Lab 05 Specification – Evaluating Algorithm Performance
Due (via your git repo) no later than 8 a.m., Wednesday, 17th April 2019.
50 points
[An individual two-week lab assignment. This is a MAKE-UP lab.]

Lab Goals

- Practice timing the performance speed of algorithms on a variety of input sizes
- Evaluate the Big-O performance of said algorithms

Suggestions for Success

- Take a look at the suggestions for successfully completing the lab assignment, which is available at: `https://www.cs.allegheny.edu/sites/amohan/resources/suggestions.pdf`
- Often times, the lab specification is purposefully developed to include open ended tasks and not exposing too low level details. The intention is to trigger you towards further refining your thinking skills. Thinking is more important to solve any algorithmic problem provided. Thinking through, may lead you to ask further clarifying questions, which is what my expectation is. So, there will be one or more open ended parts added to the lab. Attending labs regularly and steering discussions with the Professor, TA's and your classmates is going to be the success mantra for doing a great job in the labs. Waiting till the last minute, will restrict a student from both learning the content discussed and to have a negative effect to the effectiveness of the lab work produced.

Learning Assignment

If you have not done so already, please read all of the relevant "GitHub Guides", available at https://guides.github.com/, that explain how to use many of the features that GitHub provides. In particular, please make sure that you have read guides such as "Mastering Markdown" and "Documenting Your Projects on GitHub"; each of them will help you to understand how to use both GitHub and GitHub Classroom. To do well on this assignment, you should also read

- Goodrich chapter 04 [as recommended in the reading assignment section of the class slides.]

Assignment Details

In the past few classes, we have been discussing the performance of algorithms. We saw that there are three primary ways to evaluate the performance of an algorithm: measuring the running time of a section of code, counting the number of primitive operations with respect to input size, and giving a general estimate of the number of primitive operations by examining code structure. In this lab, we will write a few short functions and measure the performance of the code by these methods.

Part 01 - Measuring `String` vs `StringBuilder` Processing (15 points)

Java supports a few different methods for storing strings of characters. The first is the `String` class, which is the one we have most commonly used so far. One downside to `String`s is that they are immutable objects – once created, they cannot be changed. That means that when concatening two `String`s, we are in fact creating a brand new `String`. In other words, the following code segment:

```
String str1 = "My String";
str1 = str1 + " is awesome!";
```

actually replaces the original `str1` reference with a new `String` that is declared and initialized implicitly to the phrase "My String is awesome!". Concatenating and modifying `String`s is thus an expensive process in Java. For strings that will be modified frequently, `StringBuilder` is a better choice. `StringBuilder`s are mutable objects. By calling the `append()` method of a `StringBuilder` object, the internal character array will be altered, which is a much more efficient process than creating a whole new object.

In this section of the lab, we will test just how quickly we can append strings with both `String` and `StringBuilder` objects. You should create a class (the name is up to you) to compare the running time of these classes. You should use `System.nanoTime()` to measure this performance. In your code, you will be running a loop of various lengths, appending a single character onto one object from each class. Your `String` loop should look something like:

```
String myString = new String();
for (int i = 0; i < N; i++) {
    myString = myString + "a";
} //for
```

and your `StringBuilder` loop should look something like:

```
StringBuilder mySB = new StringBuilder();
for (int i = 0; i < N; i++) {
    mySB.append("a");
} //for
```

You should add your timing code around these loops, to measure how long it takes to run through the loops. You should time runs with loop sizes (values of `N`) of 1000, 10000, 50000, and 100000. For each of these loop sizes, you should measure 5 running times, to ensure that one outlying datapoint isn't confusing your analysis.

You should then take the average of these timings for each loop size, and then plot them on a chart. The x-axis of your chart should measure loop size, and the y-axis of your chart should note the running time.

When complete, you should see average timing results somewhat similar to what is reported in the textbook on pages 152-153.

Part 02 - Measuring Array Processing (15 points)

Since we spent some time earlier in the class discussing both arrays (1D and 2D), it would also be interesting to evaluate the performance of arrat processing.

