CPSC 319 2020

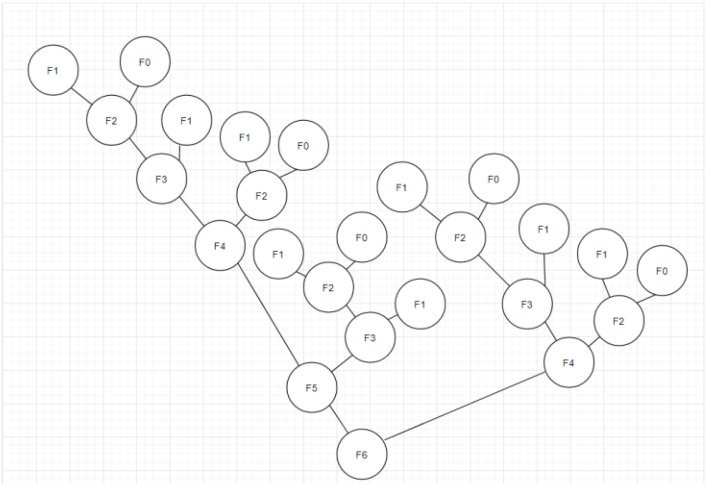
Garth Slaney

30061944

1.

a) Algorithm 1 performs redundant calculations as it only terminates at the base case.

b)

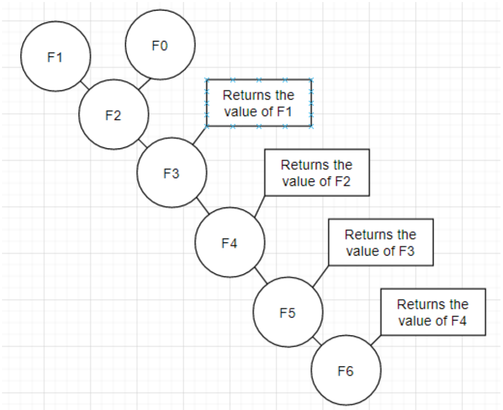


c) For F6 F5 is called once, F4 is called twice, F3 is called three times and F2 is called five times.

2.

a) Algorithm 2 doesn’t perform redundant calculations as it calculates each branch once and than returns the value of the branch.

b)



c) For F6 F5 is called once, F4 is called once, F3 is called once and F2 is called once.

3. Algorithm 2 is more cost efficient as it only calls each branch once and avoids redundant calculations while Algorithm 1 repeats calculations already made.

4. It is O(n) because graph 3 shows it being linearly proportional to n. Also, from the code for algorithm 3 the length of the array and the for loop are dependent on n and the linearity comes from the for loop.

From our code we can get the equation 5n + 4 where the 4 in front of n comes from making an array of n size, accessing two array elements, adding them and setting the value of the array.

5. It is O(n) because graph 4 shows it being linearly proportional to n. Also, from the code the linearity comes from the for loop as it is the only line dependent on n. From the code we can get the equation 3n + 6 where the 3 comes from setting the variables three times in the loop.

6. Algorithm 4 is more cost efficient as it doesn't require the creation on an array based on n and instead does all the work in the stack. We also see this as it has a lower constant in our equations.

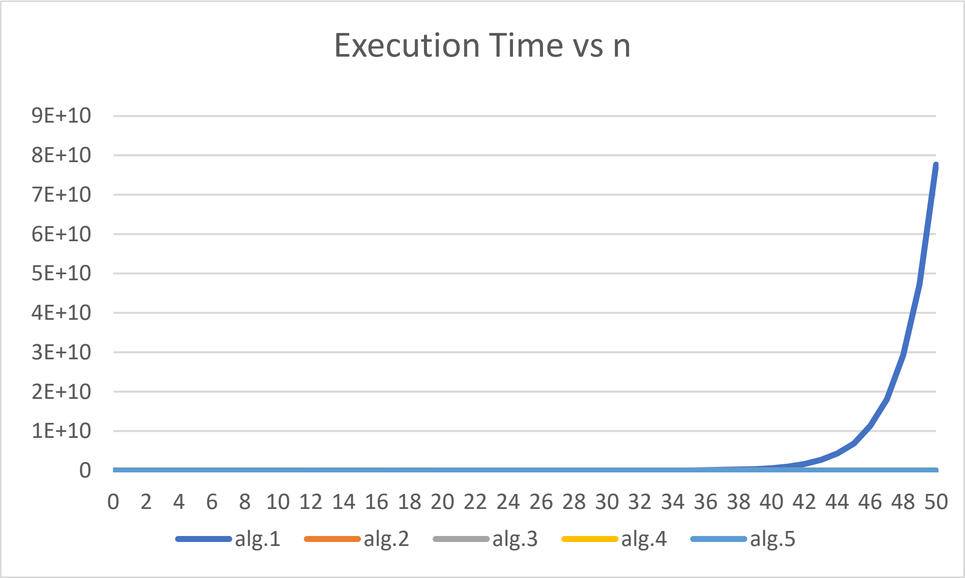
7. It is log(n) as n is halved every time. We can also see this relationship from graph 5 as running time increases as n increases making it not constant but it doesn't increase linearly. From the code we can get the algorithm

$$f(n) = \begin{cases} 1 & \text{if n} = 1 \\ C_1 + f(n/2) & \text{n} \geq 1 \end{cases}$$

