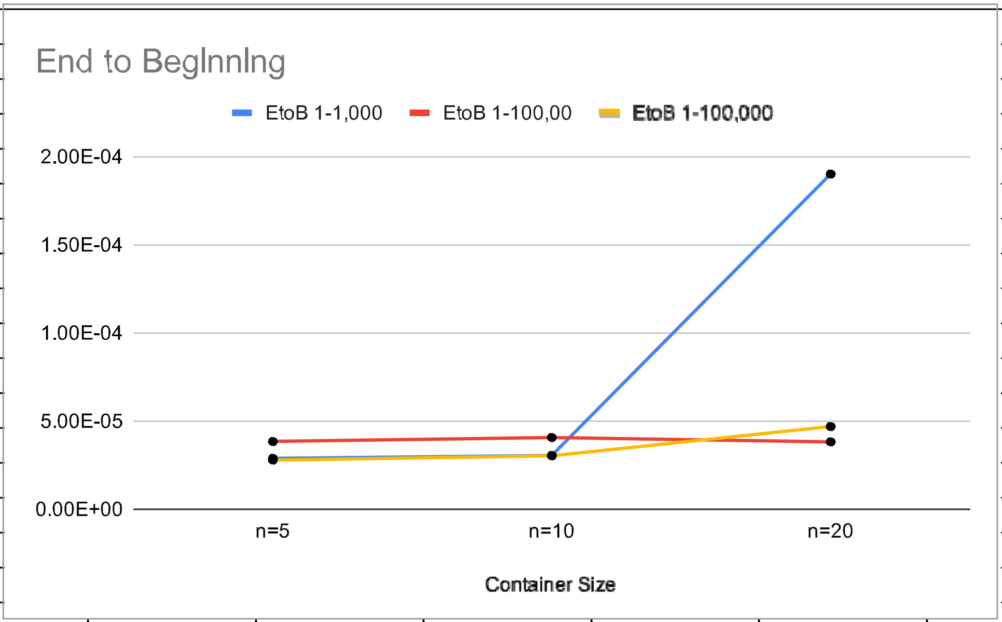
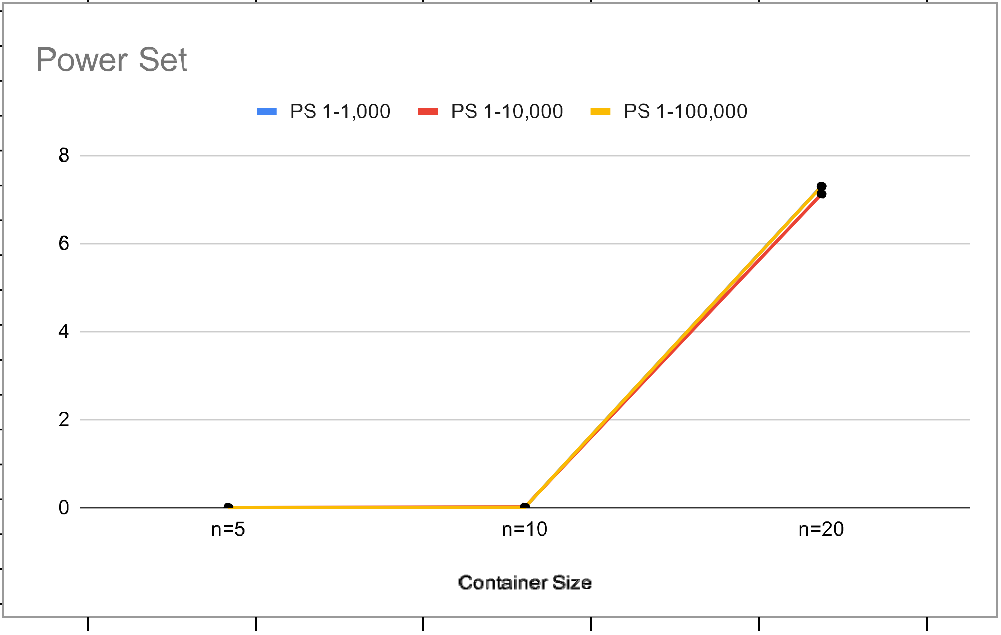
1.) Samuel Flinkfelt – sef202@csu.fullerton.edu – Submission for Project 2

