Fractional Greedy Knapsack Growth Rate Investigation
CS 4310 – Design & Analysis of Algorithms
Spring 2017

Andrew Burcroff

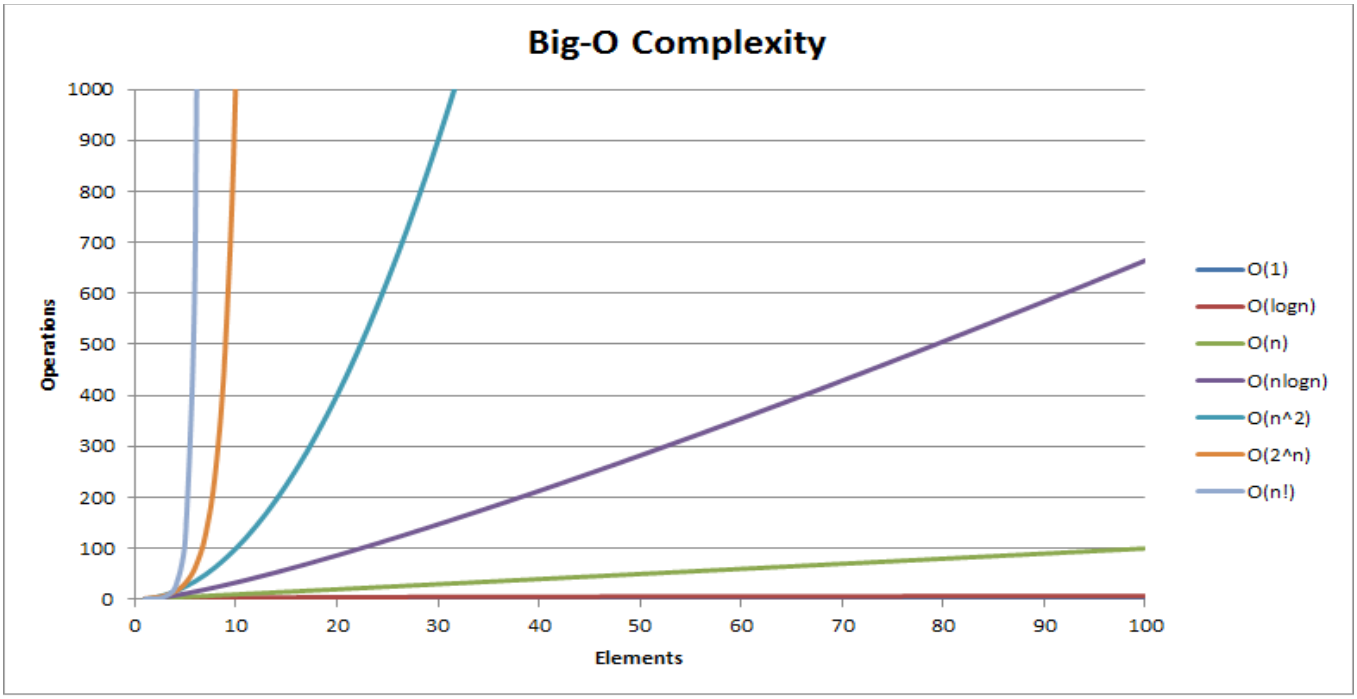
3/30/17

Hypothesis:
The Fractional Greedy Knapsack has a time complexity of O (n log (n))

Test Design:

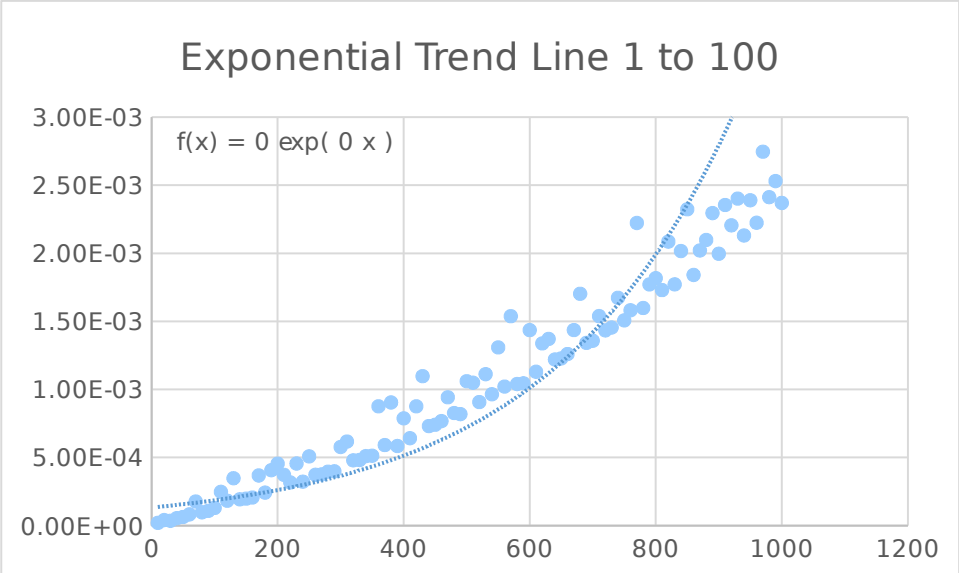
1) Implement the Fractional Greedy Knapsack algorithm in ruby
2) Run code for different total Knapsack sizes
    a) Collected 40 data points for analysis
    b) Analyze the data points to make sure that growth rate will match that of our hypothesis
3) At this point we can just look at the graph to see if our data is consisted if not visually then we must re work out experiment
4) After determining that the rate of change is similar use mathematical analysis to verify hypothesis

