Kruskal's Algorithm Growth Rate Investigation

CS4310 - Design and Analysis of Algorithms

Mariia Kravtsova

4/25/2017

Hypothesis:

Kruskal's Algorithm runs in O(m log(n)) time where n is the number of vertices and m is the number of edges.

Test Design:

I have taken the following steps to test the hypothesis:

1. Implement Kruskal's Algorithm following the pseudo code on page 428 from Algorithm Design and Applications by Michael T. Goodrich.
2. Run the code on multiple sizes of complete graphs.
    1. Test 1: range 10 - 18,000 vertices with max edges 45 - 1,619,100
    2. Test 2: range 10,000 - 100,000 vertices and 49995000 - 499950000 edges
    3. Graph and analyze test 1 and 2
    4. Run each trial 10 times and average the time
3. Analyze each graph and evaluate the trend lines, evaluate the hypothesis
4. Verify growth using mathematical analysis

Evaluation of Data:

All the files that are referred in this analysis can be found in the submission folder.

I have used the implementation of the graph made by Brian Storti and it can be found in the three files MyVertex.rb, MyEdge.rb, MyGraph.rb. Kruskal.rb is the file containing the Kruskal implementation which uses the built in Sets, Sort, and consequently union. The built in tools on ruby have been proven to be optimized for the compiler. kruskal_test.rb has the test runs of the algorithm. Commented you can see is the graph example we have been given in class, which was my example of the algorithm.

output.csv contains the data for the first test, and output3.csv contains the output for second test. I have also done some subsequent runs which are represented in output1.csv and output2.csv. However, that was just to test smaller and different implementations. For statistical analysis I used python's libraries - numpy and scipy, and for plots I used matplotlib, all this code can be found in analysis.py, and graphs are saved as png files.

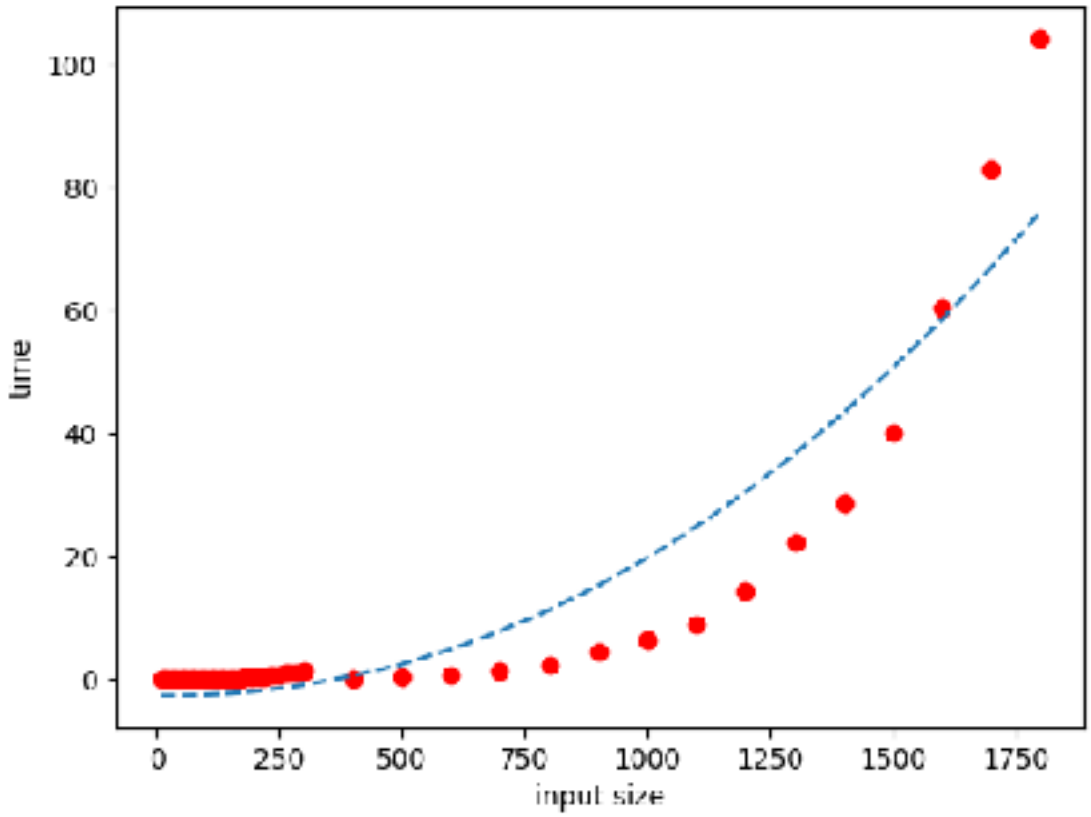
My tests are run on Intel Core i5-6400 2.7GHz with 16GB DDR4-3000 Memory.

