MATHEMATICAL, PHYSICAL, AND ENGINEERING SCIENCES

IMPROVEMENT OF BLUM'S ET. AL. SELECTION ALGORITHM

ELIEZER A. ALBACEA
Institute of Computer Science, University of the Philippines Los Baños
4031 College, Laguna, Philippines

ABSTRACT

Blum, et. al. [1] presented a selection algorithm that finds the kth smallest element of a set with n distinct elements using 5.4305n comparisons in the worst case. In this paper, we present an improvement of this algorithm that requires 5.3975n comparisons in the worst case. The contribution of this paper is not on the amount of improvement but rather on the result that the worst case of the best practical algorithm for selection can still be improved. Thus, opening the possibility of further closing the gap between the best practical worst case and the best theoretical worst case for selection.

Keywords: selection, order statistic, analysis of algorithms.

INTRODUCTION

Blum, et. al. [1973] presented an algorithm for selection that requires 5.4305n comparisons in the worst case. This was improved by Schonhage, et. al. [1976] by presenting an algorithm that requires asymptotic to 3n comparisons in the worst case. Unfortunately, the algorithm presented in [4] is much more difficult to implement than the one in [Blum et al. 1973]. From the practical point of view, therefore, the algorithms of Blum. et. al. [1973] remain the best.

One of the algorithms in [Blum et al. 1973] (called PICK1) is known to perform well as the number of elements discarded increases. In this paper, we present a new algorithm called LOCATE, that behaves in the opposite direction, i.e., the algorithm performs well as the number of elements that can be discarded decreases. By combining these two algorithms, an algorithm that requires 5.4137n comparisons is produced. Further refinement reduces this value to 5.3975n comparisons.

Notations

For convenience, the notations used in [Blum et al. 1973] will be adopted in this paper. Given a set S of n distinct elements, $k\theta S$ is the kth smallest element of S, $x\rho S$ is equal to the rank of x in S. But by arguments of symmetry, the algorithms in this paper assume that $1 \leq k \leq \lceil n/2 \rceil$. The small letter c will consistently be used to represent the size of the columns used in the algorithms presented in this paper and h (c) is the cost of sorting a c-column. The term 'c-column' refers to a column with c elements and the term 'c-sorted' refers to a c-column that is sorted in ascending order. All sorting steps involved in the algorithms arrange the elements of the set in ascending order and are carried out using the sorting algorithm in [Ford and Johnson 1959]. Further, it should be noted that in the succeeding sections, performance or running time or cost is measured in terms of number of comparisons.

Blum's et. al. Selection Algorithms

The most important contributions of their algorithms are the establishment that the selection can be done in linear time. These algorithms are similar with Hoare's selection algorithm (FIND (Hoare 1961)] except that the algorithms ensure that in every iteration about ¼ of the total number of elements being considered are discarded. This somehow explains the reason why the worst case running times of the algorithms are linear.

Their best algorithm, called PICK1, starts by dividing the input into c-columns which are then c-sorted. In the succeeding iterations, the retained elements are restored to c-sorted columns by merging. This means that the columns are c-sorted only once which explains the cost of 5.4305n comparisons. Since PICK1 will be used as a basis of some improvements to be given later, a slightly reformatted version of it will be given below.

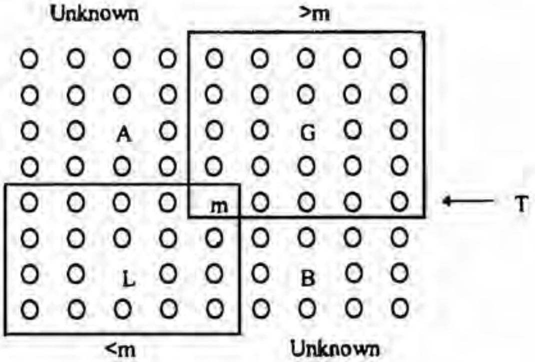

Figure 1. Division of Elements in PICK1

Algorithm: PICK1

```
Comment: Selects $k\theta S$, where $|S| = n$ and $1 \leq k \leq \lceil n/2 \rceil$.

1.  If $n \leq 45$, sort S, print $k\theta S$, and halt.
2.  Arrange S into $\lceil n/15 \rceil$ columns of length 15, and sort each column.
3.  Use PICK Ia to select $k\theta S$.
```

