligned}
X_{n,i,j} &= \sum_{\ell=1}^{i-1}\sum_{k=j-i+\ell}^{n-i+\ell}\frac{kT_{k,\ell,j-i+\ell}}{k(k+1)}-\sum_{\ell=1}^{i-1}\sum_{k=j-i+\ell}^{n-i+\ell-1}\frac{kT_{k,\ell,j-i+\ell}}{(k+1)(k+2)}+\sum_{k=j}^{n}\frac{kT_{k,i,j}}{k}-\sum_{k=j}^{n-1}\frac{kT_{k,i,j}}{k+1}\\
&= \sum_{\ell=1}^{i-1}\sum_{k=j-i+\ell}^{n-i+\ell-1}\frac{2T_{k,\ell,j-i+\ell}}{(k+1)(k+2)}+\sum_{\ell=1}^{i-1}\frac{T_{n-i+\ell,\ell,j-i+\ell}}{n-i+\ell+1}+\sum_{k=j}^{n-1}\frac{T_{k,i,j}}{k+1}+T_{n,i,j}.
\end{aligned}$$

We collect our results in the following theorem.

Theorem 2. *Let the sequence of numbers $X_{n,i,j}$, for $1 \le i \le j \le n$, satisfy the following recurrence:*

$$X_{n,i,j} = \frac{1}{n}\sum_{k=1}^{i-1}X_{n-k,i-k,j-k}+\frac{1}{n}\sum_{k=j+1}^{n}X_{k-1,i,j}+T_{n,i,j},$$

with $T_{n,i,j}$, $1 \le i \le j \le n$, an arbitrary sequence, such that $T_{n,i,j} = 0$ if $i < 1$, $j < i$ or $n < j$.
Then $X_{n,i,j}$, for $1 \le i \le j \le n$, is given by the explicit formula

$$X_{n,i,j} = \sum_{\ell=1}^{i-1}\sum_{k=j-i+\ell}^{n-i+\ell-1}\frac{2T_{k,\ell,j-i+\ell}}{(k+1)(k+2)}+\sum_{\ell=1}^{i-1}\frac{T_{n-i+\ell,\ell,j-i+\ell}}{n-i+\ell+1}+\sum_{k=j}^{n-1}\frac{T_{k,i,j}}{k+1}+T_{n,i,j}.$$

We remark that setting $i = j$ above gives an exact solution of the generic Quickselect recurrence. The solution thus obtained is slightly different from the one given in [12] and it is stated in the following corollary.

Corollary 1. *Let the sequence of numbers $X_{n,j}$, for $1 \le j \le n$, satisfy the following recurrence:*

$$X_{n,j} = \frac{1}{n}\sum_{k=1}^{j-1}X_{n-k,j-k}+\frac{1}{n}\sum_{k=j+1}^{n}X_{k-1,j}+T_{n,j}, \tag{4}$$

with $T_{n,j}$, $1 \le j \le n$, an arbitrary sequence such that $T_{n,j} = 0$ if $j < 1$ or $n < j$.
Then $X_{n,j}$, for $1 \le j \le n$, is given by the explicit formula

$$X_{n,j} = \sum_{\ell=1}^{j-1}\sum_{k=\ell}^{n-j+\ell-1}\frac{2T_{k,\ell}}{(k+1)(k+2)}+\sum_{\ell=1}^{j-1}\frac{T_{n-j+\ell,\ell,}}{n-j+\ell+1}+\sum_{k=j}^{n-1}\frac{T_{k,j}}{k+1}+T_{n,j}.$$

Recurrence (1) studied in Section 2.2 is the instance of recurrence (2) for the particular toll function $T_{n,i,j} = 1$, $1 \le i \le j \le n$. We can then obtain the exact solution of (1) applying Theorem 2, which gives after easy summations:

$$\begin{aligned}
P_{n,i,j} &= \sum_{\ell=1}^{i-1}\sum_{k=j-i+\ell}^{n-i+\ell-1}\frac{2}{(k+1)(k+2)}+\sum_{\ell=1}^{i-1}\frac{1}{n-i+\ell+1}+\sum_{k=j}^{n-1}\frac{1}{k+1}+1\\
&= \sum_{\ell=1}^{i-1}\sum_{k=j-i+\ell}^{n-i+\ell-1}2\Big(\frac{1}{k+1}-\frac{1}{k+2}\Big)+H_n-H_{n-i+1}+H_n-H_j+1\\
&= \sum_{\ell=1}^{i-1}2\Big(\frac{1}{j-i+\ell+1}-\frac{1}{n-i+\ell+1}\Big)+2H_n-H_{n-i+1}-H_j+1\\
&= H_j+H_{n-i+1}-2H_{j-i+1}+1, \quad \text{for } 1 \le i \le j \le n.
\end{aligned}$$

This proves Theorem 1.

2.4. The number of comparisons

Next we study the average behavior of the random variable $\mathcal{C}_{n,i,j}$, with $1 \le i \le j \le n$, which counts the number of comparisons in the partitioning phase between elements in the array and the pivot element, when executing the algorithm Range Quickselect until an element with a rank between i and j is found in the array $A[1..n]$.

Theorem 3. *The expected number of element comparisons $C_{n,i,j} = \mathbb{E}\left(\mathcal{C}_{n,i,j}\right)$ made while executing the algorithm Range Quickselect until an element with a rank between i and j is found in an array of size n is:*

$$\begin{aligned} C_{n,i,j} &= 2(n+1)H_n + 2(j-i+4)H_{j-i+1} - 2(j+2)H_j \\ &\quad - 2(n-i+3)H_{n-i+1} + 2n - j + i - 2 \\ &\sim 2n\log n + 2(j-i+1)\log(j-i+1) - 2j\log j - 2(n-i+1)\log(n-i+1) \\ &\quad + \mathcal{O}(\log^2 n), \quad \text{for } 1 \le i \le j \le n. \end{aligned}$$

The asymptotic equivalence holds uniformly for $1 \le i \le j \le n$ and $n \to \infty$.

Setting $i = j$ above, we obtain the average number of comparisons to select the jth smallest element out of n [10]:

$$C_{n,j,j} = 2\Big((n+1)H_n + n + 3 - (j+2)H_j - (n-j+3)H_{n-j+1}\Big), \quad \text{for } 1 \le j \le n.$$

Another immediate consequence of the theorem is that the value $C_{n,i,j}$ is always $\Theta(n)$, namely,

$$C_{n,i,j} = c(i/n, j/n)n + o(n),$$

with $c(a,b) = -2(1-a)\log(1-a) - 2b\log b + 2(b-a)\log(b-a) + 2 - (b-a)$.

The proof of this theorem is fully analogous to that of Theorem 1 in Section 2.2. First we obtain a distributional recurrence for $\mathcal{C}_{n,i,j}$, which has the same structure as the one given in Proposition 1. Here, we only have to take into account that during the partitioning phase and independently of the actual rank of the pivot, we perform exactly $n-1$ comparisons between the pivot element and the other elements in the array.

Proposition 2. *The random variable $\mathcal{C}_{n,i,j}$ satisfies the following distributional recurrence:*

$$\mathcal{C}_{n,i,j} \overset{(d)}{=} n - 1 + \mathbb{1}_{\mathcal{U}_n < i} \cdot \mathcal{C}^{(1)}_{n-\mathcal{U}_n, i-\mathcal{U}_n, j-\mathcal{U}_n} + \mathbb{1}_{\mathcal{U}_n > j} \cdot \mathcal{C}^{(2)}_{\mathcal{U}_n - 1, i, j}, \quad \text{for } 1 \le i \le j \le n,$$

and $\mathcal{C}_{n,i,j} = 0$, if $i < 1$ or $j < i$ or $j > n$, where the rank $\mathcal{U}_n$ of the pivot element is uniformly distributed on $\{1, 2, \ldots, n\}$ and independent of $(\mathcal{C}_{n,i,j})_{n,i,j}$, $(\mathcal{C}_{n,i,j}{}^{(1)})_{n,i,j}$ and $(\mathcal{C}^{(2)}_{n,i,j})_{n,i,j}$; the last two are independent copies of $(\mathcal{C}_{n,i,j})_{n,i,j}$.

Proposition 2 gives then the following recurrence for the expectation $C_{n,i,j}$ of the number of comparisons:

$$C_{n,i,j} = n - 1 + \frac{1}{n}\sum_{k=1}^{i-1} C_{n-k,i-k,j-k} + \frac{1}{n}\sum_{k=j+1}^{n} C_{k-1,i,j}, \quad \text{for } 1 \le i \le j \le n, \tag{5}$$

and $C_{n,i,j} = 0$, if $i < 1$ or $j < i$ or $j > n$.

