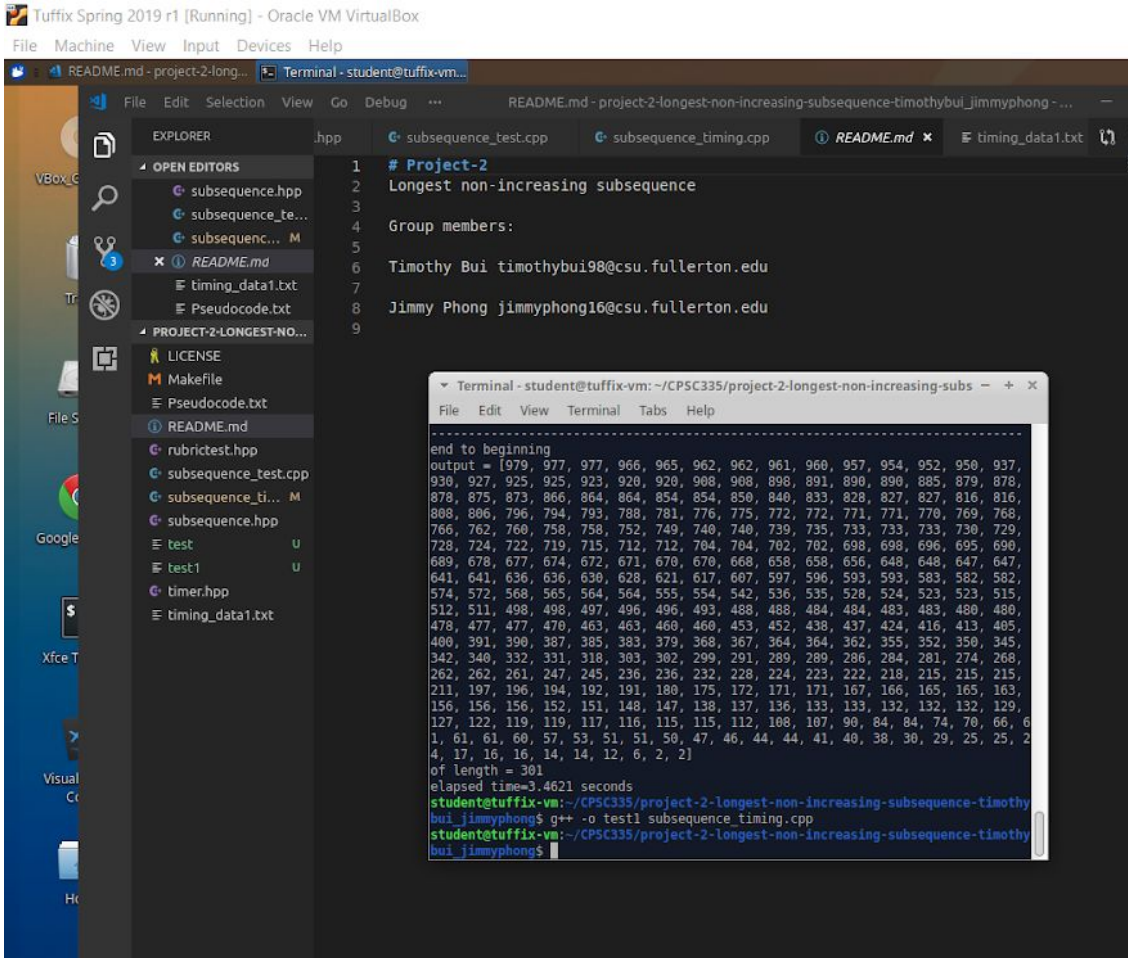
CPSC 335 Project 2 - Empirical Analysis

By: Jimmy Phong and Timothy Bui
JimmyPhong16@csu.fullerton.edu
timothybui98@csu.fullerton.edu

In this project, we were given the task to analyze two algorithms: End-toBeginning Algorithm and the Powerset Algorithm. This project description includes complete functionality of the running skeleton code provided by our professor, formatility of the report and Github presentation, and a complete analysis of the algorithms' efficiency class along with timing data to prove the efficiency classes. All files have been posted to our Github repository.

Pasted below is the screenshot of Tuffix and the IDE that we used.



Algorithm Analysis - *End-to-Beginning Algorithm*

Pseudocode

This image includes the pseudocode and step count calculation. I divided the algorithm into two parts: Part 1 and Part 2.

