Find Refined

Roland Backhouse
University of Nottingham
139 PUBLICATIONS **2,999** CITATIONS

Find Refined

Roland Backhouse[*]

May 8, 2001

Abstract

A new implementation of Hoare's *Find* program is developed. The program determines the Kth smallest element in an (unordered) array of values and how often it occurs.

We argue that *Find* provides an excellent example of the design of an algorithm that is non-trivial, correct by construction, and yet within the scope of an introductory module on (formal) program design.

Keywords: algorithm design, program correctness.

1 Introduction

The program *Find* developed by C.A.R. Hoare [Hoa61] was one of the first examples of the development of an algorithm hand-in-hand with a proof of its correctness [Hoa71, He89]. The program is rarely used, however, to illustrate the principles of program construction in programming texts, not even in any of the leading texts on formal program design.

In this paper we present an improved derivation and implementation of the program in the hope that the problem may be adopted in the future as an excellent example of the design of algorithms that are non-trivial, correct by construction, and yet within the scope of an introductory module on (formal) program design.

The main simplification in our design is the use of Dijkstra's solution to the so-called "Dutch National Flag Problem" [Dij76] rather than Hoare's partitioning program [Hoa61]. Additionally, we refine the invariant property used in the algorithm. This enhances the functionality of the algorithm in the sense that not only does it determine the Kth smallest value in an array of values (as does Hoare's algorithm) but it also determines how often that value occurs in the array.

[*]School of Computer Science and IT, University of Nottingham, Jubilee Campus, Wollaton Road, Nottingham NG8 1BB, UK. `rcb@cs.nott.ac.uk`

2 Problem Statement and Invariant

The function of *Find*, as stated by Hoare, is to find that element of an array of size N whose value is Kth in order of magnitude.

As so often happens, this natural language statement of the problem is unclear. If values can be duplicated, there may be several elements in the array that all have the Kth smallest value. (In a sports tournament, for example, there may be a tie for a particular position.) Taking this into account, the problem we want to consider is that of rearranging the elements in an array of size N into three segments. The left segment has length less than K and contains the "smallest" elements in the array; the right segment has length at most $N-K$ and contains the "largest" elements in the array; finally, the middle segment has length at least one and all values stored in the segment are equal. The Kth smallest value in the array is thus the value common to all the elements in the middle segment and the number of its occurrences is equal to the length of the middle segment.

Formally, the specification is that we are given an array a of size N and a number K such that $1 \leq K \leq N$. Assuming that the array elements are indexed from 0 onwards, it is required to rearrange them so that on termination there are indices s and l satisfying

$$\begin{aligned}
& 0 \leq s < K \leq l \leq N \\
\wedge\ & \langle \forall i,j \mid 0 \leq i < s \wedge s \leq j < N : a[i] < a[j] \rangle \\
\wedge\ & \langle \forall i,j \mid 0 \leq i < l \wedge l \leq j < N : a[i] < a[j] \rangle \\
\wedge\ & \langle \forall i,j \mid s \leq i < l \wedge s \leq j < l : a[i] = a[j] \rangle \quad .
\end{aligned}$$

In words, the first s values in the array should be strictly smaller than any other array elements and their number should be less than K; the last $N-l$ values in the array should be strictly larger than any other elements and their number should be at most $N-K$; and the remaining array elements should all be equal. (There are thus $l-s$ values in the array equal to the Kth smallest value, which is given by $a[s]$.)

This specification is a refinement of Hoare's specification; his requirement was that on termination

$$\langle \forall i,j \mid 0 \leq i < K \leq j < N : a[i] \leq a[K-1] \leq a[j] \rangle \quad .$$

This result is clearly implied by our own specification.

3 Solution

The basic idea of Hoare's algorithm is to maintain invariant the property that, for some indices s and l, the first s elements of the array are known to be "small", the next $l-s$ elements are "medium", and the remaining $N-l$ elements are "large". We use the same idea but with slightly different formulations of what it is to be "small" and "large". Specifically, we consider an algorithm that maintains two variables s and l satisfying

$$\begin{array}{ll} & 0 \le s < K \le l \le N \\ \wedge & \langle \forall i,j \mid 0 \le i < s \wedge s \le j < N : a[i] < a[j] \rangle \\ \wedge & \langle \forall i,j \mid 0 \le i < l \wedge l \le j < N : a[i] < a[j] \rangle \quad . \end{array}$$

This property is depicted in fig. 1.

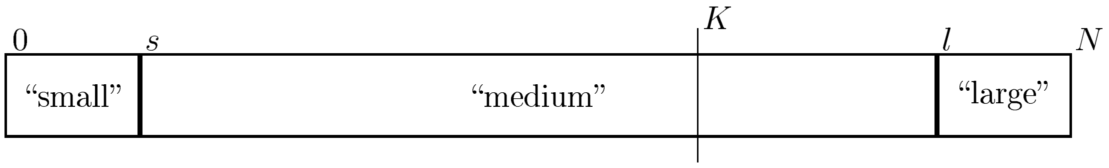

Figure 1: Find Invariant.

