

Assignment#2

Sorting

Sorting and Performance Analysis
3% of course grade

Corrections to text made Oct,6,2020

Submission Requirements

Complete the following exercise and submit electronically in the assignments folder on eLearn as an IntelliJ Project – Zip the entire folder not just the source files in MyCanvas. Please refer the course Calendar for the exact date and time of the submission. This assignment is to be completed individually.

Background

You are to evaluate the Sort Algorithms that were presented in class for efficiency. You have been provided with starter code (see MyCanvas) that includes the source code for each of the sort methods discussed (bubble, selection, insertion, merge and quick) that are to be evaluated. You must add code to each of the methods to count the number of comparisons required to completely sort the data. Ensure that you generate and use the same data for each sort. You will also measure the time required to sort the data using each method by timing the algorithm using the **System.nanoTime** method.

Requirements

Your program must report the following information:

1. The time required in nanoseconds, the number of comparisons required and the time to execute a basic step (the comparison) in ns for each of the sort methods provided. The basic step is determined by taking the time required to sort an array of a size n and dividing by the number of comparisons required for that algorithm. Report your results for data sets of 30, 300, 30000 elements.
2. Modify the `sorta`, `sortb`, `sortc`, `sortd`, `sorte` methods to count the number of comparisons and return this value from the method.

Suggested Steps:

- a. Change each of the sort methods from type `void` to type `int`.
- b. Add a local count variable to each method.
- c. Increment the count just before the comparison of two array elements in each of the sort methods.
- d. Return the count value from the method and print to the screen in the main method.

3. The `sorta` method currently counts the # of comparisons by using a global variable. This is not a good programming technique. Modify the `sorta` method to count comparisons by passing a parameter. This is a little trickier as the comparisons are done in the `part` method. You should notice that the number of comparisons can be determined before the call to the `part` method. You will need to return this value from the `sorta` method and modify the `sorta` header to pass this value into each recursive call. You will need to add the counts recursively. As an alternative to the recursive counting, you can leave the code as is and complete the `sorta` counting using the global variable.
4. Adding counting to the `sortd` method. Again, you can use a recursive technique or use a global variable. Recursive counting is preferred. If this is not possible use a global variable like `sorta`
5. Time the java standard `Arrays.sort` method for all three sizes.
6. In a *Comment* section at the TOP of your source file, provide answers to the following:
 - Identify the type of sorts for each of the methods provided. Indicate which sort (a,b,c,d,e) is bubble, quick, merge, selection or insertion.
 - List in order (fastest to slowest) your selection of algorithm to use when the array to be sorted contains 30 elements. Base this on your results
 - List in order (fastest to slowest) your selection of algorithm to use when the array to be sorted contains 30000 elements.
 - List the algorithm and the BIG O notation (time complexity, average case) for that algorithm. Does the Big O notation line up with your results for 30000 elements?
 - Which algorithm has the best performance of the basic step? Does this have any impact on your selection of fastest algorithm when sorting an array of 30000 elements? Why?
 - For the standard `Arrays.sort` method, which algorithm do the performance results most resemble.

Example Output

```

Lab#2 Sorting Algorithm Performance Analysis
=====

Comparison for array size of 30

Number of compares for sort a    = ____ time = ____ ns Basic Step = ____ ns
Number of compares for sort b    = ____ time = ____ ns Basic Step = ____ ns
Number of compares for sort c    = ____ time = ____ ns Basic Step = ____ ns
Number of compares for sort d    = ____ time = ____ ns Basic Step = ____ ns
Number of compares for sort e    = ____ time = ____ ns Basic Step = ____ ns

Repeat for array sizes of 300 and 30000

```

Marking Scheme

- Program Modifications – Counting implemented, recursive counting implemented - 20%
- Program structure – Comments, follows best programming practices - 20%
- Output displayed for 30, 300, 30000 elements correct – 20%
- Discussion in comment at top of code – Questions answered based on results – 40%