CPSC 335 Project 2
Alex Mulvaney - mulvaneya@csu.fullerton.edu
Shane Spangenberg - sjs445@csu.fullerton.edu

KEY
(1) = end_to_beginning algo
(2) = powerset algo

PSEUDOCODE
(1)

```
sequence longest_nonincreasing_end_to_beginning(sequence A):
    n = A.size()    //n size of A
    H(n, 0)         //H vector size A, initiated all 0
    for(i = n-2; i >= 0; i--)          //go down the sequence of numbers (end->beginning)
        for(j=i+1; j<n; j++)     //for each element between i+1 to the end of the array
            //if the indexed element is >= to the previous index and its follow list value
            //is < the previous then the indexed follow list element = 1+previous
            if(A[i] >= A[j] && H[i] < H[j] + 1) then
                H[i] = H[j] +1

    R(max(H))       //vector R of length = max element of H follow list
    index = max(H)-1      //count for each element in the follow list
    j = 0                 //count for R sequence
    for(i = 0; i < n; i++)    //for each element in H/A
        //if H ele is the index value the add that A index to R, dec and inc accordingly
        if(H[i] == index)
            R[j] = A[i]
            index--
            j++
    return R
```

MATHEMATICAL ANALYSIS

(1)

sequence **longest_nonincreasing_end_to_beginning**(sequence A):

```
n = A.size()                                              -1
H(n, 0)                                                   -1
for(i = n-2; i >= 0; i--)                                 -n
        for(j=i+1; j<n; j++)                              -n
                if(A[i] >= A[j] && H[i] < H[j] + 1) then  -max(3,5) = 5
                        H[i] = H[j] +1

R(max(H))                                                 -1
index = max(H)-1                                          -2
j = 0                                                     -1
for(i = 0; i < n; i++)                                    -n
        if(H[i] == index)                                 -max(1,4) = 4
                R[j] = A[i]
                index--
                j++
return R                                                  -1
```

Step-Count = 2 + (n * n * 5) + 4 + (n * 4) + 1

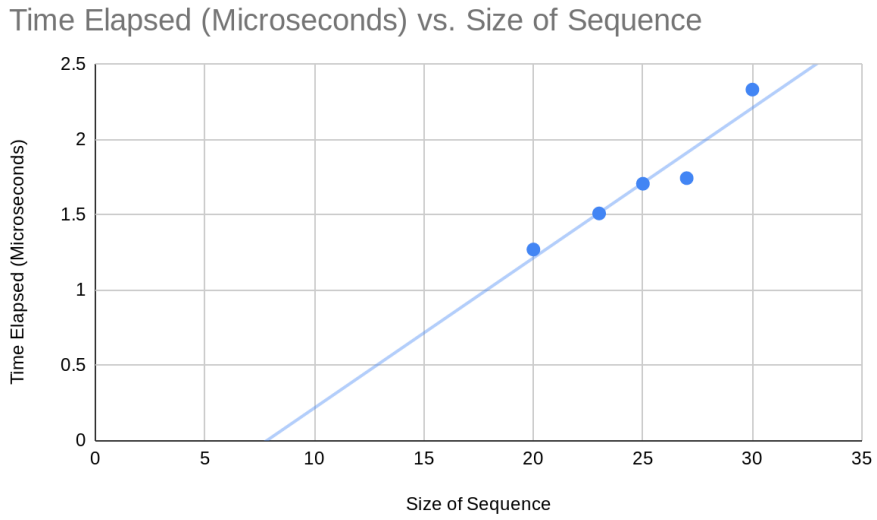
= $7 + 4n + 5n^2$

lim(n -> ∞) ($5n^2+4n+7 / n^2$) = 5; 5 ≥ 0, so $5n^2+4n+7 \in O(n^2)$

SCATTER PLOT
(1)

Power set Scatter Plot and Analysis

Size of Sequence	Time Elapsed (Microseconds)
20	1.2711
23	1.5106
25	1.708
27	1.745
30	2.3335



The beginning to end algo operates much faster as all time elapsed was observed in microseconds and not seconds.

PSEUDOCODE

(2)

```
sequence longest_nonincreasing_powerset(sequence A) begin function
n = A.size();   // We let n be the size of our list input.
sequence best;        // Initialize a new sequence, best.
stack(n+1, 0);          // Create a stack with size + 1 of initial input.
k = 0;            // Initialize k to 0.
while (true) do
   if (stack[k] < n) then  // If the value at pos[k] in the stack is less than the initial size.
     stack[k+1] = stack[k] + 1;         // value at pos[k+1] becomes value at pos[k]+1.
     ++k;         // Increment k.
   else
     Stack[k-1]++;      // Otherwise, increment value in the stack at pos[k-1].
     K--;                                  // Decrement k

   if (k == 0) then                // Break the while loop when k is 0 again.
     break;

   sequence candidate;       // Initialize a new sequence, candidate.
   for i = 1 to i<=k do
     candidate.push_back(A[stack[i]-1]);   // populate powersets into candidate
  End for
   if(is_nonincreasing(candidate) && candidate.size() > best.size()) then
     best=candidate;            // If candidate  is nonincreasing and the size of candidate is bigger
//than the size of best then best becomes our candidate.
End while
  return best;
End Function
```