2.) Empirical Timing Data

Container Size	EtoB 1-1,000	EtoB 1-100,00	EtoB 1-100,000
n=5	2.89E-05	3.85E-05	2.79E-05
n=10	3.05E-05	4.07E-05	3.04E-05
n=20	0.000190204	3.83E-05	4.70E-05
Container Size	PS 1-1,000	PS 1-10,000	PS 1-100,000
n=5	0.000104371	0.00014255	0.000105256
n=10	0.00528709	0.00946119	0.00509083
n=20	7.296	7.12049	7.2913





```
n=5
1-1000
--------------------------------------------------------------------------------
n = 5
[684, 559, 629, 192, 835]
--------------------------------------------------------------------------------
end to beginning
output = [684, 559, 192]
of length = 3
elapsed time=2.8909e-05 seconds
--------------------------------------------------------------------------------
powerset
output = [684, 559, 192]
of length = 3
elapsed time=0.000104371 seconds
--------------------------------------------------------------------------------
Program ended with exit code: 0
n=5
1-10,000
--------------------------------------------------------------------------------
n = 5
[2732, 9845, 3264, 4859, 9225]
--------------------------------------------------------------------------------
end to beginning
output = [9845, 3264]
of length = 2
elapsed time=3.848e-05 seconds
--------------------------------------------------------------------------------
powerset
output = [9845, 3264]
of length = 2
elapsed time=0.00014255 seconds
--------------------------------------------------------------------------------
Program ended with exit code: 0
n=5
1-100,000
--------------------------------------------------------------------------------
n = 5
[68268, 43567, 42613, 45891, 21243]
--------------------------------------------------------------------------------
end to beginning
output = [68268, 43567, 42613, 21243]
of length = 4
elapsed time=2.7937e-05 seconds
--------------------------------------------------------------------------------
powerset
output = [68268, 43567, 42613, 21243]
of length = 4
elapsed time=0.000105256 seconds
--------------------------------------------------------------------------------
Program ended with exit code: 0
n=10
1-1000
```

```
--------------------------------------------------------------------------------
n = 10
[684, 559, 629, 192, 835, 763, 707, 359, 9, 723]
--------------------------------------------------------------------------------
end to beginning
output = [835, 763, 707, 359, 9]
of length = 5
elapsed time=3.0535e-05 seconds
--------------------------------------------------------------------------------
powerset
output = [835, 763, 707, 359, 9]
of length = 5
elapsed time=0.00528709 seconds
--------------------------------------------------------------------------------
Program ended with exit code: 0
n=10
1-10,000
--------------------------------------------------------------------------------
n = 10
[2732, 9845, 3264, 4859, 9225, 7891, 4373, 5874, 6744, 3468]
--------------------------------------------------------------------------------
end to beginning
output = [9845, 9225, 7891, 4373, 3468]
of length = 5
elapsed time=4.072e-05 seconds
--------------------------------------------------------------------------------
powerset
output = [9845, 9225, 7891, 4373, 3468]
of length = 5
elapsed time=0.00946119 seconds
--------------------------------------------------------------------------------
Program ended with exit code: 0
n=10
1-100,000
--------------------------------------------------------------------------------
n = 10
[68268, 43567, 42613, 45891, 21243, 95939, 97639, 41993, 86293, 55026]
--------------------------------------------------------------------------------
end to beginning
output = [68268, 43567, 42613, 21243]
of length = 4
elapsed time=3.0374e-05 seconds
--------------------------------------------------------------------------------
powerset
output = [68268, 43567, 42613, 21243]
of length = 4
elapsed time=0.00509083 seconds
--------------------------------------------------------------------------------
Program ended with exit code: 0
n=20
1-1000
--------------------------------------------------------------------------------
n = 20
[684, 559, 629, 192, 835, 763, 707, 359, 9, 723, 277, 754, 804, 599, 70, 472, 600, 396, 314, 705]
```

```
--------------------------------------------------------------------------------
end to beginning
output = [835, 763, 707, 599, 472, 396, 314]
of length = 7
elapsed time=0.000190204 seconds
--------------------------------------------------------------------------------
powerset
output = [835, 763, 707, 599, 472, 396, 314]
of length = 7
elapsed time=7.296 seconds
--------------------------------------------------------------------------------
Program ended with exit code: 0
n=20
1-10,000
--------------------------------------------------------------------------------
n = 20
[2732, 9845, 3264, 4859, 9225, 7891, 4373, 5874, 6744, 3468, 705, 2599, 2222, 7768, 2897, 9893, 537, 6216, 6921,
6036]
--------------------------------------------------------------------------------
end to beginning
output = [9845, 9225, 7891, 4373, 3468, 2599, 2222, 537]
of length = 8
elapsed time=3.8294e-05 seconds
--------------------------------------------------------------------------------
powerset
output = [9845, 9225, 7891, 4373, 3468, 2599, 2222, 537]
of length = 8
elapsed time=7.12049 seconds
--------------------------------------------------------------------------------
Program ended with exit code: 0
n=20
1-100,000
--------------------------------------------------------------------------------
n = 20
[68268, 43567, 42613, 45891, 21243, 95939, 97639, 41993, 86293, 55026, 80471, 80966, 48600, 39512, 52620, 80186,
17089, 32230, 18983, 89688]
--------------------------------------------------------------------------------
end to beginning
output = [68268, 43567, 42613, 41993, 39512, 32230, 18983]
of length = 7
elapsed time=4.6957e-05 seconds
--------------------------------------------------------------------------------
powerset
output = [68268, 43567, 42613, 41993, 39512, 32230, 18983]
of length = 7
elapsed time=7.2913 seconds
--------------------------------------------------------------------------------
Program ended with exit code: 0
```