This recurrence is exactly the recurrence studied in Section 2.3 for the particular toll function $T_{n,i,j} = n - 1$, for $1 \le i \le j \le n$. Applying Theorem 2 easily leads then, for $1 \le i \le j \le n$, to an exact formula for $C_{n,i,j}$ and proves Theorem 3:

$$\begin{aligned} C_{n,i,j} &= \sum_{\ell=1}^{i-1}\sum_{k=j-i+\ell}^{n-i+\ell-1} \frac{2(k-1)}{(k+1)(k+2)} + \sum_{\ell=1}^{i-1}\frac{n-i+\ell-1}{n-i+\ell+1} + \sum_{k=j}^{n-1}\frac{k-1}{k+1} + n - 1 \\ &= \sum_{\ell=1}^{i-1}\sum_{k=j-i+\ell}^{n-i+\ell-1}\Big(-\frac{4}{k+1} + \frac{6}{k+2}\Big) + \sum_{\ell=1}^{i-1}\Big(1 - \frac{2}{n-i+\ell+1}\Big) + \sum_{k=j}^{n-1}\Big(1 - \frac{2}{k+1}\Big) + n - 1 \\ &= \sum_{\ell=1}^{i-1}\Big(-4(H_{n-i+\ell} - H_{j-i+\ell}) + 6(H_{n-i+\ell+1} - H_{j-i+\ell+1})\Big) \\ &\quad + i - 1 - 2(H_n - H_{n-i+1}) + n - j - 2(H_n - H_j) + n - 1 \\ &= 2(n+1)H_n + 2(j-i+4)H_{j-i+1} - 2(j+2)H_j - 2(n-i+3)H_{n-i+1} + 2n - j + i - 2. \end{aligned}$$

To obtain the final result we just used the basic summation formula

$$\sum_{k=1}^{n-1} H_k = n\big(H_n - 1\big). \tag{6}$$

2.5. Savings and grand averages

Given any measure of performance $\mathcal{X}_{n,i,j}$ of Range Quickselect when looking for an element whose rank falls in the range $[i..j]$, out of n elements, it is quite obvious that

$$\mathcal{X}_{n,i,j} \leq \mathcal{X}_{n,k,k},$$

for any $k \in [i..j]$. In other words, no matter what measure we consider, Range Quickselect will never perform worse than Quickselect when the sought rank k belongs to the range $[i..j]$ given as input to Range Quickselect. The inequality above of course carries over expectations, thus $X_{n,i,j} \leq X_{n,k,k}$ for $k \in [i..j]$.

It makes sense then to introduce the difference

$$\Delta X_{n,i,d} = X_{n,i,i} - X_{n,i-d,i+d}, \quad d < i < n+1-d, 0 \leq d \leq \lfloor (n-1)/2 \rfloor$$

which measures the savings of Range Quickselect over Quickselect when looking for the ith smallest element and Range Quickselect is given a range of size $2d+1$ around i. As we shall see, in some cases, $\Delta X_{n,i,d}$ does not depend (or its main order term does not depend) on i, so using the "size" d of the range to express the savings yielded by Range Quickselect turns out to be a relevant choice. Obtaining both explicit and asymptotic formulaæ for $\Delta P_{n,i,d}$ and $\Delta C_{n,i,d}$ is straightforward from the explicit expressions given by Theorems 1 and 3, and the well-known asymptotic expansion of the harmonic numbers

$$H_n = \log n + \gamma + \mathcal{O}(n^{-1}),$$

with $\gamma \approx 0.577215\ldots$ denoting the Euler–Mascheroni constant.

Another interesting set of quantities that we study in this section (and on those forthcoming) are the grand averages. We fix a size $2d+1$ for the range given to Range Quickselect and then average over all possible i, i.e., we are interested in the expected value of $\mathcal{X}_{n,i-d,i+d}$ when i is uniformly distributed in $[d+1..n-d]$. Such quantities are often called *grand averages* [14,19]. Thus,

$$\overline{X}_{n,d} = \frac{1}{n-2d} \sum_{d<i\leq n-d} X_{n,i-d,i+d}, \quad 0 \leq d \leq \lfloor (n-1)/2 \rfloor.$$

Notice that $\overline{X}_{n,0}$ is the expected value for quickselect with random rank.

As before, we will also be interested in the "grand average savings"

$$\Delta\overline{X}_{n,d} = \overline{X}_{n,0} - \overline{X}_{n,d}, \quad 0 \leq d \leq \lfloor (n-1)/2 \rfloor.$$

In the case of passes and comparisons, explicit and asymptotic expressions for the grand averages and the average savings follow easily from the explicit formulæ available for these measures of cost. The following corollary summarizes the relevant results.

Corollary 2. *Let d and i be such that $0 \leq d \leq \lfloor (n-1)/2 \rfloor$ and $d < i < n+1-d$. The asymptotic estimates below hold uniformly for all $d > 0$, unless explicitly stated otherwise, when $n \to \infty$.*

(1) *Let $\Delta P_{n,i,d} = P_{n,i,i} - P_{n,i-d,i+d}$ be the average number of passes saved if we use Range Quickselect with range $[i-d..i+d]$ instead of Quickselect with rank i. Then*

$$\begin{aligned}\Delta P_{n,i,d} &= (H_i - H_{i+d}) + (H_{n+1-i} - H_{n+1-i+d}) + 2H_{2d+1} - 2\\ &\sim 2\log d + \Theta(1).\end{aligned}$$

(2) *Let*

$$\overline{P}_{n,d} = \frac{1}{n-2d} \sum_{d<i<n+1-d} P_{n,i-d,i+d}$$

be the average number of passes made by Range Quickselect for a range of size $2d+1$ centered around a rank chosen uniformly at random. Then

$$\begin{aligned}\overline{P}_{n,d} &= 2\frac{n+1}{n-2d}(H_n - H_{2d+1}) - 1 + \frac{2}{n-2d}\\ &\sim \begin{cases} 2\log(n/d) + O(1), & \text{if } 0 < d = o(n),\\ \dfrac{2}{1-2\delta}\log(1/2\delta) - 1 + O(1/n), & \text{if } d = \delta\cdot n + o(n), \text{ with } 0 < \delta < 1/2.\end{cases}\end{aligned}$$

Furthermore, the grand average of the savings is

$$\Delta\overline{P}_{n,d} = \overline{P}_{n,0} - \overline{P}_{n,d} \sim 2\log d + O(1).$$

(3) *Let* $\Delta C_{n,i,d} = C_{n,i,i} - C_{n,i-d,i+d}$ *be the average number of element comparisons that we save if we use Range Quickselect with range* $[i-d..i+d]$ *instead of Quickselect with rank* i*. Then*

$$\begin{aligned}\Delta C_{n,i,d} &= 8 - 2(i+2)H_i - 2(n-i+3)H_{n+1-i} - 2(2d+4)H_{2d+1} + 2(i+d+2)H_{i+d} \\ &\quad + 2(n-i+3+d)H_{n+1+d-i} + 2d \\ &\sim \begin{cases} 4d\log\left(\dfrac{n}{d}\right) + \Theta(d), & \text{if } 0 < d = o(n), \\ 2c(\alpha,\delta)n - 8\log n + O(1), & \text{if } d = \delta\cdot n + o(n), \text{ with } 0 < \delta < 1/2, \end{cases}\end{aligned}$$

where

$$\begin{aligned}c(\alpha,\delta) &= \delta + (1-\alpha+\delta)\log(1-\alpha+\delta) + (\alpha+\delta)\log(\alpha+\delta) \\ &\quad - \alpha\log\alpha - (1-\alpha)\log(1-\alpha) - 2\delta\log(2\delta).\end{aligned}$$

The second asymptotic estimate holds uniformly for $i = \alpha n + o(n)$ *and* $n \to \infty$*.*

(4) *Let*

$$\overline{C}_{n,d} = \frac{1}{n-2d}\sum_{d<i<n+1-d} C_{n,i-d,i+d}$$

be the average number of comparisons made by Range Quickselect for a range of size $2d+1$ *centered around a rank chosen uniformly at random. Then*

$$\begin{aligned}\overline{C}_{n,d} &= 3n - \frac{4(d+2)(n+1)}{n-2d}(H_n - H_{2d+1}) + 5 - \frac{4(d+2)}{n-2d} \\ &\sim \begin{cases} 3n - 4(d+2)\log\left(\dfrac{n}{d}\right) + \Theta(d), & \text{if } 0 < d = o(n), \\ \left(3 + \dfrac{4\delta\log(2\delta)}{1-2\delta}\right)n + \Theta(1), & \text{if } d = \delta\cdot n + o(n), \text{ with } 0 < \delta < 1/2. \end{cases}\end{aligned}$$

Furthermore, the grand average of the savings is

$$\Delta\overline{C}_{n,d} = \overline{C}_{n,0} - \overline{C}_{n,d} \sim \begin{cases} 4d\log(n/d) + \Theta(d), & \text{if } 0 < d = o(n), \\ \dfrac{4\delta\log(1/2\delta)}{1-2\delta}n - 8\log n + \Theta(1), & \text{if } d = \delta\cdot n + o(n),\ 0 < \delta < 1/2. \end{cases}$$

To conclude, a few words on the practical significance of these findings. For instance, we can find an element whose rank is $n/2 \pm \sqrt{n}$ and save up to $\Theta(\sqrt{n}\log n)$ comparisons, or find an element of rank $\alpha n(1\pm\delta)$, for some $\delta > 0$ and save a linear number of comparisons. Savings of the order $\Theta(\sqrt{n}\log n)$ might seem too small to bother with, since the algorithm runs in linear time; however, for moderate sizes of the array, savings such as these are noticeable in practice. For instance, with a range of size $2\sqrt{n}+1$ around the desired rank, the algorithm allows us to save $\approx$1555 comparisons on average for an array of size 10 000 (the range around the sought rank is of size 201), and we save $\approx$2400 comparisons on the average when the size is 20 000 (the range is then of size $\approx$283).

3. Moves in Quickselect and Range Quickselect

We start with the definition of the quantities in our study of moves of elements in the standard Quickselect and Range Quickselect algorithms.

The random variable $\mathcal{M}_{n,p,j}$, with $1 \le p, j \le n$, counts the number of moves of the element with rank p, i.e., assignments appearing in line 8, line 13 or 17 where the right-hand side contains the pth element, in the PARTITION procedure (given as Algorithm 3 in next page) while executing the algorithm Quickselect to find an element with rank j in an array $A[1..n]$.