Algorithm: PICK1a

```
Comment: Selects $k\theta S$, where $|S| = n$ and $1 \leq k \leq \lceil n/2 \rceil$ and the set S and T are
already 15-sorted.

1.  If $n \leq 45$, sort S, print $k\theta S$, and halt.
2.  Arrange T, the set of column medians (Figure 1), into $\lceil n/225 \rceil$ columns of
    length 15, and sort each column.
3.  Use PICK1b to select $k\theta S$.
```

Algorithm: PICK1b

```
Comment: Selects $k\theta S$, where $|S| = n$ and $1\lceil \leq k \rceil\ n/2$ , and the sets S and T are
already 15-sorted.

1.  Select $m = \lceil |T|/2 \rceil\ \theta T$ using PICK1a, where T is the set of all c-column medians.
2.  Compute $m\rho S$, by partitioning $(A \cup B)$ of Figure 1 about m as follows, stopping
    as soon as it becomes apparent that $m\rho S < k$ or $m\rho S > k$.
    (a) Insert m into each column of B using binary insertion.
    (b) Insert m into each column of A using linear search technique beginning
        near the median of the original column.
3.  Apply the discard and restore procedure.
4.  Decrease n by the number of elements discarded. If $n \leq 45$, sort S, print $k\theta S$
    and halt, otherwise return to Step 1.
```

Algorithm: Discard and Restore – PICK1

```
1.  If $m\rho S = k$, halt. Otherwise, if $m\rho S > k$, then discard $G \cup \{x \mid x \in B \wedge x > m\}$, else
    discard $L \cup \{x \mid x \in A \wedge x > m\}$ and decrease k by the number of elements
    discarded.
2.  Restore S to a set of 15-columns.
3.  Restore T to a set of 15-columns.
```

Let Pb(n), Pa(n) and P(n) be the costs of algorithms PICK1b, PICK1a, and PICK1, respectively. These costs were shown in [1] to assume the values

Proof: For every pair of c-columns affected by the discard operation, exactly c elements are discarded. Since there are n/4c pairs of affected c-columns, therefore a total of c(n/4c)=n/4 elements are discarded. □

New Selection Algorithm

Using the simple discard and restore procedure outlined in the previous section, an algorithm that performs well as d decreases can be constructed. The outline of the algorithm is exactly the same as that of PICK1, except for the discard and restore procedure. An outline is given below.

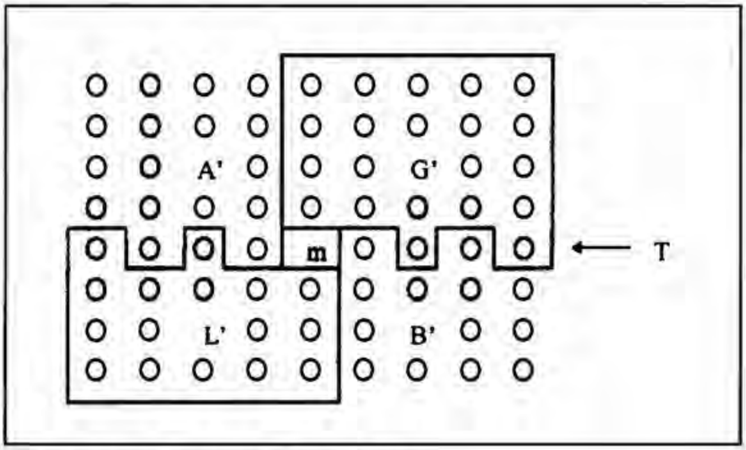

Figure 2. Divisions of Elements in LOCATE.

Algorithm: LOCATE

Comment: Exactly the same as PICK1, except that Step 3 uses LOCATEa.

Algorithm: LOCATEa

Comment: Exactly the same as PICK1a, except that Step 3 uses LOCATEb.