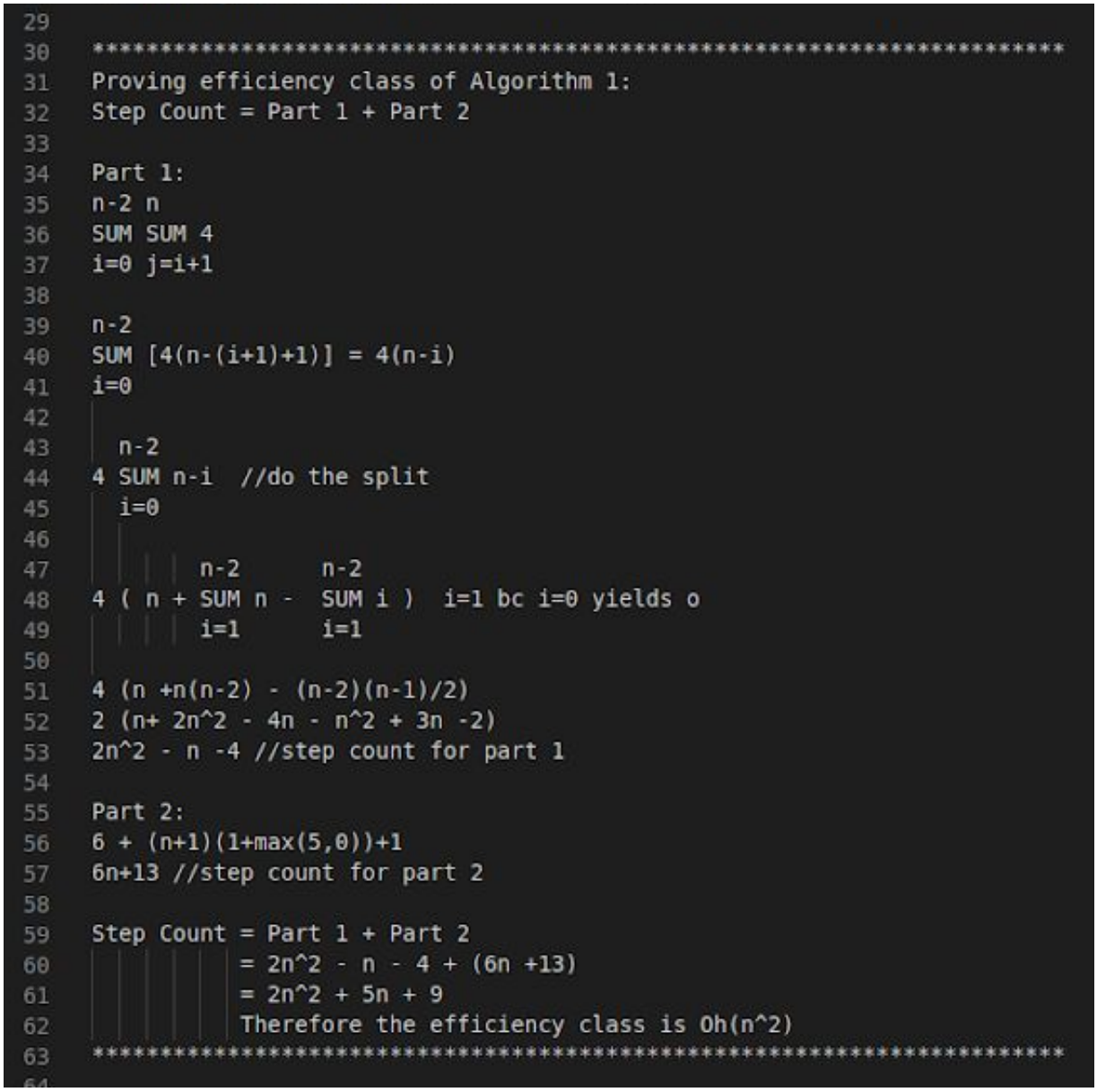
Equation: Step Count = Part 1 + Part 2

Pseudocode.txt × README.md

```
Pseudocode for mathematical analysis
This is used to solve the efficiency class using Big-Oh notation.

Algorithm 1: End-to-Beginning Algorithm
                                                    Step Count Part 1
Set n = size of vector A                            +1
initialize vector H with all zeros                  +1
for i = n-2 down to 0 do                            n-1 times
    for j= i+1 to n do                              n-i+1 times
        if(A[i] >= A[j])                            1+max(1+max(2,0),0) = 4
            if(H[i]<=H[j])
                H[i]=H[j]+1
    endfor
endfor
                                                    Step Count Part 2
max = size of H +1                                  +2
initialize vector R with max length                 +1
initialize index = max - 1                          +2
initialize j = 0                                    +1
for 0 to n do                                       n+1 times
    if(H[i] == index)                               1+max(5,0)
    {
        R[j] = A at index i
        decrement index
        increment j
    }
endfor
return sequence R with max                          +1
```

