New selection algorithm based on generating sequences

Y N Artamonov¹ and Y V Alyaeva²

¹ Department of Science Research, Federal State Budgetary Scientific Institution “Gosmetodcenter”, 51 Lyusinovskaya St., 115093 Moscow, Russia
² Institute of Information Technologies and Computer Science, National Research University “Moscow Power Engineering Institute”, 14 Krasnokazarmennaya St., 111250 Moscow, Russia

E-mail: junart@mail.ru

Abstract. The paper proposes a new deterministic selection algorithm with computational complexity \(O(n)\), called cs-select, which is a modification of the quickselect algorithm. The changes concern the selection of reference elements. Instead of selecting some elements of the sequence itself as references, the cs-select algorithm proposes to use finite segments of complete sequences, that allow to represent any natural number in a given range as the sum of elements from a selected finite segment of the complete sequence. It is theoretically justified that in the cs-select algorithm, the number of comparisons when searching for the \(k\)-th statistic in a sequence of length \(n\) can converge to the value \(2n\) with unlimited growth of \(n\) for some complete sequences. It was also shown that different variations of the complete sequences make it possible to reduce the latent constant in \(O(n)\) less than 2.

1. Introduction
The selection algorithm is a fundamental task of theoretical computer science and is widely used in processing numerical data: finding elements of the specified rank in the selection, sorting algorithms, finding nearest neighbors in numerical sequence, the problem of finding convex hulls of a set.

The formulation of this problem goes back to C.A.R. Hoare and is as follows [1]:

\[
X = \{x \mid \exists j : |x| = n\}
\]

(1)

is a set and that is a relation \(\prec\) on \(X\) and integer \(1 \leq k \leq n\). Required to find \(k\)-th smallest element, i.e. element \(x \in X\), for which there are at least \(k-1\) elements

\[
\forall j = 1, \ldots, (k-1): x_j \prec x
\]

(2)

and no less than \(k\) elements

\[
\forall j = 1, \ldots, k: x_j \prec x \lor x_j = x.
\]

(3)

Let's introduce the following notation: \(x = Q(X, k)\). In special cases, we obtain:

\[
Q(X, 1) = \min(X), Q(X, n) = \max(X), Q(X, \left\lfloor \frac{n}{2} \right\rfloor) = \text{median}(X).
\]

(4)

Usually, for convenience of analysis \(X\) is considered as a set (without repeating elements).
The first implementation of the algorithm specified in paper [1] (FIND), was proposed by C. E. R. Hoar himself, as a modification of his quicksort sorting algorithm. The FIND algorithm is also often called quickselect. The following estimation of the number of comparisons is proved for this algorithm [2]:

\[
C(n,k) = 2 \cdot ((n+1) \cdot H_n - (n-k+3) \cdot H_{n-k+1} - (k+2) \cdot H_k + n+3),
\]

where \(H_k = \sum_{i=1}^{k} \frac{1}{i} \) - partial sum of harmonic series.

For boundary cases, we have:

\[
C(n,1) = C(n,n) = 2n + o(n),
\]
\[
C(n,\text{median}) = 3.39n + o(n).
\]

As noted in the paper [3], in practice, the FIND algorithm is popular, because many other algorithms are much slower on average.

The SELECT algorithm is considered in a series of articles [3, 4, 5], which currently provides the best average result:

\[
C(n,k) = n + \min(n-k,k) + o(n).
\]

Moreover, the article [3] provides a comparative analysis of SELECT and FIND. Compared to FIND, SELECT requires only a small additional stack space for recursion. It was shown that, SELECT surpasses rather complex implementations of FIND.

It should be noted that all the considered algorithms assume the use of random selection. Therefore, the effectiveness of their application is estimated for the average case. For specially selected input data, the considered algorithms do not guarantee linear time \(O(n)\) of finding the \(k\)-th statistics. Therefore, there remains a theoretical interest in finding algorithms that guarantee \(O(n)\) for the worst case with acceptable hidden constants in \(O(n)\). So the article [6] proposes the algorithm PICK - a modification of the algorithm quickselect, PICK guarantees \(O(n)\) for the worst case. However, as shown in [6], the number of comparisons of the PICK algorithm does not exceed \(5.4305 \cdot n\). Therefore, applying the PICK algorithm in practice is not appropriate if there are such large hidden constants in \(O(n)\).

