Fractional Knapsack Growth Rate Investigation

CS4310 - Design and Analysis of Algorithms

Mariia Kravtsova

4/9/2017

Hypothesis:

The Fractional Greedy Knapsack Algorithm runs in O(n log(n)) time.

Test Design:

I have taken the following steps to test the hypothesis:

1. Test the built in sorting function of Ruby with 10,000 - 200,000 range in 10,000 increments
2. Implement Fractional Knapsack following the pseudo code on page 287.
3. Run the code on multiple sizes of the array and random sum W.
    1. Test 1: range 10,000 - 300,000 in 10,000 increments
    2. Test 2: range 1,000,000 - 10,000,000 in increments of 1,000,000
    3. Graph and analyze test 1 and 2
    4. Run each trial 10 times and average the time
4. Analyze each graph and evaluate the trend lines, evaluate the hypothesis
5. Verify growth using mathematical analysis

Evaluation of Data:

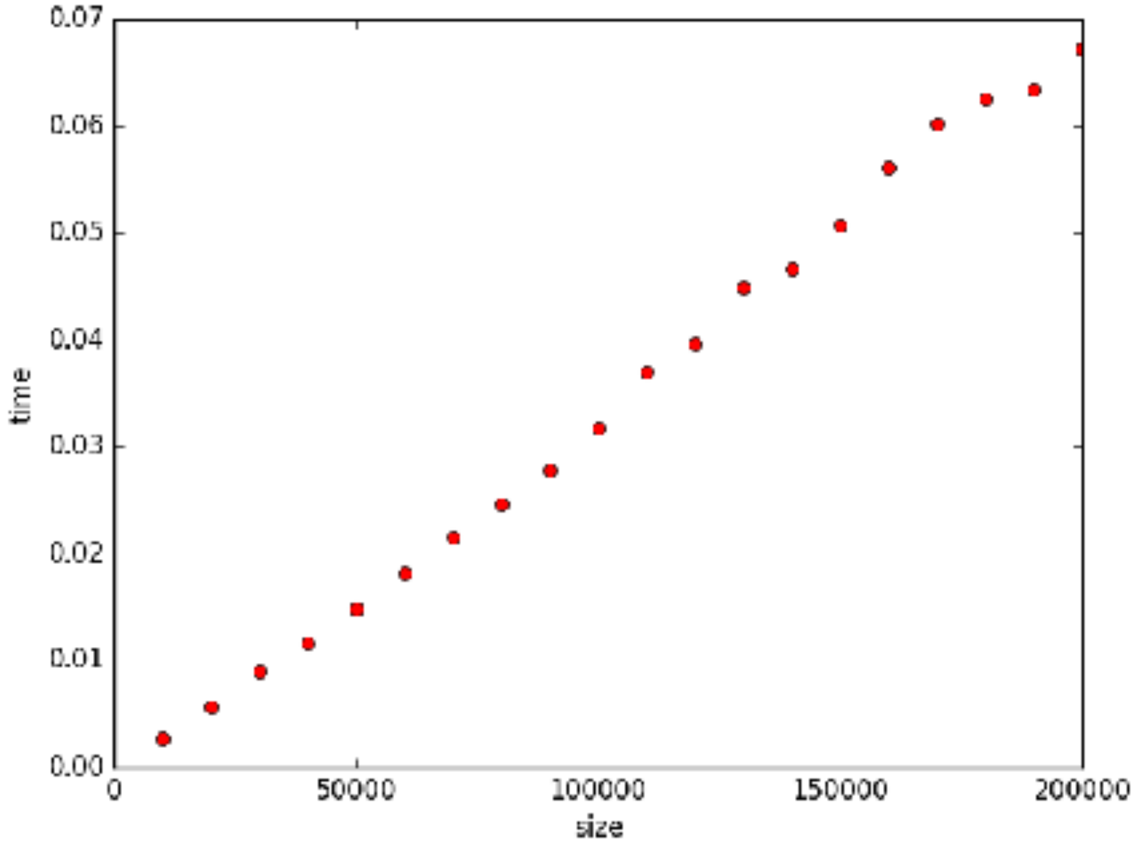
All the files that are referred in this analysis can be found in the submission folder.

