Journal of Engineering Science *Vol. XXXI, no. 1 (2024), pp. 18 - 33*
Fascicle *Electronics and Computer Science* *ISSN 2587-3474*
Topic *Computers and Information Technology* *eISSN 2587-3482*

https://doi.org/10.52326/jes.utm.2024.31(1).02
UDC 004.42

MEAN-OF-2-4 QUICKSORT

Ion Bolun*, ORCID: 0000-0003-1961-7310

Technical University of Moldova, 168 Stefan cel Mare Blvd., Chisinau, Republic of Moldova
*Corresponding author: Ion Bolun, *ion.bolun@isa.utm.md*

Received: 02. 12. 2023
Accepted: 04. 02. 2024

Abstract. By combining the Median-of-three and Regrouping-3 quicksort methods, the Joint quicksort is proposed, largely free from the shortcomings of the first two. For example, the time complexity of Joint quicksort, in case of lists of n equal elements, is $O(n)$. Analysis of the dependence of Quicksort time complexity on the ratio of the derived sublist sizes shows a relatively slow increase in sorting time as the ratio in question decreases from 0.5 to 0.1. The proposed category of Mean-of-K (MeK) sorting algorithms provides for the determination of pivot elements as the mean of K elements. It is shown that, in terms of sorting time, at $K \in [1, 4]$ and size r of the list/sublist of elements to be sorted, it is convenient to use (roughly): Insertion sort at $r \leqslant 9$, Me2 quicksort at $10 \leqslant r \leqslant 21$, Me3 quicksort at $22 \leqslant r \leqslant 46$, and Me4 quicksort at $r > 46$, yielding the Mean-of-2-4 quicksort method. It was found that the determination of pivot elements in the Median-of-three method requires more calculations than in the Mean-of-3 method; respectively, using Mean-of-3 method could also reduce sorting time. Of course, Mean-of-2-4 method could reduce this duration even further.

Keywords: *basic Quicksort, Median-of-three quicksort, Regrouping-3 quicksort, pivot element determination, time complexity, algorithm comparison.*

Rezumat. Combinând metodele de sortare rapidă Mediana-a-trei și Regrupare-3, este propusă sortarea rapidă Îmbinată, lipsită în mare măsură de neajunsurile primelor două. De exemplu, complexitatea temporală a sortării rapide îmbinate, în cazul unor liste din n elemente egale, este $O(n)$. Analiza dependenței duratei sortării Rapide de raportul dintre dimensiunile sublistelor derivate arată la o creștere relativ lentă a duratei sortării cu micșorarea raportului în cauză de la 0,5 șa 0,1. Categoria de algoritmi de sortare Media-a-K (MeK) propusă prevede determinarea elementelor pivot ca media a K elemente. Este demonstrat că, în ce privește durata sortării, la $K \in [1, 4]$ și dimensiunea r a listei/sublistei de elemente de sortat, este oportun de folosit (aproximativ): sortarea prin Inserție la $r \leqslant 9$, sortarea Me2 la $10 \leqslant r \leqslant 21$, sortarea Me3 la $22 \leqslant r \leqslant 46$ și sortarea Me4 la $r > 46$, obținând astfel metoda de sortare rapidă Media-a-2-4. S-a constatat că determinarea elementelor pivot la metoda Mediana-a-trei necesită mai multe calcule decât la metoda Media-a-3; respectiv, folosirea metodei Media-a-3 ar putea reduce și durata sortării. Bineînțeles, metoda Media-a-2-4 ar putea reduce această durată și mai mult.

Cuvinte cheie: *sortarea rapidă de bază, sortarea rapidă Mediana-a-trei, sortarea rapidă Regrupare-3, determinarea elementului pivot, complexitate temporală, comparare algoritmi.*

1. Introduction

Sorting - the ordering of entities according to a parameter (key) is widely used in computer science. As simple is the essence, so frequent sorting is encountered in practice, so wide is the multitude of approaches, and so easy it is to construct simple sorting algorithms. The first known sorting algorithm, *Radix*, based on the decimal numbering system, was proposed and implemented in electromechanical tabulators by Herman Hollerith in 1890 [1]. *Merge sort* was proposed by Jame W. Bryce and implemented in 1938 in the Collator machine, for merging cards from two different stations in a single sorting operation. In 1945 John Von Neumann implemented this method in the electronic computer "EDVAC" [1].

From the first publications in the field, appeared in the 1950s, dozens of sorting algorithms are proposed and research continues. Most of them were invented in the period 1954-1985 [2]. Approx. 30 such algorithms are described in [1] and a list of 74 chronologically systematized algorithms is published in 2014 [3].

New sorting algorithms are also proposed after 2014, including *pdqsort* published in 2021 [4] and *RevWay Sort* published in 2022 [5]. However, so far, there is no a generalized sorting algorithm that would best suit all situations in practice [6]. Thus, the search for a suitable sorting algorithm for specific situations is still current [2].

One of the most used is *Quicksort* [2]. For randomized data, especially for large lists, it is slightly faster than *Merge sort* and *Heapsort* [7].

At the same time, traditional Quicksort also has some shortcomings in certain situations, which led to the proposal of some of its developments. The best known of them is the *Median-of-3 quicksort* (Mo3), proposed in [9]. In this paper, some well-known and also newly proposed algorithms based on the traditional Quicksort algorithm are described and comparatively characterized.

2. Basic Quicksort

2.1. The essence of basic Quicksort

Quicksort was proposed by C.A.R. Hoare in 1961 [9], but also later independently in [10] and possibly by other authors. Later, some developments of it were also published. In the following, the version of basic Quicksort from [10] will be used.

Since there is a direct entity-key correspondence, in the following we will mainly operate with the keys of the respective entities called elements. Quicksort provides [9, 10] the choice, first, of the pivot element, say the last element in the list (it can be any) - s_{00}. Then the list of elements, as a result of $n - 1$ pairwise comparisons of element s_{00} with each of the other $n - 1$ elements, is regrouped (partitioned) into two sublists of elements and element s_{00}: one sublist, say G_{11}, will contain the smaller elements as s_{00}, and the second sublist, be G_{12} – elements equal to or greater than s_{00}. The comparison will be made consecutively with elements at the beginning of the list until an element greater than the pivot is identified, then with elements at the end of the list until an element smaller than the pivot is identified, and subsequently the two elements thus identified will swap with the place; the process continues in the same way until all $n - 1$ elements have been compared. Finally, the pivot element will swap with the first element in the second sublist. Thus, in the first step, the ordering of the elements is $G_{11} \rightarrow s_{00} \rightarrow G_{12}$, where for $\forall i \in G_{11}$ occurs $i \rightarrow s_{11}$, and for $\forall j \in G_{12}$ occurs $s_{00} \rightarrow j$. Element s_{00} is already in the final position.

In the second step, as a result of comparing the pivot element $s_{11} \in G_{11}$ (the last one in the sublist), $|G_{11}| > 1$, with the other elements of sublist G_{11}, sublist G_{11} is also regrouped into two sublists and element s_{11}: $G_{21} \to s_{11} \to G_{22}$, where $i|_{i\in G_{21}} \to s_{11}$ and $s_{11} \to j|_{j\in G_{22}}$. If sublist G_{11} is empty or contains only one element, i.e. $|G_{11}| \leqslant 1$, then it is no longer taken into consideration in this and the following steps. Similar actions in this step are performed on the G_{12} sublist, obtaining $G_{23} \to s_{12} \to G_{24}$. Elements s_{11} and s_{12} are already in their final positions. The process continues until all derived sublists at some step k become unitary or empty, which signifies the termination of the ordering procedure.