3.)a.) Provide pseudocode for your two algorithms.

Actual Code

```
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] && H[i] <= H[j] + 1){
      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);
}
```

PseudoCode

```
Sequence Function Longest_nonincreasing_end_to_beginning(Constant vector named A){          T.C.
        Initialize integer n = Length of A..........................................................>   1 tu
        Create Vector H(length of A, all Zeros)......................................................>   1 tu
        Forloop( initialize integer i = n – 2,  i is greater or equal to 0,  decrement i){...............>   (n-1)
                Forloop( initialize integer j = i + 1, j is less than n, increment j){......................>   (2n)
                        If(Seq A[i] is greater OR equal to A[j] AND H[i] less OR equal H[j]+1){>   2 tu
                                H[i] = H[j]+1 // H box is carried over and incremented 1........>   2 tu
                        }
                }
        } // First Time Complexity = 1 + 1 + 3 + ((n-1) * (2n) +2) = O(n^2)..............................>   O(n^2)
        Initialize integer Max = find the MAX value in (seq H) // and declare it Max...............>   1 tu
        Create Vector named R(with size of Max value + 1) ...........................................>   1 tu
        Initialize integer index = Max value minus 1..................................................>   1 tu
        Initialize integer j = 0 ......................................................................>   1 tu
        For ( initialize i = 0, i is less than n (length A), increment i){ ..............................>   (n)
                If(H[i] is the same as index){ // if it’s the first decremented number in H.......>   1 tu
                R[j] = A[i] // add the value A at index I into R at the index j ...........................>   1 tu
                Decrement index ..............................................................................>   1tu
                Increment j ...................................................................................>   1tu
                } //Second Time Complexity = 1+1+1+1+(n)+1+1+1+1 = O(n)                                      O(n)
        }// Overall Time Complexity = First + Second = O(n^2) + O(n) = O(n^2)
```

Actual Code

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

PseudoCode

```
Sequence Function Longest_nonincreasing_powerset(Constant vector named A){..................>   T.C.
Initialize integer n = Length of A.........................................................>   1 tu
Create sequence best; // it's empty........................................................>   1 tu
Create Sequence Vector named stack(Size of n + 1(length of A + 1), all 0's)................>   1 tu
Initialize integer k = 0 ..................................................................>   1 tu
Whileloop(Boolean true){ // while there is elements in A ..................................>   (n-1)
        If stack(k) is less than size of n{................................................>   1 tu
                Stack[k+1] = stack[k]+1 // stack value is carried over and incremented 1 .......>   3 tu
                Increment k ...............................................................>   1 tu
        }else{ ....................................................................................>   1 tu
                stack[k-1]++ // previous index in stack is now the next increment .................>   1 tu
                decrement k ...............................................................>   1 tu
        }
        If (k == 0){ //if k is the first element in stack..................................>   1 tu
                break while loop;} ........................................................>   1 tu
        Initialize sequence vector candidate...............................................>   1 tu
        For( initialize integer i = 0, I is less or equal to k, increment i){..............>   (n+1)
                Add to candidate vector(A[stack[i] – i]) // add the value of A}............>   1 tu
        If (candidate is non increasing AND candidate size is greater than best size){.....>   2 tu
                Return best}.......................................................................>   1 tu
        }//also include the time complexity of function Non_increasing = O(n)
        Time complexity = O((2^n) * n)                                                              O(2^n)
```

