Rowland Sanders
016845112
CECS 328 Section 11
9/11/20

Programming Assignment 1: RunForYourLife

Freddy’s Algorithm

s=1	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Trial 9	Trial 10	Average	Units
n=1000	336.952	340.102	337.234	338.722	336.927	337.221	336.153	337.112	338.781	336.457	337.566	ms
n=2000	2.661	2.624	2.656	2.611	2.634	2.679	2.615	2.645	2.617	2.609	2.635	s
n=4000	22.513	22.512	22.567	22.524	22.531	22.511	22.545	22.551	22.526	22.531	22.531	s

Predictions: n=2000 : 2.701sec , n=4000 : 21.608sec,
Complexity: n^3
Prediction justification: I came to this prediction based on the average time if n inputs were doubled. We need to cube the amount and multiply it by the average time to get an accurate prediction. So by this logic, for everytime n is doubled we need to multiply the average by 8. So we multiply 337.556 ms by 8, and the same for the next average after to get the next value. Doing so gives us an accurate representation that was very close to our actual values.
Deviation: We did see some slight deviation, less that 5%, and this can be because of other factors like outside programs and the fact that I am running this via VM.

Sophie’s Algorithm

s=1	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Trial 9	Trial 10	Average	Units
n=10000	94.134	94.144	94.132	93.991	94.072	94.163	94.122	94.131	94.102	94.156	94.115	ms
n=20000	0.421	0.423	0.429	0.431	0.427	0.425	0.432	0.426	0.431	0.429	0.427	s
n=40000	1.681	1.701	1.695	1.691	1.694	1.689	1.698	1.709	1.693	1.690	1.694	s

Prediction: n =20000 : 0.421sec, n =40000 : 1.684sec,
Complexity: n^2
I found that the runtime complexity of Sophie's algorithm was n^2 so when we double n from 10,000 inputs to the next n value of 20,000 we also multiply our average from the previous trial by 4x and from the one before that we double it to obtain the theoretical average. So for example from n=200000 we get an average of 0.427, and to predict what we want we need to multiply our average by 4 to get our prediction for n=40000.
Deviation: There is not as much deviation in this program compared to freddy’s, and was found to be under 3% overall. This can be related to the compiler or to background applications.

Rowland Sanders
016845112
CECS 328 Section 11
9/11/20

Johnny's Algorithm

s=1	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Trial 9	Trial 10	Average	Units
n=1000000	96.922	97.332	96.512	97.562	98.323	96.124	95.343	97.221	96.231	95.871	96.744	ms
n=2000000	188.221	186.234	185.891	188.451	187.201	188.561	181.992	187.221	185.910	183.102	186.278	ms
n=50000000	5.076	5.129	5.221	4.991	5.012	5.192	5.294	5.124	5.081	5.009	5.113	s

Prediction: n=2000000: 161.243 ms, n=50000000 : 5.231secs
Complexity: n * log(n)
I had a more difficult time getting an accurate calculation for Johnny's but used the same method to formulate it as I did before. So when T(n) = K Value * n * log(n), where T(n) is the average used in n=1000000. We see a more accurate calculation.
Method: K Value * 1000000log1000000 = 96.74 → k = 161.243 ms
Deviation: I saw significantly more deviation in this algorithm compared to the other ones. It may be because I am using a virtual machine, But I think more likely that any other background computer computations, like my google chrome, really affect the outcome of this algorithm

Sally's Algorithm

s=1	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6	Trial 7	Trial 8	Trial 9	Trial 10	Average	Units
n=50000000	94.331	95.112	95.295	94.987	94.345	94.872	94.511	93.998	95.107	94.416	94.697	ms
n=100000000	187.231	187.691	188.623	188.191	188.142	187.432	188.275	188.312	188.144	187.671	187.97	ms
n=200000000	374.292	375.312	375.674	375.981	373.911	375.102	374.912	375.531	376.112	375.678	375.051	ms

Prediction: n=100000000: 189.394ms, n=200000000 : 378.788ms
Complexity: n
We can multiply the time it takes to run this program by 2 when n is doubled. So when we double that again or 4n, we can multiply it by a power of 4. Knowing this, when we double n from 50M, to 100M for the n value, we are also doubling the averages that are based on the previous trials. So to get our predictions we can just multiply the current average by 2 to get the value of 2n. For example, n=100000000: 189.394ms* 2 → 378.788ms for n=200000000
Deviation: There is also a slight deviation in this algorithm that can be pointed to the compiler and background applications, especially when making a calculation of this magnitude in such a short amount of time.

Rowland Sanders
016845112
CECS 328 Section 11
9/11/20

Overall, I think that Sally's algorithm is significantly more efficient than the other algorithms. Based on the theoretical run times and the actual trials that I conducted, I can confirm that Sally's algorithm is the most effective and I would say Freddy's algorithm is the worst because of its runtimes compared to the other algorithms. I didn't even reach into the nano seconds for some tests on freddy's because of its clunkiness. I would say that Sally algorithm has the best runtime complexity and the preferred program. I also had a few problems with runtimes originally because I was using an online compiler and the data wasn't very consistent. I found that sallys program ran about 21 times faster than Freddy's and these calculations are very significant for efficiency. Even a time and a half slower is a big deal if someone were to make a larger calculation, I would expect any deviations and runtime differences to have a huge effect on the overall performance of the task. This can also be seen in the complexity of the algorithms, with Freddys complexity of n^3 while Sally's was just n. This makes a huge difference and we can see this in overall performance.