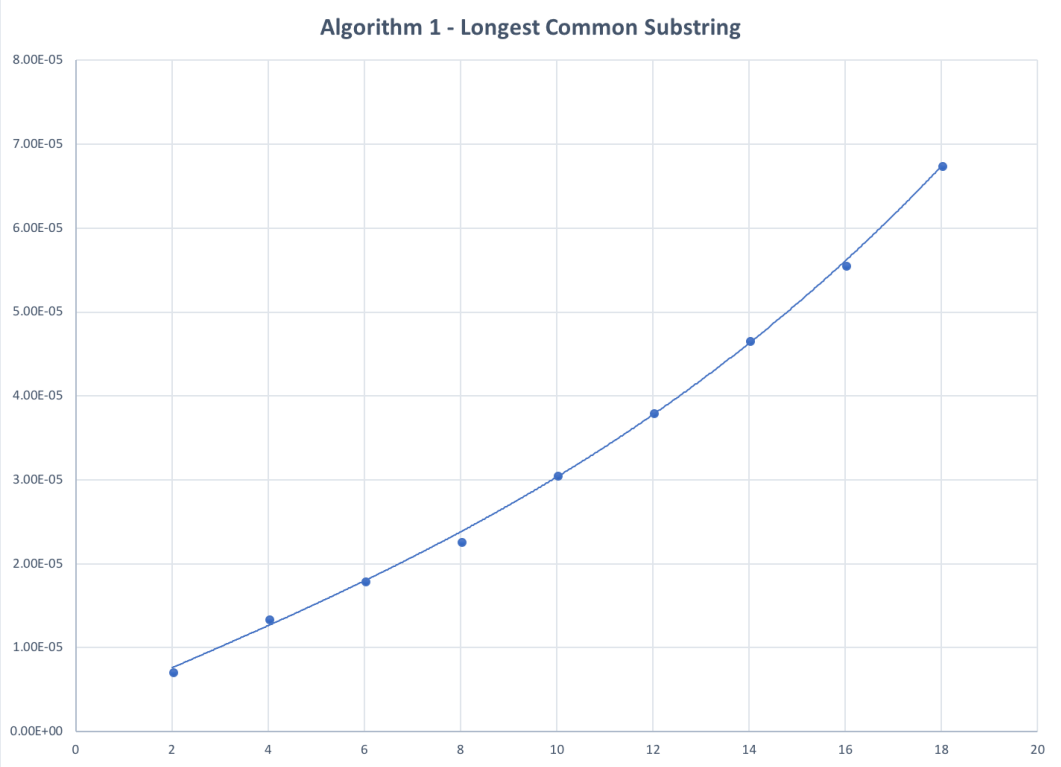
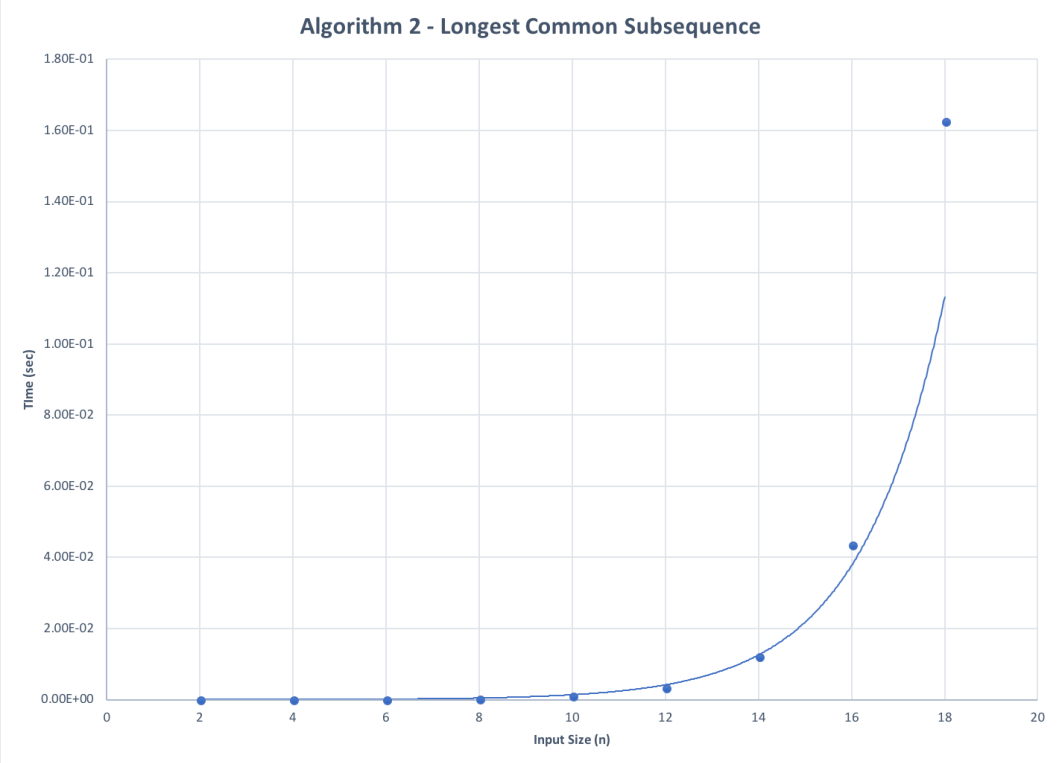
CPSC 335