Algorithm: LOCATEb

Comment: Exactly the same as PICK1b, except that Step 3 uses LOCATEa and discard and restore procedure given below.

Algorithm: Discard and Restore – LOCATE

1. If $m\rho S = k$, halt. Otherwise if $m\rho S > k$, then discard D_g, else discard D_l and update k.
2. Restore S to a set of c-columns as follows:

2. Restore S to a set of c-columns as follows:

   (a) The c-column medians form $\lfloor c/2 \rfloor$-sorted and ($\lfloor c/2 \rfloor$+1)-sorted columns since they were discarded in that form in Step 1 of LOCATEb. This means that $\lfloor c/2 \rfloor$ pairs, i.e., $\lfloor (\lfloor c/2 \rfloor +1)/2 \rfloor$ and $\lfloor \lfloor c/2 \rfloor /2 \rfloor$ pairs from those whose medians are ($\lfloor c/2 \rfloor$+1)-sorted and $\lfloor c/2 \rfloor$-sorted, respectively, from out of c affected columns can be merged immediately using the method outlined in the previous section.

   (b) Combine the extra columns and merge every $\lfloor c/2 \rfloor$-column with a ($\lfloor c/2 \rfloor$+1)-column. It should be made sure, however, that ½ of the extra columns are $\lfloor c/2 \rfloor$-columns and the other half are ($\lfloor c/2 \rfloor$+1)-columns. This arrangement can be done easily.

3. Restore T to a set of c-columns similarly. Let $Z \subset T$ be those column medians which were column medians in Step 1 of LOCATEb.

   (a) Since the elements of Z are arranged in $\lfloor c/2 \rfloor$-sorted and ($\lfloor c/2 \rfloor$+1)-sorted columns (elements of Z were discarded in that form when m was computed in Step 1 of LOCATEb), restoring Z can therefore be done by simply merging every $\lfloor c/2 \rfloor$-sorted column with a ($\lfloor c/2 \rfloor$+1)-sorted column.

   (b) Restore the new medians, which were computed after the restoration of S, by sorting them into c-columns.

Lemma 2. The cost of restoring S is (c-2)((c-1)/c)(n/4c) + (c-1)(1/c)(n/4c) comparisons.

Proof: The discard operation leaves ((c-1)/c)(n/4c) pairs of adjacent $\lfloor c/2 \rfloor$- and ($\lfloor c/2 \rfloor$+1)-columns. Hence, they can be merged using (c-2)((c-1)/c(n/4c) comparisons. The unknown columns, whose number is (1/c)(n/2c) and which are composed also of $\lfloor c/2 \rfloor$- and ($\lfloor c/2 \rfloor$+1)-columns can be restored using (c-1)(1/c)(n/4c) comparisons. □

Lemma 3. The cost of restoring T is (c-1)(n/4c)(1/c)+h(c)(n/4c)(1/c) comparisons, where h(c) is the cost of c-sorting.

Proof: The restoration of $Z \subset T$ can be done using the method used to restore S. Since |Z|= n/2c, therefore there are (n/4c) (1/c) $\lfloor c/2 \rfloor$-sorted columns in Z. The same number of ($\lfloor c/2 \rfloor$+1)-sorted columns is present. Hence, Z can be restored to c-columns in (c-1)(n/4c)(1/c) comparisons. The set of new medians, whose number is n/4c, can be c-sorted using h(c)(n/4c)(1/c) comparisons. ›

The costs of LOCATE, LOCATEa, and LOCATEb, which are denoted by L(n), La(n), and Lb(n), respectively, can be described by the following recurrence relations:

$L(n) \leq nh(c)/c + La(n)$	Cost of LOCATE.
$La(n) \leq nh(c)/c2 + Lb(n)$	Cost of LOCATEa.
	Cost of LOCATEb.
$Lb(n) \leq La(\lceil n/c \rceil)$	Compute m.
$+ (\lceil \log(\lfloor c/2 \rfloor + 1) \rceil)(n/2c)$	Insert m into B.
$+ (n/2x + d)$	Insert m into A.
$+ (c-2)((c-1)/c)(n/4c)$	Restore S. Step 2a.
$+ (c-1)(1/c)(n/4c)$	Restore S. Step 2b.
$+ (c-1)(n/4c)(1/c)$	Restore S. Step 3a.
$+ h(c)(n/4c)(1/c)$	Restore T. Step 3b.
$+ Lb(3n/4)$	Cost of succeeding iterations.