The maximum number of pairwise element comparison operations ($U_{\max}$) occurs when, for each regrouping of a list/sublist into two sublists, one and only one non-empty element sublist will be formed. Such a situation occurs if the initial list of elements is ordered or in reverse order, or if all elements are equal. In this case [10]:

$$U_{\max} = \frac{n(n-1)}{2}. \tag{1}$$

On the contrary, the minimum number of pairwise element comparison operations ($U_{\min}$) occurs when, at each regrouping of a list/sublist into two sublists, two sublists of the same size will be formed. This condition can only be met for [11]

$$n = 1 + (1 + 2(1 + 2(1 + \ldots 2(1 + 2))\ldots) = \sum_{i=0}^{k} 2^i = 2^{k+1} - 1, k = 1,2,\ldots \tag{2}$$

and in this case it takes place [10]:

$$U_{\min} = (n + 1)\log_2(n + 1) - 2n. \tag{3}$$

At each current step, the elements in the list/sublist that are regrouped are written in the same table. Each element in the regrouping list/sublist gets a comparison operation with the pivot element and, if applicable, an additional 0.5 swap operations (the swap operation between the two sublists common to the two elements); finally, one more place swap operation is performed on the pivot element with the first element in the second sublist. Of course, the concrete implementation on a specific computer also involves other operations, but the basic ones are the nominated ones.

2.2. Dependence of Quicksort laboriousness on derived sublist sizes

The influence of the deviation of the value of n from that of Eq. (2) on the laboriousness of sorting is of interest. Let n have such a value that, at each regrouping of a list/sublist into two sublists, one of them will contain kd elements, and the other $(1 - d)k$ elements, where $d \in (0, 1)$. Obviously the minimum value of the sorting time T is obtained at $d = 1/2$ for all iterations. The sorting time of the algorithm is determined [12] by the following recurrent relation $T(k,d) = k + T(\lfloor d(k - 1)\rfloor) + T(\lceil (1 - d)(k - 1)\rceil, d)$. If, in approximate calculations, to operate with fractional numbers of entities, i.e. $T(k, d) = k + T(d(k - 1)) + T((1 - d)(k - 1), d)$, then the solution of this recurrent equation is [12]

$$g(d) = \lim_{n\to\infty} \frac{T(n,d)}{T(n,\frac{1}{2})} = \frac{1}{-d\log_2 d - (1-d)\log_2(1-d)}. \tag{4}$$

The essence of the quantity $g(d)$ – how many times the duration $T(n,d)$ is greater than the duration $T(n, 1/2)$ at $d \in [1/n, 1/2]$. The graph of the function $g(d)$ at $d \in [0.025; 0.5]$ is shown in Figure 1. From Figure 1 it can be seen that the laboriousness of Quicksort increases relatively slowly when d decreases from 0.5 (the ratio between the sizes of the two derived sublists is 1:1) to approx. 0.1 (the ratio between the sizes of the two derived sublists is 1:9). Under uniform distribution of elements, the probability that the pivot element will be the one at position *s* of the final (sorted) list is 1/*n*. Obviously, $d = s/n$. Under such assumptions and taking into account that $g(d) = g(1 - d)$, the mean value ($g_{\text{med}}(n)$) of $g(d)$, at $d \in [1/n, 1/2]$ and *n* even, is determined as

$$g_{\text{med}}(n)|_{d\in[\frac{1}{n},\frac{1}{2}]} = \frac{2}{n}\sum_{s=1}^{n/2} \frac{1}{-d\log_2 d - (1-d)\log_2(1-d)}, \tag{5}$$

where $d = s/n$.

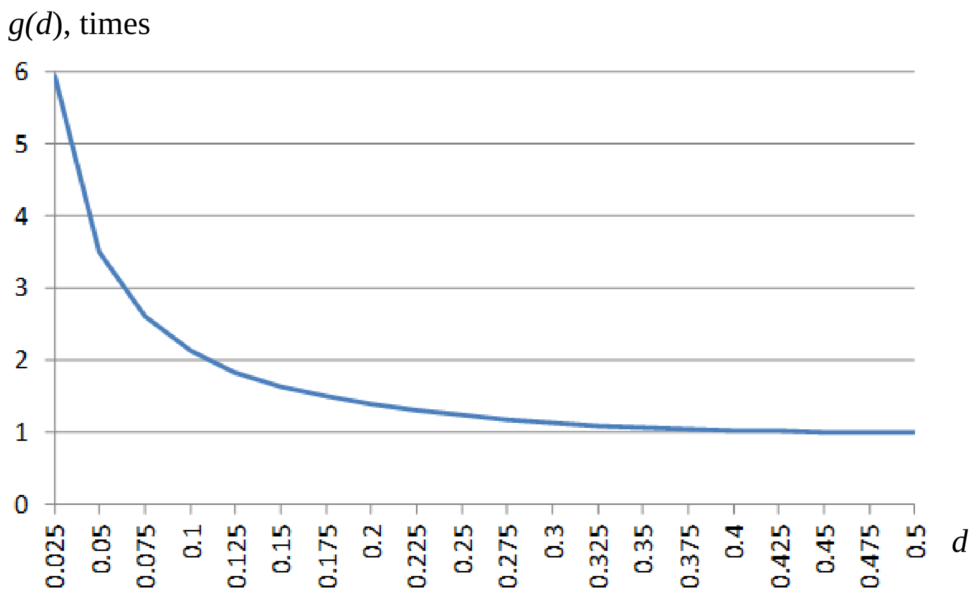

Figure 1. The increase of Quicksort laboriousness with the decrease of $d < 1/2$.

If the quantity $g(d)$ can be used for the analysis of a Quicksort algorithm apart, then the quantity $g_{\text{med}}(n)$ can be used for the comparative analysis of some Quicksort algorithms. The graph of the function $g_{\text{med}}(n)$ at $d \in [0.025; 0.5]$ is shown in Figure 2.

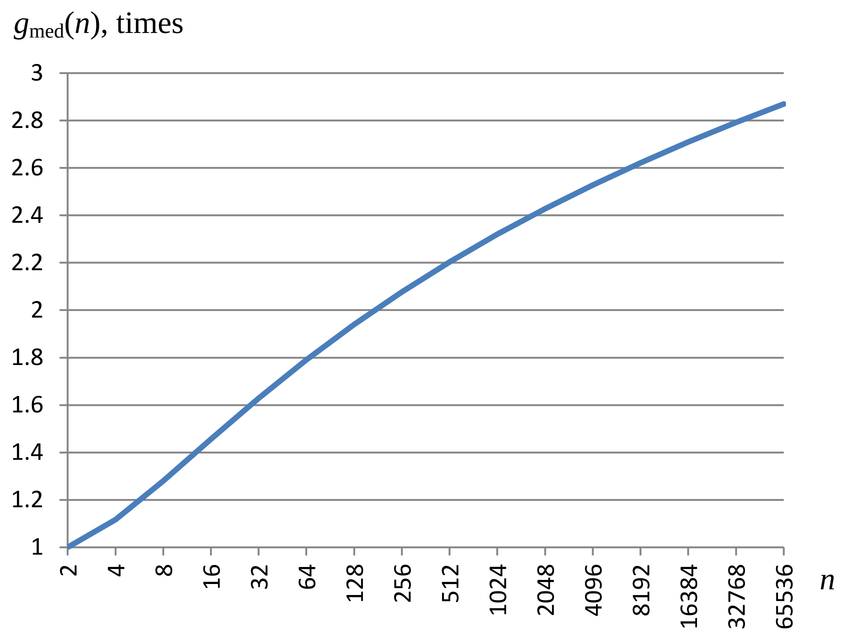

Figure 2. Dependence on *n* of the average value of $g(d)$ at $d \in [0.025; 0.5]$.

From Figure 2 it can be seen that the function $g_{\text{med}}(n)$ is increasing, that is, the efficiency of Quicksort decreases with the increase in the number n of elements of the initial list. At the same time, this growth is relatively slow, the abscissa scale being logarithmic. For example, $g_{\text{med}}(2048) = 2.43$, and $g_{\text{med}}(65536) = 2.87$.

Thus, approximately (taking into account the assumptions admitted in [12]), the average number of operations with Quicksort when sorting a list of 65536 elements is 2.87 times higher, compared to the situation when $d = 1/2$, both for the initial list as well as for all derived sublists.

3. Quicksort variants based on traditional Quicksort

As mentioned in Section 2.1, the maximum laboriousness of basic Quicksort occurs when the initial list of elements is ordered or in reverse order, or if all elements are equal. The Quicksort variants described in this section are partially or completely free of these shortcomings.

The **Median-of-three quicksort** (Mo3) [8] differs from the basic Quicksort by choosing the pivot element from three elements of the list/sublist to regroup: the first, the last, and the element in the middle position (obtained as the arithmetic mean truncated to integers of the positions of the first and last elements). These three elements are ordered and the middle element is used as the pivot element. Mo3 sorting allows reducing the volume of calculations for cases with lists of elements already ordered or in reverse order. However, this variant does not reduce the volume of calculations in the case of initial lists with entities that have equal keys.

