Analysis of Median of Medians Algorithm

David Helmbold, Spring 2005

This document contains a more detailed analysis of the SELECT or median of medians algorithm than is contained in the text. Let S be the set of n distinct elements (so no elements are repeated) being selected from, $g = n/5$ rounded down be the number of groups, and m be the "median of medians" element found by the algorithm. Let $C(n)$ be the worst case number of comparisons between elements done by the select algorithm when called on n elements.

Claim 1: The median of 5 elements can be found using 6 comparisons.
Proof: see Knuth "The Art of Computer Programming" vol 3 (Sorting and Searching) page 215 (this is $V_3(5)$ in his notation).

Claim 2: At most $7n/10+2$ elements in S are (strictly) greater than m and at most $7n/10+2$ elements in S are (strictly) less than m.
Proof: Let's consider how many elements are less than the median, the same argument can be used to bound the number of elements greater than the median of medians. There are $g = \lfloor n/5 \rfloor \geq \frac{n-4}{5}$ groups. at least $\lceil g/2 \rceil \geq \frac{n-4}{10}$ of the groups have medians greater than or equal to the median of medians, m (consider the two cases where g is even and g is odd). Each of these $\geq \frac{n-4}{10}$ groups contains three elements greater than equal to m. Therefore at least $3\frac{n-4}{10} > \frac{3n}{10} - 2$ elements are guaranteed to be greater than or equal to m. Since at least $\frac{3n}{10} - 2$ elements are greater than or equal to m, at most $\frac{7n}{10} + 2$ elements can be (strictly) less than m.

Note: this analysis is a little more detailed than that in the book and winds up with a "+2" where the book has a "+6". (Can you see where the book's analysis doesn't count some things that it could?)

Theorem: The Select algorithm does at most $50n$ comparisons in the worst case.

Proof: By induction on n. For $n \geq 1$, let $P(n)$ be the statement that $C(n) \leq 50n$.

Base Cases: $n \leq 100$.

When $n \leq 100$ the select algorithm calls selection sort, does a swap, and then returns. The selection sort algorithm does $n(n-1)/2 \leq 50n$ comparisons in this case, showing that each $P(n)$ holds for each $1 \leq n \leq 100$.

Induction step. Assume $n \geq 101$ and for all $1 \leq j < n$ that $P(j)$ holds to show that $P(n)$ also holds.

Consider an arbitrary call of select on a set of $n \geq 101$ elements. All comparisons between elements are done: to find the medians of the groups, in the first recursive call to find the median of medians, in the call to partition, or in the second recursive call on either the small keys or the large keys (but never both). Note that the second recursive call may not be needed.

We know from Claim 1 that at most $6g$ comparisons are required to find the medians of the g groups (recall $g = n/5$ rounded down). Using $P(g)$, at most $50g$ comparisons are required in the first recursive call to find the median of medians. The partition algorithm makes $n-1$ comparisons. From Claim 2, the second recursive call to select is on at most $7n/10+2$ elements. Therefore the second recursive call does at at most $\max_{1 \leq j \leq 7n/10+2} C(j)$

comparisons. Since $n > 100$, we have $7n/10 + 2 < n$ and (using $P(j)$) the second recursive call does at most $50(7n/10 + 2) = 35n + 100$ comparisons.

Adding these four bounds up gives the following bound on $C(n)$.

$$\begin{aligned} C(n) &\leq 6g + 50g + n - 1 + 35n + 100 && (1)\\ &\leq 56(n/5) + 36n + 99 && (2)\\ &\leq 47.2n + 99 && (3)\\ &\leq 50n \quad \text{(Recall that } n > 100\text{, so } 99 < n\text{)} && (4) \end{aligned}$$

This shows $P(n)$ as desired, completing the proof of the theorem.

A (potentially interesting, depending on your point of view) exercise is to tighten the inequalities in the induction to get as small a constant as possible in the bound on $C(n)$.

```
/* Pseudo code for SELECT algorithm */
/* it swaps the ith smallest element in A[first..last] into position A[first] */
/* the other elements still be in the array, but scrambled around */

select(int A[],int first, int last, int i) {

   n = last - first + 1;                /* n is the number elements to select from */
   if (i > n)  {return ERROR;}  /* there is no ith smallest element */

   if( n < = 100 ) {
      /******************** For Small n *********************/
      Run selection on A[first..last] taking at most n(n-1)/2 < 50n comparisons;
      swap A[first+i-1] with A[first]   /* put ith smallest in A[first] */
   }
   else /* n > 100 */ {
     /********** main recursion ************************/
     numGroups = n / 5;     /* integer division, round down */
     for group = 0 to numGroups-1 do {
         shift = group*5;
         /* A[first+shift] is the start of the group, A[first+shift+4] is end of group */
         find median of A[first+shift .. first+shift+4] and swap it into A[first + group];
     }  /* for group */;
     lastMedian = first+numGroups-1;
     /* now the medians of the numGroups groups are all A[first .. lastMedian] */

     /******  the first recursive call to find median of medians ******/
     select(A, first, lastMedian, numGroups/2);
     /* now median of medians is in slot A[first]  */

     /*********** partition array *********************/
     k = partition(A,first, last);   /* See partition on page 146 of text */
     /* now k is the index where the median of median winds up, the smaller elements */
     /*  will be in A[first..k-1] and larger elements will be in A[k+1..last] */

     /************ where is the ith smallest element? ********/
     if (k == first + i -1)  {
        /* ith smallest is the median of medians  in A[k]  */
        swap A[k] and A[first] and return
     } else if  (k > = first + i -1) {
        /* second recursion to find ith smallest among the "small" keys in A[first..k-1] */
        select(A, first, k-1, i);
     } else  /* k < first + i -1 */  {
        /* second recursion to find the proper element among the "large" keys */
        numSmaller = k-first+1;  /* the number of "smaller" keys not recursed on */
        newi = i - numSmaller;
        /* the ith smallest of A[first..last] is the newi smallest of A[k+1..last] */
        select(A, k+1, last, newi);
        /* ith smallest now in A[k+1], put it in A[first] */
        swap A[k+1] and A[first];
     } /* if k  - second else */
  } /* if n  - else part */
} /* select */
```