Visual and Statistical Inspection:

Firstly, I would like to point out that the graph, the sort and the data structures I choose for the Kruskal's Algorithm all play a certain role. I did some time benchmarking for those separately, but since the time of the graph is not reflected in our benchmarking I am leaving that part out.

Let's address our hypothesis, O(m log(n)) where n is the number of vertices and m is the number of edges. For graphing and analysis purposes I changed the m to the (n*(n-1))/2 which is the maximum number of edges in a complete graph, which is input of Kruskal's. So now interpreting the trend lines and making the equation is much easier.

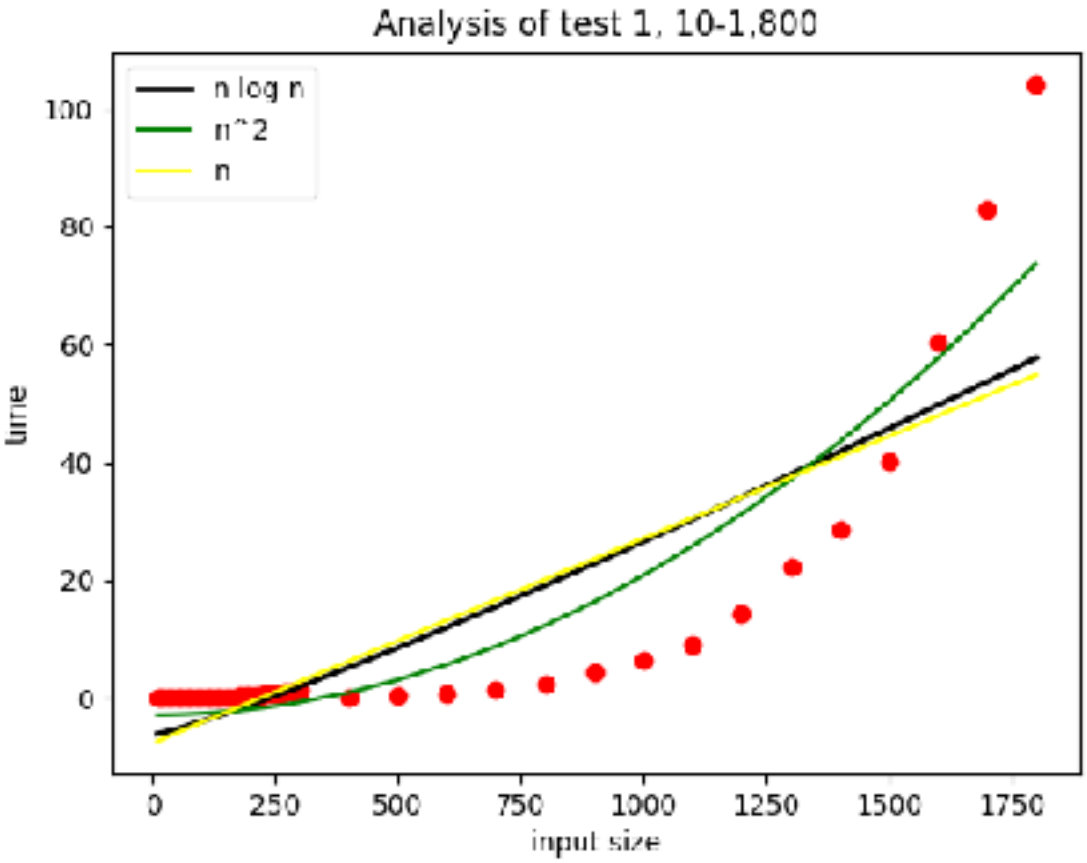
First let's graph test 1 data and see its fit, as you can see below it's actually quite good fit. I am am putting as labels just the number of vertices.



