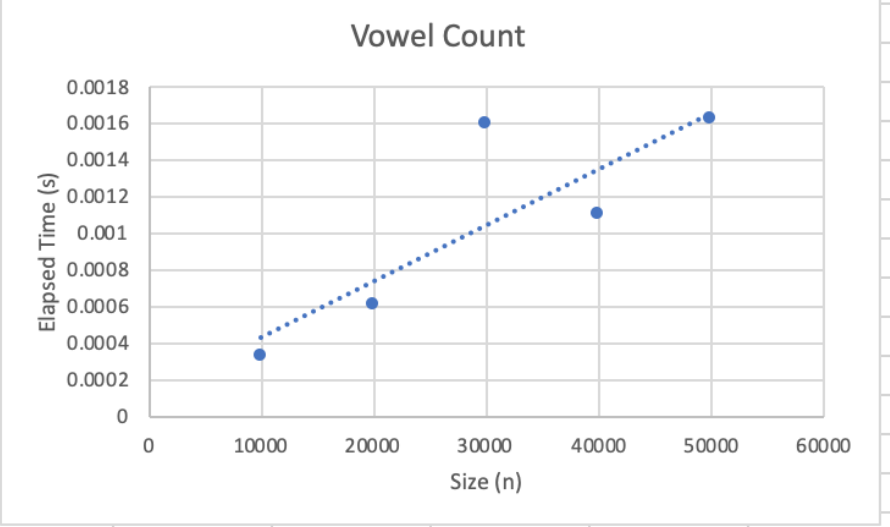
PDF for Project 2

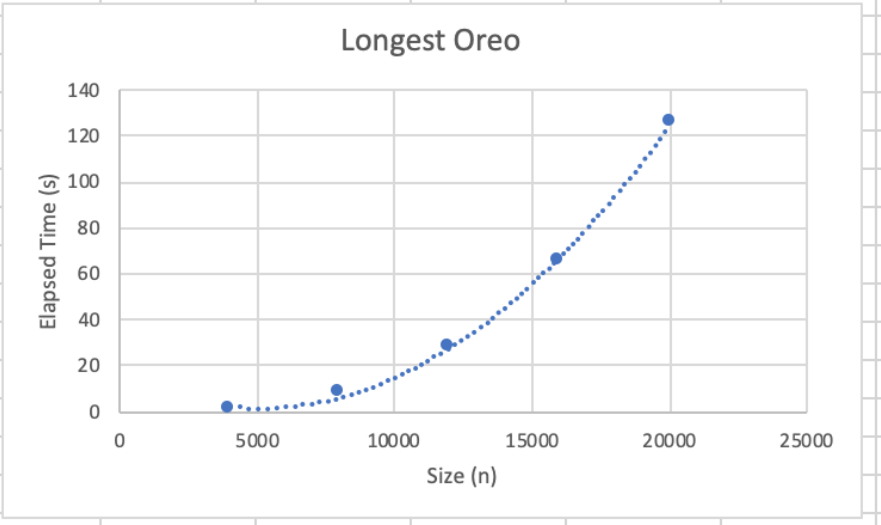
Name: Jesus Flores
Email: jflores2016@csu.fullerton.edu