The random variable $\mathcal{V}_{n,i,j}$, with $1 \le i \le j \le n$, counts the total number of moves, i.e., assignments appearing in line 8, line 13 or 17), of array elements in the procedure PARTITION when executing the algorithm RQS to find the element with rank $k \in [i..j]$ in an array $A[1..n]$.

We conclude this introduction by stating the following well-known randomness preservation property (see, e.g., [11]) of the partition algorithm PARTITION as described in Section 3.1 (we remark that this property also holds for other commonly used partition procedures). When starting with a random permutation of distinct values $a_l < a_{l+1} < \cdots < a_r$ as input data $A[l..r]$ for the partition algorithm PARTITION(A, l, r, k) it holds that after executing this procedure the left subarray $A[l..k-1]$ is itself a random permutation of $a_l < a_{l+1} < \cdots < a_{k-1}$, and the right subarray $A[k+1..r]$ is itself a random permutation of $a_{k+1} < a_{k+2} < \cdots < a_r$.

This randomness preservation property allows a recursive description of the parameters studied in this paper and is thus heavily used in the analysis carried out in what follows.

3.1. The partition procedure

There are several standard implementations of the partitioning phase used in practice for the algorithm Quickselect (and, of course, also for Quicksort). The procedure PARTITION given as Algorithm 3 is just one particular implementation, which we assume to be used in both Quickselect and Range Quickselect. While for the analysis of moves, we continue assuming that all elements are distinct, the implementation of PARTITION contemplates the more general case where repetitions may occur. At this point, we want to point out that other standard implementations of the partitioning procedure will likely lead to similar, although slightly different, results for the quantities studied here. After executing PARTITION(A, l, r, k) a pivot element v is brought to its correct position $v = A[k]$ in the array, such that all elements in the array $A[l..k-1]$ are smaller than or equal to v and all elements in the array $A[k+1..r]$ are larger than or equal to v.

To do this the procedure starts by choosing as pivot element v the first element $A[l]$ in the array $A[l..r]$, which is stored. Then, by using two pointers a and b that are initialized by $a = l$ and $b = r$, the array is scanned in an alternating way from right and from left, where each element is compared with the pivot element v. When scanning from right we search for the first element $A[b]$, which is smaller than or equal to v; this element is then stored at position $A[a]$ and one continues with scanning from left. When scanning from left we search for the first element $A[a]$, which is larger than or equal to v; this element is then stored at position $A[b]$ and one continues with scanning from right. The scan stops if $a = b$, i.e., if the two pointers a and b meet each other. Then it remains to store the pivot element v at its correct place $A[a]$ in the array and return this final location of the pivot element.

Algorithm 3 The PARTITION procedure

Require: Array $A[l..r]$
Ensure: $\forall m : (l \le m < k) \Rightarrow A[m] \le A[k]$, and $\forall m : (k < m \le r) \Rightarrow A[k] \le A[m]$

```
1:  procedure PARTITION(A, l, r, k)
2:      if l > r then return                                  ▷ Nothing will be done
3:      end if
4:      a := l; b := r; v := A[a]
5:      while a < b do
6:          while A[b] > v do b := b − 1                      ▷ Scan from right
7:          end while
8:          A[a] := A[b]
9:          a := a + 1
10:         if a < b then
11:             while A[a] < v do a := a + 1                  ▷ Scan from left
12:             end while
13:             A[b] := A[a]
14:             b := b − 1
15:         end if
16:     end while
17:     A[a] := v
18:     k := a                                                ▷ Task finished
19: end procedure
```

3.2. The number of moves of particular elements in Quickselect

We study here the average behavior of the random variable $\mathcal{M}_{n,p,j}$, with $1 \le p, j \le n$, which counts the number of moves, i.e., assignments $A[.] := a_p$ of the element with rank p in the PARTITION procedure when executing the algorithm Quickselect to find the element with rank j in an array of size n. The following theorem provides an exact formula for the expectation $M_{n,p,j} := \mathbb{E}\left(\mathcal{M}_{n,p,j}\right)$.

Theorem 4. *The expected number of moves $M_{n,p,j} = \mathbb{E}\left(\mathcal{M}_{n,p,j}\right)$ of the element with rank p during the execution the algorithm Quickselect to find the element with rank j in an array of size n, is*

$$M_{n,p,j} = \frac{1}{3}H_n + \frac{1}{6}H_j + \frac{1}{6}H_{n-p+1} - \frac{2}{3}H_{j-p+1} + \frac{1}{2} - \frac{(p-1)^2}{3n} + \frac{(p-1)(p-2)}{3(n-1)} - \frac{(p+2)(p-1)}{6j} + \frac{(p-1)(p-2)}{6(j-1)} + \frac{1}{j-p+1}, \quad \text{for } 1 \le p < j \le n,$$

$$
\begin{aligned}
M_{n,p,j} &= \frac{1}{3}H_n + \frac{1}{6}H_p + \frac{1}{6}H_{n-j+1} - \frac{2}{3}H_{p-j+1} - \frac{(p-1)^2}{3n} + \frac{(p-1)(p-2)}{3(n-1)} \\
&\quad - \frac{(p-j)(p-j-3)}{6(n-j+1)} + \frac{(p-j)(p-j-1)}{6(n-j)} + \frac{1}{3p} + \frac{2}{3(p-j+1)}, \quad \text{for } 1 \le j < p \le n, \\
M_{n,p,j} &= \frac{1}{3}H_n + \frac{1}{6}H_j + \frac{1}{6}H_{n-j+1} + \frac{1}{6} - \frac{(j-1)^2}{3n} + \frac{(j-1)(j-2)}{3(n-1)} + \frac{1}{3j} \\
&\quad + \frac{1}{12}\cdot[\![j=1]\!] - \frac{1}{12}\cdot[\![j=n]\!], \quad \text{for } p=j,\ 1 \le j \le n \text{ and } n \ge 2, \\
M_{1,1,1} &= 1.
\end{aligned}
$$

Despite the formulæ for $M_{n,p,j}$ when $p < j$ and $M_{n,p,j}$ when $p > j$ seem to be related via the substitution $p \leftrightarrow n-p+1$, $j \leftrightarrow n-j+1$, this is not the case, as the reader can readily convince herself by substituting a few values. The difference between the formula $M_{n,p,j}$ for $p > j$, call it $M^{(2)}_{n,p,j}$, and the formula $M_{n,p,j}$ for $p < j$, call it $M^{(1)}_{n,i,j}$, when we substitute p by $n+1-p$ and j by $n+1-j$ is very small, namely, $M^{(2)}_{n,p,j} = M^{(1)}_{n,p,j} + \mathcal{O}(1)$. For completeness, we will later give separate asymptotic formulæ in Corollary 3 for $M_{n,p,j}$ when $p < j$ and when $p > j$, although that should be not necessary because of the relation just noted.

To prove this theorem we start with a recursive description of $\mathcal{M}_{n,p,j}$, which is obtained by considering a call of Quickselect for an array $A[1..n]$. We assume now that the pivot element $v = A[1]$ is the kth smallest element in the array; since our input data are forming a random permutation of length n it holds that the probability that the pivot element has rank k is $1/n$, for $1 \le k \le n$.

Now we study whether the element with rank p will be moved, i.e., an assignment $A[\cdot] := \ldots$ where the right-hand side contains the element with rank i is performed, during the execution of the partition procedure Partition. We have to distinguish three cases: (1) if $k = p$ then the element with rank p (in this case this is the pivot element) will always be moved, (2) if $k < p$ the element with rank p will be moved only if it is located in the subarray $A[2..k]$; the probability that this happens is thus $\frac{k-1}{n-1}$, and (3) if $k > p$ the element with rank p will be moved only if it is located in the subarray $A[k..n]$; the probability that this happens is then $\frac{n-k+1}{n-1}$. After the partitioning phase the left subarray $A[1..k-1]$ and the right subarray $A[k+1..n]$ are each forming a random permutation of lengths $k-1$ and $n-k$, respectively.

Next we observe that if the pivot element has a rank between p and j, i.e., depending on the order of the considered elements either $p \le k \le j$ or $j \le k \le p$, the final number of moves of the element with rank p during the execution of Quickselect is already reached. This holds since then either the Quickselect algorithm terminates ($k = j$) or it continues executing in a subarray that does not contain the element with rank p. Only if $k < p \le j$ or $k < j \le p$ we proceed with a recursive call of Quickselect for the right subarray, and if $k > j \ge p$ or $k > p \ge j$ we proceed with a recursive call of Quickselect for the left subarray. In these latter cases we have to add the number of moves of the element with rank p during the execution of Quickselect occurring therein.

These considerations immediately lead to the following proposition.

