Nikhil Shaw is currently pursuing undergraduate degree in Information Technology at SRM University, Kattankulathur campus, Chennai, India.

Contact:

nikhilshaw_ra@srmuniv.edu.in

shaw.nikhil05@gmail.com
Abstract

In computer science, a selection algorithm is an algorithm for finding the \textit{kth smallest number in a list or array}; such a number is called the \textit{kth order statistic}. This includes the cases of finding the minimum, maximum, and median elements. There are \textit{O(n)} (worst-case linear time) selection algorithms, and sublinear performance is possible for structured data; in the extreme, \textit{O(1)} for an array of sorted data. Selection is a subproblem of more complex problems like the nearest neighbor and shortest path problems. Many selection algorithms are derived by generalizing a sorting algorithm, and conversely some sorting algorithms can be derived as repeated application of selection.

This new algorithm although has worst case of \textit{O(n^2)}, the average case is of near \textit{linear time for an unsorted list}.

Algorithm

Legend:

\texttt{arr}: Unsorted array of numbers.

\texttt{k}: Kth smallest element in the array.

\texttt{n}: Total number of elements in the unsorted array.

\texttt{Ns}: Closest element smaller than Kth element encountered.

\texttt{Nl}: Closest element greater than Kth element encountered.

\texttt{smallcount}: Keeps count of elements smaller than arr[j].

\texttt{largecount}: Keeps count of elements larger than arr[j].

\texttt{smalllimit}: Number of possible elements smaller than Kth element.

\texttt{largelimit}: Number of possible elements greater than Kth element.
Algorithm starts here:

1. Start

2. Input the unsorted array (arr[]), its size (n) and value of k (Kth smallest element)

3. smalllimit=k-1
   largelimit=n-k
   Ns= -infinity (minimum value possible)
   Nl= +infinity (maximum value possible)

4. Note: Keep count of smallcount, largecount, smalllimit, largelimit, Ns and Nl

   from i=0 till i<n:

   smallcount=largecount=0
   j=i+1

   if i==n-1 then:
       print the element (this is the Kth element)

   if arr[i] doesn't lie between Ns and Nl, then:

   if arr[i]<Ns then:
       decrement smalllimit by 1
else:

decrement largelimit by 1

i++

continue the loop

}

from j=i+1 till j<n:

{

if arr[j] < arr[i] then:

increment smallcount by 1

else:

increment largecount by 1

if smallcount>smalllimit then:

Decrement largelimit by 1

Nl= arr[i]

break out of the loop

else if largecount > largelimit:

Decrement smalllimit by 1

Ns= arr[i]

break out of the loop

Increment j by 1
if smallcount is equal to smalllimit and largecount is equal to largelimit
    print the element in the array (ie. print arr[i])

5. End

Analysis of the Algorithm

Best case:

When the Kth element is present in the first position in the unsorted array. For example,

Arrangement: 6 4 7 1 9 0 11 (unsorted)
            n=7, k=4
            0 1 4 6 7 9 11 (sorted)

Time complexity- O(n)

Worst case:

When Kth element is present at the end of the unsorted array and elements are arranged in alternate order smaller and greater than Kth element (from greatest to smallest). For example,

Arrangement: 11 0 9 1 7 4 6 (unsorted)
            n=7, k=4
            0 1 4 6 7 9 11 (sorted)

Time complexity: O(n^2)
Conclusion:

The algorithm works well if either the element is present in the first position of the unsorted array or when elements just smaller and greater than Kth element is present in the initial positions of the unsorted array.

Example: When elements just smaller and greater than Kth element is in 1st and 2nd position

Arrangement:  7 4 11 0 9 1 6  (unsorted)
               n=7, k=4
               0 1 4 6 7 9 11  (sorted)

In such a case also complexity is $O(n)$. 
References

wikipedia.org-