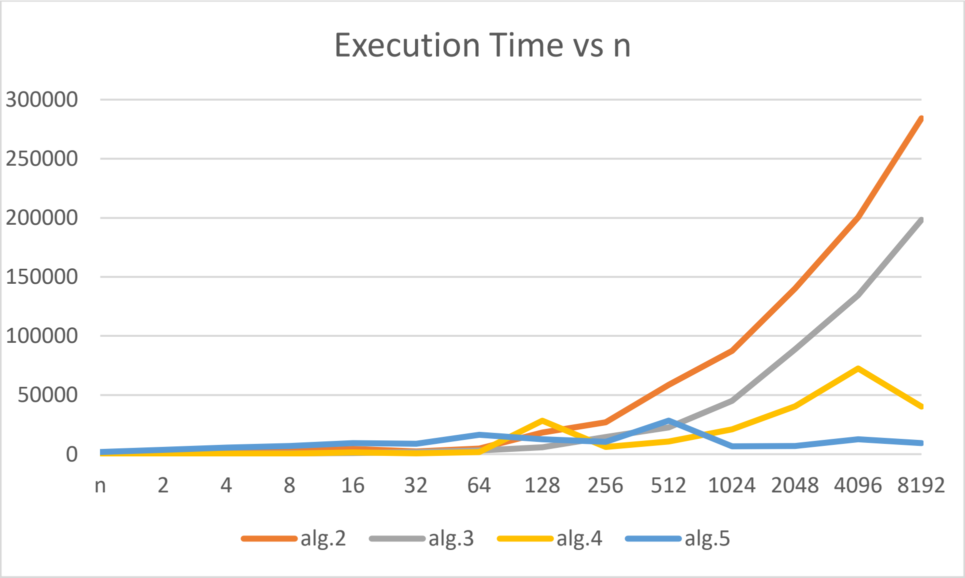
When we solve we get f(n) = 2k + f(n/2^k) for some integer k. When solving for our base case we have the final expression f(n) = c logn + d. C and d are unknown constants but as initially algorithm 5 has a high time we can assume that both are large and only when n increases greatly does it's logn become useful.

8. I would use algorithm 5 for large n values and algorithm 4 for smaller ones. Algorithm 5 is more efficient but has a larger start up time making it less efficient for calculations of smaller n. From graph 2 we can see that after 1024 algorithm 5 becomes more efficient in all case n greater than 1024. Therefore, if n is less than 1024 algorithm 4 is more efficient but otherwise algorithm 5 becomes more efficient. If we only did this calculation asymptotically algorithm 5 would be the most efficient.