The **Regrouping-3 quicksort** (R3), proposed in [10] and later in [13], operates efficiently in cases of equal elements, too. Its difference from basic Quicksort consists in regrouping each list/sublist not into two but into three sublists: the first sublist of elements smaller than the pivot element, the second – of elements equal to the pivot element, and the third – of elements larger than the pivot element. For example, the first step will obtain the sublists $G_{11} \rightarrow S_{11} \rightarrow G_{12}$, where the sublist S_{11} contains all elements equal to the pivot element s_{00}. Also, since equal elements usually occur less often, to reduce the amount of calculations, first check whether the element belongs to the subset G_{11} and only then to the S_{11} or the G_{12}. In this case, the number of operations to regroup the current subset will usually be less. The elements of sublist S_{11} are already in their final positions. In the same way, the regrouping of the new sublists is carried out.

Obviously, the number of pairwise element comparison operations (U), in the case when all n elements are equal and first the current element's membership in the first sublist is checked, is $U = 2(n - 1)$. At the same time, if there are no equal elements in the initial list, then the R3 sort requires twice the number of pairwise comparison operations than in the basic Quicksort. Moreover, R3 sorting does not reduce the amount of calculations in the case of already ordered or reverse-ordered initial lists.

Joint quicksort. Comparing the Mo3 and R3 sorts, it can be seen that they complement each other: the shortcoming of Mo3 (does not reduce the volume of calculations for initial lists of equal elements) is eliminated by the R3 sort, and the shortcoming of the R3 sort (does not reduce the volume of calculations in the case of initial lists already ordered or in reverse order) are mitigated by Mo3 sorting. So, combining these two algorithms results in a more efficient sort - Joined quicksort. This allows reducing the volume of calculations both for lists of equal elements and for lists of elements already ordered or in reverse order. The essence of Joint quicksort:

a) the pivot element is determined according to Median-of-three sorting;

b) each list/sublist is regrouped not into two but into three sublists according to Regrouping-3 sorting.

4. Determining the pivot element

Based on the idea of Mo3 sorting, the question arises: why Mo3 and not, for example, Mo2, Mo4 or, in general, Mo*K*? In what follows in this section, the variants: Me1 (Mean-of-1 - conventional average at basic Quicksort), Me2, Me3, and Me4 are investigated comparatively under certain conditions described in Section 4.1. The notation Me2, Me3, and Me4 is used rather than Mo2, Mo3, and Mo4 because the pivot keys are determined in a different way.

4.1. Description of the list of elements to be sorted and the sorting conditions

There are several variations of Quicksort. The following variant will be investigated in the section. Let be a list of r entities that have key values (elements) $\{1, 2, 3, ..., r - 1, r\}$ and are placed within the list arbitrarily. So, the distribution of entity key values is deterministic uniform with the same distance of one unit between neighboring entities in the final ordered list.

At each step, for the sublist (hereafter the list) of size r that regroups into two sublists, the pivot element u is determined as the arithmetic mean (truncated to integer) of K elements, the neighboring ones being positioned in the list at approximately equal distances. For example, at $K = 2$, for the pair of elements $\{j, k\}$ positioned on the first (1) and last (r) positions, respectively, is obtained $u = \lfloor (j + k)/2 \rfloor$. Likewise, at $K = 3$, for the triad of elements $\{j, k, l\}$, positioned on the first (1), the one in the middle ($\lfloor (1 + r)/2 \rfloor$) and, respectively, the last (r) positions is obtained $u = \lfloor (j + k + l)/3 \rfloor$.

By comparison with the pivot element, the list of r elements is regrouped into two sublists, such that each element in the first sublist is less than or equal to the smallest element in the second sublist. Regrouping into sublists continues until all sublists contain no more than one element each. Such a sort is called Mean-of-*K* (Me*K*) sort. Also:

r is even;

P_i - the probability that the list of r elements regroups into two sublists, one of which contains $i = \underline{1, r/2}$ elements, and the other contains ($r - i$) elements;

N_i - the number of different regroupings in two sublists, one of which contains $i = \underline{1, r/2}$ elements, and the other contains ($r - i$) elements, also taking into account the regroupings obtained from the end of the list;

N - the total number of different regroupings of the list of r elements into two sublists, also taking into account the regroupings obtained from the end of the list.

With such an approach, along with the sortings of the categories $K = \{2w + 1\}$, $w = 1, 2, 3, ...$, which can be seen as a generalization of the Median-of-three (Mo3) sorting, they also make sense Mean-of-*K* (Me*K*) sorts of categories $K = 2w$, $w = 1, 2, 3, ...$, some of which will be examined in this section.

The comparison of Me*K* sortings, at different values of $K = 1, 2, 3, ...$, will be carried out within the assumptions of [12], in the case of which Eqs (4) and (5) hold.

4.2. Mean-of-1 quicksort

Me1 sort involves groupings with the mean value of an element (conventional mean - the value of the element itself) used as the pivot element.

Since at Me1 as pivot element u of regrouping the list of r elements into two sublists can be any of the r elements, in total there can be r different regroupings of the same probability. At *r* even, the variants of regrouping the list of *r* elements into two sublists are: (1, *r* − 1), (2, *r* − 2), (3, *r* − 3), ..., (*r*/2 − 1, *r*/2 + 1), (*r*/2, *r*/2), (*r*/2, *r*/2), (*r*/2 + 1, *r*/2 − 1), ..., (r − 2, 2) , (*r* − 1, 1). Here, in regrouping (*x*, *y*), *x* and *y* specify the number of elements in the first and second sublists of the regrouping, respectively. It can also be seen that for *r* even and arbitrary selection of the pivot element *u* (for example, the first element in the list), the number of operations required:

- of the regrouping (1, *r* − 1) is equal to that of the regrouping (*r* − 1, 1);
- of the regrouping (2, *r* − 2) is equal to that of the regrouping (*r* − 2, 2);

………………………………..

- of the regrouping (*r*/2 − 1, *r*/2 + 1) is equal to that of the regrouping (*r*/2 + 1, *r*/2 − 1);
- of the regrouping (*r*/2, *r*/2), obtained starting from the beginning of the list, is equal to that of the regrouping (*r*/2, *r*/2), obtained starting from the end of the list.

At *r* even, there are *r*/2 cases where the first sublist has $i = \underline{1, r/2}$ elements and *r*/2 cases where the second sublist has $i = \underline{1, r/2}$ elements. Thus, in total there are *r* regroupings with *r* different pivot elements. So, at *r* even, one has:

$$P_i = 2/r,\ i = \underline{1, r/2}.$$

Under the assumptions in [12] and *r* = *n*, Eqs. (4) and (5) hold, and the dependencies *d*(*n*) and $g_{\text{med}}(n) = g_{\text{Me1}}(n)$ of *g*(*d*), at *d* ∈ [1/*n*, 1/2] and *n* even, in graphical form are shown in Figures 1 and 2.

4.3. Mean-of-2 quicksort

Me2 sorting assumes regroupings $M_{i,j}$ with the average value of two elements, either *j* and *k*, used as the (conventional) pivot element *u* = (*j* + *k*)/2, $(j,k) = \underline{1, r}, j \neq k$:

$$M_{j,k} = \left(\lfloor\frac{j+k}{2}\rfloor, r - \lfloor\frac{j+k}{2}\rfloor\right), (j,k) = \underline{1, r}, j \neq k. \tag{6}$$

The variants of regrouping the list of *r* elements into two sublists are the same as for Me1: (1, *r* − 1), (2, *r* − 2), (3, *r* − 3), ..., (*r*/2 − 1, *r*/2 + 1), (*r*/2, *r*/2), (*r*/2, *r*/2), (*r*/2 + 1, *r*/2 − 1), ..., (*r* − 2, 2), (*r* − 1, 1). Similarly, at *r* even, the number of regrouping operations required:

- of the regrouping (1, *r* − 1) is equal to that of the regrouping (*r* − 1, 1);
- of the regrouping (2, *r* − 2) is equal to that of the regrouping (*r* − 2, 2);

………………………………..