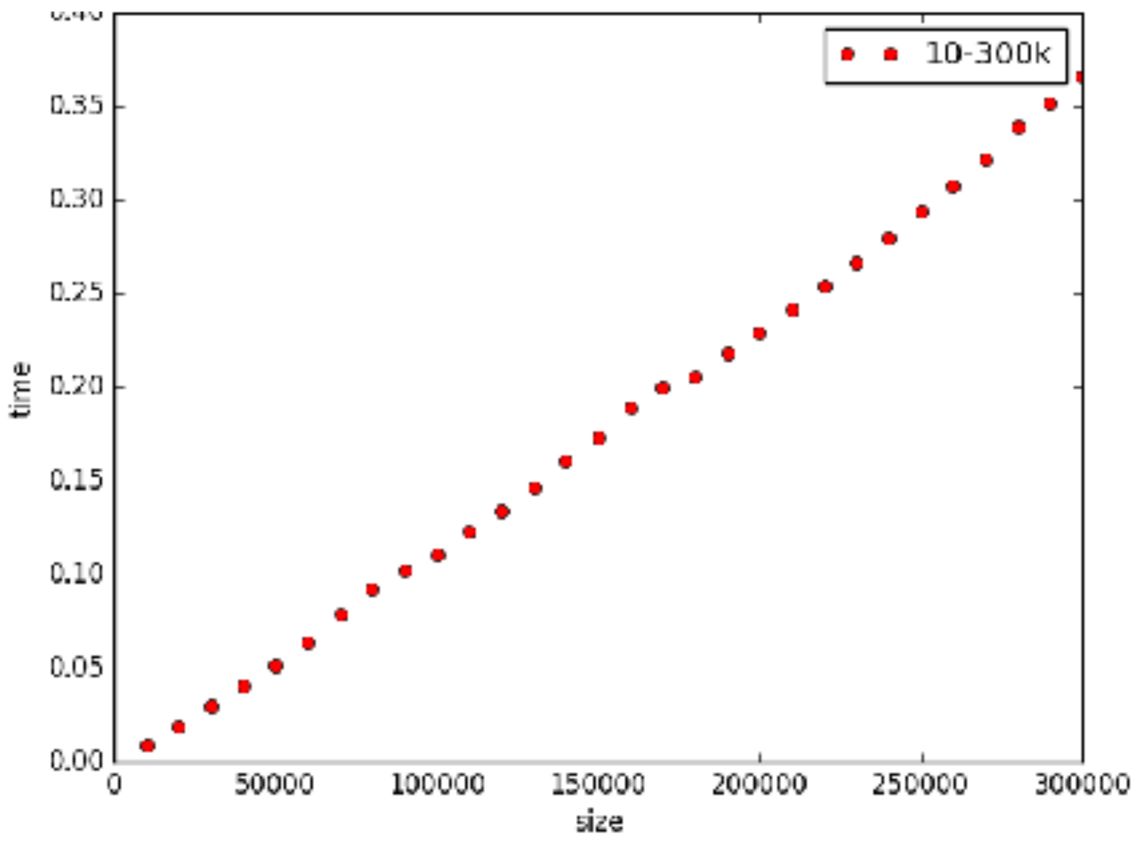
Item.rb has the implementation of the object that we create to store the benefit, weight, value, and final amount. Fractional.rb has the implementation of the actual fractional knapsack algorithm, and Fractional_test.rb has all the tests and benchmarking that have been used to get 60 data points to prove the hypothesis.

