HomeWork 3 : Order Statistics

Sameer S. Shende

February 12, 1996

1. **Order Statistics**

   Order statistics calculates the kth largest element in an array in linear time complexity. The algorithm is given below :

```
int select_kth_largest(int k, int *S, int n)
return the kth largest element from the ordered set
S of integers having n elements
if n < 50 then
begin
sort in dec. order S, return kth largest element in S
end
else
begin
divide S into ceil(n/5) sequences of 5 elements each
with upto four leftover elements
sort in dec order each 5 element sequence
let M be the sequence of medians of the 5 element sets
m = select_kth_largest(ceil(n(M)/2), M, n/5);
let S1, S2 and S3 be the sequences of elements in S
greater than, equal to and less than m respectively.
if n(S1) >=k then return select_kth_largest(k, S1, n(S1));
else
if (n(S1) + n(S2) >= k) then return m
else
return select_kth_largest (k - n(S1) - n(S2), S3);
end
```

2. **Discussion**

   Here S is appropriately padded with mininum negative integers and then the median is calculated and the sequences S1 and S3 are made. For the purpose of comparison with a quick and dirty approach, quicksort was coded for sorting the array in decreasing order and the kth largest element was found. In the implementation, S1 and S3 are arrays that are dynamically allocated each time and the memory associated with these is freed at appropriate places. Thus, the system calls have an added overhead and this appears only in the case of order statistics. The same array is then passed to the quicksort based routine to calculate the kth largest element and both return the correct kth largest element and the timings associated with each.

3. **Results**

The timings for order statistics and quicksort are compared below for different sizes of the array.

Array Size	Order Statistic Timings	Quicksort Timings
10	0.000051	0.000052
100	0.000539	0.000406
1000	0.016076	0.00579
10000	0.076752	0.099892
100000	0.654696	1.017191
1000000	6.364369	12.410070

4. **Conclusion**

We notice that the asymptotic growth rates of order statistics is linear whereas that of quicksort is non-linear. For small sizes of the array ($<$ 1000 elements), the quicksort performed slightly better than order statistics, whereas for large number of elements order statistics outperformed quicksort (for e.g., for one million elements, order statistics took 6.36 secs whereas quicksort took 12.41 secs). Thus, we conclude that the advantage of order statistics is that it has a better time complexity $O(n)$ than quicksort which has $O(nlogn)$.

But this is the case when we wish to calculate the kth largest element just once. If we had a list of inputs of requests (different k's) then quicksort would be better as the array would be sorted and order statistics would need to repeatedly invoke it for different k's.

The other disadvantage of order statistics is that it needs much more memory. At each invocation of the routine it needs to allocate two arrays S1 and S3 which could be as big as $3n/4$ where n is the size of the S input array. Although the memory is managed efficiently by freeing it when the appropriate array is not needed, this is a definite overhead especially since this is a recursive call and adds to the space overhead. This is not the case with quicksort and so we can solve a much bigger problem with quicksort.

Thus, we conclude that order statistics which has an $O(n)$ time complexity performs better than quicksort which has an $O(nlogn)$ time complexity for arrays having more than 2000 elements.

5. **Appendix**

The source code can be found in /home/grads/sameer/cis621/h3/kl.c and the script files are in log1, log2 and log5. The appendix consists of

(a) Source code listings

(b) Plots for different sizes of array showing the performance of order statistics vs. quicksort.

(c) Data used in the plots for comparison.

(d) script files showing the sample execution runs.