Project 4: Dynamic Programming

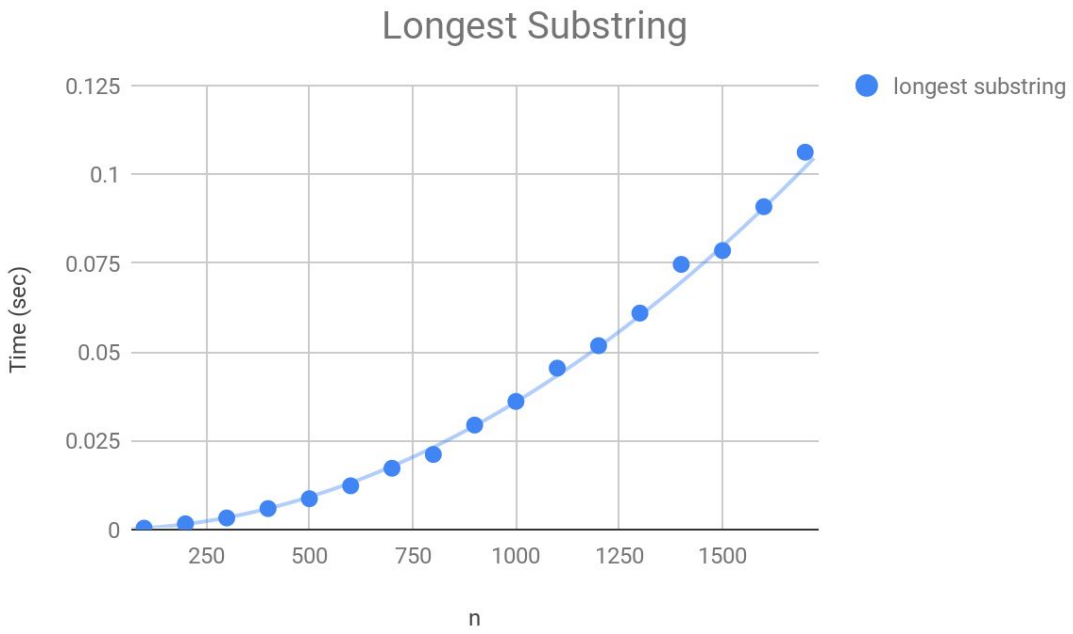
Team Members

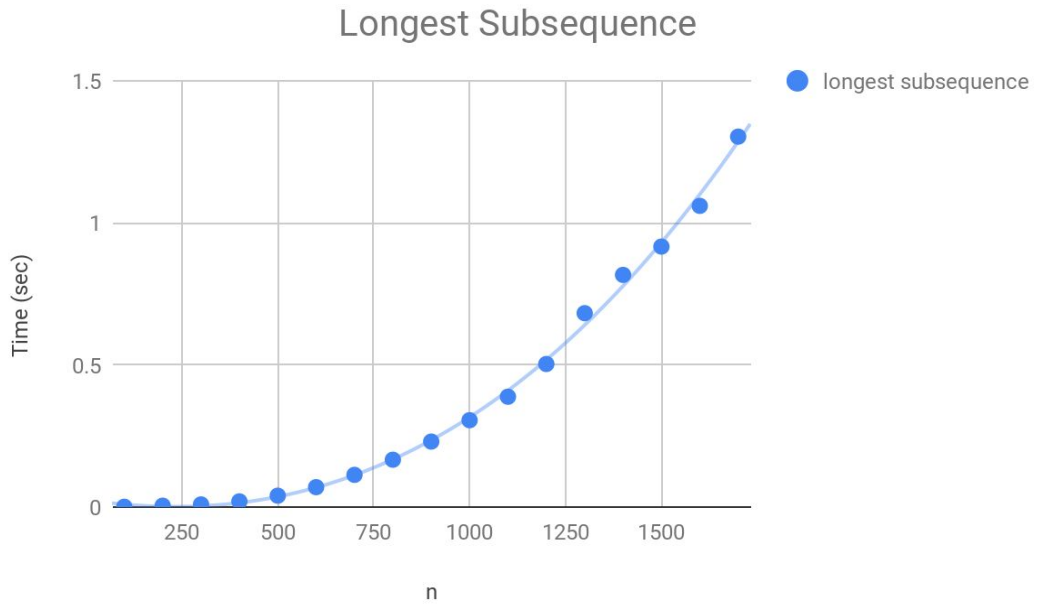
Jasper Chen jsprchn2@csu.fullerton.edu

Alex Ma alexjmma@csu.fullerton.edu

William Bui wqbui3@csu.fullerton.edu

Graphs





Questions

1. Consider the two algorithms you've implemented for the substring problem (the exhaustive search algorithm in project 3, and the dynamic programming algorithm in this project). How do their performance compare; is one significantly faster than the other, and which one; or are they roughly equivalent? If there is a speed difference, is it enough to make the difference between the algorithm being practical to use, and impractical?

   Comparing the results from Project 3 and 4, we can see that the dynamic programming version of the substring problem is significantly faster than the exhaustive search version. While the difference in speed is significant when comparing the two, the data gathered from Project 3 and 4 suggests both the exhaustive search and dynamic programming versions of longest_substring are very fast and practical to use. However, this may be due to the fact that in Project 3, very small values were used for *n* (1-15), while much larger values (up to 1700) were used in Project 4. Additional experiments will need to be performed in project 3 to investigate this suspicion. Or get dash from the incredibles to figure it out with all his math homework. Worth to watch it.

2. Answer the same questions, but for the subsequence problem.

   Comparing the data from Project 3 and 4, we see that the dynamic programming approach is much more efficient than the exhaustive search approach. In Project 3, longest_subsequence was already taking 16 seconds to run when *n* = 15. Compared with Project 4, where longest_subsequence took 1.3 seconds when *n* = 1700, we see that dynamic programming is much more practical to use than exhaustive search. Comparing the two versions of longest_subsequence was like comparing a clown on a unicycle with Darth Vader's personal flagship, the Executor.

3. Did you find implementing this algorithm to be easy, difficult, or in between? What was the hardest part, and why? How does the difficulty level compare to the exhaustive search algorithms from project 3?

   Implementing the algorithm was about in between, it wasn't really hard to figure out where to place some of the variables like "best" in the code. The part that we had some trouble in figuring out was the length of the 2D array since we kept making it one more than it should be, but studying the code for a bit we managed to figure out what we did wrong on the length. Overall this project was much more straightforward than the previous project. The hardest part was figuring how the pseudocode worked but once we got the ball rolling on that, it was smooth sailing from there. We felt that dynamic programming is much more comprehensible and straightforward than exhaustive search.