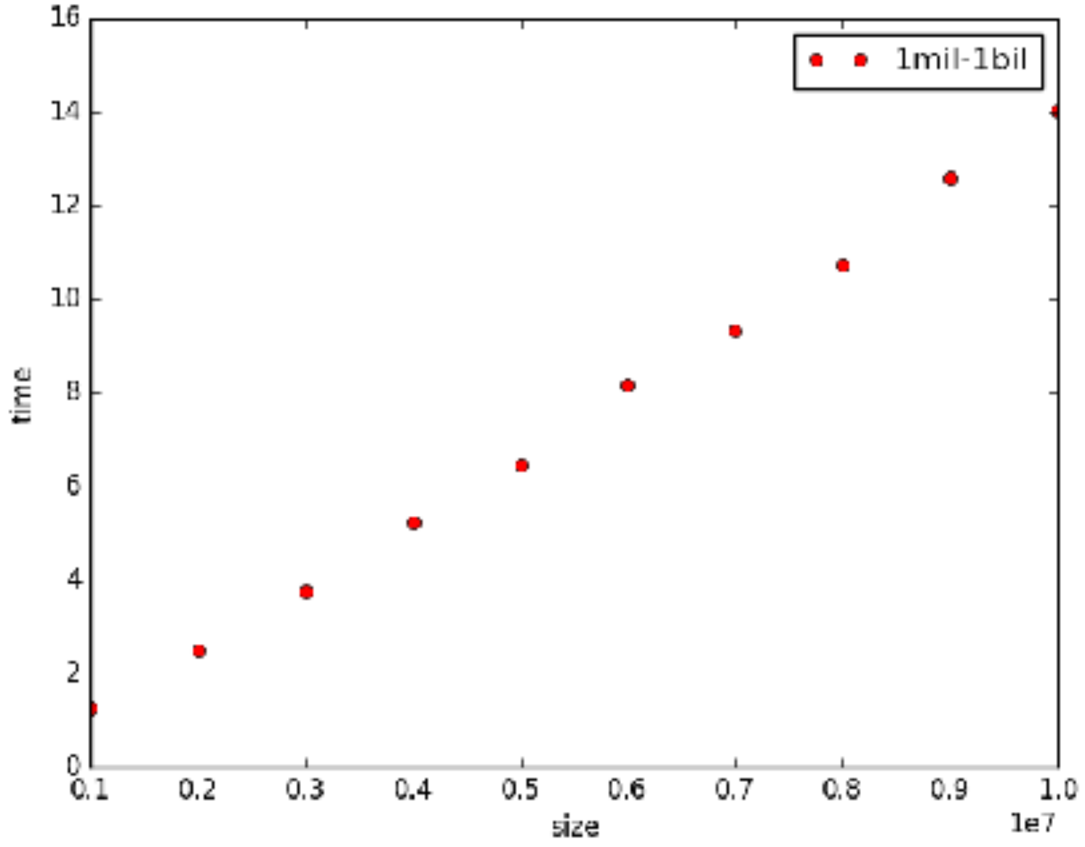
sort.csv contains 20 data points for the size and the time that sort_by function runs in to sort an array. outputsort-30.csv has 30 entries of size and time which is the result of the first test, and outputsort-1mil.csv has the result of the second test. For statistical analysis I used python's libraries - 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:



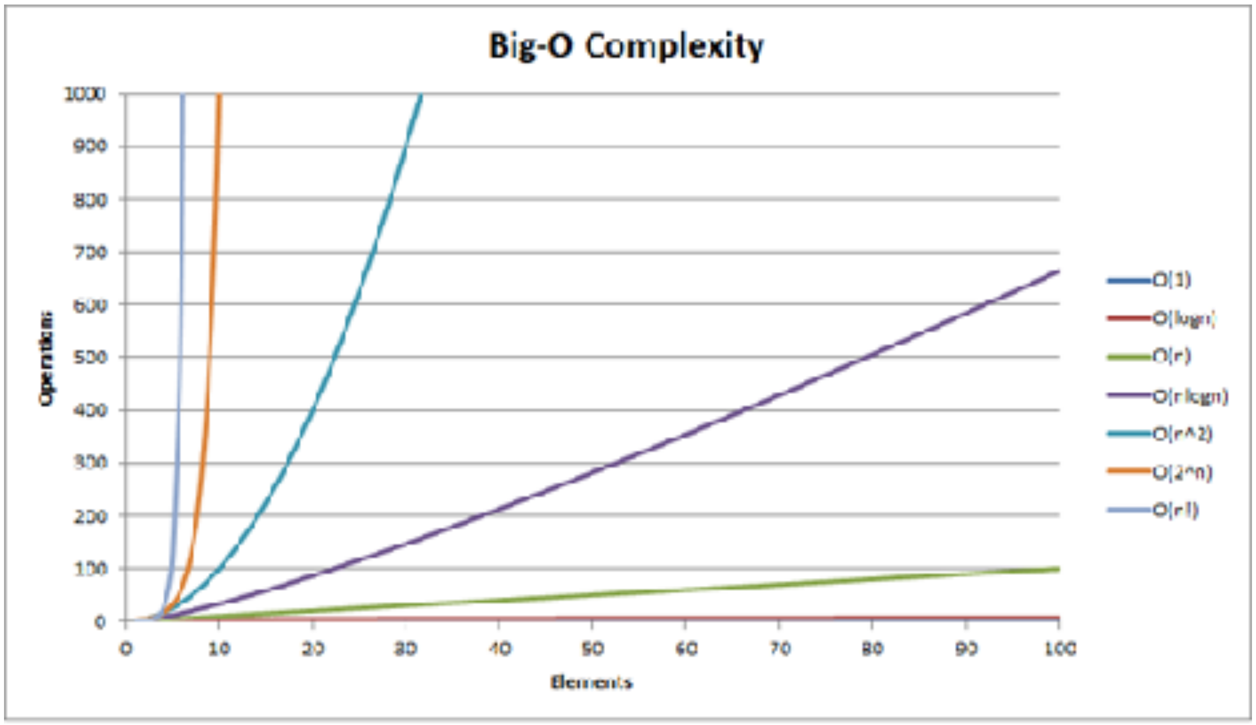




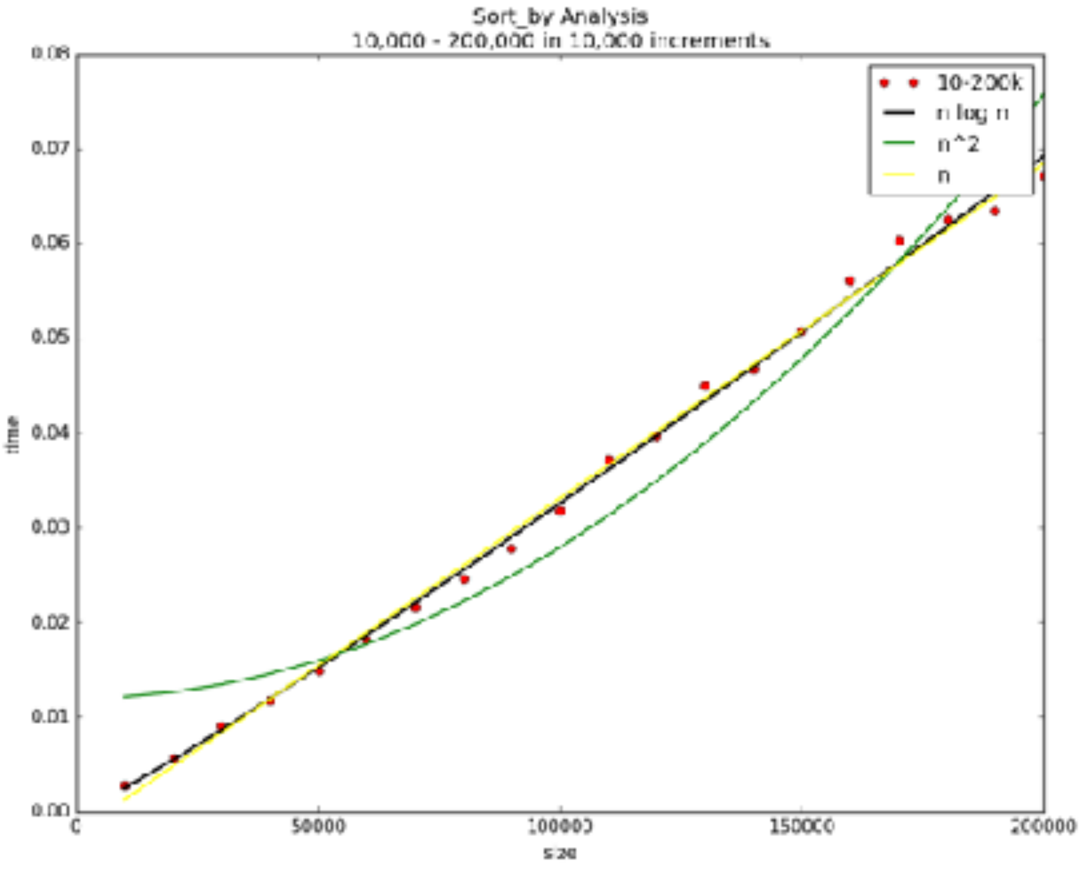
The first plot represents the result of the test to make sure that sort_by function is efficient for our purposes, which would mean that it runs in O(n log n) time. According to ruby documentation this function uses Schwartzian Transform to perform the sort.