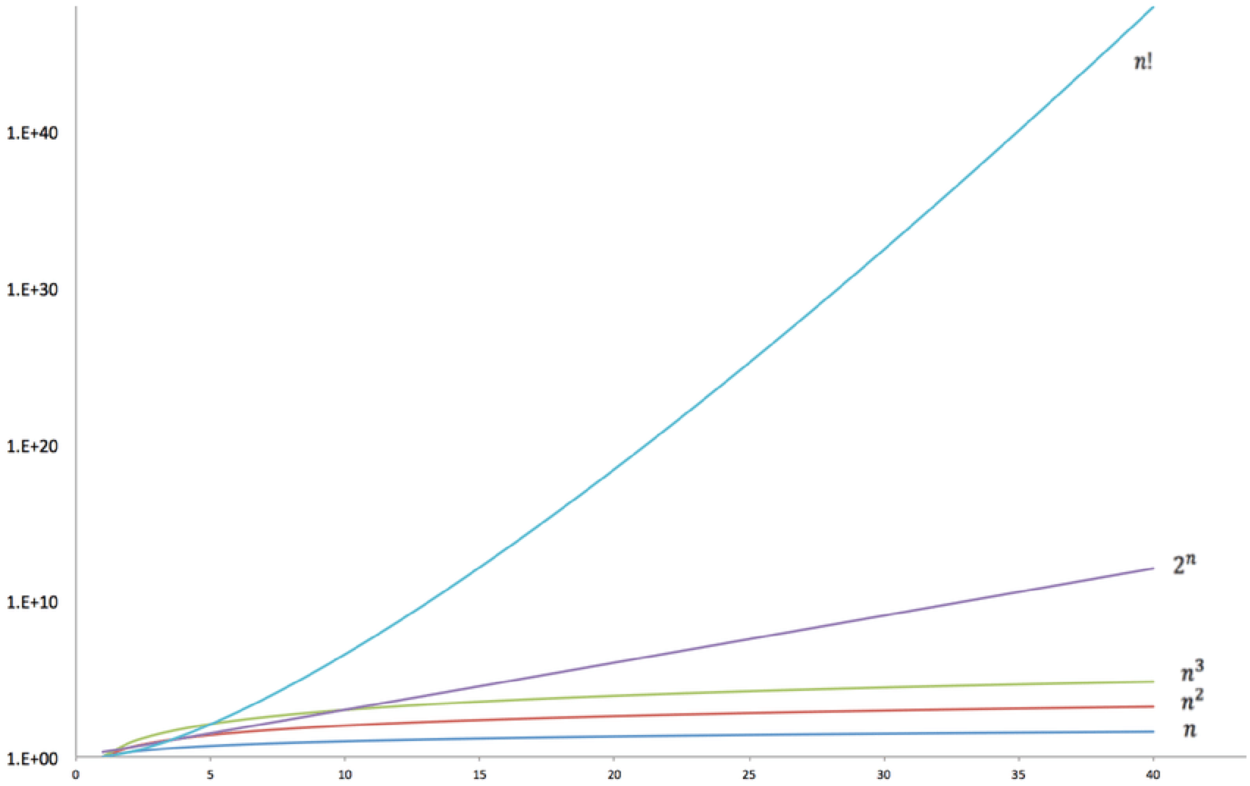
b.) I have computed that the The Beginning to End function has a overall time complexity of O(n^2) because it works off a nested loop. While the powerset function has the time complexity of O(2^n), with two with the for loop (the non_incrementing function) inside a for loop inside a while loop.

c.) There is a noticeable difference in the running speed of the algorithms. The Beginning to end function takes less time to compute then that of the powerset function, in some cases by 6-7 seconds. This logic makes sense, comparing the 2 mathematical functions to the graph below and does not surprise me.

d.) While testing, the graphs I have generated on the first page, shows that it does take less time to compute the Beginning to end function than the powerset function. The function lines match up with the graphs steep climb comparing the two functions, as the powerset has much higher seconds to run. This is probably because the powerset is going through each list with a while loop and stack trying to compare every single sequence to get a working sequence.

d.) The time efficiency data I've generated is consistent with my mathematical derived big- O efficiency class. My data has produced longer runtimes with the powerset function, then the beg_to_end function by comparing the (n=5, n=10, n=20), along with the random number ranges of (1-1000, 1-10,000, 1-100,000). These test results justify the efficiency classes.

e.) Along with the scatter plot diagrams, which are all consistent with the runtimes and data that I have produced. I have concluded that 2^n powerset takes more time than n^2 beg_to_end.


1.E+40
1.E+30
1.E+20
1.E+10
1.E+00
0
5
10
15
20
25
30
35
40
$n!$
2^n
n^3
n^2
n