- of the regrouping (*r*/2 − 1, *r*/2 + 1) is equal to that of the regrouping (*r*/2 + 1, *r*/2 − 1);
- of the regrouping (*r*/2, *r*/2), obtained starting from the beginning of the list, is equal to that of the regrouping (*r*/2, *r*/2), obtained starting from the end of the list.

So, only the cases of regroupings *i* ∈ [1, *r*/2] can be examined, but the obtained result will be multiplied by 2. At *i* ∈ [1, *r*/2], from the beginning of the list one has the regroupings:

(1, *r* − 1): 1|2, 2|1 - in total 2 cases (N_1= 2) , because $\lfloor(1 + 2)/2\rfloor = 1$ and $\lfloor(2 + 1)/2\rfloor = 1$;

(2, *r* − 2): 1|3, 1|4, 2|3 and vice versa - in total 3 + 3 = 6 cases ($N_2 = 6$), since $\lfloor(1 + 3)/2\rfloor = 2$, $\lfloor(1 + 4)/2\rfloor = 2$ and $\lfloor(2 + 3)/2\rfloor = 2$;

(3, *r* − 3): 1|5, 1|6, 2|4, 2|5, 3|4 and vice versa - in total 5 + 5 = 10 cases ($N_3 = 10$);

(4, *r* − 4): 1|7, 1|8, 2|6, 2|7, 3|5, 3|6 and vice versa - in total 7 + 7 = 14 cases ($N_4 = 14$);

…………………………..

(*i, r* − *i*): in total $N_i = (2i - 1) + (2i - 1) = 2(2i - 1)$ cases;

…………………………..

(*r*/2, *r*/2): 1|*r* - 1, 1|*r*, 2|*r* - 2, 2|*r* - 1, 3|*r* - 3, 3|*r* - 2, 1|*r* - 1, ..., (*r*/2 − 1|*r*/2 + 1), *r*/2|*r* - *r*/2; in total 2(*r* − 1) cases.

Respectively, one gets:

$N_i = 2\cdot 2(2i - 1) = 4(2i - 1),\ i = \underline{1, r/2}.$

$$N = \sum_{i=1}^{\frac{r}{2}} N_i = \sum_{i=1}^{\frac{r}{2}} 4(2i-1) = 4\sum_{i=1}^{\frac{r}{2}} (2i-1) = 8\sum_{i=1}^{\frac{r}{2}} i - \frac{4r}{2} = $$
$$= 8\left(\frac{r}{2}+1\right)\frac{r}{4} - \frac{r}{2} = r^2.$$

So,

$$P_i = \frac{N_i}{N} = \frac{4(2i-1)}{r^2}, i = \underline{1, r/2}. \tag{7}$$

Thus, under the assumptions of [12] and *r* = *n*, one obtains

$$g_{\text{Me2}}(n)|_{d\in[1/n,1/2]} = $$
$$= \sum_{d=1/n}^{1/2} P_{dn} g(d) = \frac{4}{n^2}\sum_{d=1/n}^{1/2} \frac{2dn-1}{-d\log_2 d - (1-d)\log_2(1-d)}. \tag{8}$$

In graphical form, the dependence of the mean value $g_{\text{Me2}}(n)$ on $g(d)$, at $d \in [1/n, 1/2]$ and *n* even, is shown in Figure 3.

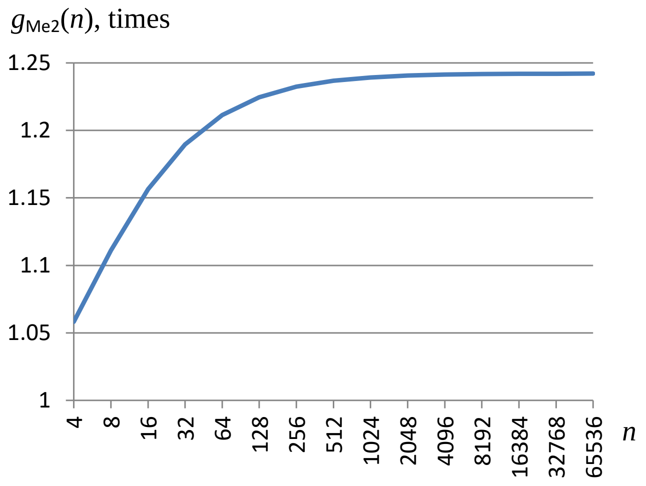

Figure 3. Dependence $g_{\text{Me2}}(n)$ at $d \in [1/n, 1/2]$.

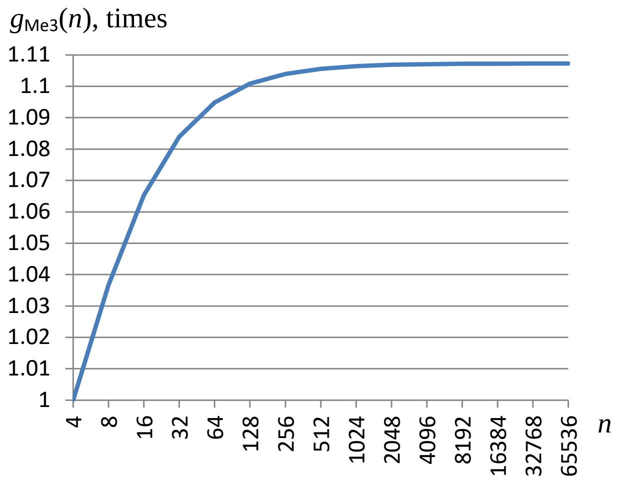

Figure 4. Dependence $g_{\text{Me3}}(n)$ at $d \in [1/n, 1/2]$.

The function $g_{\text{Me2}}(n)$ is increasing, and starting with, approximately, *n* = 2048 it changes a little: from $g_{\text{Me2}}(2048)$ = 1.240550 ≈ 1.2406 (times) and up to $g_{\text{Me2}}(65536)$ = 1.242014 ≈ 1.2420 (times).

4.4. Mean-of-3 quicksort

Me3 sorting assumes regroupings $M_{i,j,k}$ with the average value of three elements, either *j*, *k* and *l*, used as the pivot (conventional) element *u* = (*j* + *k* + *l*)/3, $(j, k, l) = \underline{1, r}, j \neq \{k, l\}, k \neq l$:

$$M_{j,k,l} = \left(\lfloor\frac{j+k+l}{3}\rfloor, r - \lfloor\frac{j+k+l}{3}\rfloor\right), (j, k, l) = \underline{1, r}, j \neq \{k, l\}, k \neq l. \tag{9}$$

In the Me3 sort, the variants of regrouping the list of *r* elements into two sublists are the same as in the Me2 sort. Likewise, only the cases of regroupings *i* ∈ [1, *r*/2], can be examined, but the obtained result will be multiplied by 2. For simplicity, the case of using three elements *j*, *k* and *l* to determine the respective pivot element will be noted *j.k.l*. In total, with the same three elements, there are 3! different variants distinguished by their order: *j.k.l*, *j.l.k*, *k.j.l*, *k.l.j*, *l.j.k*, *l.k.j* and *l.j.k*. In the following, only the variants in ascending order of the constituent elements will be explicitly specified (i.e. $j < k < l$), and the obtained result will be multiplied by 3! From the beginning of the list, in ascending order of the constituent elements and $i = \underline{1, r/2}$ (the obtained result will be multiplied by 2·3!), we have the Me3 regroupings:

(1, r − 1): in total there are $N1$ = 0 cases, because $\lfloor(1 + 2 + 3)/3\rfloor = 2 > 1$;

(2, r − 2): 1.2.3, 1.2.4, 1.2.5 and 1.3.4, for each variant *x.y.z* of which $\lfloor(x + y + z)/3\rfloor = 2$ occurs, i.e. 4 cases. Thus, in total there are $N_2 = 4·2·3!$ cases;

(3, r− 3): 1.2.6, 1.2.7, 1.2.8, 1.3.5, 1.3.6, 1.3.7, 1.4.5, 1.4.6, 2.3.4, 2.3.5, 2.3. 6 and 2.4.5 for each variant x.y.z of which $\lfloor(x + y + z)/3\rfloor = 3$ occurs, i.e. 12 cases. Thus, in total there are $N_3 = 12·2·3!$ cases;

(4, r − 4): 1.2.9, 1.2.10, 1.2.11, 1.3.8, 1.3.9, 1.3.10, 1.4.7, 1.4.8, 1.4.9, 1.5.6, 1.5. 7, 1.5.8, 1.6.7, 2.3.7, 2.3.8, 2.3.9, 2.4.6, 2.4.7, 2.4.8, 2.5.6, 2.5.7, 3.4.5, 3.4.6, 3.4.7, 3.5.6, for each variant *x.y.z* of which loc $\lfloor(x + y + z)/3\rfloor = 4$ occurs, i.e. 25 cases. Thus, in total there are $N_4 = 25·2·3!$ cases;

and so on.