Frist, let's look at the at the possibilities of time and complexity.
Here is an example of the Big O Complexity, I will be comparing too.

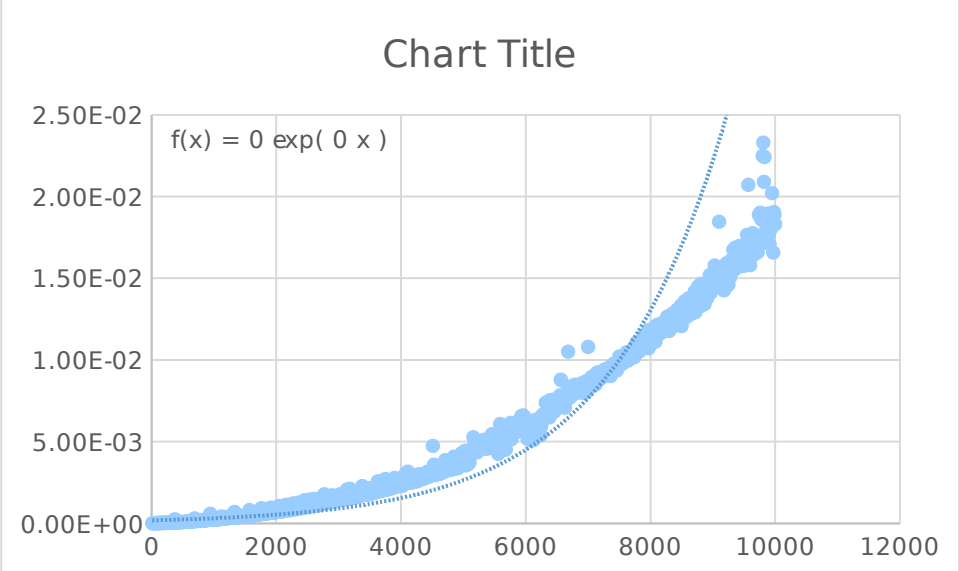


Data Evaluation:

Exponential Trend Line:

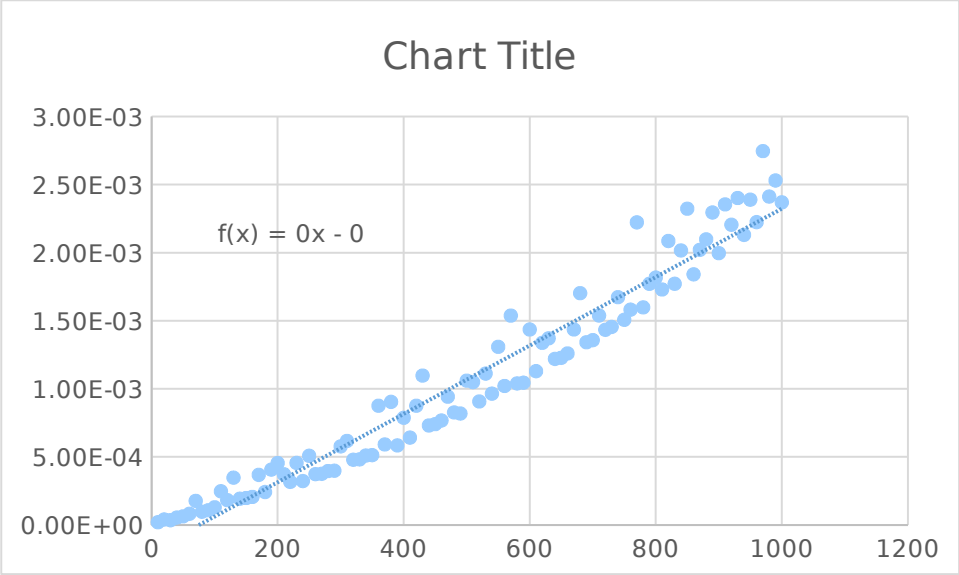


After looking at this data, it a possibility that the curve of exponential can fit this data. This data looks at 1 to 100. Now let's look at another one:



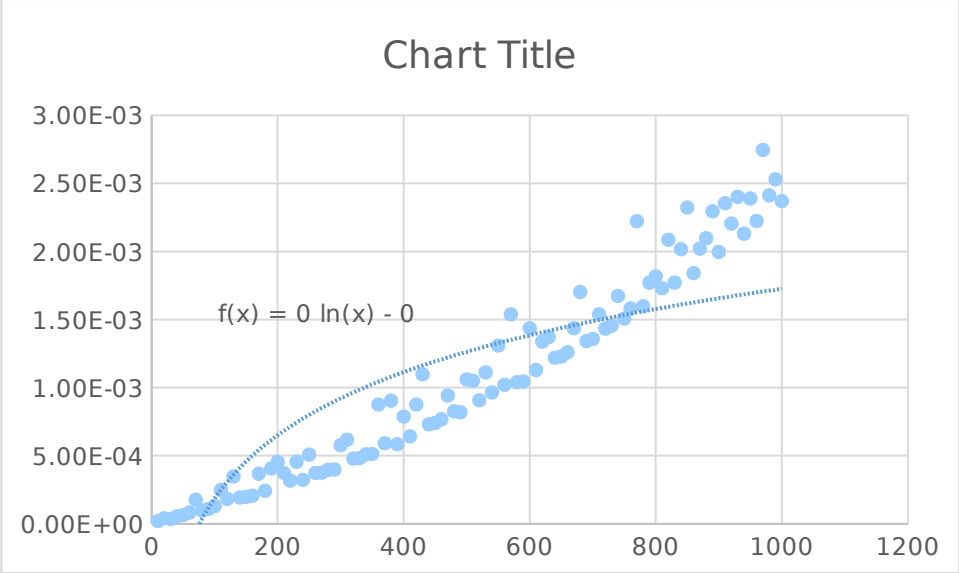
After looking at this graph with more data points. There cannot be a case that the Knapsack has a time and complexity of Exponential.

Power Trend Line:

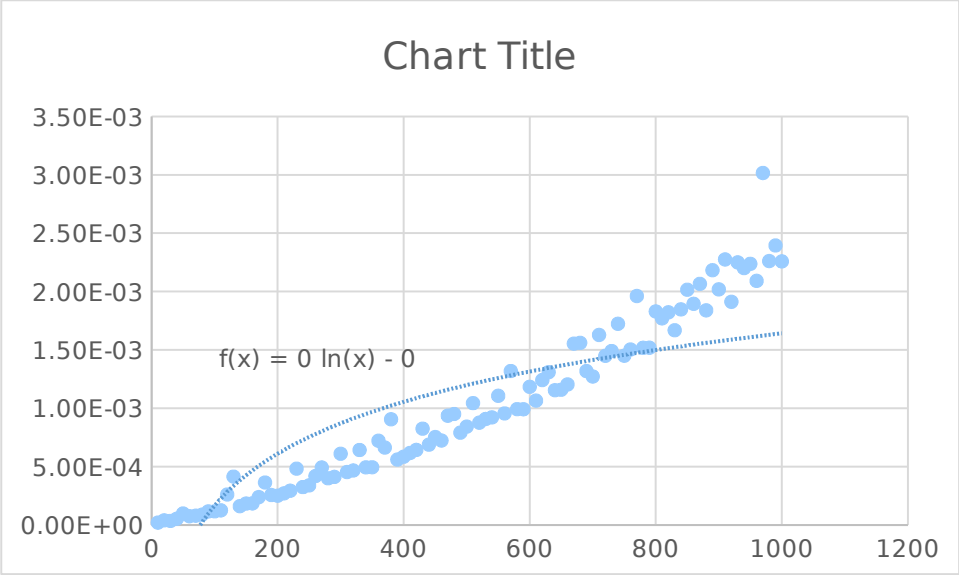


After looking at this data, it is NOT a possibility that the curve of power can fit this data. Therefore, a case not be proven of the Knapsack has a time and complexity of Power.