For this secton of the lab, you should test both insertion and retreival from arrays. For the insertion component, you can add 1000, 10000, 50000, and 100000 items (values of `N`) to an integer array. For retrieval, you can do something similar, using the index in the array. See the code on the next page.

As with the last section, you should measure 5 running times for each of the array size, to ensure that one outlying datapoint isn't confusing your analysis. Additionally, you should take the average of these timings for each array size and plot them on a chart. The x-axis of the chart should measure the array size, and the y-axis of the chart should note the running time.

```
// Add timing code around each of these code chunks and measure them separately
// Add to array
int[] myArray = new int[N];
for (int i = 0; i < N; i++) {
    myArray[i] = i;
} //for

// Get from array
for (int i = 0; i < N; i++) {
    int something = myArray[i];
} //for

// Search if the element "data" exist in the array
for (int i = 0; i < N; i++) {
    if (myArray[i] == data)
      return true;

} //for
return false;
```

Here is what one trial run of the output (in nanoseconds) looks like on my old and slow laptop for `N`=10000:

```
One run results:
   Adding to array:       193213
   Get from array:        198122
   Search in array:       179920
```

Part 03 - Additional Questions (20 points)

Please answer the following questions thoroughly:

1. Evaluate the Big-O order of growth of the following loops and explain why:

```
for (int j = 0; j < N; j++) {
    for (int k = 0; k <= j; k++) {
        total += arr[j];
    } //for
} //for
```

2. Evaluate the Big-O order of growth of the following loop and explain why:

```
for (int j = 0; j < N; j=j+2) {
   total += arr[j];
} //for
```

3. Evaluate the Big-O order of growth of the following loops and explain why:

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < N; j++) {
        for (int k = 0; k <= j; k++) {
            total += arr[k];
        } //for
    } //for
} //for
```

4. Evaluate the Big-O order of growth of the following loops and explain why:

```
for (int j = 0; j < N; j++) {
    for (int k = 1; k < N; k = k * 2) {
        total += arr[j];
    } //for
} //for
```

Submission Details

For this assignment, please submit the following to your GiHub repository by using the link shared to you by the course instructor:

1. Commented source code from the “PartOne” program under the src sub-directory.
2. Commented source code from the “PartTwo” program under the src sub-directory.
3. A readme file explaining how to compile and run your program. In addition, include a detailed self reflection of the lab exercise in the readme file. The self reflection may include answers to questions such as:
   - ”What challenges did you face in this lab?”
   - ”What did you like in this lab?”
   - ”What did you not like in this lab?”
4. A PDF file that contains your answers to Part 03 of this lab. Present your answers as clear as possible.
5. It is highly important, for you to meet the honor code standards provided by the college. The honor code policy can be accessed through the course syllabus. Make sure to add the statement ”This work is mine, unless otherwise cited.” in all your deliverables such as source code and other textual documents.

Grading Rubric

1. The lab is worth 50 points. It is expected that the submission has the complete solution to the problem above. The breakdown for this assignments points is :
   - Part one: 15 points
   - Part two: 15 points
   - Part three: 20 points
2. There will be full points awarded for the lab, if all the requirements in the lab specification are correctly implemented. Partial credits will be awarded if deemed appropriate.

3. Failure to upload the lab assignment code to your git repo, will lead you to receive ”0” points given for the lab submission. In this case, there is no solid base to grade the work.

4. There will be minimal partial credit awarded if your code doesn’t compile correctly. It is highly recommended to validate if the correct version of the code is being submitted before due date and make sure to follow the honor code policy described in the syllabus. If it is a late submission, then it is the student’s responsibility to let the professor know about it after the final submission in github. In this way, an updated version of team’s submission will be used for grading. If the team did not communicate about the late submission, then automatically, the most updated version before the submission deadline will be used for grading purposes. If the team had not submitted any code, then in this case, there is 0 points given to the team automatically for the lab work. All team members will get the same grade for the actual lab part except the points allocated to the group meetings.

5. If you needed any clarification on your lab grade, talk to the instructor. The lab grade may be changed if deemed appropriate.