Respectively, one gets:

$$N_i = \begin{cases} 0, \text{at } i = 0 \\ 8·3!, \text{at } i = 2 \\ N_{i-1} + 2·3! \begin{cases} 2(i+1) + \frac{5}{2}(i-3), \text{at } i \text{ odd} \\ 2i + \frac{5}{2}(i-2), \text{at } i \text{ even} \end{cases} & , i = \overline{3, r/2} \end{cases} .$$

$$N = \sum_{i=1}^{r/2} N_i.$$

$$P_i = \frac{N_i}{N}, i = \overline{1, r/2}.$$

Thus, under the assumptions of [12] and $r = n$, one obtains

$$g_{\mathrm{Me3}}(n)|_{d \in [1/n, 1/2]} = \sum_{d=1/n}^{1/2} P_{dn} g(d) = \sum_{d=1/n}^{1/2} \frac{P_{dn}}{-d \log_2 d - (1-d)\log_2(1-d)}. \quad (10)$$

In graphical form, the dependence of the mean value $g_{\mathrm{Me3}}(n)$ on $g(d)$, at $d \in [1/n, 1/2]$ and n even, is shown in Figure 4.

The function $g_{\mathrm{Me3}}(n)$ is increasing, and starting with, approximately, n = 4096 it changes little: from $g_{\mathrm{Me3}}(4096)$ = 1.107057 (times) to $g_{\mathrm{Me3}}(65536)$ = 1.107254 (times).

4.5. Mean-of-4 quicksort

Me4 sorting assumes regroupings $M_{i,j,k,l}$ with the average value of four elements, let j, k, l and m, used as the pivot (conventional) element $u = (j + k + l + m)/4$, $(j, k, l, m) = \underline{1, r}, j \neq \{k, l, m\}, k \neq \{l, m\}, l \neq m$:

$$M_{i,j,k,l} = \left(\lfloor\frac{j+k+l+m}{4}\rfloor, r - \lfloor\frac{j+k+l+m}{4}\rfloor\right), (j, k, l, m) = \underline{1, n}, j \neq \{k, l, m\}, k \neq \{l, m\}, l \neq m. \quad (11)$$

In the Me4 sort, the variants of regrouping the list of r elements into two sublists are the same as in the Me2 sort. Likewise, only the cases of regroupings $i \in [1, r/2]$, can be examined, but the obtained result will be multiplied by 2. For simplicity, the case of using four elements (j, k, l, m) to determine the pivot element respectively, *j.k.l.m* will be noted. In total, with the same four elements, there are 4! different variants that differ by the order of the constituent elements. In the following, only the variants in ascending order of the constituent elements will be explicitly specified (i.e. $j < k < l < m$), and the obtained result will be multiplied by 4! From the beginning of the list, in ascending order of the constituent elements and $i = \overline{1, r/2}$ (the obtained result will be multiplied by 2·4!), we have the Me4 regroupings:

(1, $r - 1$): - in total there are $N_1 = 0$ cases, because $\lfloor(1 + 2 + 3 + 4)/4\rfloor = 2 > 1$;
(2, $r - 2$): 1.2.3.4, 1.2.3.5, for each variant *x.y.z.w* of which $\lfloor(x + y + z + w)/4\rfloor = 2$ occurs, i.e. 4 cases. Thus, in total there are $N_2 = 2{\cdot}2{\cdot}4!$ cases;
(3, r − 3): 1.2.3.6, 1.2.3.7, 1.2.3.8, 1.2.3.9, 1.2.4.5, 1.2.4.6, 1.2.4.7, 1.2.4.8, 1.2.5.6, 1.2.5.7, 1.3.4.5, 1.3.4.6, 1.3.4.7, 1.3.5.6, 2.3.4.5 and 2.3.4.6, for each variant *x.y.z.w* of which $\lfloor(x + y + z + w)/4\rfloor = 3$ occurs, i.e. 12 cases. Thus, in total there are $N_3 = 16{\cdot}2{\cdot}4!$ cases;
and so on.

Respectively, one gets:

$$N_i = \begin{cases} 0, \text{ at } i = 1 \\ \Delta N_2 = 2{\cdot}2{\cdot}4!, \text{ at } i = 2 \\ N_{i-1} + \Delta N_i, i = \overline{3, n/2} \end{cases}.$$

$$\Delta N_i = \Delta N_{i-1} + 2{\cdot}4!(i-2)4{\cdot}2 + 2{\cdot}4! \begin{cases} 4, \text{at } i = 3 \\ 7, \text{at } i = 4 \\ 9, \text{at } i = 3j + 5, j = 0, 1, 2, \dots \\ 12, \text{at } i = 3j + 6, j = 0, 1, 2, \dots \\ 15, \text{at } i = 3j + 7, j = 0, 1, 2, \dots \end{cases}, \text{ at } i = \overline{3, r/2},$$

$$N = \sum_{i=1}^{r/2} N_i.$$

$$P_i = \frac{N_i}{N}, i = \overline{1, r/2}.$$

Thus, under the assumptions of [12] and $r = n$, one obtains

$$g_{\text{Me4}}(n)|_{d\in[1/n, 1/2]} = \sum_{d=1/n}^{1/2} P_{dn} g(d) = \sum_{d=1/n}^{1/2} \frac{P_{dn}}{-d\log_2 d - (1-d)\log_2(1-d)}. \quad (12)$$

In graphical form, the dependence of the mean value $g_{\text{Me4}}(n)$ on $g(d)$ at $d \in [1/n, 1/2]$ and n even is shown in Figure 5.

The function $g_{\text{Mo4}}(n)$ is increasing up to, approximately, $n = 2048$, and then weakly decreasing. Thus, $g_{\text{Me4}}(2048) = 1.062616$, $g_{\text{Me4}}(4096) = 1.062610$ (times), $g_{\text{Me4}}(8192) = 1.062604$ (times) and $g_{\text{Me4}}(65536) = 1.062598$ (times).

4.6. Mean-of-2-4 quicksort

The Me2-4 quicksort is based on the results of comparing the Me1, Me2, Me3 and Me4 sorts. As already mentioned, the approximate comparison of these sorts can be performed based on the mean value $g_{\text{Me}K}(n)$ of $g(d)$ at $d \in [1/n, 1/2]$. For this purpose, Figure 6 shows the dependences $g_{\text{Me1}}(n)$, $g_{\text{Me2}}(n)$, $g_{\text{Me3}}(n)$, and $g_{\text{Me4}}(n)$ at $d \in [1/n, 1/2]$.

According to Figure 6, Me2, Me3, and Me4 sorts are considerably more efficient (in terms of the number of operations required) than Me1, with the former's advantages over Me1 sort increasing significantly as n increases. Of course, the computation of the pivot elements requires additional computations, but at values of n not too small, they, already knowing the value of the pivot element, are much smaller than the total number of sort operations.

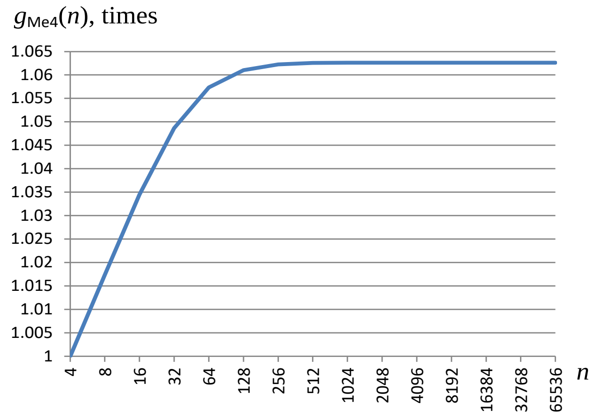

Figure 5. Dependence of mean value $g_{Me4}(n)$ at $d \in [1/n, 1/2]$.

