CS 610 Algorithms	**Programming Assignment 2**	Prof. D. Nassimi

Given an array $A[\,]$ of n elements and an integer k in the range $[1, n]$, the function

SELECT (dtype $A[\,]$, int n, int k)

returns the kth smallest element of the array. (For example, $k = 1$ is the smallest element, $k = n$ is the largest, and $k = n/2$ is the median.) In this programming assignment, you are to implement SELECT using three different algorithms as described below, and compare their performance in terms of the number of *key comparisons*. (Do not perform the key comparisons in-line. Rather, use a function to perform each key comparison, while incrementing a counter.)

1. SELECT1: Sort the array using Quicksort and pick the kth smallest element. This has average time complexity of $O(n \log n)$ and worst-case time of $O(n^2)$.

2. SELECT2: Randomized selection. (The textbook calls it Quick-Select, p.246.) This algorithm has an average time complexity of $O(n)$ and a worst-case time of $O(n^2)$.

   (a) If $n < 25$, sort the array using insertion sort, and return the kth element. Otherwise, continue.

   (b) Pick a pivot element V in random and use it to partition the elements into three sets (L, E, G) of elements less than V, equal to V, and greater than V, respectively. Let the number of elements in these sets be n_1, n_2, n_3. Use the sizes of these sets to determine where the kth smallest element falls, and make a recursive call accordingly. That is:

   If $k \leq n_1$ then SELECT2 (L, n_1, k);
   else if $k \leq n_1 + n_2$ then return (V);
   else SELECT2 $(G, n_3, k - n_1 - n_2)$.

3. SELECT3: Selection algorithm with linear worst-case time, but with a large constant factor. (This algorithm is discussed in class and is also discussed in exercise C-4.24, p. 254, of the textbook.) The algorithm is outlined below.

   (a) If $n < 25$, sort the array using insertion sort, and return the kth element. Otherwise, continue.

   (b) Divide the set of n elements into subsets ("rows") of size 5 each, and find the median of each subset. (If n is not a multiple of 5, the last row will have less than 5 elements.) To find the median of each row of 5, you can simply use bubble-sort or insertion sort. To avoid the need for a second array, you can pack the $n/5$ row-medians in front of the same array as they are found. (Use swap operations for this packing to preserve all elements.)

   (c) Make a recursive call to SELECT3 to find the median of the $n/5$ row-medians.

   (d) Use this median-of-medians as the pivot V to partition the array of n elements into three sets as before. Then make a recursive call to SELECT3 for either the left or right partition as needed.

Run experiments for the following values of n: 10 000, 100 000, and 1000 000. For each value of n, produce an array of randomly generated elements. (You may use integers.) Then use each of the three methods to find the kth smallest element for $k = n/2$, and print one line result:

Algorithm X: $n, k, A[k]$, Number of Key-Comparisons.

(Be sure the same original array of n elements is used for all three methods.) Tabulate the performance results for the three algorithms. Explain how the results compare with the expected analytical results, and how the three algorithms compare against each other.

For example, according to the analysis, SELECT3 is expected to run in $O(n)$ time. Does the experimental results confirm an $O(n)$ running time? If so, what is the constant factor and how does it compare with the other algorithms?

Submit on Moodle:

1. Source code
2. Output produced
3. Tabulated results and Concluding Discussion