Given c=15, the recurrence relations simplify to the following:

$$\begin{aligned} L(n) &\leq 42n/15 + La(n) \\ La(n) &\leq 42n/225 + Lb(n) \\ Lb(n) &\leq (60/11)(479n/1125 + d). \end{aligned}$$

The algorithm assumes its best performance when d=0, where

$$\begin{aligned} Lb(n) &\leq 2.3224n \\ La(n) &\leq 2.5091n \\ L(n) &\leq 5.3091n \end{aligned}$$

The worst performance, on the other hand, occurs when 6n/30 £ 7n/30, where

$$\begin{aligned} Lb(n) &\leq 3.4133n \\ La(n) &\leq 3.5000n \\ L(n) &\leq 6.4000n. \end{aligned}$$

However, the optimum value of c is c=21, where L(n) £ 5.2773n when d = 0 and $L(n) \leq 6.3361n$ when $6n/30 \leq d \leq 7n/30$. Hence, the following theorem is proven:

Theorem 1. There is a selection algorithm that requires 5.2773n comparisons when d = 0 and requires 6.3361n when d = 7n/30.

We have, therefore, shown the existence of an algorithm whose behaviour is opposite to that of PICK1 as d increases.

An Improvement

Let d be as described before. As noted earlier, PICK1 performs well as d increases while LOCATE performs well as d increases. By combining these two algorithms, the worst case of 5.4305 (due to PICK1) can be reduced to a new value of 5.4137n.

Let v be the value of d where the performances of PICK1 and LOCATE are equal. A new and better worst case is set by computing for d, if $d \leq n$ then apply the discard and restore procedure of LOCATE, otherwise use the discard and restore procedure of PICK1. An algorithm, called LOCATE1, employing this observation is given below. The constant d and v are as defined above.

Algorithm: LOCATE1

Comment: Exactly the same as PICK1, except that Step 3 uses LOCATE1a.

Algorithm: LOCATE1a

Comment: Exactly the same as PICK1a, except that Step 3 uses LOCATE1b.

Algorithm: LOCATE1b

Comment: Selects $k\theta S$, where $|S| = n$ and $1 \leq k \leq n$, and the sets S and T are already c-sorted.

1. Select $m = \lceil |T|/2 \rceil \theta T$ using LOCATE1a, where T is the set of all c-column medians.
2. Compute $m\rho S$, by partitioning $(A \cup B)$ of Figure 1 about m as follows, stopping as soon as it becomes apparent that $m\rho S < k$ or $m\rho S > k$.
   (a) Insert m into each column of B using binary insertion.
   (b) Insert m into each column of A using linear search technique, the search starts at elements near the medians of the c-columns.
3. If $d > v$, then use the discard and restore procedure of PICK1. Otherwise, use the discard and restore procedure of LOCATE.
4. Decrease n by the number of elements discarded. If $n \leq 45$, sort S, print k0S and halt, otherwise return to Step 1.

Let L1(n), L1a(n), and L1b(n) be the costs of LOCATE1, LOCATE1a, and I OCATE1b, respectively. Since the performance of the algorithm is optimum when $c = 15$, in order to simplify the succeeding discussions, c will be set to this value.