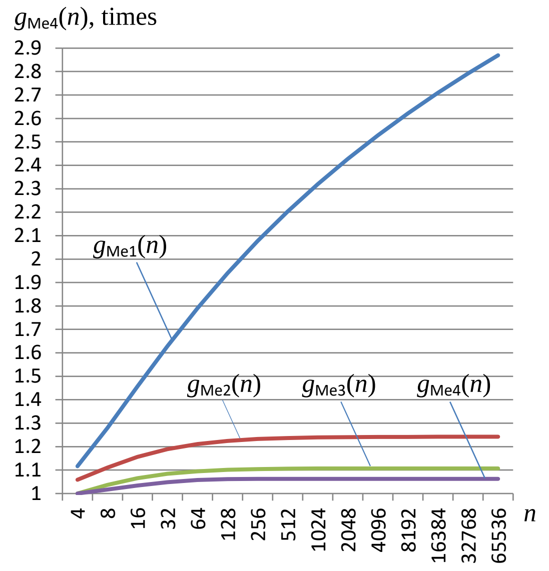

Figure 6. Dependences of mean values $g_{Me1}(n)$, $g_{Me2}(n)$, $g_{Me3}(n)$, and $g_{Me4}(n)$ at $d \in [1/n, 1/2]$.

In general, relationships take place

$$g_{Me1}(n) > g_{Me2}(n) > g_{Me3}(n) > g_{Me4}(n) \tag{13}$$

and

$$g_{Me1}(n) - g_{Me3}(n) > g_{Me2}(n) - g_{Me3}(n) > g_{Me3}(n) - g_{Me4}(n). \tag{14}$$

So, with the increase in the number K of elements, on the basis of which the pivot element is determined, the advantage of using a larger number of such elements decreases and may even be negative (due to the increase in the time needed to calculate the pivot elements). Thus, relationships can be expected to occur

$$g_{MeK}(n) - g_{Me(K+1)}(n) > g_{Me(K+1)}(n) - g_{Me(K+2)}(n),\ K = 1, 2, 3, \ldots \tag{15}$$

Therefore, it may not be appropriate to use too large values for K. However, the use of Me4 sorting might be appropriate in some cases, especially at relatively large values of n. For this purpose, only the dependences of the mean values $g_{Me2}(n)$, $g_{Me3}(n)$ and $g_{Me4}(n)$ of $g(d)$, at $d \in [1/n, 1/2]$ are shown in Figure 7.

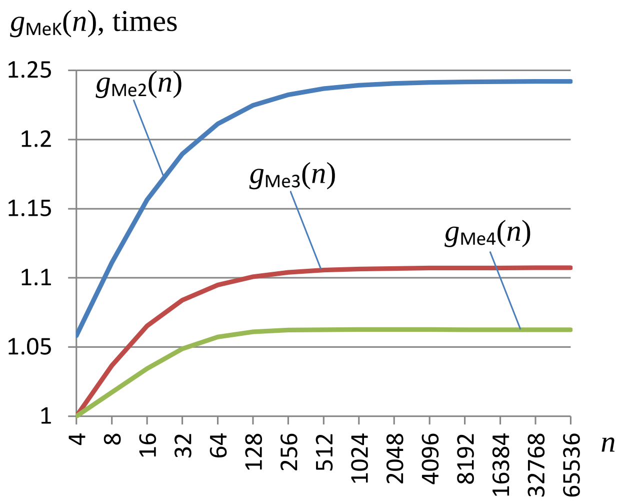

Figure 7. Dependences of mean values $g_{Me2}(n)$, $g_{Me3}(n)$, and $g_{Me4}(n)$ at $d \in [1/n, 1/2]$.

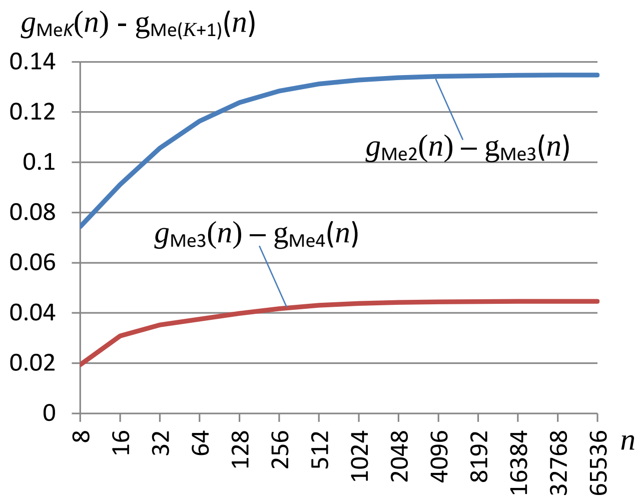

Figure 8. Dependences of differences $g_{MeK}(n) - g_{Me(K+1)}(n)$ at $K = \{2, 3\}$.

From Figure 7 it can be seen that the value of $g_{MeK}(n)$ with increasing K stabilizes (to some extent) at lower values of n.

A clearer quantitative difference in the efficiency of different sortings is presented by the value of the differences $g_{MeK}(n) - g_{Me(K+1)}(n)$, $K = 1, 2, 3, \ldots$ For $K = 1$, $K = 2$ and $K = 3$, some of these are shown in Figure 8 and Table 1.

Since for the sorts Me1, Me2, Me3 and Me4 the sorting times $T(n, 1/2)$ are approximately equal, the sizes $100(g_{MeK}(n) - g_{Me(K+1)}(n))$, $K = 1, 2, 3$ means, roughly, how much (in %) the Me(K+1) sort is more efficient than the MeK sort.

From Table 1 it can be seen that the differences $g_{Me1}(n) - g_{Me2}(n)$, $g_{Me1}(n) - g_{Me3}(n)$, $g_{Me1}(n) - g_{Me4}(n)$, $g_{Me2}(n) - g_{Me3}(n)$ and $g_{Me3}(n) - g_{Me4}(n)$ are increasing with respect to n, the first three of which, at n = 65536, reach considerable values, respectively (approximately): 1.628, 1.762 and 1.807. So only at very small values of n can it be reasonable to use the Me1 sorting over the Me2 one. At the same time, if the difference $g_{Me2}(n) - g_{Me3}(n)$ is relatively large, constituting (approximately) 0.133 at n = 1024 and 0.135 at n = 65536 (i.e. a reduction of the laboriousness of sorting by more than 13%), then the $g_{Me3}(n) - g_{Me4}(n)$ is not negligible in some cases, being (approximately) 0.044 at n = 1024 and 0.045 at n = 65536 (i.e. a reduction in sorting laboriousness of more than 4.4%).

Table 1

Absolute value of differences $g_{MeK}(n) - g_{MeL}(n)$

n	$g_{Me1}(n) - g_{Me2}(n)$	$g_{Me1}(n) - g_{Me3}(n)$	$g_{Me1}(n) - g_{Me4}(n)$	$g_{Me2}(n) - g_{Me3}(n)$	$g_{Me3}(n) - g_{Me4}(n)$
8	0.169002	0.243352	0.262709	0.074350	0.019357
16	0.299643	0.390807	0.421634	0.091164	0.030827
32	0.438706	0.544349	0.579620	0.105643	0.035271
64	0.578794	0.695242	0.732756	0.116448	0.037514
128	0.715023	0.838784	0.878588	0.123761	0.039804
256	0.844680	0.973064	1.014813	0.128384	0.041749
512	0.966614	1.097782	1.140836	0.131168	0.043054
1024	1.080642	1.213428	1.257245	0.132786	0.043818
2048	1.187120	1.320822	1.365054	0.133702	0.044232
4096	1.286613	1.420823	1.465270	0.134210	0.044448
8192	1.379800	1.514288	1.558846	0.134488	0.044558
16384	1.467298	1.601936	1.646549	0.134638	0.044614
32768	1.549702	1.684419	1.729061	0.134717	0.044642
65536	1.627536	1.762296	1.806952	0.134760	0.044656

Of course, the laboriousness of the sorting algorithms depends on the particularities of the implementation (programming language, computer, etc.). In approximate calculations, the laboriousness of operations will be considered (in conventional operations): of reading a number - c operations; of adding two numbers - a operations; of comparing two numbers - a operations; of exchange with the place of two elements - s operations; of dividing two numbers - h operations; of determining the position of the middle element of a list for the Me3 sorting - two operations of reading the positions of the first and last elements of the list, an operation of addition and an operation of division of two numbers (truncated to integers), so in total $a + 2c + h$ operations; of determining the positions of two non-marginal elements positioned at approximately equal distances from the neighboring elements (out of the four) in the list for the Me4 sorting – three addition operations, six read operations and one divide operation, so a total of $3a + 6c + h$ operations.

