COMPSCI 260P | Algorithms | Project 1 Report
(K^{th} Smallest Element : DSelect vs Quick Select Algorithms analysis)

CONTENTS:

I) QUICK SELECT ALGORITHM

Pseudocode:

Algorithm quickSelect(S, k): *Input:* Sequence S of n comparable elements, and an integer $k \in [1, n]$ *Output:* The kth smallest element of S

if n = 1 then return the (first) element of S

pick a random element x of S remove all the elements from S and put them into three sequences:

L, storing the elements in S less than x; E, storing the elements in S equal to x; G, storing the elements in S greater than x.

if $k \leq |L|$ then quickSelect(L, k)

else if $k \leq |L| + |E|$ then

return x // each element in E is equal to x

else

quickSelect(G, $k - |L| - |E|$)

Complexity of Quick Select Algorithm:

Analysis:

0(n)

To prove that quick select runs in O(n) time, we use linearity of expectation.

E(X+Y)=E(X)+E(Y) and E(cX)=c E(X), where we use E(Z) to denote the expected value of expression Z.

Let t(n) denote the running time of randomized quick select on a sequence of size n. Since, a randomized quick select algorithm depends on the outcome of random events, its running time t(n) is a random variable. We are interested in bounding E(t(n)). A pivot used to partition the input set is considered a good pivot if it partitions S so that size of L and G is at most 3n/4 and bad otherwise. A pivot is good with probability ½. Let g(n) denote the number of consecutive recursive invocations before getting a good invocation. Then,

t(n) <= bn g(n) + t (3n/4)

where b>0 is a constant. We are focusing on the case when n is larger than 1 for we can easily characterize a closed form that t(1)=b. Applying linearity of expectation:

E (t(n)) <= E (bn · g(n) + t(3n/4)) = bn · E (g(n)) + E (t(3n/4))

For good recursive call (probability=1/2), E(g(n))=2

For n>1, T(n)<= T(3n/4)+2bn

General case gives us: $T(n) \leq 2bn * \sum_{i=0}^{\log_{\frac{4}{3}} n} (3/4)^i$

Thus, we obtain T(n) is O(n).

Implementation Design Choices:

1) Array for storing N integers, from which k^{th} smallest number has to be selected.
2) Used Java Random() class for generating random number as the pivot in the array.
3) Used Java getCpuTime() function, to compute current CPU time, before and after

execution of Quick sort method.

4) Executed for N lying in range 5000 to 1000000 in intervals of 5000 (200 data points), and 20 different values of k for each N. Averaged 20 values to get elapsed time of 1 value of N.

II) DETERMINISTIC SELECT ALGORITHM

Pseudocode:

Algorithm DeterministicSelect(S, k):

Input: Sequence S of n comparable elements, and an integer $k \in [1, n]$

Output: The kth smallest element of S

if n = 1 then return the (first) element of S

Divide S into $g = \lceil n/5 \rceil$ groups, $S_1,\ldots, S_g$, such that each of groups $S_1, \ldots, S_{g-1}$ has 5 elements and group S_g has at most 5 elements. for i ← 1 to g do

Find the baby median, x_i, in S_i (using any method) x ← DeterministicSelect($\{x_1, \ldots, x_g\}$, $\lceil g/2 \rceil$) remove all the elements from S and put them into three sequences:

• L, storing the elements in S less than x • E, storing the elements in S equal to x • G, storing the elements in S greater than x.

if $k \leq |L|$ then DeterministicSelect(L, k)

else if $k \leq |L| + |E|$ then

return x // each element in E is equal to x

else

DeterministicSelect(G, $k - |L| - |E|$)

Implementation Design choices:

1) Used array to store integer values.
2) Used Java Random() class to get random numbers within certain range.
3) Used Java getCpuTime() function to calculate elapsed time for execution of DSelect algorithm.
4) Calculated elapsed times for number of array elements lying between 5000 and 1000000, in intervals of 5000. (Hence 200 data points)
5) Executed a value of N, for each group (3,5,7 or 9) with 20 different values of k, and calculated average time elapsed for N (number of elements).

Complexity of DSelect Algorithm:

1) **For Groups of 5**

   If we divide the input into groups of 5, after partitioning the input with the median of medians, say m, as the pivot element, we get a lower bound on the number of elements that are greater than m and lesser than m. We get atleast 3/10 elements of the array that are lesser than m and atleast 3/10 elements that are greater than m.
   Recurrence relation becomes:
   $T(n) <= cn + T(n/5)+T(7n/10)$
   for some constant c.
   Number of comparisons done is 24n.
   On solving the above recurrence, we get the algorithm runs in **O(n)** time.

2) **For Groups of 3**
   If we divide the input into groups of 3,after partitioning the input with the median of medians, say m, as the pivot element, we get a lower bound on the number of elements that are greater than m and lesser than m. We get at least ($\lceil 1/2 \lceil n/3 \rceil \rceil$-2) elements that are greater than m and some smaller than m.
   Recurrence relation for runtime is:
   $T(n)<= T(\lceil n/3 \rceil)+ T(2n/3 + 4) + O(n)$
   The above recurrence does not satisfy O(n), as in intermediate steps, we are left with sub problems of size n. This suggests that the problem was not reduced to an optimal size. Hence, overall times does not come to be linear.
   Runtime for select with groups of 3, comes to be **O(n log n).**

