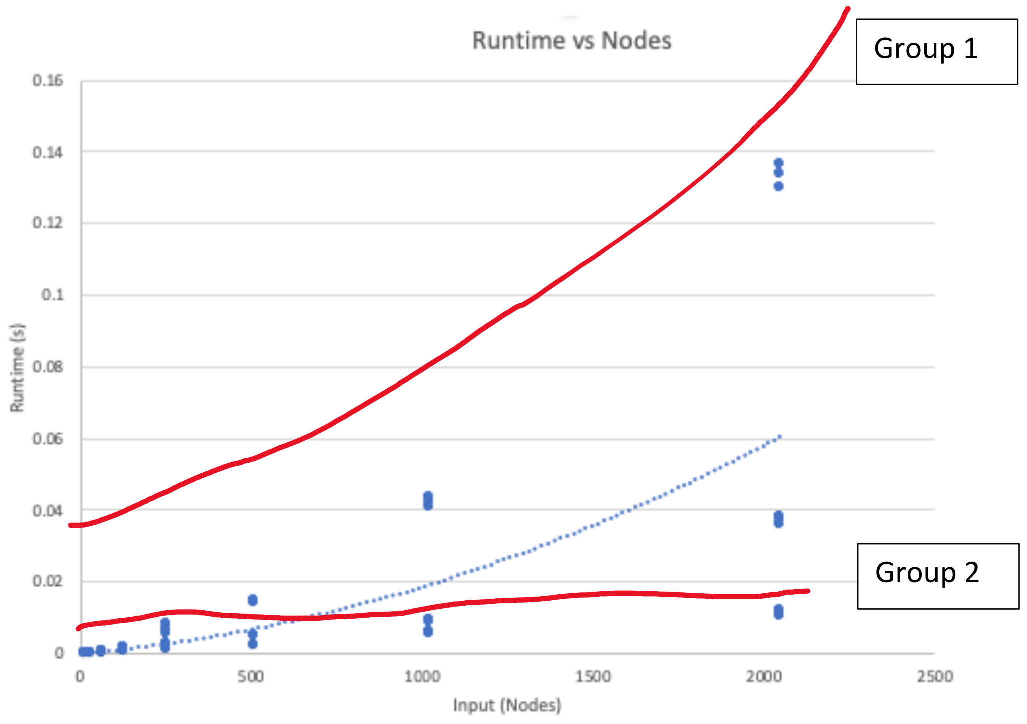
Zachary McNulty (zmcnulty, ID: 1636402)

CSE 417 HW 2 Report

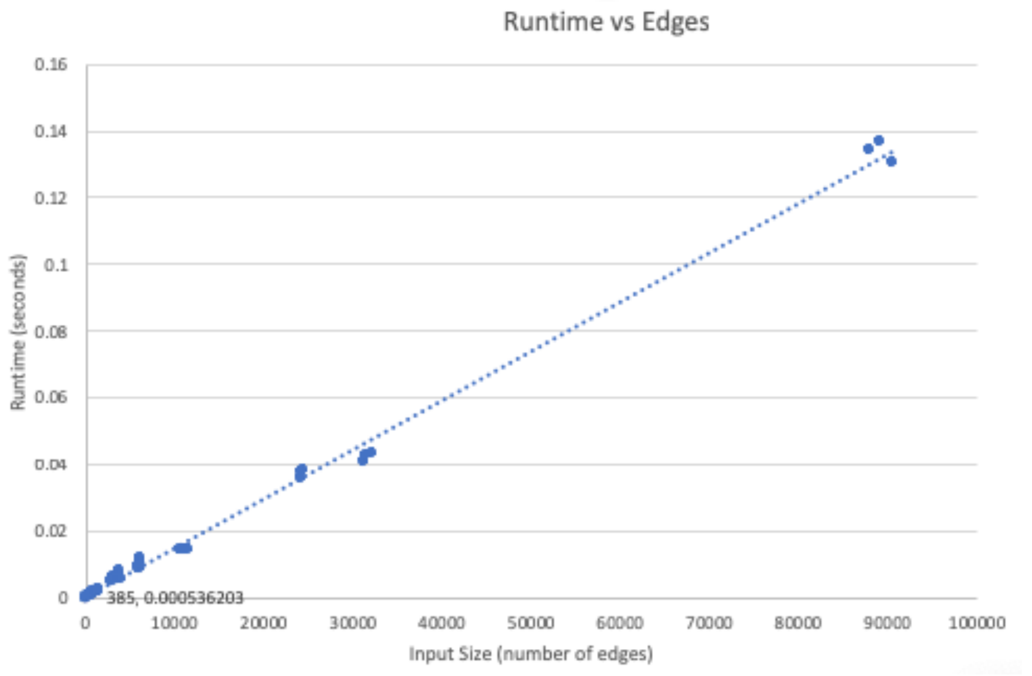
I ran the algorithm I developed for finding the biconnected components on all of the provided sample graphs (with the help of a quick bash script). I used my 2015 MacBook air for the entire project. Below is some basic information on the processor and memory of the computer I ran the testing on.

