Project 4: dynamic programming

Group members: Brad Dodds bradleydodds@csu.fullerton.edu

CWID: 889763546

Introduction

In this project I have implemented and analyzed two dynamic programming algorithms similar to project 3. In project 3 the time complexity for the longest common subsequence problem was O(2^n * n). Now I have implemented an algorithm solving the same problem but with a different approach with a time complexity of O(n^3).

Problem1: longest common substring

input: a string a of length m and a string b of length n
output: the longest string s such that s is a substring of both a and b; in the case of ties, use the substring that appears first in a

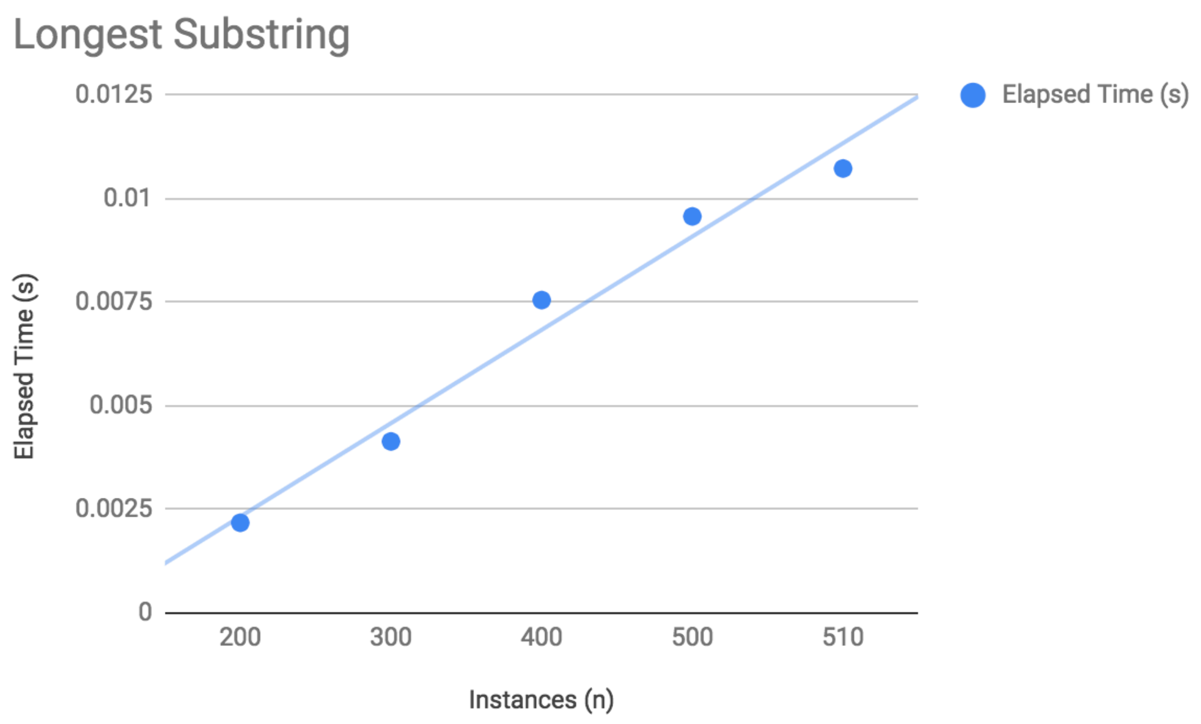
Problem2: longest common subsequence

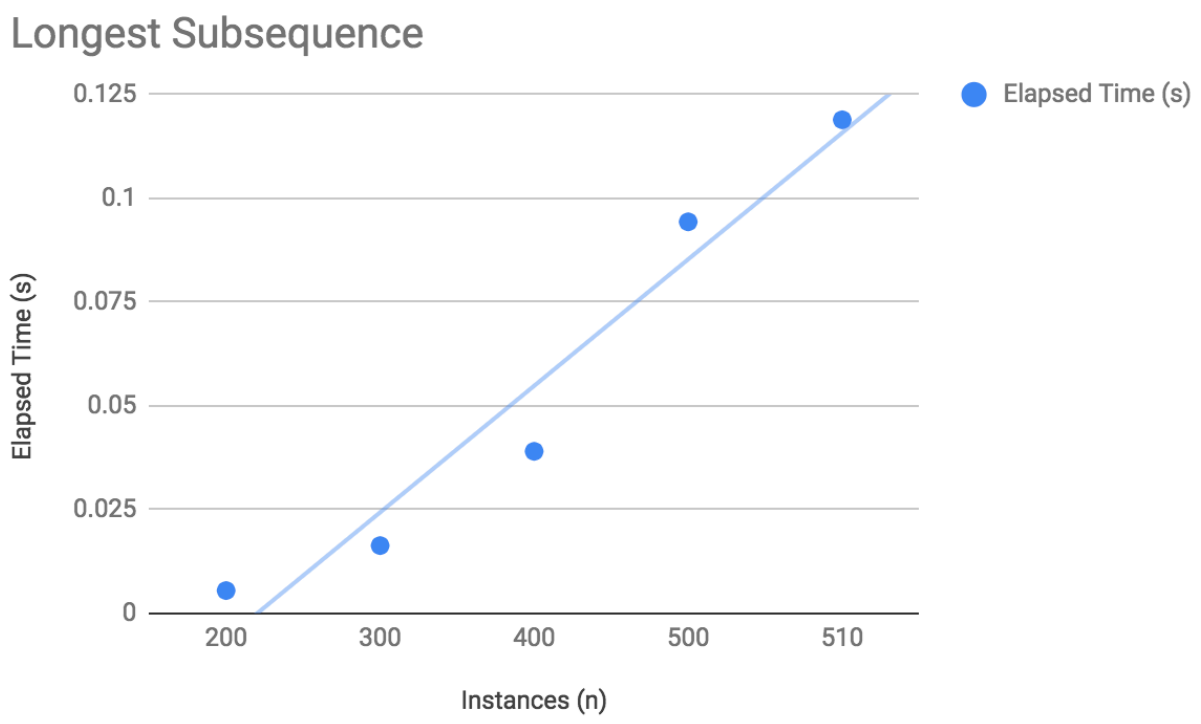
input: a string a of length m and a string b of length n
output: the longest string s such that s is a subsequence of both a and b; in the case of ties, use the substring that appears first in a

Empirical vs. Mathematical

After finalizing the project, I can conclude that empirically observed time efficiency data is consistent with the mathematically derived big- O efficiency classes. The longest subsequence problem executed as expected, much faster than in project 3. Analyzing the time complexities through pseudocode and analyze the time it took to execute justifies the comparison of approaches.

Performance





Conclusion:

a.) Substring Problem Comparison to Project 3

The performance the substring algorithm is faster than project 3's implemented algorithm. I kept the input roughly the same and noticed a slight difference on the smaller inputs of n but a more noticeable difference on larger instances of n.

For example:

Project 3: instance(n) = 500 executed at 0.0891387 (s)

Project 4: instance(n) = 500 executed at 0.00957563 (s)

b.) Subsequence Problem Comparison to Project 3

The performance the subsequence is substantially faster than the algorithm implemented in project 3. The inputs of n were much larger in this project than they were in project 3 and executed much faster. The execution time had a drastic change.

For example:

Project 3: instance(n) = 25 executed at 31.2 (s)

Project 4: instance(n) = 500 executed at 0.118926 (s)

c.) Algorithm implementation

I found implementing the subsequence algorithm to be a little bit easier than project 3 because it was much simpler to implement with the given pseudocode. The hardest part of this project was comparing the rows and columns of the subsequences. I had some difficulty with the 2D arrays being out of bounds. When comparing the difficulty to project 3's exhaustive search algorithms it was much easier because in project 3 I had to create more supporting functions.