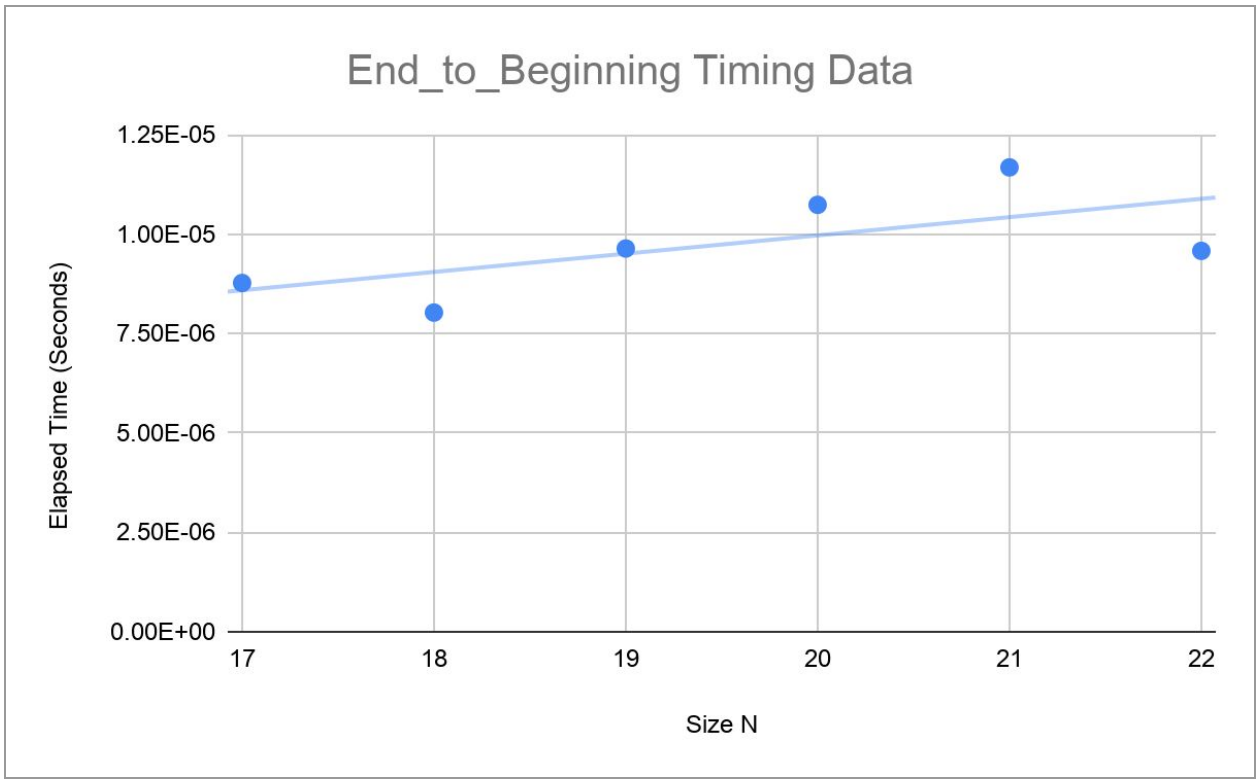
# Project 2 Brief Written Report

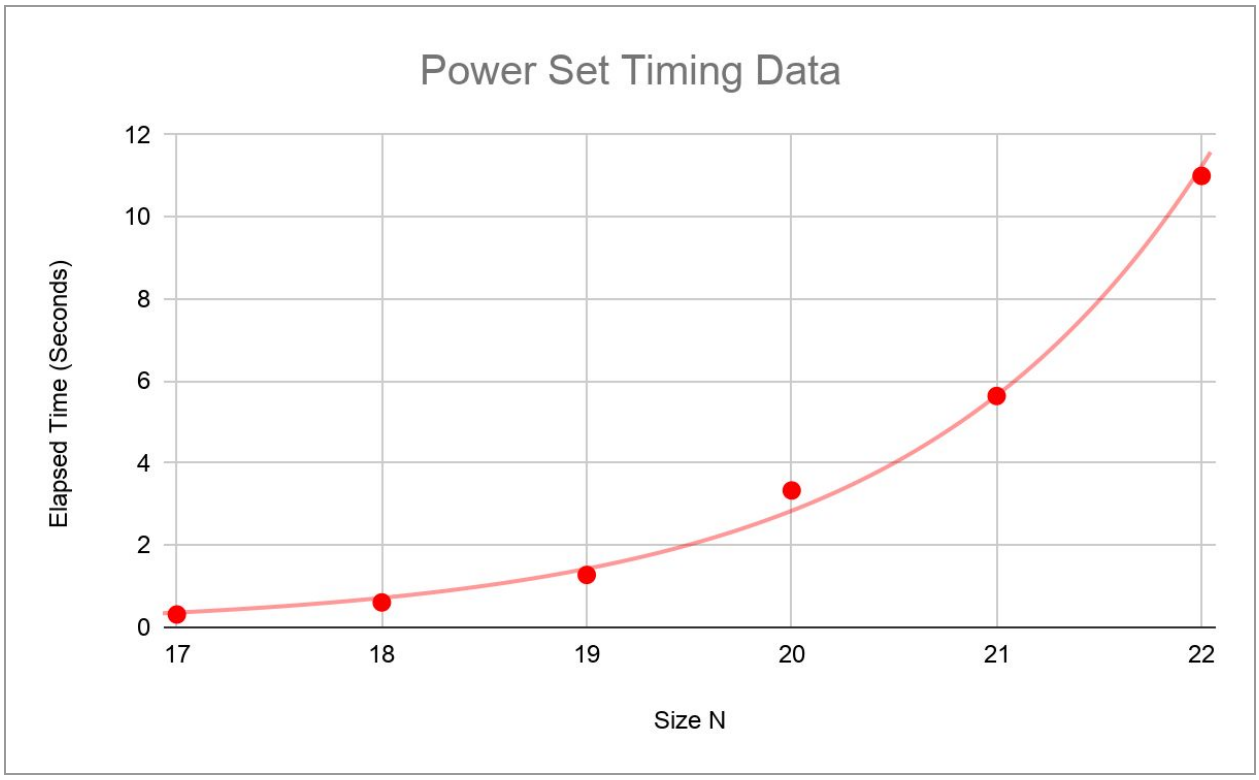
## Group Member Names:

Holland Ho ho.holland1999@csu.fullerton.edu

James Talavera jwolfw@csu.fullerton.edu

**Two Scatter Plots:**



a) Pseudocode listings:

1. End-to-Beginning Algorithm

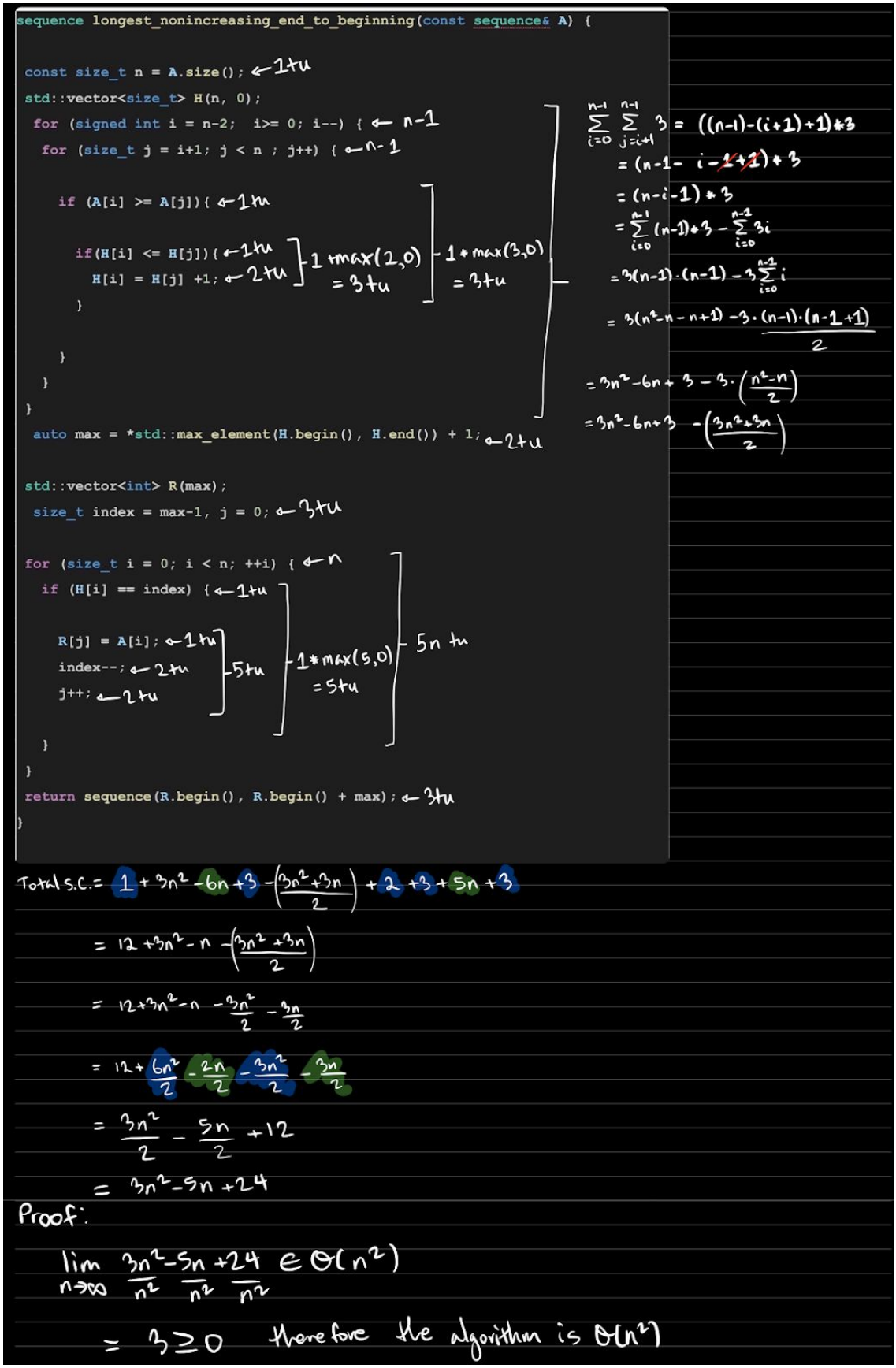
```
sequence longest_nonincreasing_end_to_beginning(const sequence& A) {

 const size_t n = A.size();
 std::vector<size_t> H(n, 0);
  for (signed int i = n-2;  i>= 0; i--) {
   for (size_t j = i+1; j < n ; j++) {

     if (A[i] >= A[j]){

       if(H[i] <= H[j]){
         H[i] = H[j] +1;
       }

     }
   }
 }
  auto max = *std::max_element(H.begin(), H.end()) + 1;

 std::vector<int> R(max);
  size_t index = max-1, j = 0;

 for (size_t i = 0; i < n; ++i) {
   if (H[i] == index) {

     R[j] = A[i];
     index--;
     j++;

   }
 }
 return sequence(R.begin(), R.begin() + max);
}
```

2. Exhaustive Algorithm

```
sequence longest_nonincreasing_powerset(const sequence& A) {
 const size_t n = A.size();
 sequence best;
 std::vector<size_t> stack(n+1, 0);
 size_t k = 0;
 while (true) {

   if (stack[k] < n) {
     stack[k+1] = stack[k] + 1;
     ++k;
   } else {
     stack[k-1]++;
     k--;
   }

   if (k == 0) {
     break;
   }

   sequence candidate;
   for (size_t i = 1; i <= k; ++i) {
     candidate.push_back(A[stack[i]-1]);
   }

   if (is_nonincreasing(candidate) && candidate.size() > best.size()) {
     best = candidate;
   }
 }
 return best;
}
```

## b) Efficiency Class Mathematical Analysis:

### 1. End_to_Beginning




Total s.c. $= 1 + 3n^2 - 6n + 3 - \left(\frac{3n^2+3n}{2}\right) + 2 + 3 + 5n + 3$