Proposition 3. *The random variable $\mathcal{M}_{n,p,j}$ satisfies, for $1 \le p, j \le n$, the following distributional recurrence:*

$$
\begin{aligned}
\mathcal{M}_{n,p,j} &\overset{(d)}{=} \mathbb{1}_{\mathcal{U}_n<p}\cdot\mathcal{M}^{(1)}_{n-\mathcal{U}_n,p-\mathcal{U}_n,j-\mathcal{U}_n} + \mathbb{1}_{\mathcal{U}_n>j}\cdot\mathcal{M}^{(2)}_{\mathcal{U}_n-1,p,j} + \mathcal{T}_{n,p,\mathcal{U}_n}, \quad \text{for } 1 \le p \le j \le n, \\
\mathcal{M}_{n,p,j} &\overset{(d)}{=} \mathbb{1}_{\mathcal{U}_n<j}\cdot\mathcal{M}^{(1)}_{n-\mathcal{U}_n,p-\mathcal{U}_n,j-\mathcal{U}_n} + \mathbb{1}_{\mathcal{U}_n>p}\cdot\mathcal{M}^{(2)}_{\mathcal{U}_n-1,p,j} + \mathcal{T}_{n,p,\mathcal{U}_n}, \quad \text{for } 1 \le j < p \le n,
\end{aligned}
$$

and $\mathcal{M}_{n,p,j} = 0$, if $\min(p,j) < 1$ or $\max(p,j) > n$. The rank $\mathcal{U}_n$ of the pivot element is uniformly distributed on $\{1,2,\ldots,n\}$ and independent of $(\mathcal{M}_{n,p,j})_{n,p,j}$, $(\mathcal{M}^{(1)}_{n,p,j})_{n,p,j}$ and $(\mathcal{M}^{(2)}_{n,p,j})_{n,p,j}$, which are independent copies of $(\mathcal{M}_{n,p,j})_{n,p,j}$.

Here the random variable $\mathcal{T}_{n,p,k}$ is the indicator function of the event that the element with rank p is moved during the execution of the partition procedure Partition *for a randomly chosen permutation of length n leading to a pivot element of rank k. It holds then, for $1 \le p, k \le n$:*

$$
\mathbb{P}\left\{\mathcal{T}_{n,p,k} = 1\right\} = \begin{cases} \dfrac{k-1}{n-1}, & k < p, \\ \dfrac{n-k+1}{n-1}, & k > p, \\ 1, & k = p, \end{cases}
$$

and $\mathbb{P}\left\{\mathcal{T}_{n,p,k} = 0\right\} = 1 - \mathbb{P}\left\{\mathcal{T}_{n,p,k} = 1\right\}$.

We remark here that in the distributional recurrence given as Proposition 3 the random variables $\mathcal{T}_{n,p,k}$ and $\mathcal{M}^{(1)}_{n-k,p-k,j-k}$ (and also $\mathcal{T}_{n,p,k}$ and $\mathcal{M}^{(2)}_{k-1,p,j}$) are *dependent* as can be checked easily for concrete examples (e.g., for $n = 3$ and $i = j = 1$). Thus Proposition 3 will only allow to treat the expectation $M_{n,p,j}$ of the number of moves, whereas a study of higher moments would require a more refined description of $\mathcal{M}_{n,p,j}$.

However, Proposition 3 immediately leads to a recurrence for the expected value $M_{n,p,j}$. It is here advantageous to distinguish between the cases $p < j$, $p = j$ and $p > j$.

We start with the case $p < j$, where we obtain, for $1 \le p < j \le n$:

$$\begin{aligned} M_{n,p,j} &= \frac{1}{n}\sum_{k=1}^{p-1} M_{n-k,p-k,j-k} + \frac{1}{n}\sum_{k=j+1}^{n} M_{k-1,p,j} + \frac{1}{n}\sum_{k=1}^{n} \mathbb{E}\left(\mathcal{T}_{n,p,k}\right) \\ &= \frac{1}{n}\sum_{k=1}^{p-1} M_{n-k,p-k,j-k} + \frac{1}{n}\sum_{k=j+1}^{n} M_{k-1,p,j} + \frac{1}{n}\left(1 + \sum_{k=1}^{p-1}\frac{k-1}{n-1} + \sum_{k=p+1}^{n}\frac{n-k+1}{n-1}\right) \\ &= \frac{1}{n}\sum_{k=1}^{p-1} M_{n-k,p-k,j-k} + \frac{1}{n}\sum_{k=j+1}^{n} M_{k-1,p,j} + \frac{(p-1)(p-2)}{2n(n-1)} + \frac{(n-p)(n-p+1)}{2n(n-1)} + \frac{1}{n}. \end{aligned}$$

To get an exact solution of $M_{n,p,j}$ we can thus apply Theorem 2 for the particular toll function

$$T_{n,p,j} = \frac{(p-1)(p-2)}{2n(n-1)} + \frac{(n-p)(n-p+1)}{2n(n-1)} + \frac{1}{n}, \quad 1 \le p < j \le n.$$

We omit here the computations leading to the exact formula of $M_{n,p,j}$, $1 \le p < j \le n$, given in Theorem 4, since nothing more is required than basic summation formulæ.

For the case $p = j$ we obtain, for $1 \le j \le n$:

$$\begin{aligned} M_{n,p,j} &= \frac{1}{n}\sum_{k=1}^{j-1} M_{n-k,p-k,j-k} + \frac{1}{n}\sum_{k=j+1}^{n} M_{k-1,p,j} + \frac{1}{n}\sum_{k=1}^{n} \mathbb{E}\left(\mathcal{T}_{n,p,k}\right) \\ &= \frac{1}{n}\sum_{k=1}^{i-1} M_{n-k,p-k,j-k} + \frac{1}{n}\sum_{k=j+1}^{n} M_{k-1,p,j} \\ &\quad + \begin{cases} \dfrac{(j-1)(j-2)}{2n(n-1)} + \dfrac{(n-j)(n-j+1)}{2n(n-1)} + \dfrac{1}{n}, & \text{for } n \ge 2, \\ 1, & \text{for } n = 1. \end{cases} \end{aligned}$$

Thus, an exact solution of $M_{n,p,j}$ with $p = j$ can be obtained by introducing $M'_{n,j} := M_{n,p,j}$ and applying Corollary 1 for the particular toll function

$$T_{n,j} := \begin{cases} \dfrac{(j-1)(j-2)}{2n(n-1)} + \dfrac{(n-j)(n-j+1)}{2n(n-1)} + \dfrac{1}{n}, & \text{for } 1 \le j \le n \text{ and } n \ge 2, \\ 1, & \text{for } j = n = 1. \end{cases}$$

After carrying out the computations occurring, which are omitted here, we obtain the exact formula of $M_{n,j,j}$, $1 \le j \le n$, given in Theorem 4.

Finally we consider the case $p > j$, where we obtain, for $1 \le j < p \le n$:

$$\begin{aligned} M_{n,p,j} &= \frac{1}{n}\sum_{k=1}^{j-1} M_{n-k,p-k,j-k} + \frac{1}{n}\sum_{k=p+1}^{n} M_{k-1,p,j} + \frac{1}{n}\sum_{k=1}^{n} \mathbb{E}\left(\mathcal{T}_{n,p,k}\right) \\ &= \frac{1}{n}\sum_{k=1}^{j-1} M_{n-k,p-k,j-k} + \frac{1}{n}\sum_{k=p+1}^{n} M_{k-1,p,j} + \frac{(p-1)(p-2)}{2n(n-1)} + \frac{(n-p)(n-p+1)}{2n(n-1)} + \frac{1}{n}. \end{aligned}$$

When introducing $M'_{n,p,j} := M_{n,j,p}$ this recurrence can be written as follows, with $1 \le p < j \le n$:

$$M'_{n,p,j} = \frac{1}{n}\sum_{k=1}^{p-1} M'_{n-k,p-k,j-k} + \frac{1}{n}\sum_{k=j+1}^{n} M'_{k-1,p,j} + \frac{(j-1)(j-2)}{2n(n-1)} + \frac{(n-j)(n-j+1)}{2n(n-1)} + \frac{1}{n}.$$

An exact solution of $M'_{n,p,j}$, $1 \le p < j \le n$, can be obtained by applying Theorem 2 for the particular toll function

$$T_{n,p,j} = \frac{(j-1)(j-2)}{2n(n-1)} + \frac{(n-j)(n-j+1)}{2n(n-1)} + \frac{1}{n}, \quad 1 \le p < j \le n.$$

After back substitution we thus obtain an exact solution of $M_{n,p,j}$, with $1 \le j < p \le n$, which is given in Theorem 4. Again the straightforward computations are omitted.

Last but not least, we can obtain asymptotic equivalents with little effort.

Corollary 3. *The expected number of moves $M_{n,p,j} = \mathbb{E}\left(\mathcal{M}_{n,p,j}\right)$ of the element with rank p when executing the algorithm Quickselect to find the element with rank j in an array of size n has the following asymptotic equivalents, which hold for $n \to \infty$ and uniformly for the given range of p and j:*

$$M_{n,p,j} = \frac{1}{3}\log n + \frac{1}{6}\log j + \frac{1}{6}\log(n-p+1) - \frac{2}{3}\log(j-p+1) + \mathcal{O}(1), \quad 1 \le p < j \le n,$$
$$M_{n,p,j} = \frac{1}{3}\log n + \frac{1}{6}\log p + \frac{1}{6}\log(n-j+1) - \frac{2}{3}\log(p-j+1) + \mathcal{O}(1), \quad 1 \le j < p \le n,$$
$$M_{n,j,j} = \frac{1}{3}\log n + \frac{1}{6}\log j + \frac{1}{6}\log(n-j+1) + \mathcal{O}(1), \quad 1 \le j \le n.$$