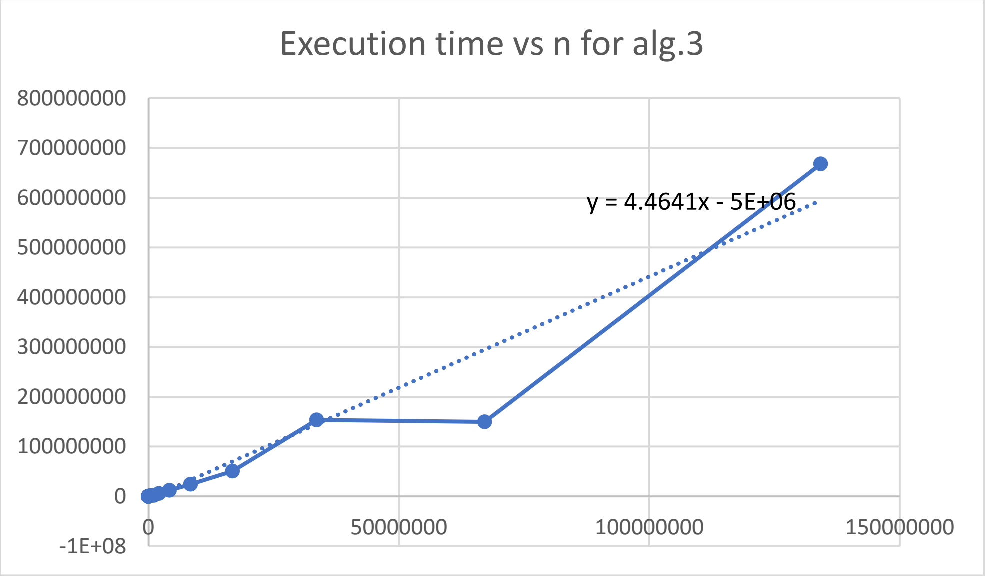
Graph 1



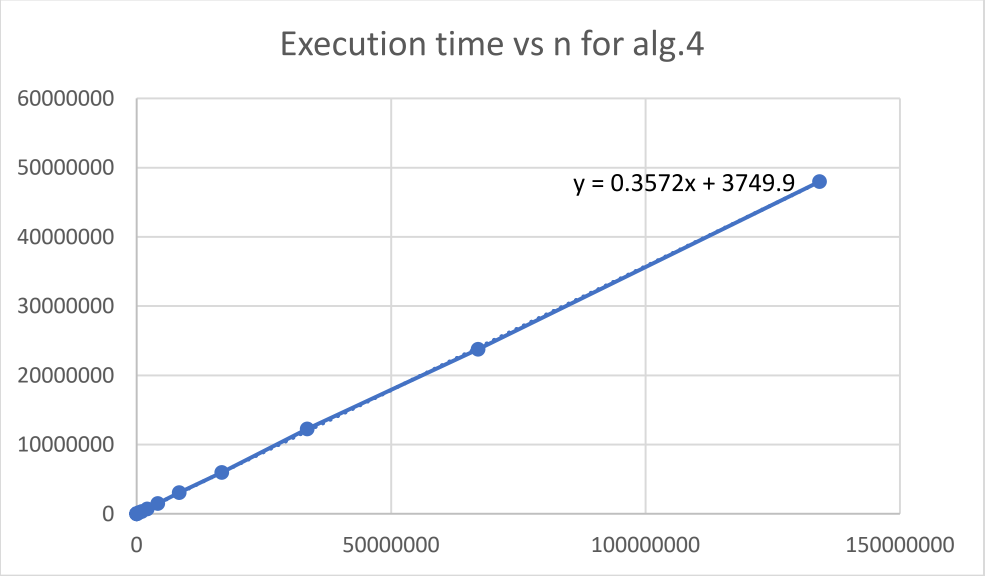
Graph 2



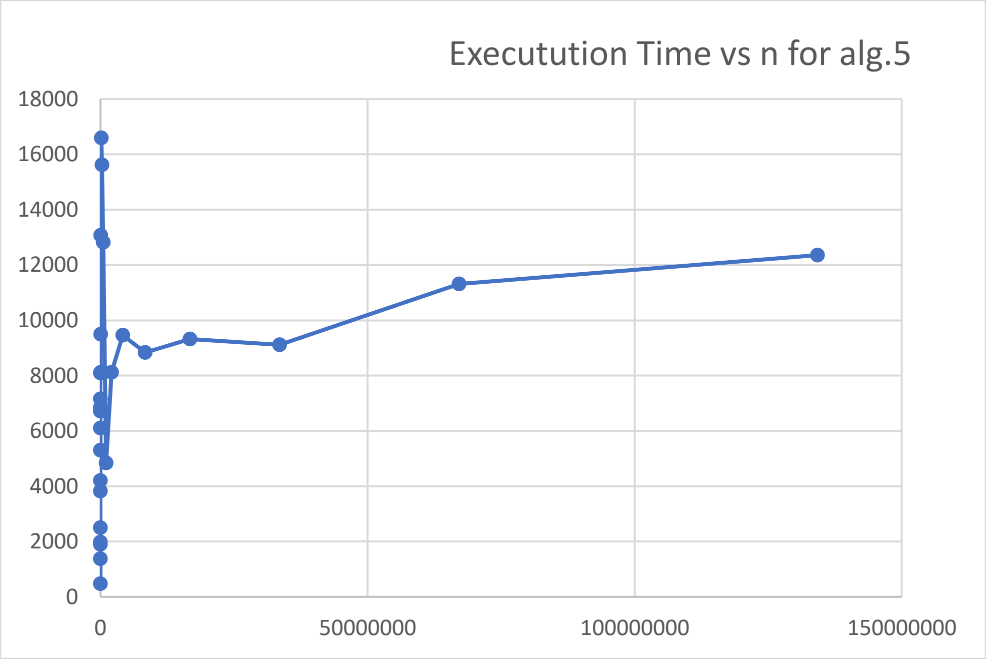
Graph 3



Graph 4



Graph 5


Executution Time vs n for alg.5
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
0
50000000
100000000
150000000