The second plot is the result of the first test, and third plot is the result of the second test. The y axes represents time, and x is the size. One might note that on the plot of the second test the size is represented in decimals because the size numbers are fairly large and will make this plot hard to read.

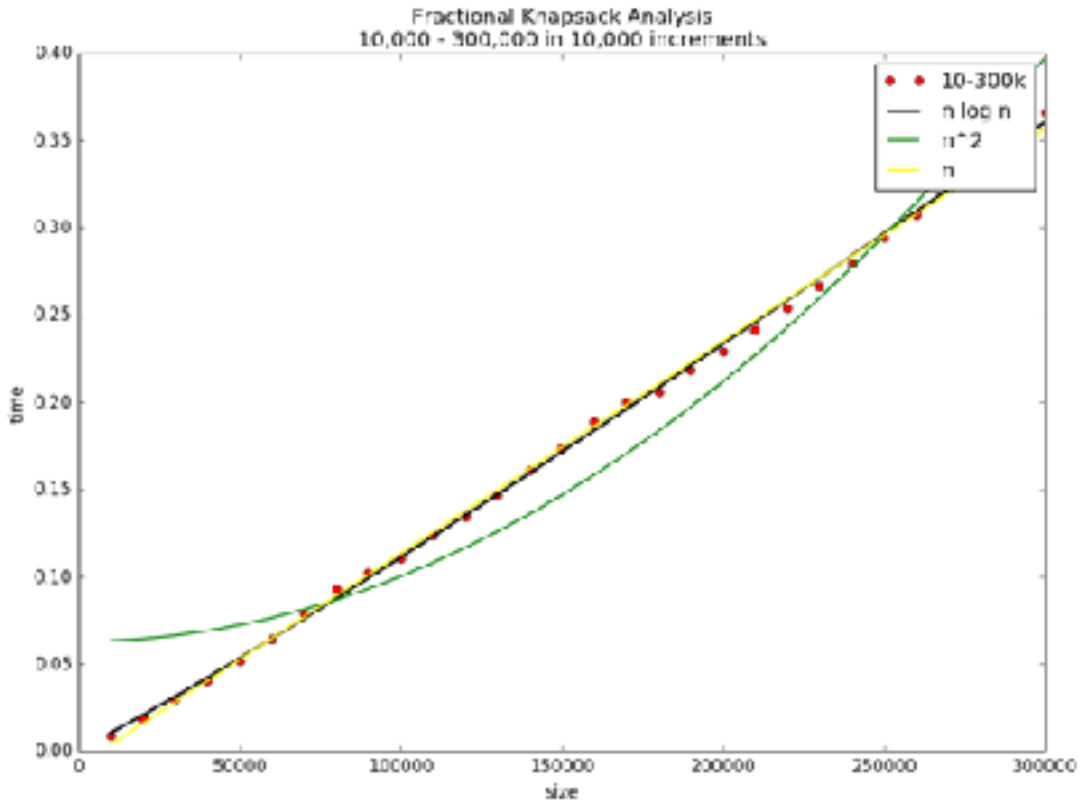
Just plotting the data points without zooming out on the graph or performing any statistical analysis we might make the assumption that our sort and knapsack runs in linear time. There are not a lot of curves here, and we can observe some bumps, but we must keep in mind that the O(n log(n)) function grows fairly slowly. For reference of function growth I am including Big O Complexity chart below, which I referenced from online source during my analysis.



Now that we have defined what the data points look like and what Big O functions look like I plotted it against our data, and run some statistical analysis for it.