Project 3: Polynomial versus Exponential Time

Devontae Reid devontae.reid@csu.fullerton.edu

Scatterplots





Questions

a. Write your pseudocode for subsequence detection.
   a. Longest Common Substring

```
lcs( s , t):
        n = min(s.length,t.length)                  1
        for each index i from 0 to min:                     |
                if (s[i] != t[i])                           |       n
                        return s.substr(0,i)        1       |
        return s.substr(0,n)                        1

exhaustive_ls(a,b):
        best = empty string                                 1
        m = a.length                                        1
        n = b.length                                        1

        for each index i in a from 0 to m:
                for each index j in b from 0 to n:                  |       |
                        x = lcs(a.substr(i,m), b.substr(j,n))  O(n)  |       |
                        if x.length > best.length                   |   n   |   n
                                best = x                    1       |       |
        return best                                         1
```

a. Longest Common Subsequence

```
detect_subsequence( cand_sub, cand_supsub):
        cand1Ptr = empty char array                     1
        cand2Ptr = empty char array                     1

        copy( cand1Ptr, cand_sub)                       O(n)
        copy( cand2Ptr, cand_supsub)                    O(n)

        while cand1Ptr != null and cand2Ptr != null                 |
                if cand1Ptr == cand2Ptr++                           |       n
                        cand1Ptr++                          1       |
        return !cand1Ptr                                    1


generate_powerset(s):
        results = empty vector                          1

        powerset_size = pow(2,s.length)                 1
        for each index i from 0 to powerset_size                    |
                tempString = ""                         1   |       |
                for each index j from 0 to s.length             |   | 2^n
                        if i and (1<<j)                         |  n  |
                                tempString += s[i]      1       |     |
                results.push_back(tempString)           1              |
        return results                                  1

exhaustive_ls(a,b):
        best = empty string                     1
        shorter = empty string                  1
        longer = empty string                   1

        if a.length > b.length
                shorter = b                     1
                longer = a                      1
        else
                shorter = a                     1
                longer = b                      1

        candSubs = generate_powerset(shorter)   O(2^n n)

        for candSub in candSubs:                                                    |
                if detect_subsequence(sub,longer) and sub.length > best.length      n      |  n
                        best = sub              1                                          |

        return best                             1
```

b. Analyze your subsequence detection algorithm; state and justify a time efficiency class.

   a. Longest Common Substring

      $T(n) = 1 + 1 + 1 + n(n(n + 1)) + 1$
      $= n(n^2 + n) + 4$
      $= n^3 + n^2 + 4 \therefore O(n^3)$

   b. Longest Common Subsequence

      $T(n) = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 2^n n + n(n + 1) + 1$
      $= 2^n n + n^2 + n + 8 \therefore O(2^n n)$

c. Is there a noticeable difference in the performance of the two algorithms? Which is faster, and by how much? Does this surprise you?

   There really isn't a big difference in performance the time also show that the first algorithm is faster by milliseconds. The first algorithm is faster because it doesn't have to spend time dealing with looping through powersets because powersets doesn't finish the loop until 2^n. This does not surprise me.

d. Are your empirical analyses consistent with your mathematical analyses? Justify your answer.

   The empirical analyses are consistent with my mathematical analyses because each data point is close to the trendline.

e. Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.

   Yes, exhaustive search is easy to implement when you understand the pattern for exhaustive search algorithm.

f. Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.

   No, I didn't notice exponential running time being that much slower that the first algorithm that was a polynomial.