Logarithmic Trend Line:

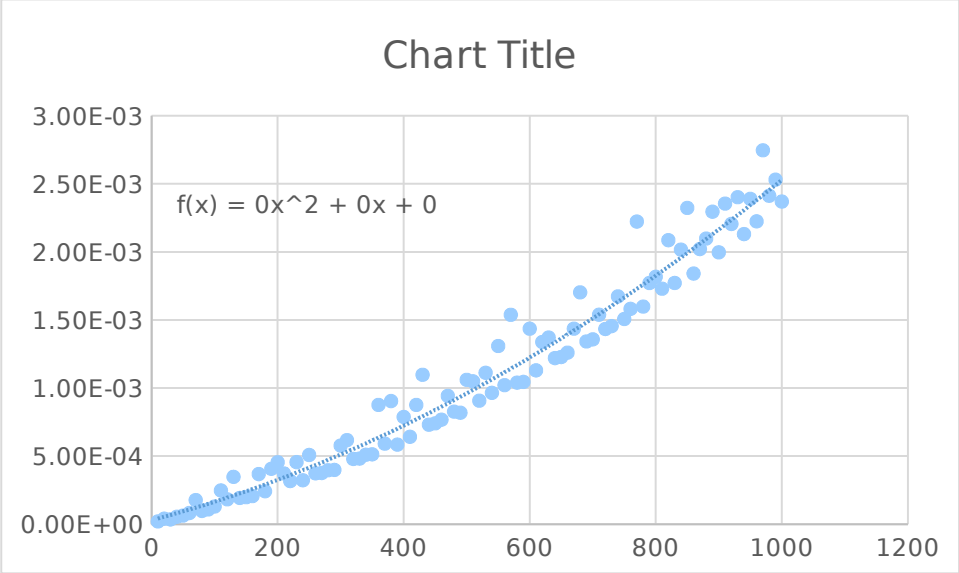


After looking at this data, it is a possibility that the curve of logarithmic can fit this data. Now let's look at more data with more input.

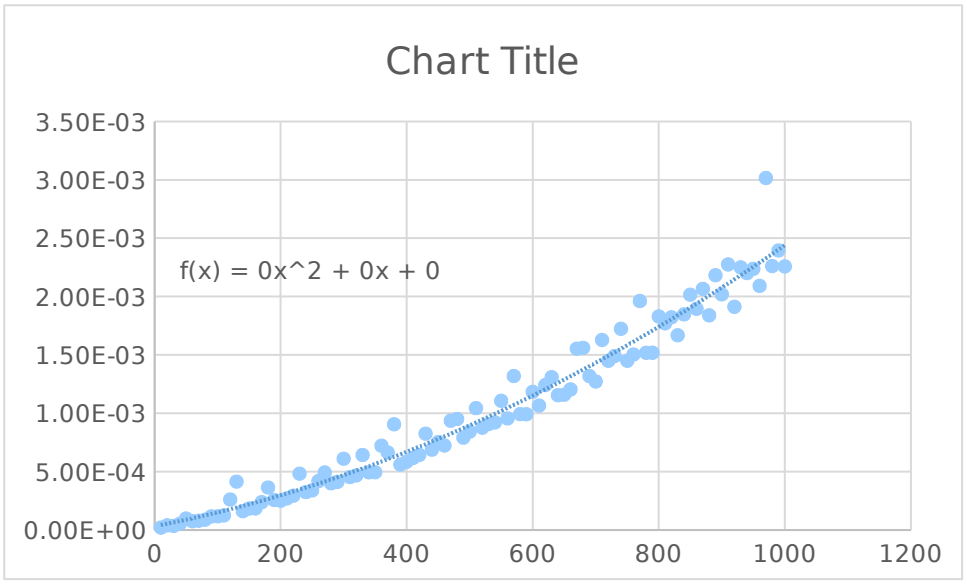


After looking at the data, a case cannot be proven the Knapsack has a time and complexity of Logarithmic.