We can see the dots are the 45 data points, and the blue line is the fit of m log(n) which is really (n*(n-1))/2 * log(n). We can see that the fit is not great, but if we look at

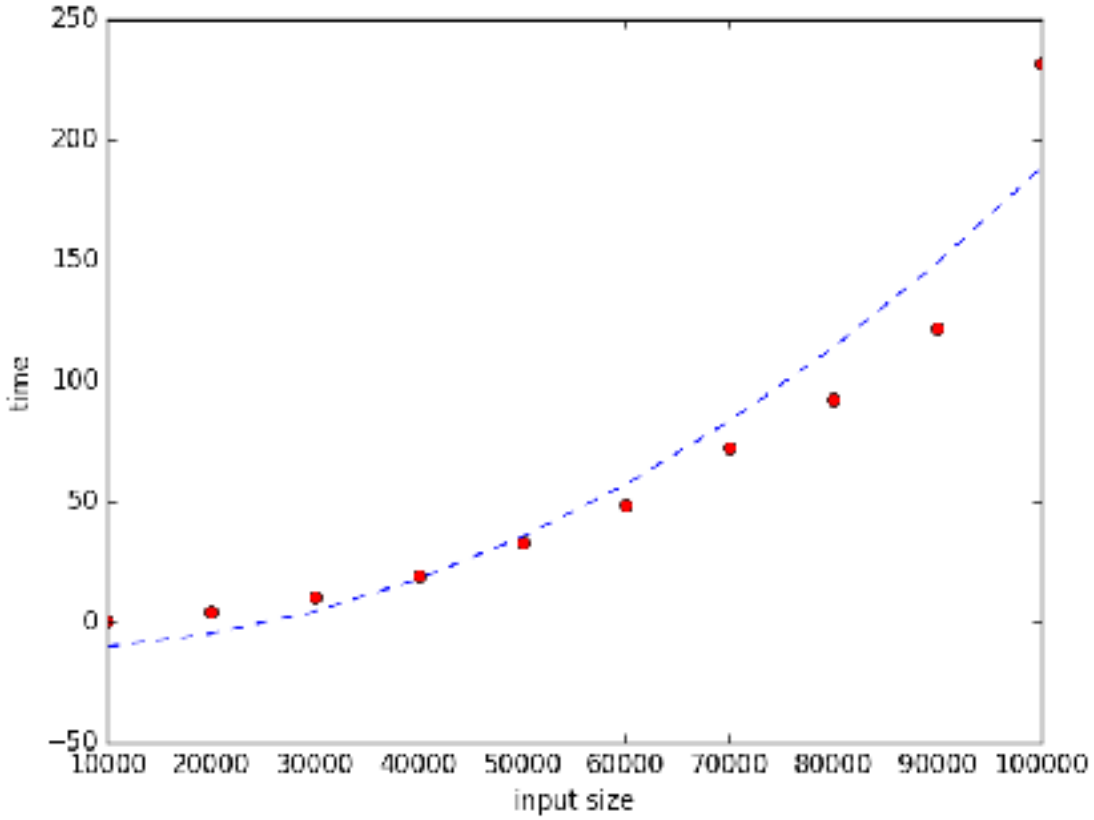
equation it is y = 4.735e-05x - 2.882 and r^2 amount is at 0.9689, and error rate of 0.05%.

Certainly we must compare this with the fit for n^2, n log n, and n, which are our upper and lower comparison for the Bog O complexities.

