Michael Edelman

Professor Avraham Leff

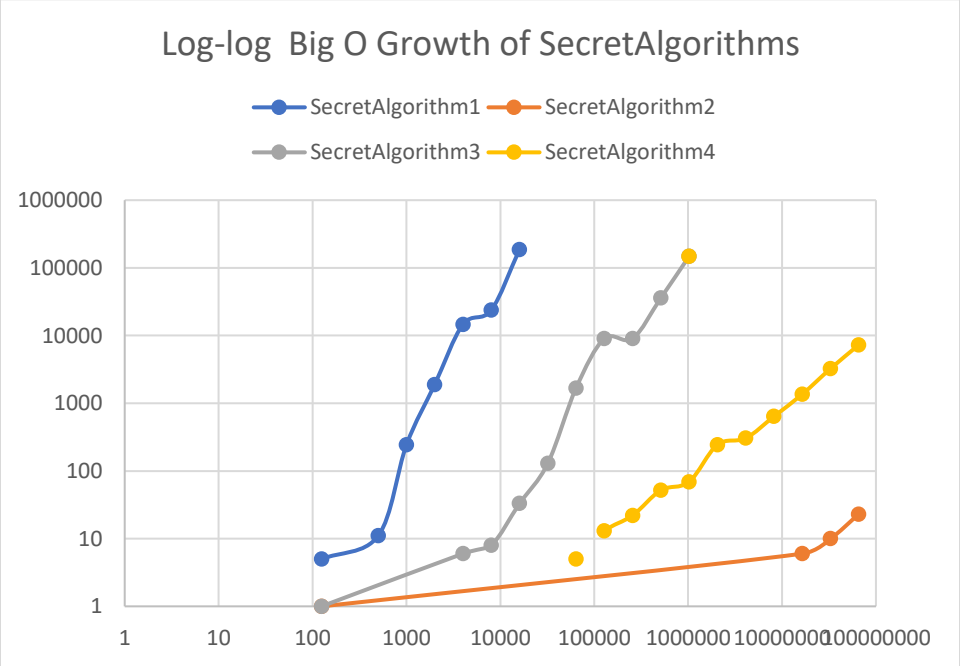
Estimate Secret Algorithms Assignment

To analyze 4 secret algorithms, I wrote a computer program that executed each algorithm repeatedly and with every repetition I doubled the input size. I kept track of each algorithms execution time by noting System.timeinmillis() before and after the execution, and then finding their difference. I calculated the ratio of execution time over the previous execution time to see the factor of difference. for every doubling of the input size. I also polymorphically processed all of the secret algorithms at once by instantiating one of each, putting them in an array and then iterating through each with an enhanced for loop. Within that enhanced for loop is the code to generate and print out experimental data for a BigOMeasurable algorithm, which is similar to Sedgewick's. I also did two tweaks to the data as the loops iterated to ensure A) The algorithm wasn't given an input size that would cause it to be running for more than 200 seconds for each execution and B) The program would only print out data results if the execution time was greater than 5 seconds. I did A) to ensure the testing did not take unnecessarily long and B) to ignore all the algorithm executions that could be small enough input where Big O trends would be impossible to see or unreliable.

Let *F*(n) = T be a function where n is input size and T is execution time.

Algorithm	Big O Classification
SecretAlgorithm1 was determined to be a cubic algorithm due to the time ratios being ~8 more than half the time. We know this is cubic because when you double the base value of a cubic function the output is 8X the value. (2n)^3/(n^3) -> 8(n^3)/(n^3) = 8. Two of the other values did not display a 8 ratio at all. One much more, one much less. I'm assuming the value that was a greater ratio than 8 was due to something additional be performed that generally doesn't occur(such as resizing an array). As for the lesser than 8 ratio this might be due to the properties of the inputs generated by the algorithm, bringing about a faster execution time.	Cubic (n^3)
SecretAlgorithm2 was an odd one because there was no foreseeable affect of the execution time when doubling the input until very high values. This where my code tweak(of getting rid of executions that take less than 5 seconds) comes in handy. It left us with only three useable data	Linear(n)