Three Scatter Plots

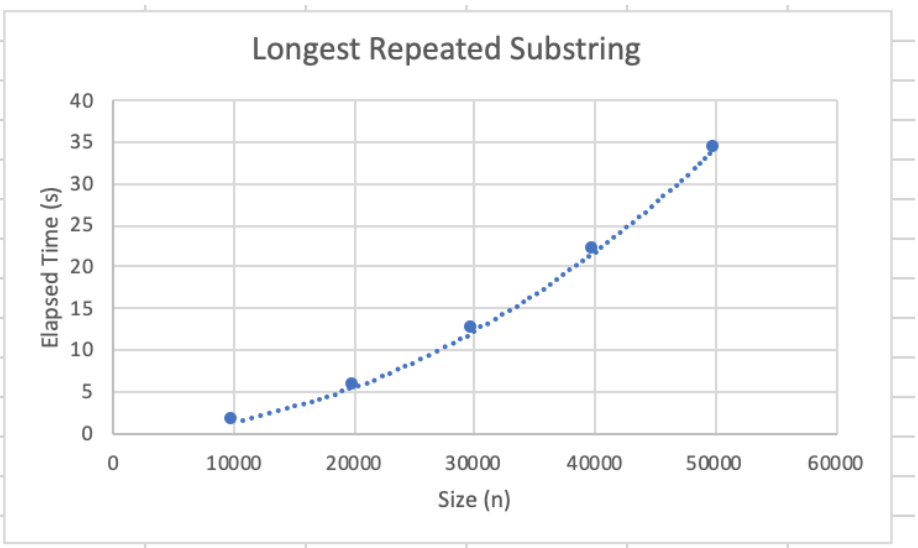
Size X-axis	Time Y-axis
10000	0.00031544
20000	0.00059774
30000	0.00159258
40000	0.00109758
50000	0.00161894



Size X-axis	Time Y-axis
4000	1.10143
8000	8.29531
12000	27.4599
16000	65.0935
20000	125.262



Size X-axis	Time Y-axis
10000	1.40475
20000	5.48399
30000	12.3175
40000	21.822
50000	34.0583



Pseudocode for my three algorithms:

```
vowel_count(string s)
    if s.length == 0
        return 0
    int vowel_count = 0
    for i = 0 i <s.length i++
        if s[i] = a or e or i or o or u or A or E or I or O or U
            ++vowel_count
    return vowel_count


longest_oreo(string s)
    assert s.size >0
    if s.length == 1 or s[0] == s[s.length -1]
        return s
    string longest_o
```

```
    string current_o
    int n = s.length -1
    for int i = 0 i<n i++
        for int j = n j>i j--
            current_o = ""
            if s[i] == s[j]
                for int a = i a<=j a++
                    current_o+=s[a]
                if current_o.length > longest_o.length
                    longest_o = current_o
    return longest_o

longest_repeated_substring(string s)
    assert s.size >0
    if s.length ==1
        return ""
    string substring
    string longest_sub
    int a = 0
    int b = 0
    for i = 0 i <s.length -1 i++
        for j = i+1 j<s.length j++
            substring = ""
            a = i
            b = j
            if substring.length == 0 and s[a] == s[b]
                substring+= s[a]
```

```
            ++a
            ++b
        while s[a] == s[b] and s[a-1] == s[b-1] and b<s.length
            substring+= s[a]
            ++a
            ++b
        if longest_sub.length < substring.length
            longest_sub = substring
    return longest_sub
```

The efficiency class that I derived for vowel count was O(3n+3), which I proved to be 0(n). The efficiency class that I derived for longest Oreo was O(10n^3 +6), which I proved to be O(n^3). The efficiency class that I derived for longest repeated substring was O(40n^3 +7), which I proved to be O(n^3).

Yes, there is a noticeable difference between the algorithms. The longest Oreo algorithm was by far the most time-consuming algorithm. The size inputs are different for this algorithm than the other two because who knows when the Oreo algorithm would've stopped running had I used the other size inputs. The vowel count algorithm is much faster than the other two algorithms. Both of the algorithms took seconds if not minutes to finish running. The vowel count algorithm on the other hand wasn't even close to hitting the one second mark when it came to finishing running.

Yes, the fit lines on my scatter plot are consistent with the efficiency classes that I derived. For my longest Oreo and longest repeating substring algorithms, I got a polynomial for my fit line, which corresponds to their 0(n^3) time. For my vowel count algorithm, I got a linear line for my fit line, which corresponds to its 0(n) time. However, I did have at outlier for my vowel count when the size was 30,000. I believe that this was due to the fact that I went onto a website called unit-conversion to create my random strings. The ratio of vowels to words may have been a bit more than expected when size was 30,000.

The evidence is consistent with the hypothesis stated on the first page. The efficiency classes that I came to a conclusion to corresponds to their respective graph fit line. There may be a few outliers when testing out algorithms, something which happened to me, but aside from that the evidence is consistent with the hypothesis.