

Lab 4.5: Evaluating Algorithms

Summary

In this lab, we analyze and compare different algorithms for solving the famous computer science Three Sum Problem and Traveling Salesman Problem(TSP). By doing so, we learn how to evaluate the performance of algorithms and how different approaches affect performances dramatically.

Procedures

1. Collect run-time results by using the Stopwatch method of algs4 for different programs for the same problem using different input sizes.
2. Use Google Sheet to organize, plot, best fit and analyze data.
3. Based on the data, predict run-times for larger data size.
4. Compare predicted with actual results.

Techniques

1. Average the multiple results collected for each size to reduce error.
2. Log-transform data from curves to lines to simplify the best fit process.

Notation

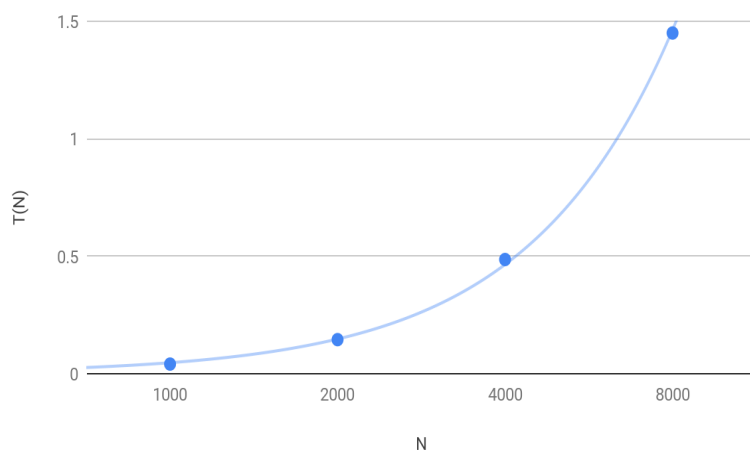
N : Problem size

$T(N)$: Runtime in seconds for problem size N

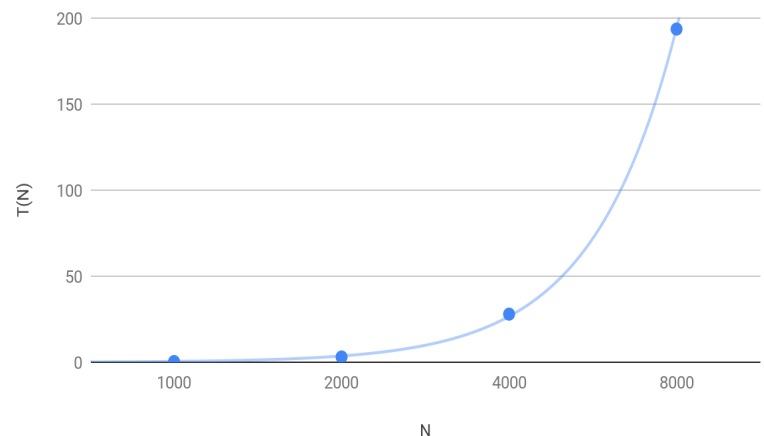
Data & Analysis

Through procedure 1. and 2., we get the plots of runtime $T(N)$ v.s problem size(N):

$T(N)$ vs. N for ThreeSumFast



$T(N)$ vs. N for TreeSum

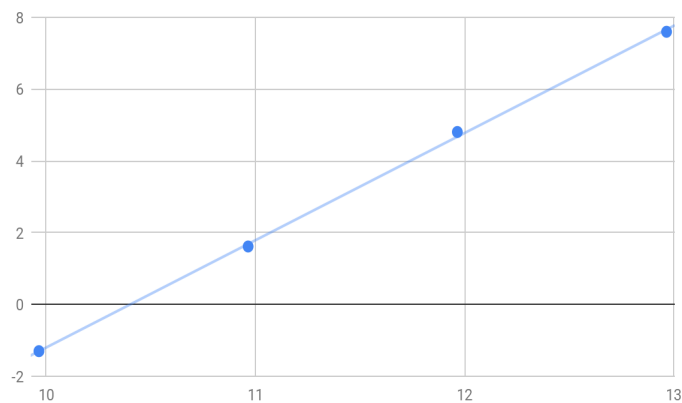


Viewing these two best fit lines for ThreeSum and ThreeSumFast, we see that they have a similar exponential trend. However, notice that their scale for y-axis is