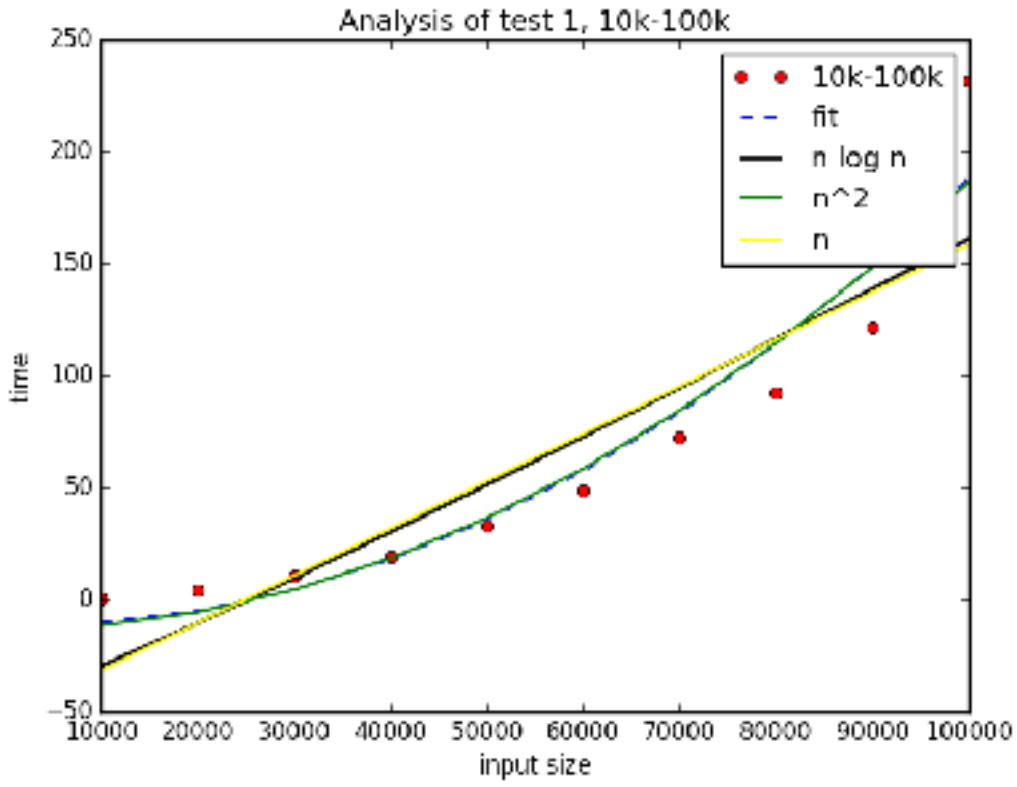


The r^2 for the polynomial is about 0.9, linear at 0.57, and n log n at 0.83. This looks strange, but I think we should trust the numbers since they are more accurate and following the trend line in generality, vs graphs are zoomed in version.

For the second test I have 10 data points where number of vertices ranges between 10,000 to 100,000. The equation for the graph bellow is y = 4.134e-07 x - 17.83. Again the red dots are the data points and the dashed blue line is the fit.



The fit is looking much better in this instance. Which we can also notice by the r squared values. For out modified m log n it is at 0.97 with an about 3% error. The polynomial at 0.91, linear at 0.71, and n log n at 88.3



Conclusion:

To be honest, Kruskal's algorithm is not very easy to analyze since it took a very long time to run, so I had to keep the number of vertices low. Besides when you multiply number of vertices with (n*(n-1))/2 the number of edges needs to be created and added to the graph is a lot. However, throughout the analysis of r^2 we do notice a fairly strong relationship for the O(m log n) with values of 0.96 and improved value of 0.97.

From just looking at the graphs I had my own concerns so I run it through Excel and my numbers matched with some rounding errors. It is my conclusion that based on my analysis I believe that my implementation does indeed support the hypothesis that Kruskal's Algorithm runs in O(m log(n)) time where n is the number of vertices and m is the number of edges.