3) **For Groups of 7**

   If we divide the input into groups of 7, after partitioning the input with the median of medians, say m, as the pivot element, we get a lower bound on the number of elements that are greater than m and lesser than m as:
   For elements greater than m,half of the sublists consisting of 7 elements has atleast 4 elements greater than m. We ignore sublist containing m and the last sublist which has

sublist of size at most 7. Thus, number of elements greater than m is atleast
$4*(\lceil 1/2\lceil n/7 \rceil \rceil -2) >= 2n/7 – 8.$
Similarly, the number of elements that are smaller than m is also $2n/7 – 8$.
This algorithm calls recursively on a problem of size at most $n- (2n/7 – 8)= 5n/7 + 8$.
Recurrence relation for runtime is
$T(n) <= T(\lceil n/7 \rceil)+ T(5n/7 + 8) + O(n)$

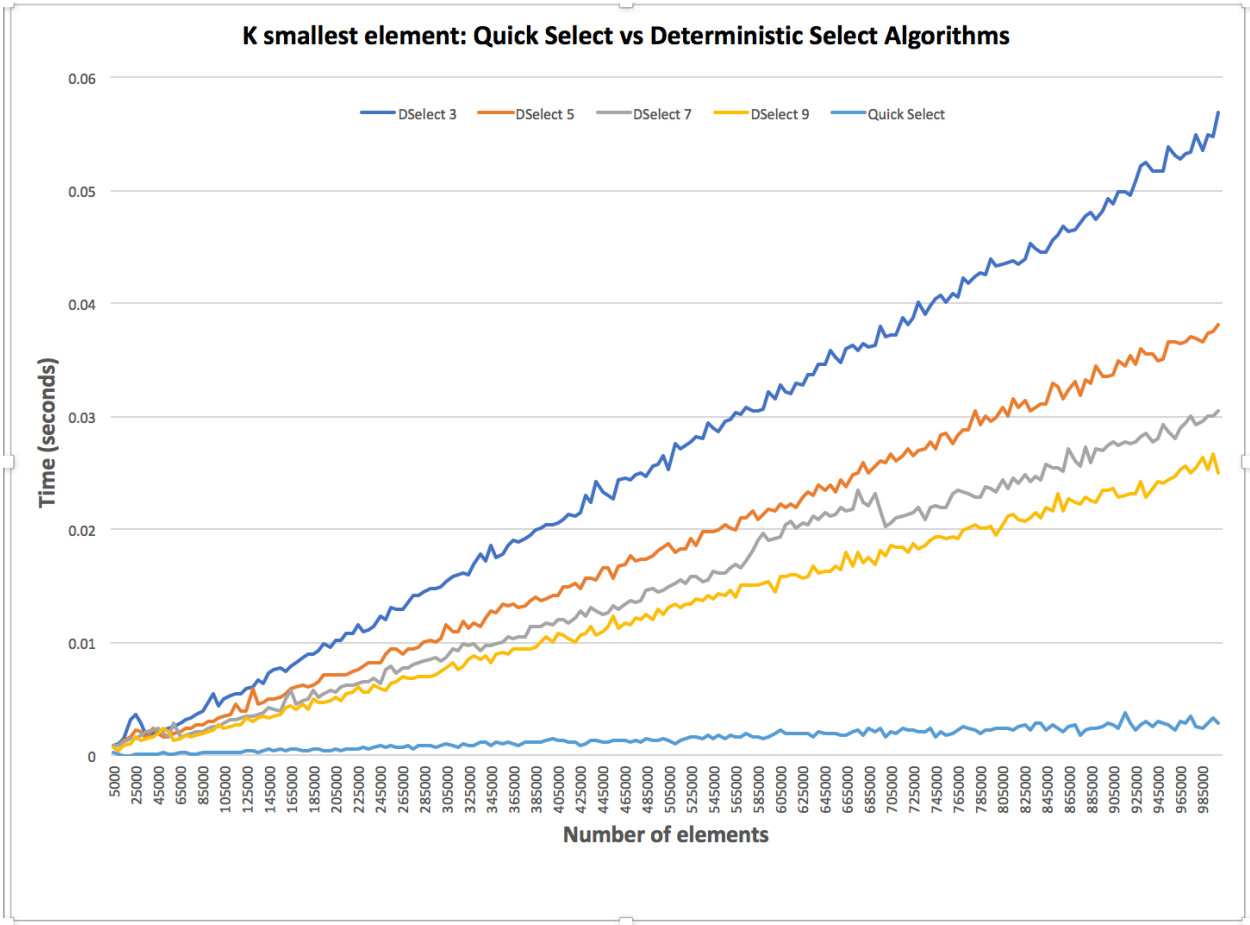
On solving the above recurrence, we find that for $n > 56$, we can find value of c for which the above equation runs in linear time.
Hence, runtime is **O(n).**

4) **For Groups of 9**
Just like above groups, for groups of 9, we get number of elements greater than m as
$5*\lceil 1/2\lceil n/9 \rceil -2 \rceil > = (5n/18 -20)$.
Recurrence relation becomes:
$T(n) = T(n/9) + T(13n/18 +20)$
On solving the above relation, we find that the algorithm runs in **O(n)** time for groups of 9.

III) Best asymptotic characterization:

The best running time is obtained from grouping of 5 or 7 elements of array. It provides fewer comparisons as compared to other groupings and finds the kth smallest element in linear time. Groups of 3 does not reduce the size of the sub problem and hence, it does not lead to linear time complexity.

IV) EXPERIMENTAL RESULT (Quick Select vs Deterministic Select)



Upon running both the algorithms for 200 different values of N (lying in the range 5000-1000000), we find that DSelect grouping in 5 or 7 results in best time complexity which is O(n), as the size of subproblem is reduced to less than N. DSelect grouping of 3 leads to O(nlog n) time complexity,as the size of problem is not reduced. Quick select algorithm provides linear time complexity, but its worst case behavior is $O(N^2)$, when every time an inappropriate random pivot is selected.

(References: Algorithm design and applications by Goodrich and Tamassia)