In particular, we get the following important estimates when $j = \beta n + o(n)$, $0 < \beta < 1$:

$$M_{n,p,j} \sim \frac{1}{6}\log\beta + \frac{1}{6}\log(1-\alpha) - \frac{2}{3}\log(\beta-\alpha) + \frac{\alpha^2}{6\beta^2} - \frac{2\alpha}{3\beta} + \frac{1}{2} - \frac{\alpha}{3} + \frac{\alpha^2}{3},$$
for $p = \alpha n + o(n)$, and $0 < \alpha < \beta < 1$,
$$M_{n,p,j} \sim \frac{1}{6}\log\alpha + \frac{1}{6}\log(1-\beta) - \frac{2}{3}\log(\alpha-\beta) + \frac{(1-\alpha)^2}{6(1-\beta)^2} - \frac{2(1-\alpha)}{3(1-\beta)} + \frac{1}{2} - \frac{\alpha}{3} + \frac{\alpha^2}{3},$$
for $p = \alpha n + o(n)$, and $0 < \beta < \alpha < 1$,
$$M_{n,p,j} \sim \frac{2}{3}(1-\kappa)\log n, \quad \text{for } |j-p| \sim Kn^{\kappa}, \text{ with } 0 < \kappa < 1 \text{ and } K \neq 0,$$
$$M_{n,p,j} \sim \frac{2}{3}\log n, \quad \text{for } |j-p| = \mathcal{O}((\log n)^{\kappa}) \text{ for some } \kappa > 0.$$

3.3. The total number of moves in Range Quickselect

Now we study the average behavior of the random variable $\mathcal{V}_{n,i,j}$, with $1 \le i \le j \le n$, which counts the total number of moves, i.e., assignments $A[\cdot] := *$ of array elements, in the partition procedure PARTITION when executing the algorithm RQS to find the element with rank $k \in [i..j]$ in an array $A[1..n]$.

Theorem 5. *The expected total number of moves $V_{n,i,j} = \mathbb{E}\left(\mathcal{V}_{n,i,j}\right)$ of array elements in the partition procedure* PARTITION *when executing the algorithm* RQS *to find the element with rank $k \in [i..j]$ in an array $A[1..n]$ filled with a random permutation of length n, for $1 \le i \le j \le n$, is given by the following exact formula:*

$$V_{n,i,j} = \frac{2}{3}(n+1)H_n - \frac{1}{6}(4j+1)H_j - \frac{1}{6}(4n-4i+5)H_{n-i+1} + \frac{2n}{3} + \frac{1}{3}(2j-2i+1)H_{j-i+1} - \frac{j}{3} + \frac{i}{3} + \frac{1}{2}, \quad \text{for } 1 \le i < j \le n,$$
$$V_{n,j,j} = \frac{2}{3}(n+1)H_n - \frac{1}{6}(4j+1)H_j - \frac{1}{6}(4n-4j+5)H_{n-j+1} + \frac{2n}{3} + \frac{7}{9} - \frac{1}{36}[\![j = 1 \vee j = n]\!], \quad \text{for } 1 \le j \le n \text{ and } n \ge 2,$$
$$V_{1,1,1} = 1.$$

Asymptotically, for $i = \alpha n + o(n)$ and $j - i = \delta n + o(n)$,

$$V_{n,i,j} \sim \frac{n}{3}\big(2\delta\log\delta - 2(1-\alpha)\log(1-\alpha) - 2(\alpha+\delta)\log(\alpha+\delta) + 2 - \delta\big), \quad 0 < \delta < 1-\alpha.$$

We derive this theorem from a recursive description of $\mathcal{V}_{n,i,j}$, which is again obtained by considering a call of RQS for an array $A[1..n]$. We assume that the pivot element $v = A[1]$ is the kth smallest element in the array; the probability that this happens is $1/n$, for $1 \le k \le n$.

Now we want to count the total number of moves, i.e., an assignment $A[.] := *$ (appearing in line 8, line 13 or 17), of array elements in the partition procedure PARTITION. We distinguish between two cases:

(1) if $k = 1$ then there is exactly one move during the partitioning phase, namely the assignment of the pivot element in line 17,
(2) if $k \ge 2$ then there may occur the following two situations:

- Element $A[k]$ has a rank in the range $1..(k-1)$ and exactly ℓ elements with a rank in the range $1..(k-1)$ are located in the subarray $A[k..n]$. It follows then that exactly $\ell-1$ elements with a rank in the range $(k+1)..n$ are located in the subarray $A[2..k-1]$. In this situation we obtain then that exactly 2ℓ, i.e., ℓ (line 8) $+\ell-1$ (line 13) $+1$ (line 17), moves are carried out during the partitioning phase. By elementary combinatorial considerations we get the following probability that this event occurs:

$$\frac{1}{(n-1)!}\binom{k-2}{\ell-1}\binom{n-k}{\ell-1}(k-1)!(n-k)! = \frac{\binom{k-2}{\ell-1}\binom{n-k}{\ell-1}}{\binom{n-1}{k-1}}, \quad \text{for } 1 \le \ell \le k-1.$$

(3) Element $A[k]$ has a rank in the range $(k+1)..n$ and exactly ℓ elements with a rank in the range $1..(k-1)$ are located in the subarray $A[k+1..n]$. It follows then that exactly ℓ elements with a rank in the range $(k+1)..n$ are located in the subarray $A[2..k]$. In this situation we obtain then that exactly $2\ell+1$, i.e., ℓ (line 8) $+\ell$ (line 13) $+1$ (line 17), moves are carried out during the partitioning phase. This gives the following probability that this event occurs:

$$\frac{1}{(n-1)!}\binom{k-2}{\ell-1}\binom{n-k}{\ell}(k-1)!(n-k)! = \frac{\binom{k-2}{\ell-1}\binom{n-k}{\ell}}{\binom{n-1}{k-1}}, \quad \text{for } 1 \le \ell \le k-1.$$

Of course, after the partitioning phase the left subarray $A[1..k-1]$ and the right subarray $A[k+1..n]$ are each forming a random permutation of lengths $k-1$ and $n-k$, respectively. But as can be shown easily (permuting the elements with a rank in the range $1..(k-1)$ and of the elements with a rank in the range $(k+1)..n$, respectively, in the input data array leads to easy-describable permutations of the elements in the subarrays $A[1..k-1]$ and $A[k+1..n]$ after the partitioning phase) even more is true. Namely, if we consider only those permutations, such that the number of moves in the procedure PARTITION is exactly $\tilde{\ell}$, with an arbitrary $\tilde{\ell}$, then it also holds that after the partitioning phase the left subarray $A[1..k-1]$ and the right subarray $A[k+1..n]$ are each forming a random permutation of lengths $k-1$ and $n-k$, respectively.

Thus the number of moves during the partitioning phase is independent of the number of moves, which are made during a recursive call of RQS for the right subarray $A[k+1..n]$ (if $k<i$) or the left subarray $A[1..k-1]$ (if $k>j$) and that have to be added to get the total number of moves. This independence property appearing in the distributional recurrence stated in the following proposition would allow also to study higher moments of $\mathcal{V}_{n,i,j}$ or could be a starting point for considerations concerning the limiting distributional behavior of $\mathcal{V}_{n,i,j}$ (see, e.g., [4,7] for limiting distribution results studying the parameter "number of comparisons" in Quickselect).

Proposition 4. *The random variable $\mathcal{V}_{n,i,j}$ satisfies, for $1 \le i \le j \le n$, the following distributional recurrence:*

$$\mathcal{V}_{n,i,j} \overset{(d)}{=} \mathbb{1}_{\mathcal{U}_n<i} \cdot \mathcal{V}^{(1)}_{n-\mathcal{U}_n,i-\mathcal{U}_n,j-\mathcal{U}_n} + \mathbb{1}_{\mathcal{U}_n>j} \cdot \mathcal{V}^{(2)}_{\mathcal{U}_n-1,i,j} + \mathcal{T}_{n,\mathcal{U}_n}, \quad \text{for } 1 \le i \le j \le n,$$

and $\mathcal{V}_{n,i,j} = 0$, if $i<1$, $j<i$ or $j>n$, where the sequences $(\mathcal{U}_n)_n$, $(\mathcal{T}_{n,k})_{n,k}$, $(\mathcal{V}^{(1)}_{n,i,j})_{n,i,j}$ and $(\mathcal{V}^{(2)}_{n,i,j})_{n,j}$ of random variables are all independent. Here $\mathcal{V}^{(1)}_{n,i,j}$ and $\mathcal{V}^{(2)}_{n,i,j}$ are independent copies of $\mathcal{V}_{n,i,j}$, whereas $\mathcal{U}_n$ is uniformly distributed on $\{1,2,\ldots,n\}$. Furthermore, $\mathcal{T}_{n,k}$ is, for $1 \le k \le n$, distributed as follows:

$$\mathbb{P}\left\{\mathcal{T}_{n,1}=1\right\} = 1,$$

$$\mathbb{P}\left\{\mathcal{T}_{n,k}=2\ell\right\} = \frac{\binom{k-2}{\ell-1}\binom{n-k}{\ell-1}}{\binom{n-1}{k-1}}, \quad \text{for } k \ge 2 \text{ and } 1 \le \ell \le k-1,$$

$$\mathbb{P}\left\{\mathcal{T}_{n,k}=2\ell+1\right\} = \frac{\binom{k-2}{\ell-1}\binom{n-k}{\ell}}{\binom{n-1}{k-1}}, \quad \text{for } k \ge 2 \text{ and } 1 \le \ell \le k-1.$$

Proposition 4 immediately gives the following recurrence for the expectation $V_{n,i,j}$ of the total number of moves:

$$V_{n,i,j} = \frac{1}{n}\sum_{k=1}^{i-1} V_{n-k,i-k,j-k} + \frac{1}{n}\sum_{k=j+1}^{n} V_{k-1,i,j} + \frac{1}{n}\sum_{k=1}^{n} \mathbb{E}\left(\mathcal{T}_{n,k}\right), \quad \text{for } 1 \le i \le j \le n, \tag{7}$$

and $V_{n,i,j}=0$, if $i<1, j<i$ or $j>n$.