In this article, based on the results presented in [7], the authors propose a deterministic algorithm called cs-select, it guarantees \(O(n)\) in the worst case, with a hidden constant close to 2 for many practically important cases. The experiments conducted by the authors confirm that the proposed algorithm is on average comparable to quickselect in terms of computational complexity.

2. The description of cs-select algorithm

By analogy with quickselect and quicksort, the cs-select algorithm is a modification of the cs-sort sorting algorithm described in the paper [7].

Regular algorithms for generating sequences are considered in [7], which are defined as follows: for a given natural number \(n\) let’s title a finite sequence of natural numbers \(B = [b_1, b_2, \ldots, b_m]\) \(b_i \geq b_{i+1}\) as generating sequence , if any natural number \(k < n\) can be represented as the sum of the elements of this sequence:

\[
\forall k < n : k = \sum_{i=1}^{m} e_i \cdot b_i, e_i \in \{0,1\},
\]
at that $n = \sum_{i=1}^{m} b_i$.

To obtain generating sequences in [7], the properties of the following CAlgorithm(n,c) algorithm are analyzed under the condition $\left\lceil \frac{n}{c} \right\rceil \leq \frac{n+1}{2}$:

Step 1. $A = n, i = 1$;

Step 2. Until $A \neq 0$ repeat: $B[i] = \left\lceil \frac{A}{c} \right\rceil, A = A - B[i], i = i + 1$;

Step 3. Return $B$.

In [7], it was proved, in particular, that with the $c = 2$ the CAlgorithm(n,c) algorithm guarantees the generation of a generating sequence $B$ with its minimum length $m = \left\lceil \log_2 n \right\rceil + 1$. The basic idea of cs-sort and cs-select is to use a generator sequence $B$ to separate the elements of the sorted sequence. This on the one hand makes similar cs-sort and quicksort, and on the other shows the differences: in the cs-sort algorithm the elements of the sorted sequence are not used as reference elements, third-party elements of the generating sequence are used as reference elements. At the same time, obtaining the result at the end of the elements of the generating sequence is guaranteed. Below are all the steps of the proposed algorithm cs-select, assuming that at the input of the cs-select algorithm are a set $X, |X| = n$, generating sequence $B = \text{CAlgorithm}(n,c)$, as well as the number of $k$-th ordinal statistics of the set $X$.

**Auxiliary function iter-find(X,B,A,k)**

Step 1 (recursion bases). If $|B| = \emptyset$, then return X[1]. If $k \leq 0$, then return X[1]. If $|X| = 1$, then return X[1].

Step 2 (separation). $Less = \emptyset, More = \emptyset, Len = 0, A = A + B[1], i = 1$.

1: Loop While $i < |X|$

2: If $X[i] \geq A$, then push(X[i],More), otherwise push(X[i],Less), Len = Len + 1;

3: $i = i + 1$;

4: End loop While.

Step 3 (recursion). If $Len \geq k$, to call iter_find(Less,B \ B[1],A - B[1],k), otherwise call iter_find(More,B \ B[1],A,k - Len).

**The main function cs-select(X,B,k)**

Step 1. Call iter_find(X, B, 0, k).

3. Analysis of the computational complexity of the cs-select algorithm

The speed of the cs-select algorithm is determined by two factors:

1. the length of the generating sequence $B$,
2. the cardinality of set $X$, as well as the cardinality of subsets $More \subset X, Less \subset X$.

As shown in the article [7], the length of the generating sequence $B$ obtained by the CAlgorithm(n,c) for given $c \geq 2, n = \max(X)$, is bounded from above: $|B| \leq \left\lceil \log_c n \right\rceil + 1$.

For the simplicity of estimating changes in the cardinality of sets $X, More, Less$ let's $c = 2$ and assume that as an input of the cs-select algorithm is some permutation of the elements of the set: $\{i | i \in N, i \leq n\}$ by way of $X$. 


Then, with respect to each element of the generated sequence $B$, sets $X, More, Less$ are divided approximately in half, and the number of comparisons performed in point 2, step 2, of the cs-select algorithm can be estimated as:

$$\sum_{i=0}^{\lfloor \log_2 \frac{n}{2^i} \rfloor} \frac{n}{2^i} \leq C(n,k) \leq \sum_{i=0}^{\lfloor \log_2 \frac{n}{2^i} \rfloor} \frac{n}{2^i}.$$  

(9)

If $n \to \infty$ then $\sum_{i=0}^{\lfloor \log_2 \frac{n}{2^i} \rfloor} \frac{n}{2^i} \to 2n$, $\sum_{i=0}^{\lfloor \log_2 \frac{n}{2^i} \rfloor} \frac{n}{2^i} \to 2n$.