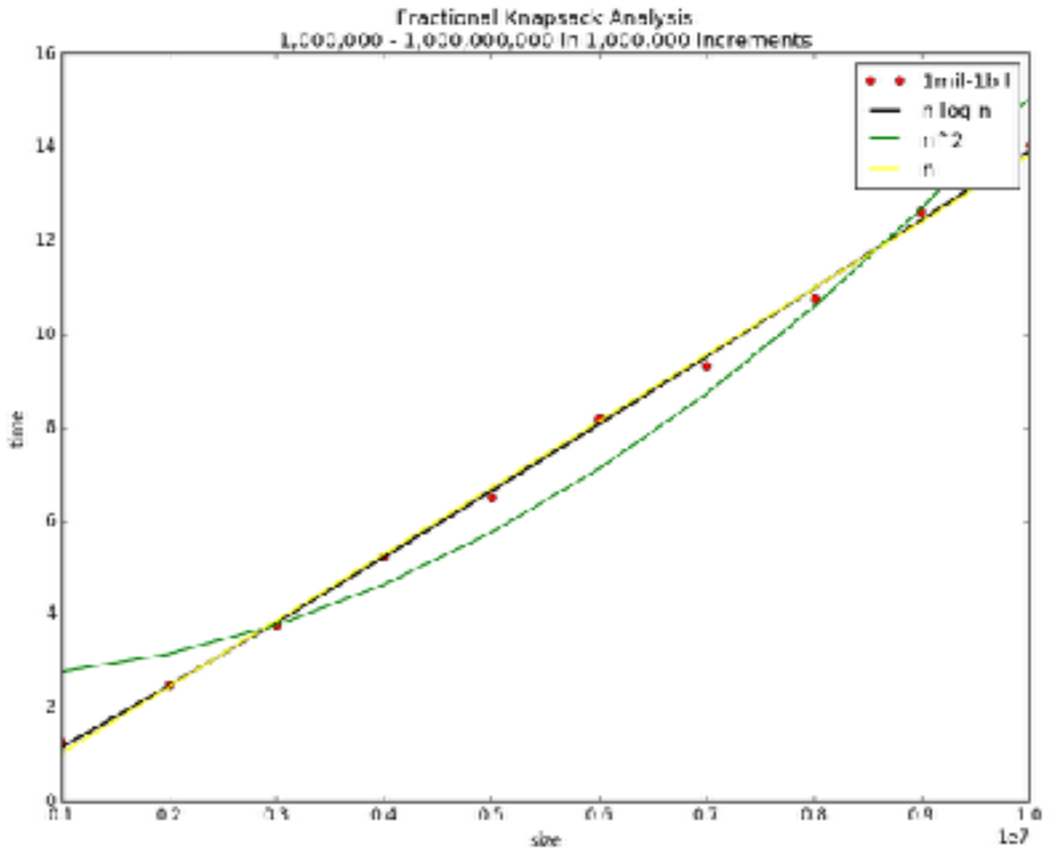


In the graph above I have the results of running the sort_by function plotted against n log n, n^2 and n trend lines. The formula of sort data is y = 3.537e-07*x - 0.002313. Even though it looks like the data fits both O(n) and O(n log n), the r squared, which is a measure of how close the data fits the specific growth line, is 0.99 for O(n log n) line, vs the linear fit of 0.86 and polynomial of 0.32. Since score of 1 lets us prove the hypothesis that the sort runs in O(n log n) time I can now proceed safely using this sort in my fractional knapsack without slowing it down.



In the plot above I did similar analysis as the sort on the Fractional knapsack test 1. The equation in is y = 1.215e-06 * x - 0.008199, and the r squared with the O(n log n) trend line is 0.99, and 0.876 for the linear, and 0.3 for the O(n^2).

Since I want to observe the trend of the line as the input becomes large the test two has less data points to save on running time, but we still get a good idea when we perform statistical analysis and plot the data, as shown in the graph below. The equation is y = 1.422e-06 * x - 0.4047, and r squared value for the O(n log n) is 0.99 with the error of 0.095, I think it is safe to say that it will follow this trend.



Conclusion:

Through steps of plotting the data, running the statistical analysis, and comparing the data point with O(n log n), O(n^2) and O(n) I have given a fair chance for this algorithm to prove or disprove the hypothesis. This a reasonable comparison of the functions that could have possibly worked with this algorithm. Of course we could have compare it with logarithmic function and so forth, but I picked the higher and lower growth functions, and I think it paid off to see how it run.

The analysis and the results that I have presented above support the hypothesis that the fractional greedy knapsack algorithm runs in O(n log n) time.