Algorithm 489

The Algorithm SELECT—for Finding the *i*th Smallest of *n* Elements [M1]

Robert W. Floyd [Recd 26 Sept. 1974]
Computer Science Department, Stanford University, Stanford, CA 94305
and
Ronald L. Rivest, M.I.T. Project MAC,
545 Technology Square, Cambridge, MA 02139

Key Words and Phrases: selection, medians, quantiles
CR Categories: 5.30, 5.39

Language: Algol (not strictly Algol 60)

Description

SELECT will rearrange the values of array segment $X[L : R]$ so that $X[K]$ (for some given K; $L \leq K \leq R$) will contain the $(K-L+1)$-th smallest value, $L \leq I \leq K$ will imply $X[I] \leq X[K]$, and $K \leq I \leq R$ will imply $X[I] \geq X[K]$. While *SELECT* is thus functionally equivalent to Hoare's algorithm *FIND* [1], it is significantly faster on the average due to the effective use of sampling to determine the element T about which to partition X. The average time over 25 trials required by *SELECT* and *FIND* to determine the median of n elements was found experimentally to be:

n	500	1000	5000	10000
SELECT	89 ms.	141 ms.	493 ms.	877 ms.
FIND	104 ms.	197 ms.	1029 ms.	1964 ms.

The arbitrary constants 600, .5, and .5 appearing in the algorithm minimize execution time on the particular machine used. *SELECT* has been shown to run in time asymptotically proportional to $N + \min(I, N-I)$, where $N = L - R + 1$ and $I = K - L + 1$. A lower bound on the running time within 9 percent of this value has also been proved [2]. Sites [3] has proved *SELECT* terminates.

The neater Algol 68 construct:
while ⟨boolean expression⟩ **do** ⟨statement⟩
is used here instead of the Algol 60 equivalent:
for *dummy* := 1 **while** ⟨boolean expression⟩ **do** ⟨statement⟩

References

1. Hoare, C.A.R. Algorithm 63 (*PARTITION*) and Algorithm 65 (*FIND*), *Comm. ACM 4* (July 1961), 321.
2. Floyd, Robert W., and Ronald L. Rivest. Expected time bounds for selection. Stanford CSD Rep. No. 349, Apr., 1973).
3. Sites, Richard. Some thoughts on proving clean termination of programs. Stanford CSD Rep. 417, May 1974.

Algorithm

```
procedure SELECT (X,L,R,K);
  value L,R,K;  integer L,R,K;  array X;
begin
  integer N,I,J,S,SD,LL,RR;  real Z, T;
  while R > L do
  begin
    if R - L > 600 then
    begin
      comment Use SELECT recursively on a sample of size S
        to get an estimate for the (K-L+1)-th smallest element
        into X[K], biased slightly so that the (K-L+1)-th
        element is expected to lie in the smaller set after partition-
        ing;
      N: = R - L + 1;
      I := K - L + 1;
      Z := ln(N);
      S := .5 × exp(2×Z/3);
      SD := .5 × sqrt(Z×S×(N-S)/N) × sign(I-N/2);
      LL := max(L,K-I×S/N+SD);
      RR := min(R,K+(N-I) × S/N+SD);
      SELECT(X,LL,RR,K)
    end;
    T := X[K];
    comment The following code partitions X[L : R] about T. It
      is similar to PARTITION but will run faster on most ma-
      chines since subscript range checking on I and J has been
      eliminated.;
    I := L;
    J := R;
    exchange(X[L],X[K]);
    if X[R] > T then exchange(X[R],X[L]);
    while I < J do
    begin
      exchange(X[I],X[J]);
      I := I + 1; J := J - 1;
      while X[I] < T do I := I + 1;
      while X[J] > T do J := J - 1;
    end;
    if X[L] = T then exchange(X[L],X[J])
      else begin J := J + 1;  exchange(X[J],X[R]) end;
    comment Now adjust L, R so they surround the subset con-
      taining the (K-L+1)-th smallest element;
    if J ≤ K then L := J + 1;
    if K ≤ J then R := J - 1;
  end
end SELECT
```