Efficiency Class Proof

This image contains the proof of the algorithm's efficiency class. This is also split into two parts.



```
29
30  ********************************************************************************
31  Proving efficiency class of Algorithm 1:
32  Step Count = Part 1 + Part 2
33
34  Part 1:
35  n-2 n
36  SUM SUM 4
37  i=0 j=i+1
38
39  n-2
40  SUM [4(n-(i+1)+1)] = 4(n-i)
41  i=0
42  |
43  |  n-2
44  4 SUM n-i  //do the split
45  |  i=0
46  |  |
47  |  |  |  n-2      n-2
48  4 ( n + SUM n -  SUM i )  i=1 bc i=0 yields o
49  |  |  |  i=1      i=1
50  |
51  4 (n +n(n-2) - (n-2)(n-1)/2)
52  2 (n+ 2n^2 - 4n - n^2 + 3n -2)
53  2n^2 - n -4 //step count for part 1
54
55  Part 2:
56  6 + (n+1)(1+max(5,0))+1
57  6n+13 //step count for part 2
58
59  Step Count = Part 1 + Part 2
60             = 2n^2 - n - 4 + (6n +13)
61             = 2n^2 + 5n + 9
62             Therefore the efficiency class is Oh(n^2)
63  ********************************************************************************
64
```

Algorithm Analysis - *Powerset Algorithm*

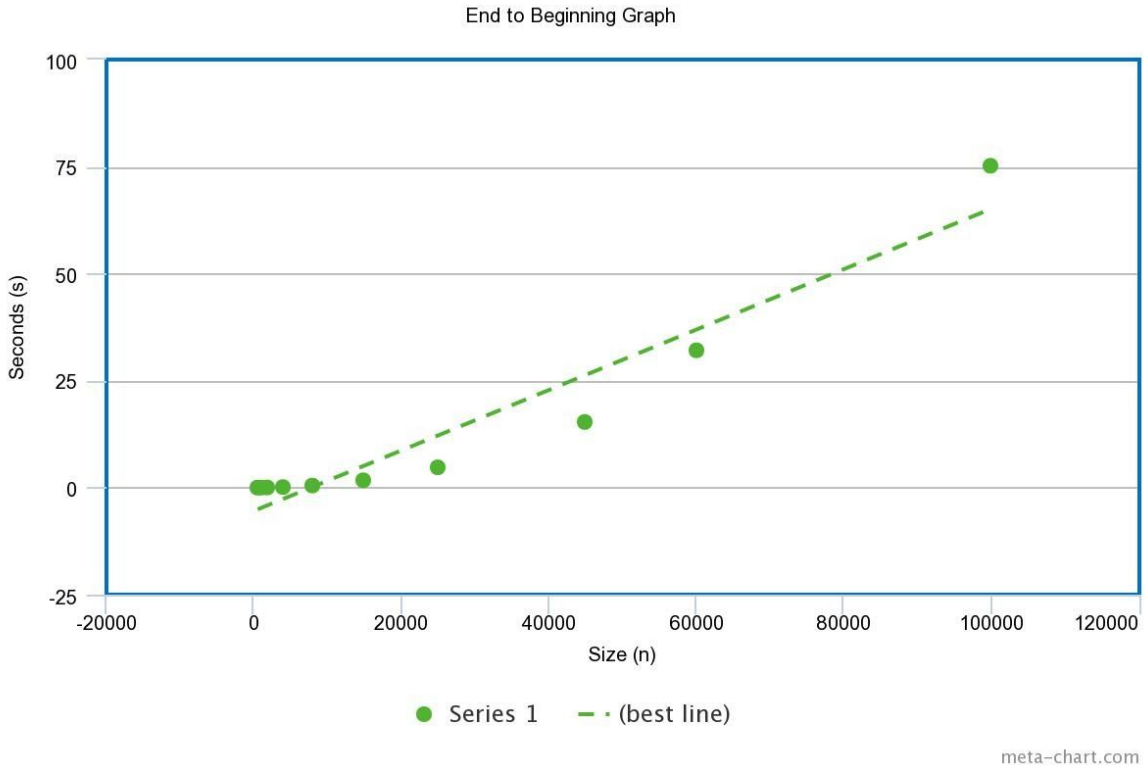
```
**************************************************************************
Algorithm 2: Powerset Algorithm
                                                Step Count
Set n = size of Vector A                        +1
initialize sequence Best                        +1
initialize vector Stack with n+1 length         +1
initialize k = 0                                +1

while true                                      2^n

  if (Stack[k] < n)                             1+max(4,4)
    Stack[k+1] = Stack[k] + 1
    k = k+1
  else
    Stack[k-1] = Stack[k-1] + 1
    k = k-1

  if k = 0                                      1+max(1,0)
    break

  initialize sequence Candidate                 +1
  for i = 1 to n do                             n
     push to candiate(A[Stack[i]-1])            +4
  endfor

   if candidate is nonincreasing                max(max(1,0),0)
     if size of Candidate > size of Best
       Best = Candidate

endwhile

return sequence Best                            1
**************************************************************************
```

Efficiency Class Proof - This image contains the proof of the algorithm's efficiency class.

