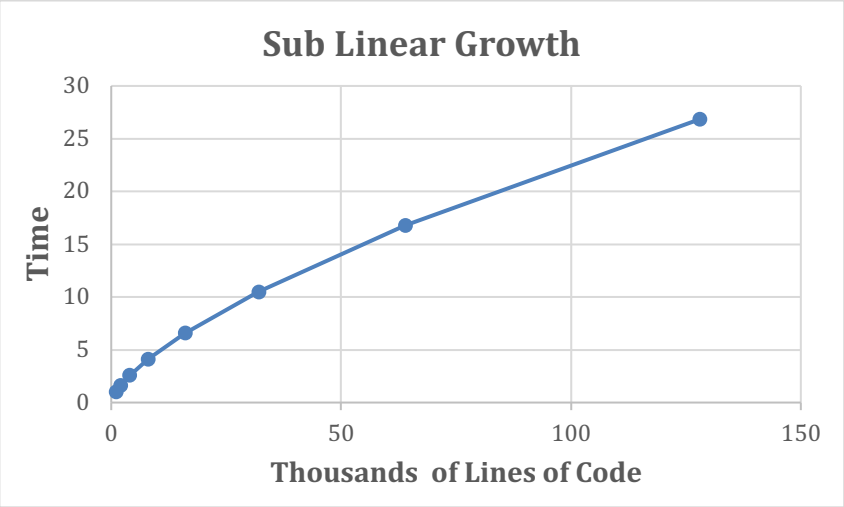
COMP 412, Supplemental Material
Note on Timing Plots

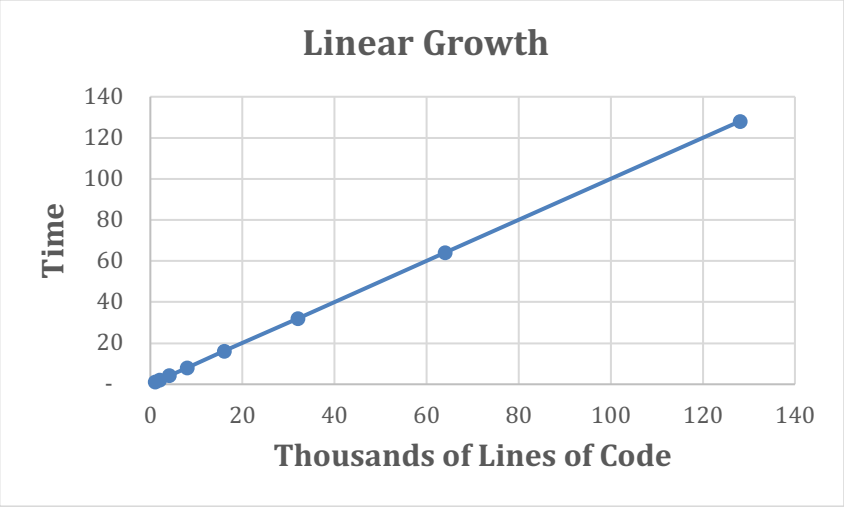
The various programming assignments in COMP 412 ask you to run one or more scalability tests on your submitted program. In each assignment, we strongly suggest that the "plot" of times versus input size should be linear. In most situations, the meaning of linear is clear.

When the input size doubles, the runtime should grow by no more than a factor of two. The following plots show Sublinear Growth, Linear Growth, Quadratic Growth and Cubic Growth.