It holds that $\mathbb{E}\left(\mathcal{T}_{n,1}\right)=1$, whereas for $k \ge 2$ we obtain:

$$\begin{aligned}\mathbb{E}\left(\mathcal{T}_{n,k}\right) &= \frac{1}{\binom{n-1}{k-1}}\sum_{\ell=1}^{k-1}\binom{k-2}{\ell-1}\binom{n-k}{\ell-1}2\ell + \frac{1}{\binom{n-1}{k-1}}\sum_{\ell=1}^{k-1}\binom{k-2}{\ell-1}\binom{n-k}{\ell}(2\ell+1)\\ &= \frac{(n-k+1)(2k-1)-1}{n-1},\end{aligned}$$

where we used the Chu–Vandermonde identity (see, e.g., [3]). Easy computations give then

$$\frac{1}{n}\sum_{k=1}^{n}\mathbb{E}\left(\mathcal{T}_{n,k}\right)=\begin{cases}\dfrac{n}{3}+\dfrac{5}{6}, & \text{for } n\geq 2,\\ 1, & \text{for } n=1.\end{cases}$$

Thus (7) can be written as follows:

$$V_{n,i,j}=\frac{1}{n}\sum_{k=1}^{i-1}V_{n-k,i-k,j-k}+\frac{1}{n}\sum_{k=j+1}^{n}V_{k-1,i,j}+T_{n,i,j},\quad \text{for } 1\leq i\leq j\leq n, \tag{8}$$

with $T_{1,1,1}=1$ and $T_{n,i,j}=\frac{n}{3}+\frac{5}{6}$, for $1\leq i\leq j\leq n$ and $n\geq 2$.

An exact solution of this recurrence can be obtained by simply applying Theorem 2, which shows Theorem 5; the straightforward computations are omitted here.

We remark here that setting $i=j$ leads to results concerning the total number of moves in standard Quickselect, e.g., $\mathcal{V}_{n,j,j}$ is the random variable that counts the total number of moves made by Quickselect when selecting the jth smallest element out of n.

As we have done for passes and comparisons, we can compare the savings of Range Quickselect relative to Quickselect. The following corollaries provide the exact and asymptotic formulæ for the savings and the grand average.

Corollary 4. *Let $\Delta V_{n,i,d}=V_{n,i,i}-V_{n,i-d,i+d}$, that is, the average number of data moves that we save if we use Range Quickselect with range $[i-d..i+d]$ instead of Quickselect with rank i, for $d<i<n+1-d$ and $0\leq d\leq\lfloor(n-1)/2\rfloor$. Then*

$$\begin{aligned}\Delta V_{n,i,d} &= \frac{1}{6}(4i+1)(H_{i+d}-H_i)+\frac{1}{6}(4n-4i+5)(H_{n+1+d-i}-H_{n+1-i})\\ &\quad+\frac{2d}{3}(H_{i+d}+H_{n+1+d-i}+1-2H_{2d+1})+\frac{5}{8}[\![d>0]\!]\\ &\sim\begin{cases}\dfrac{4}{3}d\log\left(\dfrac{n}{d}\right)+\Theta(d), & \text{if } 0<d=o(n),\\ \dfrac{2}{3}c(\alpha,\delta)n-8\log n+O(1), & \text{if } d=\delta\cdot n+o(n), \text{ with } 0<\delta<1/2,\end{cases}\end{aligned}$$

where

$$\begin{aligned}c(\alpha,\delta) &= \delta+(1-\alpha+\delta)\log(1-\alpha+\delta)+(\alpha+\delta)\log(\alpha+\delta)\\ &\quad-\alpha\log\alpha-(1-\alpha)\log(1-\alpha)-2\delta\log(2\delta).\end{aligned}$$

The first asymptotic estimate holds uniformly for all $d=o(n)$ and $n\to\infty$. The second asymptotic estimate holds uniformly for $i=\alpha n+o(n)$ and $n\to\infty$.

Observe that for any valid i and d, $\Delta V_{n,i,d}\sim\frac{1}{3}\Delta C_{n,i,d}$; actually, $V_{n,i,j}\sim\frac{1}{3}C_{n,i,j}+\frac{7}{6}P_{n,i,j}+O(1)$.

Corollary 5. *Let*

$$\overline{V}_{n,d}=\frac{1}{n-2d}\sum_{d<i<n+1-d}V_{n,i-d,i+d},\quad 0\leq\lfloor(n-1)/2\rfloor,$$

that is, $\overline{V}_{n,d}$ is the average total number of moves made by Range Quickselect for a range of size $2d+1$ centered around a rank chosen uniformly at random. Then

$$\begin{aligned}\overline{V}_{n,d} &= n-\frac{(4d+1)(n+1)}{3(n-2d)}\left(H_n-H_{2d+1}\right)+\frac{1}{2}-\frac{4d+1}{3(n-2d)},\quad \text{for } d\geq 1,\\ \overline{V}_{n,0} &= n-\frac{n+1}{3n}H_n+\frac{7}{9}-\frac{1}{18n},\quad \text{for } n\geq 2,\\ \overline{V}_{1,0} &= 1.\end{aligned}$$

Moreover, it holds

$$\overline{V}_{n,d}\sim\begin{cases}n-\dfrac{(4d+1)}{3}\log\left(\dfrac{n}{d}\right)+\Theta(d) & \text{if } 0<d=o(n),\\ \left(1+\dfrac{4\delta\log(2\delta)}{3(1-2\delta)}\right)n+\Theta(1), & \text{if } d=\delta\cdot n+o(n), \text{ with } 0<\delta<1/2.\end{cases}$$

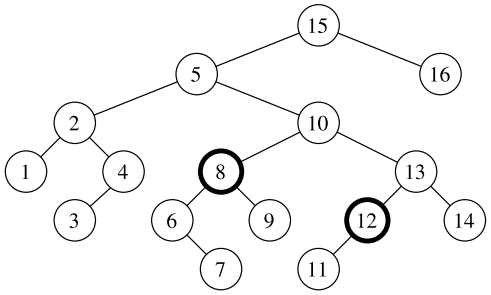

Fig. 1. An example of the parameters $\mathcal{A}_{n,i,j}$, $\mathcal{S}_{n,i,j}$ and $\mathcal{D}_{n,i,j}$.

The first asymptotic estimate holds uniformly for all $d = o(n)$, when $n \to \infty$. Furthermore, the grand average of the savings is

$$\Delta\overline{V}_{n,d} = \overline{V}_{n,0} - \overline{V}_{n,d} \sim \begin{cases} \dfrac{4d}{3}\log(n/d) + \Theta(d), & \text{if } 0 < d = o(n), \\ \dfrac{4\delta\log(1/2\delta)}{3(1-2\delta)}n - \dfrac{1}{3}\log n + \Theta(1), & \text{if } d = \delta\cdot n + o(n),\ 0 < \delta < 1/2. \end{cases}$$

4. Some parameters in binary search trees

Binary search trees are binary trees generated by successively inserting elements into an originally empty tree via a simple recursive algorithm (see for instance [21]). If element x has to be inserted into an empty tree one creates a new node containing x. If element x has to be inserted into a non-empty tree one has to compare x with the element k of the root: if $x < k$ then x will be inserted into the left subtree, whereas if $x \geq k$ then x will be inserted into the right subtree.

For the average-case analysis of the quantities considered for binary search trees we also always use the "random permutation model", i.e., we assume that all $n!$ permutations of a sequence of distinct values $a_1 < a_2 < \cdots < a_n$ are chosen with equal probability as input data to generate a binary search tree of size n.

We define now the three parameters for random binary search trees we will consider in this paper.

The random variable $\mathcal{A}_{n,i,j}$, with $1 \leq i \leq j \leq n$, counts the number of common ancestors (in a rooted tree B a node v is an ancestor of node w if v is lying on the unique path from the root of B to w) of the nodes with rank i and j in a random binary search tree of size n.

The random variable $\mathcal{S}_{n,i,j}$, with $1 \leq i \leq j \leq n$, counts the size of the subtree rooted at the least common ancestor of the nodes with rank i and j in a random binary search tree of size n (i.e., the size of the smallest subtree containing the nodes with rank i and j).

Finally, the random variable $\mathcal{D}_{n,i,j}$, with $1 \leq i \leq j \leq n$, is the *distance* (number of edges) in the unique path from the ith node to the jth node in a random BST of size n.

An example of a binary search tree together with the quantities considered in this paper is given as Fig. 1. The binary search tree depicted is of size 16 and was generated by inserting the elements [15, 5, 10, 16, 8, 2, 13, 12, 1, 14, 6, 4, 7, 9, 3, 11], in that order. The nodes $i = 8$ and $j = 12$ have $\mathcal{A}_{16,8,12} = 3$ common ancestors (nodes 15, 5, and 10). The size of the subtree rooted at the least common ancestor of nodes $i = 8$ and $j = 12$ (which is node 10) is $\mathcal{S}_{16,8,12} = 9$. The distance between the two nodes is $\mathcal{D}_{16,8,12} = 3$.