points. But those showed a clear execution time ratio of ~2 (the average is 1.99!). This is characteristic of a linear growth which is directly proportional to input size so when you double the input size the execution time is doubled as well. (2n)/n = 2.	
SecretAlgorithm3 gave us 10 data points of which the average was 4.75. the average was raised by an outlier of 12.95 other wise the average would be 3.73. The closest trend observed here is Quadratic which would quadruple the output for every doubling of the input. (2n)^2/n^2 -> 4(n^2)/(n^2) = 4. I would postulate the reasons for the outliers in this data set to be the same as mentioned for Algorithm1(nature of input and additional implementation detail.	Quadratic(n^2)
SecretAlgorithm4 gave us ample data points and practically all of them were around 2. As expressed for algorithm2 this would be linear growth.	Linear(n)



Big O Analysis for: SecretAlgorithm1

Input Size: 125 Running Time: 5.00 (ms) Doubling Ratio: Infinity

Input Size: 500 Running Time: 11.00 (ms) Doubling Ratio: 5.50

Input Size: 1000 Running Time: 242.00 (ms) Doubling Ratio: 22.00

Input Size: 2000 Running Time: 1888.00 (ms) Doubling Ratio: 7.80

Input Size: 4000 Running Time: 14595.00 (ms) Doubling Ratio: 7.73

Input Size: 8000 Running Time: 23781.00 (ms) Doubling Ratio: 1.63

Input Size: 16000 Running Time: 186387.00 (ms) Doubling Ratio: 7.84

Big O Analysis for: SecretAlgorithm2

Input Size: 125 Running Time: 1.00 (ms) Doubling Ratio: Infinity

Input Size: 16384000 Running Time: 6.00 (ms) Doubling Ratio: 2.00

Input Size: 32768000 Running Time: 10.00 (ms) Doubling Ratio: 1.67

Input Size: 65536000 Running Time: 23.00 (ms) Doubling Ratio: 2.30

Big O Analysis for: SecretAlgorithm3

Input Size: 125 Running Time: 1.00 (ms) Doubling Ratio: Infinity

Input Size: 4000 Running Time: 6.00 (ms) Doubling Ratio: 6.00

Input Size: 8000 Running Time: 8.00 (ms) Doubling Ratio: 1.33

Input Size: 16000 Running Time: 33.00 (ms) Doubling Ratio: 4.13

Input Size: 32000 Running Time: 129.00 (ms) Doubling Ratio: 3.91

Input Size: 64000 Running Time: 1671.00 (ms) Doubling Ratio: 12.95

Input Size: 128000 Running Time: 9023.00 (ms) Doubling Ratio: 5.40

Input Size: 256000 Running Time: 9032.00 (ms) Doubling Ratio: 1.00

Input Size: 512000 Running Time: 35835.00 (ms) Doubling Ratio: 3.97

Input Size: 1024000 Running Time: 147883.00 (ms) Doubling Ratio: 4.13

Big O Analysis for: SecretAlgorithm4

Input Size: 125 Running Time: 0.00 (ms) Doubling Ratio: Infinity

Input Size: 64000 Running Time: 5.00 (ms) Doubling Ratio: 1.67

Input Size: 128000 Running Time: 13.00 (ms) Doubling Ratio: 2.60

Input Size: 256000 Running Time: 22.00 (ms) Doubling Ratio: 1.69

Input Size: 512000 Running Time: 52.00 (ms) Doubling Ratio: 2.36

Input Size: 1024000 Running Time: 69.00 (ms) Doubling Ratio: 1.33

Input Size: 2048000 Running Time: 244.00 (ms) Doubling Ratio: 3.54

Input Size: 4096000 Running Time: 304.00 (ms) Doubling Ratio: 1.25

Input Size: 8192000 Running Time: 639.00 (ms) Doubling Ratio: 2.10

Input Size: 16384000 Running Time: 1356.00 (ms) Doubling Ratio: 2.12

Input Size: 32768000 Running Time: 3241.00 (ms) Doubling Ratio: 2.39

Input Size: 65536000 Running Time: 7280.00 (ms) Doubling Ratio: 2.25