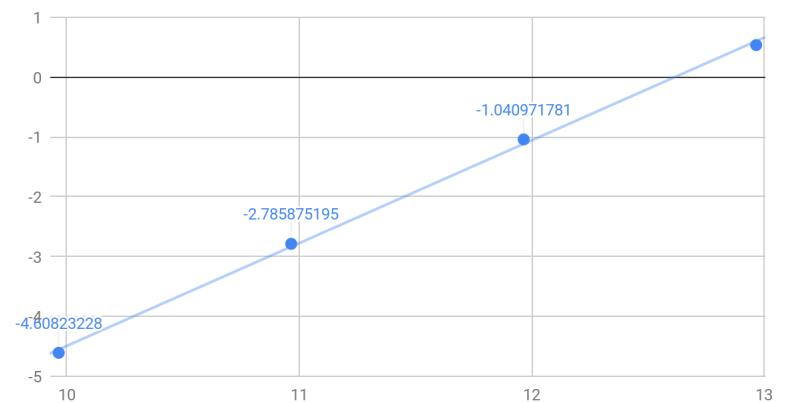
different—ThreeSum has a significantly larger one, which means that it runs much more slower than ThreeSum Fast.

To break down their differences in the relationship of runtime vs input size, we log transform them:

lg(T(N)) vs. lg(N) for ThreeSum



lg(T(N)) vs. lg(N) for ThreeSumFast

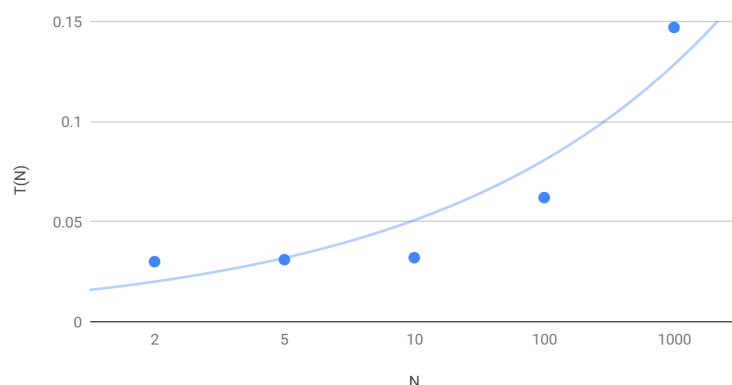


We best fit them log-transformed lines with the function: $lg(T(N)) = a * lg(N) + b$, use the a and b values to deduce the original runtime vs problem size function, and predict the runtime for the problem size of 16K:

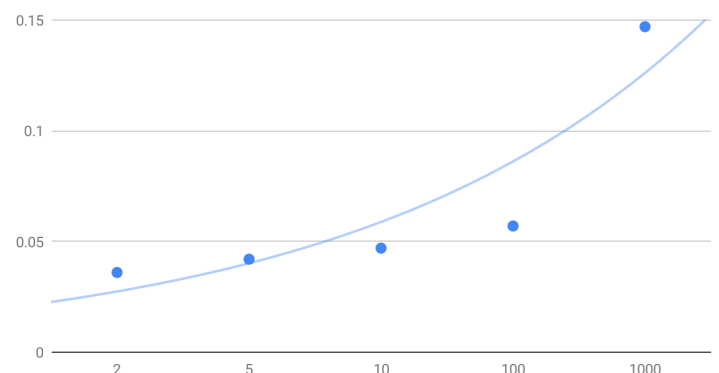
Algorithm	a (approx.)	b (approx.)	T(N) vs N Function	Predicted / Actual Runtime for N=16k	Error
ThreeSum	3.0	-31.1	$T(N) = 4 \times 10^{-10} N^{3.0}$	5.0 / 5.7	-14%
ThreeSumFast	1.7	-21.7	$T(N) = 3 \times 10^{-7} N^{1.7}$	1606.6 / 1499.6	-7%

Using similar methods, we get the plots of runtime T(N) v.s problem size(N) for algorithms for the TSP:

T(N) vs. N NearestInsertion



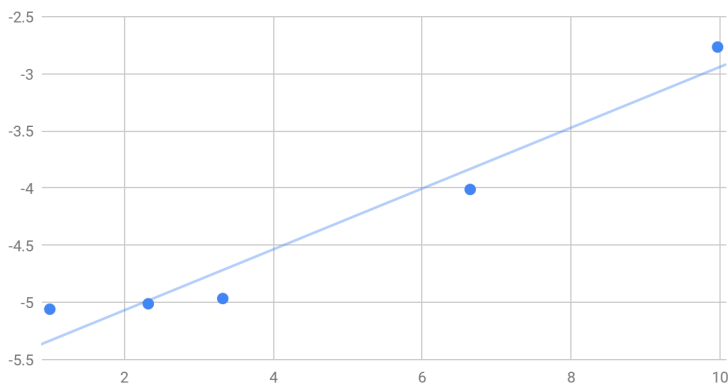
N vs. T(N) SmallestInsertion



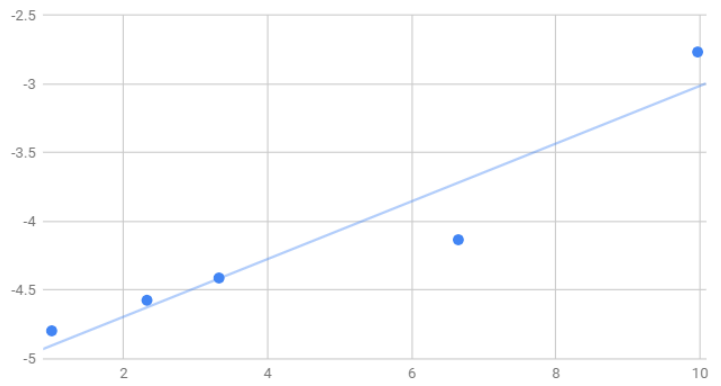
We see that NearestInsertion and SmallestInsertion graphs have almost the same exponential trend-line. This is reasonable since their algorithm procedures both loop through the tour once every insertion. Even though their data seems a little off from their trend lines, they actually follow an exponential trend. Their graphs look off because the y-axis is not labeled by scale.

Their log-transformed data:

lg(T(N)) vs. lg(N) NearestInsertion



lg(N) and lg(T(N)) SmallestInsertion




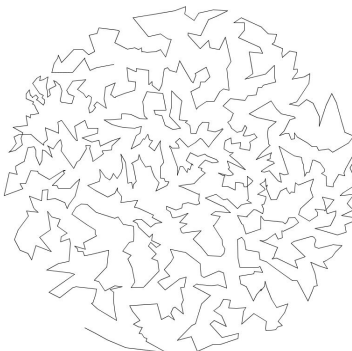
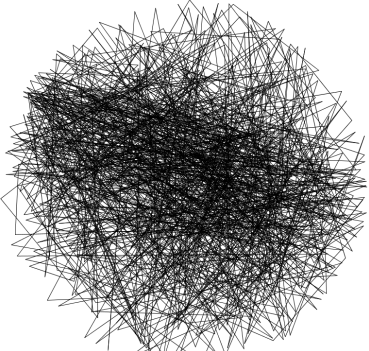
Best fitted and predicted result(also with OneByOne insertion, which is not analyzed without log-transform since the each insertion is dependent on the tour size, meaning that its run time should be $\sim aN$):

Algorithm	a (approx.)	b (approx.)	T(N) vs N Function	Predicted / Actual Runtime for N=16k	Predicted RunTime Error	Tour distance (for 1k.txt)
NearestInsertion	0.26	0.02	$T(N) = 0.02N^{0.26}$	0.42 / 43	-99.9%	27868
SmallestInsertion	0.21	0.03	$T(N) = 0.03N^{0.21}$	0.31 / 63	-99.9%	17265
OneByOne	N/A	N/A	$T(N) = 0.00009N^{0.03}$	7.5 / 1.5	400%	327693

Despite NearestInsertion and SmallestInsertion have significantly larger runtime for larger problems than OneByOne, they have significantly smaller tour distance. Minimalizing the tour distance is very important in the TSP problem, so unless the problem size is super large and the urgency is very high, we would always choose

SmallestInsertion and NearestInsertion over OneByOne. We would then conclude SmallestInsertion has the best performance among the three since it runs similarly fast as NearestInsertion's yet has smaller tour distance.

Finally, a brief look of the final TSP routes to grasp the physical sense of what these algorithms' performances are:

Input	NearestInsertion	SmallestInsertion	OneByOne
tsp1000.txt			
tsp100.txt	