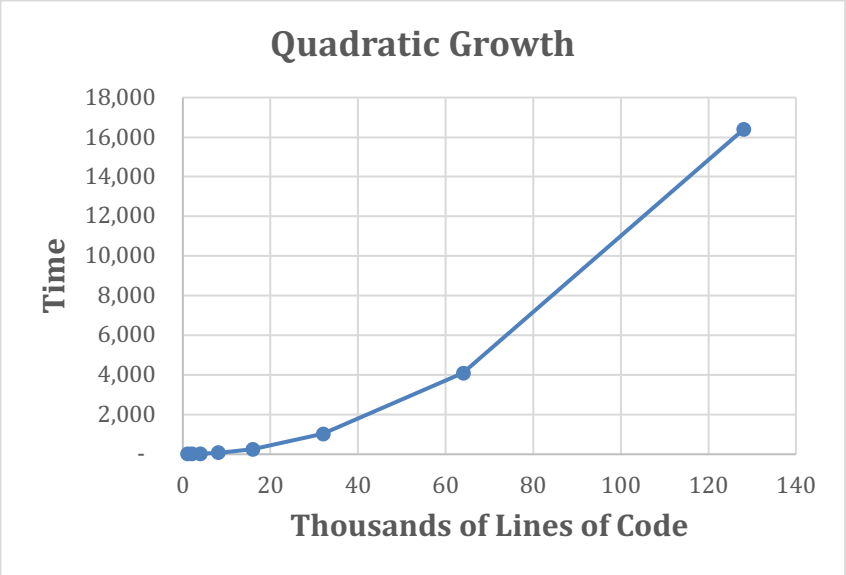
Input Size	Sublinear Growth
1,000	1.0
2,000	1.6
4,000	2.6
8,000	4.1
16,000	4.6
32,000	10.5
64,000	16.8
128,000	26.8



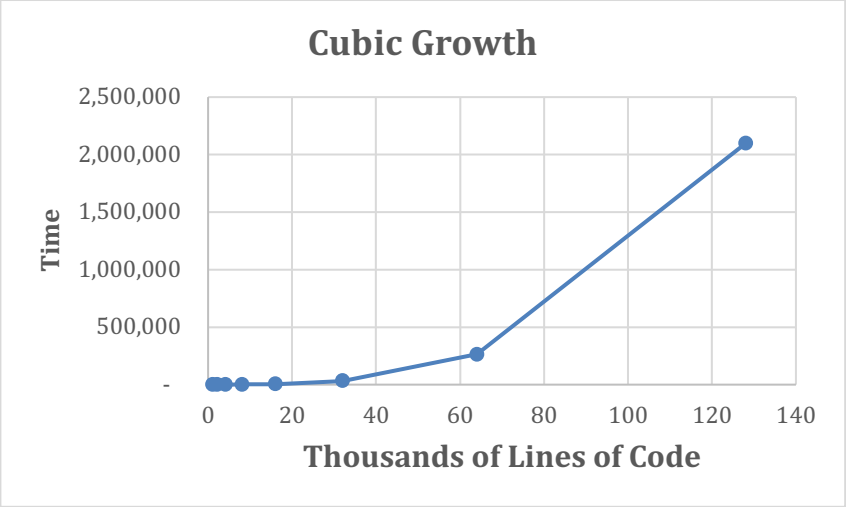
Input Size	Linear Growth
1,000	1
2,000	2
4,000	4
8,000	8
16,000	16
32,000	32
64,000	64
128,000	128



Input Size	Quadratic Growth
1,000	1
2,000	4
4,000	16
8,000	64
16,000	256
32,000	1,024
64,000	4,096
128,000	16,384



Input Size	Cubic Growth
1,000	1
2,000	8
4,000	64
8,000	512
16,000	4,096
32,000	32,768
64,000	262,144
128,000	2,097152