The computation is complete when it is the case that all values in the "medium" segment are known to be equal. That is when, in addition to the invariant property, we have:

$$\langle \forall i,j \mid s \le i < l \wedge s \le j < l : a[i] = a[j] \rangle \quad .$$

We introduce a boolean value *done* whose function is to indicate when this property has been established. Formally the loop invariant is

$$\begin{array}{ll} & 0 \le s < K \le l \le N \\ \wedge & \langle \forall i,j \mid 0 \le i < s \wedge s \le j < N : a[i] < a[j] \rangle \\ \wedge & \langle \forall i,j \mid 0 \le i < l \wedge l \le j < N : a[i] < a[j] \rangle \\ \wedge & (done \Rightarrow \langle \forall i,j \mid s \le i < l \wedge s \le j < l : a[i] = a[j] \rangle) \quad . \end{array}$$

It is straightforward to establish the invariant. The assignment

$$s, l, done := 0, N, (N \le 1)$$

initialises the "small" and "large" segments to the empty set. (The initialisation of *done* could of course be to false. Assigning it the value $N \le 1$ enables us to play safe —the specification cannot be satisfied if $N = 0$— and simplifies the average-case running time analysis below.)

The bound function we use is the pair (*done* , $l-s$), ordered lexicographically. That is, the loop body should either set *done* to true or decrease $l-s$.

In order to make progress to the termination condition whilst maintaining the invariant, Hoare made the observation that it is crucial to choose a borderline value that is known to be in the "medium" segment. Values in the "medium" segment are indexed by numbers j such that $s \le j < l$. As $s < K \le l$ when the loop body is executed, the choice of $a[K-1]$ as borderline value is always appropriate, so this is the one we will choose. (Any value in the "medium" segment will do, but choosing the ($K-1$)th guarantees termination after exactly one iteration in the fortunate circumstance that the array is already sorted.) As the loop body may swap the chosen array element with another, it is wise to record this value in some local variable, X. A crucial property of X, which is immediate from our invariant property is that all values in the small segment are (strictly) less than X and all values in the large segment are (strictly) greater than X.

In summary, the algorithm we are aiming to develop thus has the basic structure shown below.

```
{ $1 \le K \le N$ }
$s, l, done := 0, N, (N \le 1)$ ;
{ Invariant:    $0 \le s < K \le l \le N$
             $\wedge\ \langle \forall i,j \mid 0 \le i < s \wedge s \le j < N : a[i] < a[j] \rangle$
             $\wedge\ \langle \forall i,j \mid 0 \le i < l \wedge l \le j < N : a[i] < a[j] \rangle$
             $\wedge\ (done \Rightarrow \langle \forall i,j \mid s \le i < l \wedge s \le j < l : a[i] = a[j] \rangle)$
  Bound function:   $(done\ ,\ l-s)$ ordered lexicographically. }
do $\neg done$ $\rightarrow$   { choose borderline value in medium segment }
             $X := a[K-1]$
             { $\langle \forall i \mid 0 \le i < s : a[i] < X \rangle \wedge \langle \forall j \mid l \le j < N : X < a[j] \rangle$ } ;
             reduce ( $done$ , $l-s$ ) whilst maintaining invariant
od
{    $0 \le s < K \le l \le N$
  $\wedge\ \langle \forall i,j \mid 0 \le i < s \wedge s \le j < N : a[i] < a[j] \rangle$
  $\wedge\ \langle \forall i,j \mid 0 \le i < l \wedge l \le j < N : a[i] < a[j] \rangle$
  $\wedge\ \langle \forall i,j \mid s \le i < l \wedge s \le j < l : a[i] = a[j] \rangle$ }
```

The key insight at this point is that it is possible to use the well-known Dutch National Flag program to sort the elements in the "medium" segment into values less than X (the "red" values), values equal to X (the "white" values) and values greater than X (the "blue" values).

We do not give the details of the implementation of the Dutch National Flag program here as the program is so well known. In our algorithm the line

$$DNF(s, l, (<X), (=X), (>X), m, n)$$

indicates a call of the Dutch National Flag program applied to the segment of the array a delimited by s and l with predicates *red*, *white* and *blue* set to ($<X$), ($=X$) and ($>X$), respectively. The values of m and n returned by the call delimit the segment of the (partially sorted) array a all of whose values equal X. This segment is known to

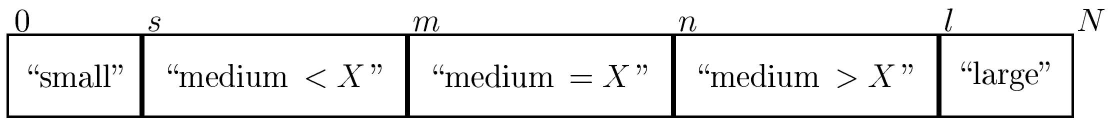

Figure 2: Outcome after applying the Dutch National Flag program

be non-empty since it includes the array element previously stored at index $K{-}1$. Fig. 2 depicts the relationships between the variables.