Processor: 1.6 GHz Intel Core i5
Memory: 8 GB 1600 MHz DDR3

I arbitrarily chose zero to be the start vertex for my algorithm every time, although I tested the algorithm using other start vertices as well, including articulation points, and it was still successful. I excluded the time it took to read the input files and generate the data structure to store the edge information in order to get a more specific picture of the performance of the algorithm itself. Below is a short summary of my results.



In this first graph, we see the input size (measured in the number of vertices in the graph) plotted against the runtime of the algorithm. Under this measure, the algorithm appears approximately polynomial order with respect to the input size, i.e. O($|V|^2$). However, it is clear the given polynomial fit poorly captures the data: there is a large amount of error in prediction. On closer inspection, there are two distinct groups. The lower data points are from sparse graphs where $|E| << |V|^2$ and the upper data points come from more complete graphs. The runtimes of the former group appear to have an almost linear relationship with the input size, while the latter group's runtimes clearly exhibit a superlinear relationship. This suggests that the number of edges may play a significant role in the runtime behavior of the algorithm. Below, we explore this suspicion.



This graph plots the runtime versus the input size, as measured by the number of edges in the graph. Here, unlike before, we see that the linear relationship between edges and runtime captures the data very nicely. Unlike before, we do not see any distinct subgroups in the data, and the relation holds for large input sizes. This suggests my algorithm may be of the order O(|E|).

Both graphs suggest upperbounds consistent with our claim that the algorithm is theoretically graph linear— O(|E| + |V|) --- as discussed in the section justifying the algorithm. For sparse graphs, |E| ~ |V|, giving a bound of O(|V|) which is consistent with the linear like behavior of the lower group in the first graph, and for non-sparse graphs $|E| \sim |V|^2$, which is consistent with the polynomial behavior of the upper group in the first graph. As the algorithm primarily deals with connected graphs, it is usually the case that |E| > |V|, so the relationship between the algorithms runtime vs edge count is likely more important as the edges dominate the vertices. Again, the fact that we get two separate groups in the first graph suggests that the vertex count does not tell the full story. As a result, I feel the relationship between edge count and runtime is more informative, as the number of vertices alone is not enough information to predict the long-term behavior of the algorithm.

I feel the first graph supports the big-O dogma. If we were to only observe small input sizes, the vertex count may appear a reasonable way to approximate the runtime of the algorithm. However, as the input size grows, it becomes clear there are other factors at play that dominant the runtime relationship: the small differences in runtime at low inputs due to the sparsity of the graphs quickly diverge as the input size grows and the linear component due to the number of vertices fades away. This supports the big-O philosophy that, in the long-run, lower order functions quickly become negligible and that the relationship between runtime and the input size is truly appropriately defined by the upper level terms.