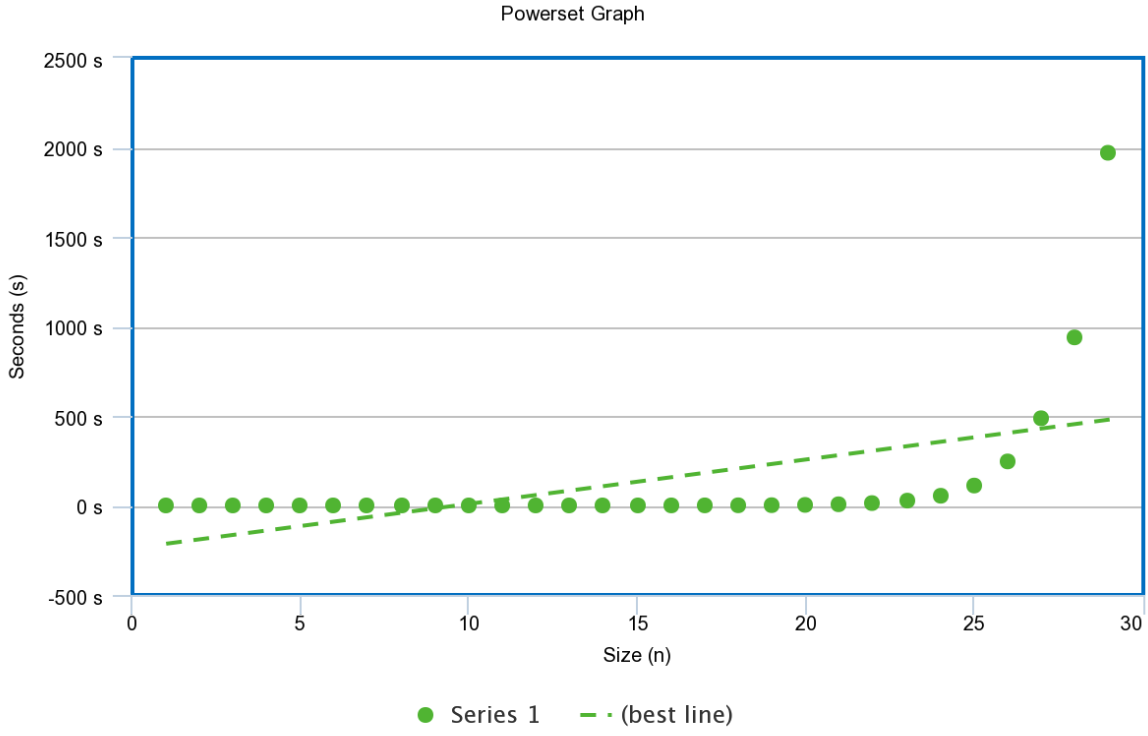
```
**************************************************************************
Proving efficiency class of Algorithm 2:

Step count f(n) = 1+1+1+1 + 2^n(1+max(4,4) + 1+max(1,0) + 1+(4n) + (max(max(1,0),0)) + 1
                  4 + 2^n(5 + 2 + 1+4n + 1) + 1
           f(n) = 5 + 2^n(9+4n) -> O(n * 2^n)
**************************************************************************
```

Scatter Plot with empirical timing data

Below is the result of running the subsequence_timing.cpp test file.



Powerset Algorithm



Analysis Questions

1. What is the efficiency class of each of your algorithms, according to your own mathematical analysis? (You are not required to include all your math work, just state the classes you derived and proved.)

The efficiency class for the End-to-Beginning Algorithm is $O(n^2)$, while the efficiency class for the Powerset Algorithm is $O(n * 2^n)$.

2. Is there a noticeable difference in the running speed of the algorithms? Which is faster, and by how much? Does this surprise you?

According to the data obtained, there is a significant difference in the running speed of the algorithms. The end-to-beginning algorithm is magnitudes faster than the powerset algorithm. Even if the latter is exhaustive, this level of difference in running speed is extremely surprising.

3. Are the fit lines on your scatter plots consistent with these efficiency classes? Justify your answer.

No, because the fit lines are a linear representation of the data of the scatterplot while the actual efficiency classes are not linear but exponential and logarithmic.

4. Is this evidence consistent or inconsistent with the hypothesis stated on the first page? Justify your answer.

1. Exhaustive search algorithms are feasible to implement, and produce correct outputs.
2. Algorithms with exponential or factorial running times are extremely slow, probably too slow to be of practical use.

Exhaustive search algorithms can be implemented, but are not feasible since the run time scales exponentially as the size of "n" increases in a sequence. Algorithms that scale exponentially in runtime are most likely too slow in practicality when scaled up to real world applications.