Project 2

(Due Apr 2/9, 2017)

Description:

In this project, we implement the *quicksort* in four versions:
1) the *original* version
2) the *randomization* version
3) the *median-of-3* version
4) the *median-of-5* version

Then we will compare their performances.

Requirements:

1. (*Programming Languages*) You can use either C/C++ or Java to write this program. If you want to use other programming languages, you may need to talk to me first.

2. (*Generate Data*) First you need to generate the data. Our data is composed of three 1000-element arrays. The first array is a random array, for which you can use a random function or method to generate an integer array containing 1000 random integers with values between 1 and 1000 inclusive. The array elements do not have to be distinct. The next two arrays are ordered arrays described below.

3. (*Ordered Data*) Use two 1000-element arrays to store ordered data: one in *ascending* order and one in *descending* order. These two arrays contain consecutive integers from 1 to 1000.

4. (*Display Data*) Before you sort the arrays, you need to display the *unordered* array on the screen. You display these 1000 integers in 50 lines with each line containing 20 numbers, separated by commas. You do not need to display the other two *ordered* arrays.

5. (*Develop*) You need to develop one function or method for each of the four versions of the *quicksort*. For each function or method, the input is the array to be sorted, and the output is the sorted array. You can use other supporting functions or methods when necessary, for example, for doing the *median-of-3* and the *median-of-5* features.

6. (*Median-of-5*) Write a function or method to find the *median* of an array with 5 elements in up to 6 comparisons. Note: When the number of elements of a subarray is less than 5, sort it directly. There is no need to do further partitions.

7. (*Count operations*) In this algorithm, you will count two types of operations: *comparisons* and *swaps*. In each of your *quicksort* implementation function or method, create two counters to count the number of those two operations. In order to do the tally conveniently, you can use global variables or instance variables when necessary.

8. (*Ignore overhead*) When we count the number of comparisons and the number of swaps, only count those that are involving the original array elements or copies of the elements. Do not count the part that are used for the overhead.

9. (*Display results*) Run four versions of the *quicksort* in order with appropriate information displayed. When you run each version on three arrays, print out the number of comparisons, the number of swaps, and then the sorted array in the same way as described in *4. (Display Data)*.

10. (*Display running time*) When you run each version on each array, clock the running time, and display it for comparison. We want to get some ideas for the performances of these four versions on different inputs. (Just compare and think about it. You do not need to write your observations and conclusions out.)