Polynomial Trend Line:

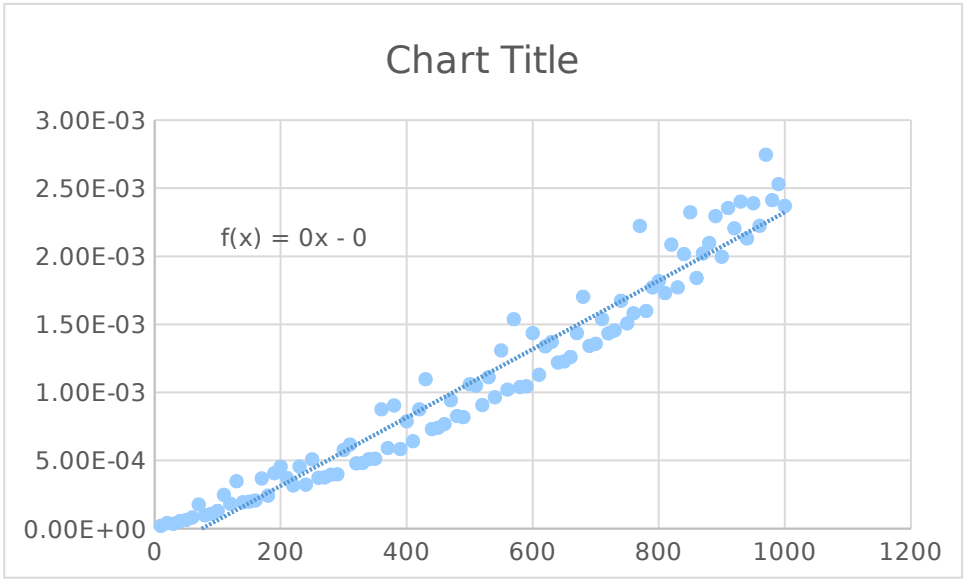


After looking at this data, it is a possibility that the curve of polynomial can fit this data. Let's look at more data.



There can be case for polynomial graph for the case of fitting the Knapsack.

Linear Trend Line:



After looking at this data, it is not a possibility that the curve of linear can fit this data. Therefore, it no more cases that linear can be the time and complexity of the knapsack.

How does the Knapsack work?

The knapsack problem is a where given a set of items, each with a weight and a value, determine the number of each item, so that the total weight is less than or equal to a given limit. The total value is as

large as possible. The problem comes into play when someone is faced by a fixed-size and needs to get the most items with into the knapsack with the largest value without going over.

Analysis

Look at the pseudocode, trying to prove that the Fractional knapsack has a time and complexity of O (n log n), the max heap contains a priority queue which removes the highest value in my case. This takes an average time of O (log n) based on the number of items, N, in the queue. Therefore, there are N times that it needs to go through, that becomes O (n). Then the time and complexity together is O (n log n). Figuring out the weight for each item in the queue, takes average of O (n) times based on it goes through the queue and finds the weight for each item. Overall, the best case for this algorithm for time and complexity can be O (n log n).

Conclusion:

The analysis results above are what we should have expected with the Knapsack Code. Therefore, I can conclude that the data input fails to accept my hypothesis by looking at the polynomial graph.

The data shows me that the graph is polynomial. Here are the equations that each time and complexity gave me:

Exponential: y = $.0001e^{.0034x}$
Power: y = 3e – 06x - .002
Logarithmic: y = .0007ln(x) - .009
Polynomial: 1e -09 x^2 1e – 06x – 3e-05
Linear: 3e-06x – 0.002

Originally I hypothesized that the logarithmic equation would be best fitting for the knapsack. However, after analyzing the data I concluded that the polynomial equation would best fit the graph as opposed to the logarithmic equation. Next I analyzed my theoretical analysis to determine where and what errors occurred. After looking at my code and stating the theoretical analysis is correct, I would look at how I tested the knapsack. After this I would run more tests with more data input, this set of data would contain larger ranges to get a better concept of more substantial values.

The standard method of analyzing the pseudocode of the fractional knapsack would have been to push items onto the heap after performing my calculations. Instead I chose to create a new method called “New Item” and have that push. My original testing took a long time to complete, in the future I would readjust how I test the knapsack with benchmark to shorten the computing time. One method of doing this involves eliminating unnecessary items when declaring a new item. If there are too many items declared in my for loop then the computing time increases; therefore I would create a statement that could regulate the capacity of the heap. In this method I would have a statement that would determine if the heap is full: if so the program would cease creating new items for this specific range.

The pseudocode of the knapsack has only four parameters, instead I passed in 5 parameters which could have had an effect on the testing. After the observing the pseudocode I didn’t see any pushing in my code of the Knapsack. Next time, I should use one class instead of the three classes to test the knapsack, this would make the testing much clearer to understand. Therefore, there is less code to understand and code corrections can be made more easily.

Overall, the testing and the implementation of the Knapsack appeared to be done correctly. Next time I use the knapsack I would implement the equation differently by using less classes; this would ensure that there is less overhead in the code. In addition, this would provide a different way to test the code using the benchmark method.