Therefore if $n \to \infty$ then $\forall k : C(n,k) \to 2n$.

In practice, in the CAlgorithm($n,c$) we can use the value $c \neq 2$, obtaining generating sequences $B$, generally longer than $1 \log_2 + n$. However, for some values $k$, you can obtain a hidden constant in $O(n)$ less than 2. The results of such calculations are shown in table 1.

| Table 1. Assessment of the computational complexity of the cs-select algorithm for different $n$, $k$, $c$. |
|---|
| $n$ | $k$ | $c=2$ | $c=2.5$ | $c=2$ | $c=2.5$ | $c=2$ | $c=2.5$ |
| | | $S=(50,25,13,6,3,2,1)$ | $S=(40,24,15,9,3,2,1,1)$ | $S=(51,26,13,7,4,2,1)$ |
| $S(n,k)/n$ | $S(n,k)/n$ | $S(n,k)/n$ |
| 1 | 1.92 | 1.9 | 1.96 |
| 25 | 1.98 | 1.89 | 1.98 |
| 100 | 2.0 | 2.16 | 1.92 |
| 75 | 2.01 | 2.27 | 1.99 |
| 100 | 2.03 | 2.51 | 1.95 |

As can be seen from table 1, with $c=2$, the values of the hidden constant have the smallest spread with an average value of 1.988; for $c=2.5$, we obtain a much larger spread in values; in some cases, this gives a hidden constant less than for $c = 2$. However, on average, with $c=2.5$, the value of the hidden constant is 2.146. The last column of table 1 demonstrates the ability to modify the generating sequence in such a way that the best result with the average value of the hidden constant 1.95 is achieved. As can be seen from the columns for $c = 2$, $c = 2.5$ of table 1, the number of comparisons increases with increasing $k$. For the initial segment of $k$ values, the hidden constant is less than 2. In the generative sequence of the last column of table 1, this initial segment is stretched due to the fact that the sequence $S = (51,26,13,7,4,2,1)$ is generative for a slightly larger number $n = 104$ while maintaining the same length. It is interesting to note that the smallest value of the hidden constant is when finding the median value. All this demonstrates the potential for further lowering the theoretical limit for the hidden constant.

4. Experimental comparison

In addition to theoretical estimates, it is interesting to compare the cs-select algorithm with existing algorithms for finding the $k$-th statistics in practice. Since the main idea of the cs-select algorithm coincides with the quickselect algorithm, and the difference lies in the selection of elements relative to which the sequence is divided, quickselect was chosen as the comparison base. It should be noted that the author’s version of the quickselect algorithm aimed to finding the $k$-th statistics in sets, i.e. does not allow repetition of elements. The quickselect algorithm used on multisets should include verification that the multiset obtained at the next step of the recursion does not consist of the same elements, and therefore cannot be divided. Such verification leads to additional time costs. In the cs-select algorithm, on the contrary, separation occurs in any case at the end of the elements of the
generating sequence without the need for accounting for the repetition of elements. This fact is an
important advantage of the cs-select algorithm.

It is also important to note the limitations to the practical use of the cs-select algorithm. Since the
number of steps of the algorithm depends on the length of the generating sequence, which in turn is
determined by the maximum number in the set \( X \), or indirectly by its digit, the cs-select algorithm
turns out to be a priori ineffective in the case of sufficiently sparse sequences of high-digit numbers.

Implementation of cs-select, quickselect algorithms made by authors in the Common Lisp
programming language (source code and results are presented in the github repository [8]) became the
Platform for conducting experiments.

The experiment used random sequences \( X \) of different lengths from numbers ranging from 1 to
\( 10^{17} - 1 \) with possible repetitions of elements. For each such sequence, \( Q(X,1) \), \( Q(X,[0.5n]) \), \( Q(X,n) \), \( Q(X,[0.25n]) \), \( Q(X,[0.75n]) \) were searched, averaged values of the number of
comparisons on a sample of 100 sequences of a given length \( n \) were calculated for them. The cs-select
algorithm always used generating sequences with \( c = 2 \). The quickselect algorithm used a random
selection of a reference element.

The results of the experiment are shown in tables 2, 3.