$= 12 + 3n^2 - n - \left(\frac{3n^2+3n}{2}\right)$

$= 12 + 3n^2 - n - \frac{3n^2}{2} - \frac{3n}{2}$

$= 12 + \frac{6n^2}{2} - \frac{2n}{2} - \frac{3n^2}{2} - \frac{3n}{2}$

$= \frac{3n^2}{2} - \frac{5n}{2} + 12$

$= 3n^2 - 5n + 24$

Proof:

$\lim_{n\to\infty} \frac{3n^2-5n+24}{n^2} \in \Theta(n^2)$

$= 3 \geq 0$ therefore the algorithm is $\Theta(n^2)$

## 2. Power Set

```
sequence longest_nonincreasing_powerset(const sequence& A) {
 const size_t n = A.size();                                  ← 1tu
 sequence best;
 std::vector<size_t> stack(n+1, 0);                          ← 1tu   } 3tu
 size_t k = 0;                                               ← 1tu
 while (true) {                                              ← 2^n
   if (stack[k] < n) {                                       ← 1tu
     stack[k+1] = stack[k] + 1;                              ← 2tu   } 4tu
     ++k;                                                    ← 2tu
   } else {
     stack[k-1]++;                                           ← 3tu   } 5tu
     k--;                                                    ← 2tu
   }                                                                 } 1*max(4,5) = 1*9 = 9tu

   if (k == 0) {                                             ← 1tu
     break;
   }

   sequence candidate;
   for (size_t i = 1; i <= k; ++i) {                         ← n
     candidate.push_back(A[stack[i]-1]);                     ← 1tu   } n
   }

   if (is_nonincreasing(candidate) && candidate.size() > best.size()) {   ←   2n-1+2 = 2n+1 tu  } (2n+1)*max(1,0) = 2n+1
     best = candidate;                                       ← 1tu
   }
 }
 return best;                                                ← 1tu
}
```

Total S.C = $3 + 2^n * (9 + 1 + n + 2n + 1) + 1$

$= 4 + 2^n \cdot (11 + 3n)$

$= 4 + 2^n \cdot 11 + 3n \cdot 2^n$

$= 3n \cdot 2^n + 2^n(11) + 4$

Proof:

$$\lim_{n \to \infty} \frac{3n \cdot 2^n}{n2^n} + \frac{2^n(11)}{n2^n} + \frac{4}{n2^n} \in \Theta(n \cdot 2^n)$$

$= 3 \geq 0$ therefore the algorithm is $\Theta(n \cdot 2^n)$

**c)** There is a noticeable difference in the running speed between the two algorithms. The End_to_Beginning Algorithm has a running speed growing at an exponential rate steadily, with each increase to the size of n making minor increments in the elapsed time. While the Exhaustive Algorithm (Power Set), had a running speed that grew exponentially quickly with every increase to the size of n. So, the faster algorithm would be the End_to_Beginning Algorithm. The End_to_Beginning Algorithm is faster than the Exhaustive one by a significant amount as the Exhaustive Algorithm starts growing exponentially with longer running times as size n increases sooner than the End_to_Beginning Algorithm. It does not come to a surprise as when analyzing the algorithm, the Exhaustive method uses a while loop which does not always narrow down to a set amount of iterations versus the End_to_Beginning method uses for loops which have a defined amount of iterations.

**d)** The fit lines on the scatter plots are consistent with the efficiency classes as the End_to_Beginning fit line is exponential growth, however, the increments are much more significant after more runs, with the beginning run time data being quick as expected. While the fit line for the Power Set began increasing was quicker than compared to the first method, which is expected as the values with Power Set's time complexity starts off higher and increments significantly larger. Examples being that with each increment to the size of n, End_to_Beginning running times go from 9.64E-06 to 1.07E-05 whereas Power set running times would go from 3.33644 to 5.63829 displaying the different growth rates.

**e)** According to the hypotheses made in the beginning, with our gathered evidence, it shows to be that the two are consistent. The exhaustive search was easy to implement and produced the correct outputs just as the End_to_Beginning algorithm, however just as the hypothesis stated the algorithm would be too slow to be of practical use; as shown when the size of n increased to 25, the Exhaustive method lead to running times going all the way into the minutes from the prior times that consisted of seconds.