Under these assumptions, the consideration of operations with the determination of the pivot element can be carried out in the following way. Let the list/sublist, at the given sort step, consist of r elements and, in the case of sorting:

- Me1, the last element of the list is taken as the pivot element u. Only one read operation is then required, i.e. c conventional operations (conv. ops.);
- Me2, to determine the pivot (conventional) element u, the first and last elements of the list are taken. Then two read operations are required, one operation to add the two elements and one operation to divide; so in total $a + 2c + h$ conv. ops.;
- Me3, to determine the pivot (conventional) element u, the first, last and middle elements of the list are taken. Then there are required: one operation to determine the position of the middle element, three read operations, two operations to add the three elements and one divide operation; so in total $a + 2c + h + 3c + 2a + h = 3a + 5c + 2h$ conv. ops.;
- Me4, to determine the pivot (conventional) element u, the first, last and two non-marginal elements positioned at approximately equal distances from the neighboring elements (among the four) in the list are taken. Then it is necessary: an operation to determine the positions of the two non-marginal elements, four reading operations, three operations to add the four elements and a dividing operation; so in total $3a + 6c + h + 4c + 3a + h = 6a + 10c + 2h$ conv. ops.

Given the pivot element, it remains to determine the number of operations with regrouping the list/sublist of r elements into two sublists. Let the list of r elements be sorted in ascending order. The pivot element u (in the Me1 sort last in the list, and in the Me2, Me3 and Me4 sorts - a conventional element that is considered not to be contained in the list) is compared with the list elements as in the basic Quicksort. That is, the pivot element is consecutively compared with the elements at the beginning of the list until an element, say j, greater than the pivot is identified, then with the elements at the end of the list until an element, say k, smaller than the pivot is identified, and subsequently, the two elements thus identified change with the place; the process continues in the same way until all $r - 1$ (at Me1) or r (at 2, Me3, and Me4) elements have been compared. At each comparison of the pivot element u with another element there can be two options, each of probability 1/2: $j < u$ or $j > u$, respectively, $k < u$ or $k > u$. If $j < u$ or $k > u$, then each comparison has two read operations (of elements u and j or u and k, respectively) and a comparison operation of the two elements - a total of $2c + a$ conventional operations, since the elements j and k remain in place. But if $j > u$ and $k < u$, then each comparison has two read operations (of elements u and j or u and k, respectively), one comparison operation of the two elements and 0.5 exchange operations with the place of elements j and k - in total $2c + a + 0.5s$ conventional operations. In the case of sorting Me1 at the end, additionally, the pivot element will be swapped with the first element in the second sublist – a total of 2 read operations and a swap operation with the place of two elements.

So, if the pivot element is known, the number of conventional operations with regrouping the list/sublist of r entities into two sublists is roughly equal to $(r - 1)(a + 2c)/2 + (r - 1)(a + 2c) + 0.5s)/2 = (r - 1)(2a + 4c + 0.5s)/2$ on Me1 sort (since the pivot element is an element of the list) and with $r(a + 2c)/2 + r(a + 2c + 0.5s)/2 = r(2a + 4c + 0.5s)/2$ conv. ops. on sorts Me2, Me3, and Me4 (since the pivot element is a conventional one and is considered not to correspond to any element of the list).

Thus, the total number ($R_{\text{Me}K}$) of conventional operations with the regrouping of the list/sublist of r entities into two sublists is approximately:

- $R_{\text{Me1}}(r) = (r - 1)(2a + 4c + 0{,}5s)/2 + c$, when Me1 sorting;
- $R_{\text{Me2}}(r) = r(2a + 4c + 0{,}5s)/2 + a + 2c + h$, when Me2 sorting;
- $R_{\text{Me3}}(r) = r(2a + 4c + 0{,}5s)/2 + 3a + 5c + 2h$, when Me3 sorting;
- $R_{\text{Me4}}(r) = r(2a + 4c + 0{,}5s)/2 + 6a + 10c + 2h$, when Me4 sorting.

Let's determine at what values of r it is appropriate to use each of the sorts under discussion. Broadly speaking, using the MeK sort is as time complexity as using the Me(K+1) sort at

$$G_{\text{Me}(K+1),K}(r) = (R_{\text{Me}(K+1)}(r) - R_{\text{Me}K}(r))/R_{\text{Me}K}(r) = g_{\text{Me}K}(r) - g_{\text{Me}(K+1)}(r). \quad (16)$$

Example 1. Let: $a = 1$ conv. ops.; $c = 0.5a$; $h = s = 3a$.

Then $R_{\text{Me1}}(r) = 2.75r - 2.25$ conv. ops.; $R_{\text{Me2}}(r) = 2.75r + 5$ conv. ops.; $R_{\text{Me3}}(r) = 2.75r + 11.5$ conv. ops. and $R_{\text{Me4}}(r) = 2.75r + 17$ conv. ops.

Based on Eq. (16), we obtain: $(R_{\text{Me2}}(r) - R_{\text{Me1}}(r))/R_{\text{Me1}}(r) = 7.75/(2.75r - 2.25)$ conv. ops.; $(R_{\text{Me3}}(r) - R_{\text{Me2}}(r))/R_{\text{Me2}}(r) = 6.5/(2.75r + 5)$ conv. ops.; $(R_{\text{Me4}}(r) - R_{\text{Me3}}(r))/R_{\text{Me3}}(r) = 5.5/(2.75r + 11.5)$ conv. ops.. Then, taking into account the data of Table 1, it can be concluded that, approximately, it is appropriate to use the sorting: Me1 at $r \leqslant 12$, Me2 at $12 < r \leqslant 21$, Me3 at $21 < r \leqslant 46$ and Me4 at $r > 46$. At the same time, in [9] it is shown that for $r \leqslant 9$ instead of Mo3 sorting it is appropriate to use Insertionsort. This result can also be extended to Me3 sorting, mostly close to Mo3, and MeK sorting, respectively. So, for sorting Me1 only $10 \leqslant r \leqslant 12$ remains - very small area.

Thus, one can roughly conclude that for lists/sublists of size:

- $r \leqslant 9$ is appropriate to use Insertionsort;
- $10 \leqslant r \leqslant 21$ it is appropriate to use Me2 sorting;
- $22 \leqslant r \leqslant 46$ it is appropriate to use the Me3 sorting;
- $r > 46$ it is appropriate to use the Me4 sorting.

Based on the result of Example 1, it may be appropriate the Mean-of-2-4 sorting which for lists/sublists of size r uses: Insertionsort at $r \leqslant 9$, Me2 sort at $10 \leqslant r \leqslant 21$, Me3 sort at $22 \leqslant r \leqslant 46$, and Me4 sort at $r > 46$. Also, if the initial list of elements is known to contain multiple equal elements then the Mean-of-2-4 sort can be combined with the Regrouping-3 sort.

5. Comparing Median-of-three and Mean-of-3 quicksorts

The version of Mo3 sort proposed in [8], for sublists of size $r \leqslant 9$, uses Insertionsort. That is why when comparing Mo3 and Me3 sorts it is appropriate to consider the same conditions, i.e. whether Insertionsort is used or not. Either in both use Insertionsort at $r \leqslant 9$.

Also, for the list of elements to be sorted described in Section 4.1, the procedures used to determine the pivot elements lead to the same regroupings of lists into sublists. So, the time complexity difference between Mo3 and Me3 sorts for each list/sublist is only determined by the number of operations required to determine the pivot element. In turn, the two procedures for determining the pivot element differ only in that in the case of Mo3, after determining the three elements, they are sorted and the middle element is used as the pivot element; while in the case of Me3, after determining the three elements, their arithmetic mean is calculated, which serves as the pivot element.

