SOFT 437 - Assignment 3

Michael Montagnese 20019319

**What is Instrumentation?**

Instrumentation is a computer software which is used to analyze the software performance as well as diagnose errors and log trace information. It works by monitoring certain components of code by measuring methods and certain blocks of code. In the case of this program, instrumentation tracks the time consumption of blocks, function calls, and statements.

**Purpose**

The purpose of the experiment is to determine what components of the sorting algorithms require the most time to run. We can determine which sorting algorithm is more time efficient. As well, you can determine where the bottleneck within the code is and find the block of code which is creating a bottleneck in the performance. This allows us to determine exactly where improvements should be made within the algorithm to make it more efficient. When these programs are then scaled up to sort large data sets, being able to have a more efficient code would be very important.

**Experiment Variables**

*Boolean isActive*

This variable is used to activate the instrumentation. When it is set to ‘true’ the instrumentation will run and be logged. It is automatically set to false through a constructor, this means the user has to explicitly set it to true in order for the program to work.

*Stack<Long>* *startTimes*

This variable stores a stack of type ‘Long’ which keeps track of when each instrumented block of code or function start. The value of the start time is taken by doing System.nanoTime().

*String buffer*

This variable stores the outputs which are eventually put into the log file. It contains the formatting of the indents as well as outputs such as total runtimes and comments. The string is then just split and put into the file. It gets filled up as the instrumentation is used.

**Additional Functions**

*Private String indentation()*

This function determines whether or not an indentation is required for the current instrumentation being logged. This is determined based on the number of values being stored in the startTimes stack.

**Test Cases**

The test case created is shown in the image below and it is used to ensure that the instrumentation works correctly. The instrumentation is included directly into the Quick Sort and Bubble Sort to allow for consistency of the sorting algorithms. The results are reproducible because the populateArray function always creates an array of the same size, whatever you set it to. The function is always filled with values from 1 to 99 999 inclusively. Both sorting algorithms are completed using the same randomly generated array. This allows for better comparison between the two sorting algorithms.

**Analysis**

There is some overhead in the instrumentation itself. For instance, the log is created by using a string and updating and storing the string while the program is running. Thus, space is needed to store the variable ‘buffer’ throughout the program. There is also the space required for storing the startTimes stack. There is also resources required to access these variables throughout the program. Writing the ‘buffer’ variable to the log file is also something that is required. Writing the information to the log file is what takes up the bulk of the overhead.

The bubble sort has an average run time of O(n2) where as quicksort has an average runtime of O(n\*log(n)). In an example using 100 elements in the same randomly created array, bubble sort was completed in a negligible amount of time which was listed as 0ms and quicksort completed the sort in a similar time with a value of 0ms. The bubble sort algorithm takes more time which is expected.

In that same test the time to populate the array was 1ms and the total time for the main program was 3ms. The time for the main, in itself, is without the other functions is also 3ms.

The bubble sort program with instrumentation included would definitely increase the runtime of the program as there is a large amount of overhead. The new runtime is 2460ms with the instrumentation which is a lot larger that without. This is as anticipated as writing and storing the information takes a lot of time.

The quick sort algorithm instrumentation also performed as expected by increasing the overall runtime with the overhead. The runtime of the algorithm is now 62ms. The quicksort algorithm may seem like the overhead would impact it more, but since it uses recursion instead of loops it actually causes for less overhead. The quicksort algorithm ran 2398ms faster than the bubble sort did with the same amount of instrumentation included. Running both algorithms with 10 000 values in the array and with no instrumentation bubble sort took 141ms and quick sort took 62ms.