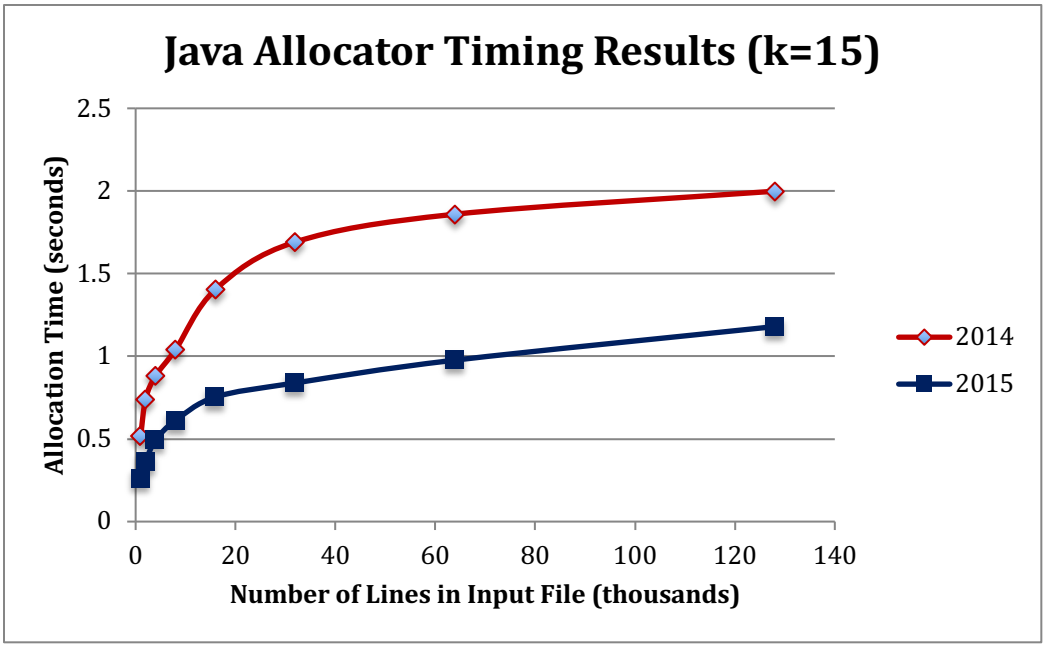


Each plot shows Time as a function of Lines of Code. In each case, the code takes 1 unit of time for 1,000 lines of code. The plots make clear the difference caused by a change in asymptotic complexity. Notice that the scale on the **Time** axis changes radically between the plots.

All four plots use linear scales on both axes. A common mistake that COMP 412 students make is to use a logarithmic scale on one or both axes. Using anything but a linear scale obscures the simple meaning of the chart.

Lab 1 and Lab 2 should show linear growth in the size of the input file. Specifically, we expect that the runtime of the submitted lab should grow by a factor of less than two when the input size doubles. There may be small variation, but the plot should show a straight line, as in the Linear Growth plot above. The **Time** value for 1,000 lines of ILOC will probably not be one. The slope may vary, but the plot should be a line.

Java Programmers: Submissions in Java will almost certainly not look like the Linear plot. In fact, they are likely to resemble the Sublinear plot. The graph below shows the fastest Java Register Allocators (now Lab 2) submitted in the Fall 2014 and Fall 2015 semesters. (For the record, CLEAR processors are now about twice as fast as when these results were measured.)



Notice the unexpected shape of the plot. The shape is an artifact of Java, rather than an artifact of the students' labs. Java uses a technique called *just-in-time compilation*, where the Java environment compiles and optimizes frequently executed code at runtime. Thus, the timing numbers for the smaller files — out to 32,000 lines — include the cost of compiling most of the frequently executed code. Once that code is compiled, the timing curve settles down to look much more like a linear plot with a relatively shallow slope (indicating a very low cost per additional line of input). In practical terms, the cost per line of code should decrease as the JIT compiles and optimizes more of the register allocator. That gives the curve its sublinear look. (See Chapter 14 in Engineering a Compiler, 3rd Edition for more on JIT compilation.)

If you write your lab in Java, don't be misled by the data on the small files. You want to look at the last three data points. They should be roughly linear or trending slightly sublinear (indicating a little more JIT compilation.)

The other artifact that you may see in a Java plot is a sharp uptick in time between two of the data points for the larger files. Such behavior often indicates that the Java VM was forced to perform one or more collections. (See the supplemental video on **Sustainable Memory Use** for more information.) If you suspect that collector activity is making your plot look nonlinear, you can run the timing test with a larger heap size. Use the -Xmn and -Xms command line options; you can hard code those into your shell script. (See **man java** on **CLEAR**).

The version of your code that you submit should run in the standard heap. You can experiment with larger heap sizes to better understand your complexity and to create additional data for your lab reports in Lab 2 and Lab 3. But, the submitted code should not manipulate the heap size.