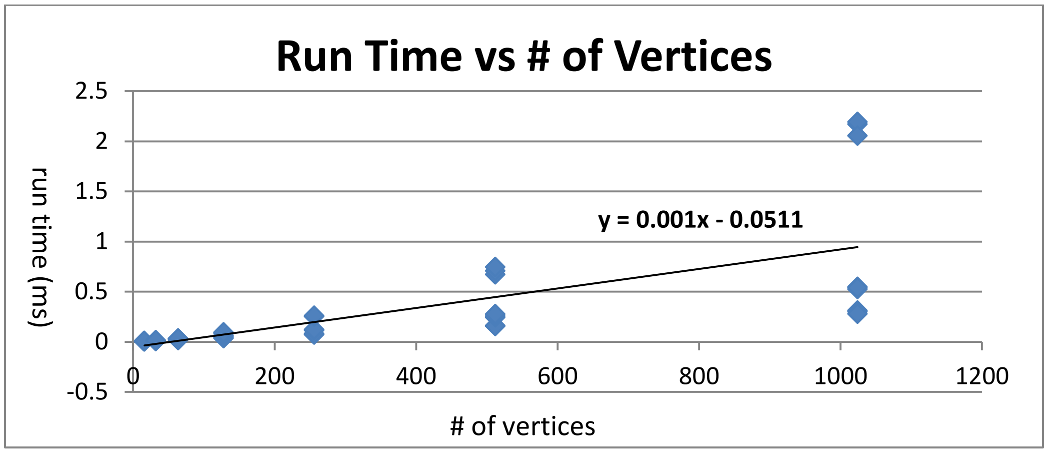
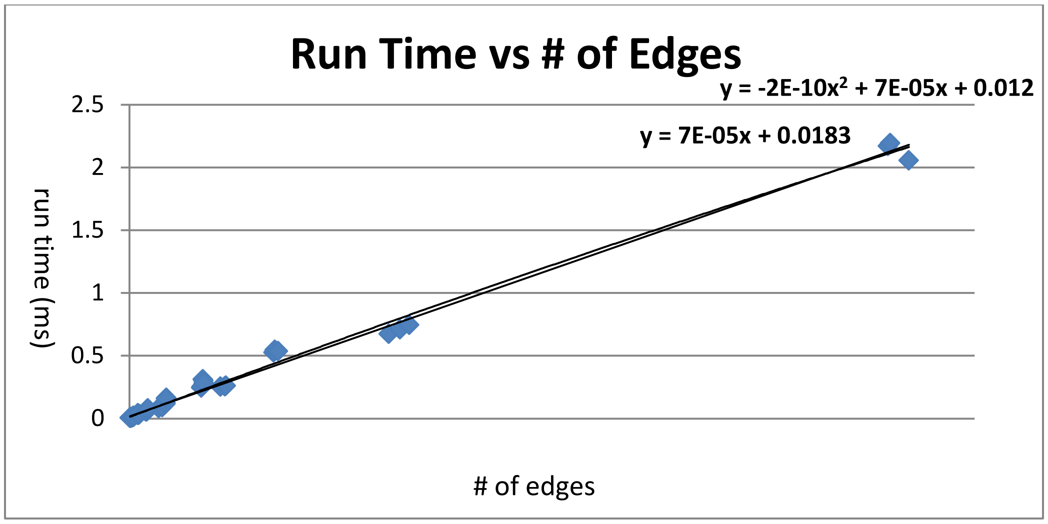
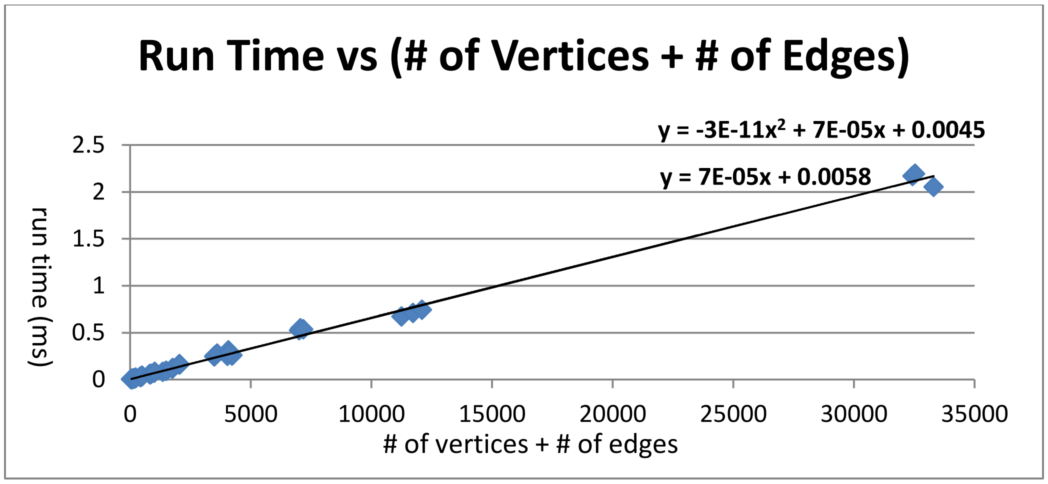
Chun-Wei Chen
mijc0517
CSE 421
Assignment #2 Report