Thus, the number of operations required to determine a pivot element in the Mo3 and Me3 sorts differs only at the last stage: in Mo3 – sorting the three elements, and in Me3 – calculating the arithmetic mean of the three elements. When Mo3 sorting, for the three-element sorting procedure described in [14], on average, 10.5 operations to read one element and 3 operations to compare two elements are required – in total, at the complexities of the operations used in Example 1, 8.25 conventional operations. In Me3 sorting, for the procedure of calculating the arithmetic mean of three elements, 3 operations of reading an element, 2 operations of addition and one operation of division are required - in total, at the complexities of operations used in Example 1, 6.5 conventional operations.

So, when determining a pivot element, the Me3 sort requires 1.75 fewer conventional operations than the Mo3 sort. Given that the Me2-4 sort is preferable to the Me3 sort, the number of sort operations, when using the Me2-4 sort and the list of elements to sort described in Section 4.1, is even smaller compared to the Mo3 sort. It should also be noted that for some lists of elements to be sorted, different from the one described in Section 4.1, in some cases the pivot elements of the Me3 sort may be more successful, and for others - the pivot elements of the Mo3 sort may be more successful and, respectively, the solutions obtained. That is why, in specific cases, it may be appropriate to make the comparison in question through computer simulation.

6. Conclusions

By combining the Median-of-three and Regrouping-3 quicksort methods, the Joint quicksort is proposed, which allows reducing the volume of calculations both for lists of equal elements and for lists of already ordered elements or in reverse order. Also is introduced the category of Mean-of-*K* (Me*K*) quicksort algorithms that differs from the Median-of-three sort [9] by determining the pivot element as the mean of *K* elements, positioned, in the list to regroup, at approximately equal distances among them. For the basic Quicksort method (Me1) under some assumptions, $g(d)$ dependencies are analyzed – how many times the sorting time $T(n,d)$ is greater than the time $T(n,1/2)$ at $d \in [1/n, 1/2]$. The laboriousness of the Me1 sort increases relatively slowly as d decreases from 0.5 (the ratio between the sizes of the two derived sublists is 1:1) to approx. 0.1 (the ratio in question is 1:9).

For the comparative analysis of Me1, Me2, Me3, and Me4 methods, the average value ($g_{\text{med}}(n)$) of $g(d)$ at $d \in [1/n, 1/2]$ is determined. The calculation results show that the dependencies $g_{\text{Me1}}(n)$, $g_{\text{Me2}}(n)$, $g_{\text{Me3}}(n)$, and $g_{\text{Me4}}(n)$ at $d \in [1/r, 1/2]$, where $g_{\text{Me}K}(n)$ is $g_{\text{med}}(n)$ on Me*K* sorting, are increasing, that is, the efficiency of quick sort decreases as the number of elements of the initial list increases. At the same time, this growth is slower and slower, especially at higher values of *K*. Thus: $g_{\text{Me1}}(1024) \approx 2.320$, $g_{\text{Me1}}(2048) \approx 2.428$ and $g_{\text{Me1}}(65536) \approx 2.870$; $g_{\text{Me2}}(1024) \approx 1.239$, $g_{\text{Me2}}(2048) \approx 1.241$ and $g_{\text{Me2}}(65536) \approx 1.242$; $g_{\text{Me3}}(1024) \approx 1.106$, $g_{\text{Me3}}(2048) \approx 1.107$ and $g_{\text{Me3}}(65536) \approx 1.107$; $g_{\text{Me4}}(1024) \approx 1.063$, $g_{\text{Me4}}(2048) \approx 1.063$ and $g_{\text{Me4}}(65536) \approx 1.063$. Moreover, Eqs. (13) hold, i.e. the sorts with the higher *K* value, at $K \in \{1, 2, 3, 4\}$, are more efficient regarding the time of sorting.

Since for $K \in \{1, 2, 3, 4\}$, the sorting time $T(n,1/2)$ practically does not depend on *K*, the quantities $100(g_{\text{Me}K}(n) - g_{\text{Me}(K+1)}(n))$, $K = 1, 2, 3$ means, roughly, how much (in %) the time of the Me(*K*+1) sort is faster than that of the Me*K* sort. As a result of calculations, it is found that the differences $g_{\text{Me1}}(n) - g_{\text{Me2}}(n)$, $g_{\text{Me2}}(n) - g_{\text{Me3}}(n)$ and $g_{\text{Me3}}(n) - g_{\text{Me4}}(n)$ are increasing, reaching at $n = 65536$ values, respectively (approximately): 1.628, 0.135 and 0.045. So, at $n =$

65536, using Me4 sorting allows to reduce the sorting time by approx. 4.5% compared to Me3, which is sometimes not negligible.

The results of the comparison of Median-of-three and Mean-of-3 sorts show that the Mean-of-3 sort requires fewer calculations with the determination of pivot elements and, respectively, could also reduce the time of the sort. Of course, Mean-of-2-4 sorting could reduce this time even further. At the same time, cases are not excluded when the Median-of-three quicksort could be more efficient than the Mean-of-3 quicksort or even the Mean-of-2-4 quicksort, if the pivot elements in the Median-of-three quicksort, for the respective lists of elements, would be significantly more successful. Such cases could be identified by computer simulation.

Conflicts of Interest: The author declares no conflict of interest.

References

1. Knuth, D. *The Art of Computer Programming. Sorting and Searching.* 2nd Edition. Addison-Wesley, Reading, Massachusetts, USA, 1998, 780 p.
2. Heineman, G.T.; Pollice, G.; Selkow, S. *Algorithms in a Nutshell*, 2nd Edition. O'Reilly Media, Sebastopol, California, USA, 2015, 424 p.
3. Kadam, P.; Kadam, S. Root to Fruit (2): An Evolutionary Approach for Sorting Algorithms. *Oriental Journal of Computer Science & Technology* 2014, 7(3), pp. 369-376.
4. Peters, O.R.L. *Pattern-defeating Quicksort.* arXiv:2106.05123v1 [cs.DS] 9 Jun 2021. Available online: https://arxiv.org/pdf/2106.05123.pdf (accessed on 25.09.2023).
5. Saha, S.; Sarkar, S.; Patra, R.; Bhattacharjee, S. New Sorting Algorithm - RevWay Sort. In: *Computational Advancement in Communication, Circuits and Systems. Lecture Notes in Electrical Engineering*, Mitra M., Nasipuri M. Kanjilal, M.R. (eds), Springer, Singapore,2022, 2022, pp. 203–210.
6. Levintin, A. *Introduction to Design and analysis of Algorithms.* 2nd Edition. Pearson, New Delhi, India, 2013, 544 p.
7. Skiena, S.S. *The Algorithm Design Manual.* Springer, New York, USA, 2008, 730 p.
8. Sedgewick, R. *Quicksort.* Stanford University, Stanford Computer Science Report STAN-C.S~75-492, Ph.D. Thesis, May 1975, 344 p.
9. Hoare, C.A.R. Partition (Algorithm 63); Quicksort (Algorithm 64); Find (Algorithm 65). *Comm. ACM* 1961, 4, pp. 321-322.
10. Bolun, I.T. One modification of Johnson's ordering algorithm. In: *Systems and means of integrated information processing.* ISRP, Chisinau, Republic of Moldova, 1981, pp. 43-46 [in Russian].
11. Bolun, I.T. Algorithm of ordering of jobs in a two-machine flow-shop. In: *Mathematical support for automated control systems.* Stiinta, Chisinau, Republic of Moldova, 1984, pp. 102-112 [in Russian].
12. Filmus, Y. Solving recurrence relation with two recursive calls. *Computer Science Stack Exchange.* Available online: https://cs.stackexchange.com/q/31930 (accessed on 20.10.2023).
13. Bentley, J. *Programming Pearls.* 2nd Edition. Addison-Wesley Professional, New York, USA, 1999, 283 p.
14. Sedgewick, R. The Analysis of Quicksort Programs. *Acta Informatica* 1977, 7, pp. 327—355.

Citation: G Bolun, I. Mean-of-2-4 Quicksort. *Journal of Engineering Science* 2024, XXXI (1), pp. 18-33. https://doi.org/10.52326/jes.utm.2024.31(1).02.