When $d > v$, obviously LOCATE1 reduces to PICK1. Hence, the cost of LOCATE1 is given by the following:

$$\begin{aligned} L1b(n) &\leq (5n/(n + 5d)(13197n/27000 + 3546d/1575) \\ L1a(n) &\leq 42n/225 + L1b(n) \\ L1(n) &\leq 42n/15 + L1a(n). \end{aligned}$$

On the other hand, when $d \leq v$, LOCATE1 reduces to LOCATE. Hence, the cost of LOCATE1 is given by the following:

$$\begin{aligned} L1b(n) &\le (60/11)(479n/1125 + d) \\ L1a(n) &\le 42n/225 + L1b(n) \\ L1(n) &\le 42n/15 + L1a(n). \end{aligned}$$

The critical value is equal to the value of d in the equation

$$(60/11)(479n/1125 + d) = (5n/(n+5d))(13197n/27000 + 3546d/1575).$$

Simplifying the equation will result to $d = v = 0.01918n$ where with this value of d $L1(n)$ simplifies to $L3(n) \le 5.4137n$. Since $\max(h(s)/s) < 5.4137n$ when $s \le 45$, the induction is justified Clearly, this value is an improvement to the algorithm in [1].

Hence, the following theorem can be stated:

Theorem 2. There is a selection algorithm that requires 5.4137n comparisons in the worst case.

Further Improvement

Let 1_A be the number of elements in A which are less than m. The relationship between 1_A and d is given by the following lemma:

Lemma 4. $1_A \le d$.

Proof: Two cases must be considered, either L or G is discarded.

Case 1. If L is discarded, clearly d = 1A. Hence, the lemma holds.
Case 2. If G is discarded, then

$$|L| + 1_A + 1_B = k + 1 < \lceil n/2 \rceil + 1 \le |L| + |B| \le |L| + g_B + 1_B..$$

From the leftmost and rightmost terms, $g_B \le 1_A$. Since $d = g_B$, $1_A \le d$ follows. □

Since $1A \le d$, Steps 2a and 2b of PICK1b, i.e., the insertion of m into B using binary insertion and the insertion of m into A using sequential search technique, can be performed in any order without changing the cost of PICK1b as a whole. The lemma also implies that the performance of PICK1 improves as the size of 1_A increases.

Using the approach used in LOCATE1, we can improve the worst case by finding an algorithm that performs well as the value of 1A decreases. One such algorithm is obtained by making a slight modification an Step 2 of LOCATE1b. The said modification is incorporated into the algorithm below. This time v stands for the value of 1_A where the performances of PICK1 and LOCATE match.

Algorithm: LOCATE2

Comment: Exactly the same as PICK1, except that Step 3 uses LOCATE2a.

Algorithm: LOCATE2a

Comment: Exactly the same as PICK1a, except that Step 3 uses LOCATE2b.

Algorithm: LOCATE2b

Comment: Selects $k\theta S$, where $|S| = n$ and $1 \leq k \leq n$, and the sets S and T are already c-sorted.

1. Select $m = \lceil |T|/2 \rceil \theta T$ using LOCATE2a, where T is the set of all c-column medians.
2. Compute $m\rho S$, by partitioning $(A_1 \cup B)$ of Figure1 about m as follows, stopping as soon as it becomes apparent that $m\rho S < k$ or $m\rho S > k$
   - (a) Insert m into each column of A using linear search technique, the search starts at elements near the medians of the c-columns.
   - (b) If $l_A \geq v$ then insert m into each column of B, using binary search technique. Otherwise, insert m into each column of B using linear search technique, starting form the elements next to the medians of the c-columns, stopping when the number of elements greater than m counted so far is equal to $(l_A + 1)$ or when B runs out of elements greater than m, whichever comes first.
3. If $l_A \geq v$ then use the discard and restore procedure of PICK1. Otherwise, use the discard and restore procedure of LOCATE.
4. Decrease n by the number of elements discarded. If $n \leq 45$, sort S, print kqS and halt, otherwise return to Step 1.

To simplify the discussion, let c assume its optimum value which is $c = 15$. Let L2(n), L2a(n), and L2b(n) be the costs of LOCATE2, LOCATE2a, and LOCATE2b, respectively.

When $l_A \geq v$, obviously LOCATE2 reduces to PICK1 (Lemma 4). Hence, the cost of LOCATE2 is given by the following recurrence relations:

$$\begin{aligned} L2b(n) &\leq (5n/(n+5l_A))(13197n/27000 + 3546l_A/1575) \\ L2a(n) &\leq 42n/225 + L2b(n) \\ L2(n) &\leq 42n/15 + L2a(n) \end{aligned}$$

However, when $l_A < v$ the cost is the same as that of LOCATE1 except for Step 2 of LOCATE2b.