**Table 2.** The average number of comparisons for sequences of different lengths, obtained
on the basis of the cs-select algorithm.

| \( n \) | \( Q(X,1) \) | \( Q(X,[0.25n]) \) | \( Q(X,[0.5n]) \) | \( Q(X,[0.75n]) \) | \( Q(X,n) \) |
|---|---|---|---|---|---|
| 10 | 18.65 | 22 | 24.15 | 23.84 | 23.61 |
| 100 | 197.52 | 206.02 | 210.05 | 206.79 | 207.06 |
| 1000 | 2000.58 | 2024.16 | 2031.53 | 2010.55 | 2005.54 |
| 10000 | 19994.11 | 20034.36 | 20092.17 | 20043.37 | 20014.85 |
| 100000 | 200004.34 | 200154.56 | 200243.28 | 200135.17 | 200062.48 |
| 1000000 | 2000050.3 | 2000328.90 | 2000714.60 | 2000658.80 | 2000136.60 |

**Table 3.** The average number of comparisons for sequences of different lengths, obtained
on the basis of the quickselect algorithm.

| \( n \) | \( Q(X,1) \) | \( Q(X,[0.25n]) \) | \( Q(X,[0.5n]) \) | \( Q(X,[0.75n]) \) | \( Q(X,n) \) |
|---|---|---|---|---|---|
| 10 | 20.58 | 31.93 | 31.33 | 29.5 | 21.19 |
| 100 | 206.69 | 309.91 | 323.55 | 325.08 | 206.70 |
| 1000 | 2075.94 | 2974.03 | 3369.36 | 3188.59 | 1940.85 |
| 10000 | 20404.60 | 31161.61 | 34515.53 | 31773.43 | 19332.77 |
| 100000 | 210805.88 | 308239.38 | 348473.56 | 311948.80 | 197159.25 |
| 1000000 | 2025686.80 | 3172760.50 | 3371014.50 | 3138005.80 | 1982103.80 |

The results obtained in table 2 demonstrate the stabilization of the number of comparisons at the \( 2n \)
level. Indeed, for \( n = 10 \) elements, when finding the median, the number of comparisons was 20% higher than the \( 2n \) level. For the last considered \( n = 100000 \) elements, when finding the median, the excess was only 0.03% of the \( 2n \) level. As can be seen from table 2, finding the median value is the most laborious.

Table 3 shows the average number of comparisons obtained by the quickselect algorithm on the
same sequences as in the cs-select algorithm. As can be seen from table 3, in general, the quickselect
algorithm in the form of the implementation proposed in [8] is inferior to cs-select. For this algorithm,
finding the median is also the most laborious.
However, here the excess of the number of comparisons over the 2n level was 68%. At the same time, it should be noted that when finding the maximum value of $Q(X,n)$, the quickselect algorithm is always more efficient than cs-select.

5. Conclusions
In general, we can say that the cs-select algorithm should be attributed to deterministic algorithms for selecting the $k$-th statistic in a sequence of integers, allowing their repetition, with a computational complexity $O(n)$ and a hidden constant in $O(n)$ close to the value of 2 for sufficiently large sequences. It should also be noted that there are potential opportunities to reduce the hidden constant by selecting a suitable generating sequence.

References
[1] Hoare C A R 1961 Algorithm 65: FIND Comm. ACM 4 321–2
[2] Knuth D E 1973 The Art of Computer Programming. Sorting and Searching (Addison-Wesley, Reading, Mass.)
[3] Krzysztof C Kiwiel 2005 On Floyd and Rivest's SELECT algorithm Theoretical Computer Science vol 347(1-2) 214-38
[4] Floyd R W and Rivest R L 1975 Expected time bounds for selection Comm. ACM 18 165–72
[5] Floyd R W and Rivest R L 1975 The algorithm SELECT—for finding the ith smallest of n elements (Algorithm 489) Comm. ACM 18 173
[6] Blum M R, Floyd R W, Pratt V R, Rivest R L and Tarjan R E 1972 Time bounds for selection J. Comput. System Sci. 7 448–61
[7] Artamonov Y N 2017 Using the Complete Sequences for Sorting Natural Numbers. The Bulletin of Irkutsk State University. Series Mathematics 22 3-17 doi.org/10.26516/1997-7670.2017.22.3
[8] Source code for comparative analysis of algorithms cs-select, quickselect https://github.com/junaart/science/tree/master/Voronezh