The development of the algorithm is completed by noting that there are three possible outcomes of the computation of m and n: either $K \leq m$ or $m < K \leq n$ or $n < K$.

In the case that $K \leq m$, the invariant is maintained and $l{-}s$ decreased by the assignment $l := m$. In the case that $n < K$, the invariant is maintained and $l{-}s$ decreased by the assignment $s := n$. Finally, in the case that $m < K \leq n$, the invariant is maintained and the computation is completed by the assignment $s, l, done := m, n, \mathsf{true}$.

This completes the development of the algorithm which is shown in fig. 3.

4 Time Complexity

As the preceding sections seek to demonstrate, Hoare's *Find* program is a good example of the techniques of developing programs that are "correct by construction". But it also provides a good example of complexity analysis at a more elementary level than its parent *Quicksort*.

Find is a variation on *Quicksort* in that, whereas *Quicksort* partitions an unordered array into two subarrays that are then ordered, *Find* executes the same partitioning process but then proceeds by (partially) ordering just one of the two subarrays.

Like *Quicksort* the worst-case time complexity of *Find* is discouraging. It is not difficult to construct an example array for which the *Find* program requires

$$N + (N{-}1) + \ldots + (N{-}K{+}1)$$

swap operations to achieve its task. So, if K is small compared to N, the worst-case complexity is $O(N^2)$. Hoare [He89] discusses alternative ways of choosing the borderline element in order to reduce the risk of such pathological behaviour. (Hoare's analysis is for *Quicksort* but applies equally to *Find*.)

The average-case running time of the algorithm is substantially better. Let C_n denote the average number of comparisons (of array elements with the borderline value X) executed by the program. (The number of swap operations is always smaller than this number.) Then, we have

$$C_1 = 0$$

and

```
$s, l, done$ := $0, N, (N \le 1)$ ;
{ **Invariant:**      $0 \le s < K \le l \le N$
                 $\wedge$ $\langle \forall i,j \mid 0 \le i < s \wedge s \le j < N : a[i] < a[j] \rangle$
                 $\wedge$ $\langle \forall i,j \mid 0 \le i < l \wedge l \le j < N : a[i] < a[j] \rangle$
                 $\wedge$ $(done \Rightarrow \langle \forall i,j \mid s \le i < l \wedge s \le j < l : a[i] = a[j] \rangle)$
  **Bound function:**   $(done, l - s)$ ordered lexicographically. }
do $\neg done$ $\rightarrow$ $X := a[K-1]$ { borderline value in “medium” region };
               { apply Dutch National Flag program to the segment
                 delimited by $s$ and $l$ with predicates $red$, $white$ and
                 $blue$ set to $(<X)$, $(=X)$ and $(>X)$, respectively.
                 Return the boundary values in $m$ and $n$. }
               $DNF(s, l, (<X), (=X), (>X), m, n)$;
               { Extend either the “small” or “large” segment, or terminate. }
               if $n < K$ $\rightarrow$ $s := n$
               □ $m < K \le n$ $\rightarrow$ $s, l, done := m, n,$ true
               □ $K \le m$ $\rightarrow$ $l := m$
               fi
od
```

Figure 3: Finding the Kth smallest value.

$$C_{n+1} \;=\; n+1+\frac{\langle \Sigma\, k \mid 1 \leq k \leq n : C_k \rangle}{n} \quad .$$

(For *Quicksort*,

$$C_{n+1} \;=\; n+1+2\times\frac{\langle \Sigma\, k \mid 1 \leq k \leq n : C_k \rangle}{n} \quad ,$$

the factor “2” reflecting the fact that two subarrays have to be sorted [GKP89].) It is easy to verify that $C_{n+1} = 2 \times n$. So the average case running time is linear in the number of array elements, independently of K.

5 Conclusion

Find is a very practical but non-trivial algorithm for determining the Kth smallest value in an unorderd array. Its development is a good illustration of the formal derivation of an algorithm that is correct by construction and can, in our view, be convincingly presented at an elementary level.

Quicksort is favoured by many authors as an illustration of good program design. There is therefore no reason why *Find* should not receive the same amount of favour.

References

[Dij76] Edsger W. Dijkstra. *A Discipline of Programming.* Prentice-Hall, 1976.

[GKP89] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics.* Addison-Wesley, 1989.

[He89] C.A.R. Hoare and C.B. Jones (editor). *Essays in Computing Science.* Prentice Hall, 1989.

[Hoa61] C.A.R. Hoare. Algorithm 63, partition; algorithm 64, quicksort; algorithm 65, find. *Communications of the ACM*, 4(7):321–322, 1961.

[Hoa71] C.A.R. Hoare. Proof of a program: Find. *Communications of the ACM*, 14(1):39–45, January 1971.