Here's a quick summary of the laptop I used for this assignment – Intel(R) Core(TM) i7-2670QM CPU @ 2.20GHz, 2201 Mhz, 4 Core(s), 8 Logical Processor(s), 6 GB DDR3 SDRAM, and a 6M L3 cache. All the timing data was collected on this laptop. The algorithm of finding biconnected components I implemented was executed 500 times on all 63 test cases provided, and the averages run time (in milliseconds) for each case was collected and shown on the graphs below.

I plotted three different graphs, run time vs # of vertices, run time vs # of edges and run time vs # of vertices + # of edges, using Microsoft Excel based on the data I collected. Here are the graphs:







All numbers of vertices of input graphs are power of 2, from 16 to 1024.
From the graph "Run Time vs # of Vertices," it's clear that number of vertices does not dominate the run time of the algorithm since the run time of different test cases with same number of vertices are varied.

Bearing the theoretical big-O bounds of the algorithm (linear time), the graph "Run Time vs # of Edges" demonstrates that number of edges dominates the run time of the algorithm. Comparing "Run Time vs (# of Vertices + # of Edges)" with "Run Time vs # of Edges," we can also get the same conclusion that vertices does not dominate the run time, which we got in the above paragraph. And that's also the reason why the last two graphs have so little difference between each other.

Both "Run Time vs (# of Vertices + # of Edges)" and "Run Time vs # of Edges" are very informative; however, by looking at graph "Run Time vs (# of Vertices + # of Edges)," we cannot figure out whether number of vertices or number of edges dominates the algorithm without looking other graphs, so I think "Run Time vs # of Edges" is the most informative among these three.

I showed a quadratic trend line along with the linear trend line on the last two graphs. The quadratic trend line overlaps the linear one a lot of points if we simply look at the graphs. I also showed the equations of two trend lines on the graphs. We can see that the coefficients of the x^2 terms are really small ($-2*e^{-10}$ and $-3*e^{-11}$); therefore, theoretical linear time O(n+m) (or O(V+E)) is good bound for the algorithm.

Let's figure out how number of edges versus number of vertices has impact on the performance. First, fix the number of edges. The ratio of number of edges to number of vertices shows how many times smaller number of vertices n is, which is the same as we've concluded above – n does not dominate the performance. Now, fix the number of vertices. The ratio shows how much bigger number of edges m is, compared to n. And m is just n * ratio, which clearly shows that run time increases linearly when m gets larger. This also confirms what we've concluded above.