Note: the is_nonincreasing() helper function is used…

```
is_nonincreasing (sequence A) begin function
  For i = 0 to A.size()-1 do
   if(A.at(i) < A.at(i+1)) // If the value at index i is less than i+1 it is not a non-increasing subsequence.
   then
     return false;
  End for
  return true;
```

MATHEMATICAL ANALYSIS

(2)

Note: the is_nonincreasing() helper function is used…

```
Bool is_nonincreasing (sequence A) begin function
  For i = 0 to A.size()-1 do                                  -n times
    if(A.at(i) < A.at(i+1))                                   -max(3,0)=3
    then
      return false;                                           -1
  End for
  return true;                                                -1
```

S.C. for helper function = 3n+1

Main Algorithm

```
sequence longest_nonincreasing_powerset(sequence A) begin function
n = A.size();                                                 -1
sequence best;                                                -1
stack(n+1, 0);                                                -1
k = 0;                                                        -1
while (true) do                                               -2^n
    if (stack[k] < n) then                                    -1+max(4, 3)=5
      stack[k+1] = stack[k] + 1;                              -3
      ++k;                                                    -1
    else
      Stack[k-1]++;                                           -2
      K--;                                                    -1

    if (k == 0) then                                          -1+max(1,0)=2
      Break;                                                  -1

    sequence candidate;                                       -1
    for i = 1 to i<=k do                                      -n times 2 = 2n
      candidate.push_back(A[stack[i]-1]);                     -2
   End for
    if(is_nonincreasing(candidate) && candidate.size() > best.size()) then
                                                              -(3n+1)+1+max(1,0)=3n+3
      best=candidate;                                 -1
End while
  return best;                                                -1
End Function
```

Step Count = $4 + 2^n * (7 + 2n + 3n + 3) + 1 = (2^n * 11) + (2^n * 5n) + 5$

$$\lim_{n\to\infty}(2^n * 11) + (2^n * 5n) + 5 = \lim_{n\to\infty}((2^n * 11)/(2^n * n) + (2^n * 5n)/(2^n * n) + 5/(2^n * n) = 5 \geq 0$$

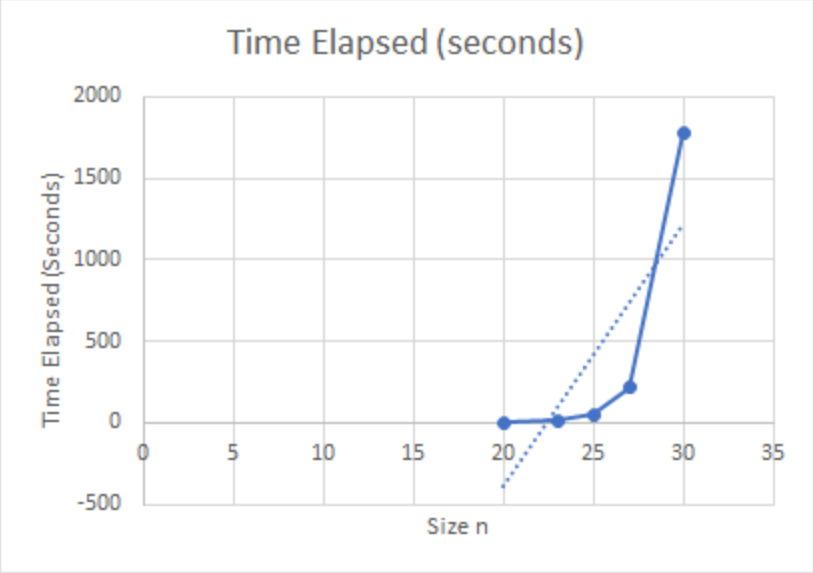
Therefore, $(2^n * 11) + (2^n * 5n) + 5 \in O(2^n * n)$

Time complexity of the algorithm is $O(n * 2^n)$

SCATTER PLOT
(2)

Power Set Scatter Plot and Analysis

Size of Sequence	Time Elapsed (seconds)
20	1.511
23	13.1293
25	52.901
27	215.488
30	1780.03



Based on the empirical data we can conclude the algorithm does fall in $O(n * 2^n)$ because it grows exponentially.

Questions

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

   Comparing the two algorithms the end to beginning algorithm performed extremely faster than the powerset algorithm. In the case of size 30 the end to beginning algorithm finished in just 0.0000077 whereas the powerset algorithm finished in 30 minutes. This is a huge difference and does not surprise me based on the fact that the powerset algorithm has an exponential time complexity.

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

   The fit line is not consistent with the end-to-beg algo because it is linear even though the time it takes per size grows exponentially. Maybe with a larger sample of n's we would see more exponential values

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

   The hypothesis stated that algorithms with exponential times are extremely slow, probably too slow to be of practical use. The graph confirms this case because as we can see once the size grows to 30 the time it takes to complete the algorithm takes way too long to be used by an organization. Someone would not wait 30 minutes for the algorithm to finish in a practical situation.