Both $\mathcal{A}_{n,i,j}$ and $\mathcal{D}_{n,i,j}$ have received attention in the literature [19,22,2]. The corresponding results in the following subsections are thus alternative derivations, using Theorem 2, of the formulæ that were already known. Other authors have also investigated the number of common ancestors and the distance between two randomly chosen nodes in a random binary search tree [15]. The results given here (Section 4.2) about the size of the subtree rooted at the least common ancestor of two given nodes are new, to the best of our knowledge.

4.1. Common ancestors

We consider now the random variable $\mathcal{A}_{n,i,j}$, with $1 \leq i \leq j \leq n$, which counts the number of common ancestors of the nodes with rank i and j in a random binary search tree of size n.

We find that the distribution of $\mathcal{A}_{n,i,j}$ has been dealt with already in Section 2.

Theorem 6. *The random variable $\mathcal{A}_{n,i,j}$ and the number of passes made by Range Quickselect $\mathcal{P}_{n,i,j}$, which has been defined in Section 2.2, are equally distributed, i.e.,*

$$\mathcal{A}_{n,i,j} \stackrel{(d)}{=} \mathcal{P}_{n,i,j}.$$

Therefore, the expected number of common ancestors $A_{n,i,j} = \mathbb{E}\left(\mathcal{A}_{n,i,j}\right)$ of the nodes with rank i and j in a random binary search tree of size n is, for $1 \le i \le j \le n$, given by the following exact and asymptotic formula (which uniformly holds for $1 \le i \le j \le n$ and $n \to \infty$):

$$A_{n,i,j} = H_j + H_{n-i+1} - 2H_{j-i+1} + 1 = \log n + \log(n-i+1) - 2\log(j-i+1) + \mathcal{O}(1).$$

This can be shown easily, where we use a recursive description of $\mathcal{A}_{n,i,j}$, which is obtained via the decomposition of a binary search tree of size $n \ge 1$ into the root node and its left and right subtree. Assuming the random permutation model we get that with probability $1/n$ the root node has rank k, for all $1 \le k \le n$. In any case the root node is a common ancestor of the nodes with rank i and j. If $i \le k \le j$ then there are no further common ancestors, since the nodes with rank i and j are lying in different subtrees of the root. Only if $k < i$ or $k > j$ the nodes with rank i and j are lying in the same subtree and one has to add the common ancestors contained in the left subtree ($k > j$) or the right subtree ($k < i$), respectively.

We get then the following proposition.

Proposition 5. *The random variable $\mathcal{A}_{n,i,j}$ satisfies, for $1 \le i \le j \le n$, the following distributional recurrence:*

$$\mathcal{A}_{n,i,j} \overset{(d)}{=} 1 + \mathbb{1}_{\mathcal{U}_n < i} \cdot \mathcal{A}^{(1)}_{n-\mathcal{U}_n, i-\mathcal{U}_n, j-\mathcal{U}_n} + \mathbb{1}_{\mathcal{U}_n > j} \cdot \mathcal{A}^{(2)}_{\mathcal{U}_n - 1, i, j}, \quad \text{for } 1 \le i \le j \le n,$$

and $\mathcal{A}_{n,i,j} = 0$, if $i < 1$ or $j < i$ or $j > n$, where $\mathcal{U}_n$ is uniformly distributed on $\{1, 2, \ldots, n\}$ and independent of $(\mathcal{A}^{(1)}_{n,i,j})_{n,i,j}$ and $(\mathcal{A}^{(2)}_{n,i,j})_{n,i,j}$, which are independent copies of $(\mathcal{A}_{n,i,j})_{n,i,j}$.

Since it follows from the proposition above and Proposition 1 that $\mathcal{A}_{n,i,j}$ and $\mathcal{P}_{n,i,j}$ satisfy the same distributional recurrence, the first part of Theorem 6 follows. The remaining part is an immediate consequence of Theorem 1.

4.2. The size of the subtree rooted at the least common ancestor

Now we study the random variable $\mathcal{S}_{n,i,j}$, with $1 \le i \le j \le n$, which counts the size of the subtree rooted at the least common ancestor of the nodes with rank i and j in a random binary search tree of size n.

We show the following theorem concerning an exact formula for the expectation

$$S_{n,i,j} := \mathbb{E}\left(\mathcal{S}_{n,i,j}\right).$$

Theorem 7. *The expected size $S_{n,i,j} = \mathbb{E}\left(\mathcal{S}_{n,i,j}\right)$ of the subtree rooted at the least common ancestor of the nodes with rank i and j in a random binary search tree of size n, for $1 \le i \le j \le n$, is given by the following exact and asymptotic formulæ (which holds uniformly for $1 \le i \le j \le n$ and $n \to \infty$):*

$$\begin{aligned} S_{n,i,j} &= (j-i+1)\big(H_j + H_{n-i+1} - 2H_{j-i+1} + 1\big) \\ &= (j-i+1)\big(\log j + \log(n-i+1) - 2\log(j-i+1) + \mathcal{O}(1)\big). \end{aligned}$$

To show this theorem we start with a recursive description of $\mathcal{S}_{n,i,j}$, which follows easily from the decomposition of a binary search tree of size $n \ge 1$ into the root node and its left and right subtree. We only have to take into account that if the root node has rank k, with $i \le j \le k$, then the least common ancestor of the nodes with rank i and j is the root itself and thus the size of the subtree is the size n of the whole tree, whereas if $k < i$ or $k > j$ the least common ancestor of the nodes with rank i and j is contained in the right subtree or the left subtree, respectively, and one has to consider them. This immediately leads to the following proposition.

Proposition 6. *The random variable $\mathcal{S}_{n,i,j}$ satisfies, for $1 \le i \le j \le n$, the following distributional recurrence:*

$$\mathcal{S}_{n,i,j} \overset{(d)}{=} \mathbb{1}_{\mathcal{U}_n < i} \cdot \mathcal{S}^{(1)}_{n-\mathcal{U}_n, i-\mathcal{U}_n, j-\mathcal{U}_n} + \mathbb{1}_{\mathcal{U}_n > j} \cdot \mathcal{S}^{(2)}_{\mathcal{U}_n - 1, i, j} + n \cdot \mathbb{1}_{i \le \mathcal{U}_n \le j}, \quad \text{for } 1 \le i \le j \le n,$$

and $\mathcal{S}_{n,i,j} = 0$, if $i < 1$ or $j < i$ or $j > n$, where $\mathcal{U}_n$ is uniformly distributed on $\{1, 2, \ldots, n\}$ and independent of $(\mathcal{S}^{(1)}_{n,i,j})_{n,i,j}$ and $(\mathcal{S}^{(2)}_{n,i,j})_{n,i,j}$, which are independent copies of $(\mathcal{S}_{n,i,j})_{n,i,j}$.

Proposition 6 leads then to the following recurrence for the expectation $S_{n,i,j}$:

$$\begin{aligned} S_{n,i,j} &= \frac{1}{n}\sum_{k=1}^{i-1} S_{n-k,i-k,j-k} + \frac{1}{n}\sum_{k=j+1}^{n} S_{k-1,i,j} + n \cdot \frac{1}{n}\sum_{k=i}^{j} 1 \\ &= \frac{1}{n}\sum_{k=1}^{i-1} S_{n-k,i-k,j-k} + \frac{1}{n}\sum_{k=j+1}^{n} S_{k-1,i,j} + j - i + 1, \quad \text{for } 1 \le i \le j \le n, \end{aligned} \tag{9}$$

and $S_{n,i,j} = 0$, if $i < 1$ or $j < i$ or $j > n$.

An exact solution of $S_{n,i,j}$, $1 \le i < j \le n$, can be obtained by applying Theorem 2 for the particular toll function $T_{n,i,j} = j - i + 1$, for $1 \le i \le j \le n$. Straightforward computations lead then to the exact formula given in Theorem 7. The results for $S_{n,i,j}$ given there follows immediately from Theorem 6, since it holds

$$S_{n,i,j} = (j - i + 1)A_{n,i,j}, \tag{10}$$

which can be easily proved by induction using (9); note however that $\mathcal{S}_{n,i,j} \overset{(d)}{\neq} (j - i + 1)\mathcal{A}_{n,i,j}$.

4.3. Distance

Finally, we consider the parameter $\mathcal{D}_{n,i,j}$. As in previous subsections, let us begin with the main result, which gives the expected value of $\mathcal{D}_{n,i,j}$.

Theorem 8. *The expected distance $D_{n,i,j} = \mathbb{E}\left(\mathcal{D}_{n,i,j}\right)$ from the node with rank i to the node with rank j in a random binary search tree of size n, for $1 \le i \le j \le n$, is given by the following exact and asymptotic formula (which holds uniformly for $1 \le i \le j \le n$ and $n \to \infty$):*

$$\begin{aligned} D_{n,i,j} &= 4H_{j-i+1} - (H_j - H_i) - (H_{n+1-i} - H_{n+1-j}) - 4 \\ &= 4\log(j + 1 - i) - (\log j - \log i) - (\log(n + 1 - i) - \log(n + 1 - j)) + \mathcal{O}(1). \end{aligned}$$

To prove the theorem, we will first deduce the distributional recurrence that $\mathcal{D}_{n,i,j}$ satisfies. If both nodes i and j lie on the same subtree, the value of $\mathcal{D}_{n,i,j}$ is defined recursively inside that subtree. But when the root is occupied by the kth element, with $i \le k \le j$, then the distance is given by the sum of the depth of i in a random BST of size $k - 1$ plus the depth of j (actually the $(j - k)$th element) in a random BST of size $n - k$ plus 2. Since $\mathcal{A}_{n,i,i}$ is the depth of the ith node in a random BST of size n plus 1, we have the next proposition.