Lemma 5. The cost of Step 2 of LOCATE2b when $l_A < v$ is $2(n/30 + l_A)$.

Proof: Step 2a of LOCATE2b obviously needs one comparison per column plus the number of elements in A which are less than m. Hence, a cost of $(n/30 + l_A)$. With regards to Step 2b of LOCATE2b, two cases must be considered. The first is when $g_B > l_A$ where g_B is the number of elements in B which are greater than m. Clearly, this costs $(n/30 + l_A)$. The second case is when $g_B \geq l_A$, where the cost is less than $(n/30 + l_A)$. Therefore, Step 2 of LOCATE2b costs $2(n/30 + 1)$. □

With this step settled, the cost of LOCATE2b when $lA < v$ can be written as follows:

$$\begin{array}{rll}
L2b(n) \leq & L2a(\lceil n/15 \rceil) & \text{Select m.} \\
+ & 2(n/30 + l_A) & \text{Compute mrS.} \\
+ & 13(14/15)(n/60) & \text{Restore S. Step 2a.} \\
+ & 14(1/15)(n/60) & \text{Restore S. Step 2b.} \\
+ & 14(n/60)(1/15) & \text{Restore T. Step 3a.} \\
+ & 42(n/60)(1/15) & \text{Restore T. Step 3b.} \\
+ & L2b(3n/4) & \text{Cost of succeeding iterations.}
\end{array}$$

This recurrence relation simplifies to $L2b(n) \leq (60/11)(404n/1125 + 2l_A)$. The critical value v is equal to the value of $l_A A$ in the equation

$$(60/11)(404n/1125 + 2l_A) = (5n/(n+5l_A))(13197n/27000 + 3546l_A/1575).$$

Simplifying the equation will result to $l_A = v = 0.04144n$ where with this value of l_A $L2(n)$ simplifies to $L2(n) \leq 5.3975n$. Clearly, this value is an improvement to the previous algorithm. Figure 3 shows the performance of LOCATE2 (line below the intersection) on different values of l_A, where $0 \leq l_A \leq 7n/30$. The theorem below follows:

Theorem 3. The selection problem can be solved using 5.3975n comparisons in the worst case.

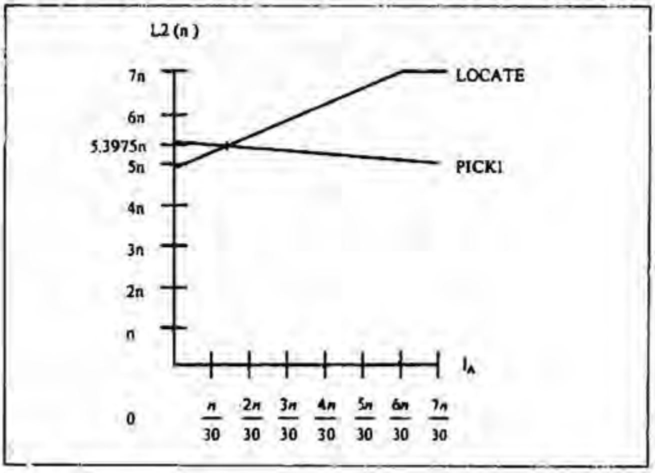

Figure 3. Performance of PICK1 and LOCATE.

REFERENCES

Blum, M., Floyd, R. W., Pratt V., Rivest, R.L., and Tarjan, R.E. Time bounds for selection, J. Computing and Systems Science 7 (1973), 448-461.

Ford, L.R., and Johnson, S.M. A tournament problem, The American Mathematical Monthly 66 (1959), 387-389,

Hoare, C.A.R. Partition: Algorithm 63; and Find: Algorithm 65, Comm. ACM 4 (1961), 321-322.

Schonhage, A., Paterson, M., and Pippenger, N. Finding the median, J. Computing Systems Science 13 (1976), 184-199.