CS 361 – Lab #7 – Randomized Quick Select

One very useful variation of the Quicksort algorithm is called Randomized Quick Select (RQS). It is a O(n) algorithm that finds the k-th largest or k-th smallest element of an array without having to sort the array first. For this lab, we will assume we are looking for the k-th smallest number. Sorting the array is unnecessary if all you want is this "order statistic." For example, finding the median value of an array. There is nothing actually wrong with sorting the array, except for computational waste: the best sorting algorithms in general run in O(n log n) time, and this amount of work is overkill for what we want.

In today's lab you will implement the RQS algorithm, as described in class. Create a new folder in your account and call it lab07. Download the two source files Array.java and Driver.java from my directory online: cs.furman.edu/~chealy/cs361/lab07. Today, all you need to do is make modifications to the quickSelect() method in Array.java. But first it would be a good idea to study the existing program. In Driver.java, you will see that 3 tests of the RQS algorithm will be performed. Array.java contains our data structure and implementation.

Let's accomplish the implementation in two phases:

1. Implement quickSelect(). There is a comment at the top of the function that gives you the entire pseudocode that you need to translate into Java. When you have completed the implementation, run the Driver program. Verify that all the tests succeed. If any of them fails, you need to debug your algorithm.

2. Now that your basic implementation works, we want to empirically observe its computational complexity. Look at test cases 1 and 2. They differ in size by a factor of 10. If we have a linear time algorithm, then we should expect about a 10-fold increase in the number of operations performed by your RQS algorithm.

   Notice that in the Array class there is an attribute called ops. Its purpose is to count all of the operations performed in rqs() and quickSelect(). Add code to your quickSelect() implementation so that it increments ops appropriately according to the number of operations evaluated or executed when your program runs.

   Now, run Driver again. Run it several times because there is randomness in the algorithm. You'll probably notice considerable variation in the number of operations each time you run the program. But in the long run, there should be several million operations performed overall. Is your ratio about 10 to one?

That's all! When you are done, please e-mail me your Array.java file.