Proposition 7. *The random variable $\mathcal{D}_{n,i,j}$ satisfies, for $1 \le i \le j \le n$, the following distributional recurrence:*

$$\begin{aligned} \mathcal{D}_{n,i,j} \overset{(d)}{=}\ & \mathbb{1}_{\mathcal{U}_n < i} \cdot \mathcal{D}^{(1)}_{n-\mathcal{U}_n, i-\mathcal{U}_n, j-\mathcal{U}_n} + \mathbb{1}_{\mathcal{U}_n > j} \cdot \mathcal{D}^{(2)}_{\mathcal{U}_n - 1, i, j} \\ & + \mathbb{1}_{i \le \mathcal{U}_n \le j} \cdot \left(\mathcal{A}^{(1)}_{\mathcal{U}_n - 1, i, i} + \mathcal{A}^{(2)}_{n-\mathcal{U}_n, j-\mathcal{U}_n, j-\mathcal{U}_n}\right), \quad \text{for } 1 \le i \le j \le n, \end{aligned}$$

and $\mathcal{D}_{n,i,j} = 0$, if $i < 1$ or $j < i$ or $j > n$, where $\mathcal{U}_n$ is uniformly distributed on $\{1, 2, \ldots, n\}$ and independent of $(\mathcal{A}_{n,i,j})_{n,i,j}$, $(\mathcal{D}^{(1)}_{n,i,j})_{n,i,j}$, $(\mathcal{D}^{(2)}_{n,i,j})_{n,i,j}$, $(\mathcal{A}^{(1)}_{n,i,j})_{n,i,j}$, $(\mathcal{A}^{(2)}_{n,i,j})_{n,i,j}$, which are independent of each other. Also, the $(\mathcal{D}^{(1)}_{n,i,j})_{n,i,j}$ and $(\mathcal{D}^{(2)}_{n,i,j})_{n,i,j}$ are independent copies of $(\mathcal{D}_{n,i,j})_{n,i,j}$, and the $(\mathcal{A}^{(1)}_{n,i,j})_{n,i,j}$ and $(\mathcal{A}^{(2)}_{n,i,j})_{n,i,j}$ are also independent copies of $(\mathcal{A}_{n,i,j})_{n,i,j}$.

Standard manipulation of the distributional recurrence above yields the following recurrence for the expectations $D_{n,i,j}$:

$$\begin{aligned} D_{n,i,j} &= \frac{1}{n}\sum_{k=1}^{i-1} D_{n-k,i-k,j-k} + \frac{1}{n}\sum_{k=j+1}^{n} D_{k-1,i,j} + \frac{1}{n}\sum_{k=i}^{j}\left(A_{k-1,i,i} + A_{n-k,j-k,j-k}\right) \\ &= \frac{1}{n}\sum_{k=1}^{i-1} D_{n-k,i-k,j-k} + \frac{1}{n}\sum_{k=j+1}^{n} D_{k-1,i,j} \\ &\quad + \frac{1}{n}\left((j - i)(H_i + H_{n-j+1} - 4) + 2(j - i + 1)H_{j-i+1} - 2\right), \quad \text{for } 1 \le i \le j \le n, \end{aligned} \tag{11}$$

and $D_{n,i,j} = 0$, if $i < 1$ or $j < i$ or $j > n$.

The last step is to use Theorem 2 with the toll function $T_{n,i,j} = ((j - i)(H_i + H_{n-j+1} - 4) + 2(j - i + 1)H_{j-i+1} - 2)/n$ to obtain the closed form for $D_{n,i,j}$ given in Theorem 8. It is worth mentioning that this result may be obtained in a more direct manner by noticing that

$$\mathcal{D}_{n,i,j} \overset{(d)}{=} \mathcal{P}_{n,i,i} + \mathcal{P}_{n,j,j} - 2\mathcal{P}_{n,i,j}.$$

5. Final remarks

The results in the paper are witnesses to the power of generating functions as tools for the solution of many recurrences, in particular, divide-and-conquer recurrences such as those arising in the analysis of Quicksort and Quickselect.

The explicit solution of the general trivariate recurrence for $X_{n,i,j}$ (Theorem 2) has allowed us to obtain exact formulæ for the expected value of several parameters describing the performance of Range Quickselect; that was our original motivation. But as we have shown it is also very useful to analyze several other parameters, including some in connection to random binary search trees, like common ancestors of two given nodes, size of the subtree rooted at the least common ancestor of two

given nodes, or the distance between two given nodes. These quantities had received some attention in past literature, and here our machinery has provided an alternative way to derive the corresponding formulæ. Theorem 2 is also fundamental in the analysis of the number of moves in which a particular element i gets involved during the execution of Quickselect to select the jth smallest element out of n.

Using other techniques not presented in this paper, we have also been able to analyze Multiple Range Quickselect, the obvious extension of Range Quickselect to the problem of multiple selection. We have preliminary results on the average number of passes and comparisons of Multiple Range Quickselect, and we are currently working in the computation of grand averages and in the comparison of Multiple Range Quickselect with Multiple Quickselect.

Several open problems remain. For instance, higher order moments and distributional properties of the number of comparisons or the number of passes in Range Quickselect, as well as other parameters like the number of moves of particular elements. The analysis of expected worst-case quantities of the type $\hat{X}_{n,d} := \mathbb{E}\left(\max_{d<i<n+1-d} \mathcal{X}_{n,i-d,i+d}\right)$ is also of interest; this type of parameters (with $d = 0$) has received often attention in the literature; see for instance [1,4]. However, the analysis of $\hat{X}_{n,d}$ seems at least as difficult as the previous existing analysis and worth a full length paper.

Other interesting questions concern the performance of Range Quickselect in the presence of repeated elements, or variations of the algorithm where the pivot is not chosen at random, but from small samples [16].

Acknowledgements

We thank the anonymous referees who revised the earlier version of the paper and gave many useful comments and advice to improve the overall quality of the paper.

References

[1] L. Devroye, On the probabilistic worst-case time of FIND, Algorithmica 31 (2001) 291–303.
[2] L. Devroye, R. Neininger, Distances and finger search in random binary search trees, SIAM J. Comput. 33 (3) (2004) 647–658.
[3] R.L. Graham, D.E. Knuth, O. Patashnik, Concrete Mathematics, 2nd ed., Addison-Wesley, Reading, MA, 1994.
[4] R. Grübel, U. Rösler, Asymptotic distribution theory for Hoare's selection algorithm, Adv. Appl. Probab. 28 (1996) 252–269.
[5] C.A.R. Hoare, FIND (Algorithm 65), Commun. ACM 4 (1961) 321–322.
[6] C.A.R. Hoare, Quicksort, Comput. J. 5 (1962) 10–15.
[7] H.-K. Hwang, T.-H. Tsai, Quickselect and the Dickman function, Combin. Probab. Comput. 11 (2002) 353–371.
[8] P. Kirschenhofer, H. Prodinger, Comparisons in Hoare's Find algorithm, Combin. Probab. Comput. 7 (1998) 111–120.
[9] P. Kirschenhofer, H. Prodinger, C. Martínez, Analysis of Hoare's FIND algorithm with median-of-three partition, Random Structures Algorithms 10 (1) (1997) 143–156.
[10] D.E. Knuth, Mathematical analysis of algorithms, in: Information Processing'71, Proc. of the 1971 IFIP Congress, North-Holland, Amsterdam, 1972, pp. 19–27.
[11] D.E. Knuth, The Art of Computer Programming: Sorting and Searching, 2nd ed., vol. 3, Addison-Wesley, Reading, MA, 1998.
[12] M. Kuba, On Quickselect, partial sorting and multiple Quickselect, Inform. Process. Lett. 99 (5) (2006) 181–186.
[13] H.M. Mahmoud, Average-case analysis of moves in quick select, in: C. Martínez, R. Sedgewick (Eds.), Proc. of the 6th Workshop on Analytic Algorithmics and Combinatorics, ANALCO, SIAM, 2009, pp. 35–40. Available from: http://www.siam.org/proceedings/analco/2009/anl09_006_mahmoudh.pdf.
[14] H.M. Mahmoud, R. Modarres, R.T. Smythe, Analysis of quickselect: an algorithm for order statistics, RAIRO, Theor. Inform. Appl. 29 (4) (1995) 255–276.
[15] H.M. Mahmoud, R. Neininger, Distribution of distances in random binary search trees, Ann. Appl. Probab. 13 (2003) 253–276.
[16] C. Martínez, D. Panario, A. Viola, Adaptive sampling strategies for quickselect, ACM Trans. Algorithms 6 (3) (2010) 53:1–53:32 + appendices (14 pages).
[17] C. Martínez, A. Panholzer, H. Prodinger, On the number of descendants and ascendants in random search trees, Electron. J. Combin. 5 (#R20) (1998) Available from: http://www.combinatorics.org.
[18] C. Martínez, H. Prodinger, Moves and displacements of particular elements in Quicksort, Theoret. Comput. Sci. 410 (21–23) (2009) 2279–2284.
[19] H. Prodinger, Multiple Quickselect — Hoare's Find algorithm for several elements, Inform. Process. Lett. 56 (3) (1995) 123–129.
[20] U. Rösler, L. Rüschendorf, The contraction method for recursive algorithms, Algorithmica (2001) 29:3–33.
[21] R. Sedgewick, P. Flajolet, An Introduction to the Analysis of Algorithms, Addison-Wesley, Reading, MA, 1996.
[22] R. Seidel, C. Aragon, Randomized search trees, Algorithmica 16 (1996) 464–497.