Mina Moslehpour
Yi Wei Lee
CPSC335

Project 2

- Analyze your pseudocode mathematically and write its efficiency class using Big-Oh notation. (You need to compute the total number of steps of the algorithm.)

```
Longest_nonincreasing_end_to_beginning(vector A) {
    n = size of A                                              1
    vector H with n amount of value 0                          1
    for i >= 0 to n-2 do {                                     n
        for j = i+1 to n do {                                  n
            if A[i] >= A[j] and H[i] <= H[j] {                 1 + max(2,0) + 1 + max(2,0) = 4
                H[i] = H[j] + 1
            }
        }
    }

    max = size of H + 1                                        2
    index = max – 1 and j = 0                                  3

    for i = 0 to n do {                                        n
        if H[i] is equal to index do {                         1 + max(3,0) = 4
            R[j] = A[j]
            Decrement index
            Add j
        }                                                      4 * n = 4n
    }

    return R                                                   1
    }
```

(1 + 1 + (n * n * 4)) + (2 + 3 + n + 4n) = 2 + 4n^2 + 5 + 5n = 4n^2 + 5n + 7 = O(n^2)

```
longest_nondecresing_powerset(sequence A) {
        n = size of A                                                                   1
        creating new sequence best;                                                     1
        Vector size  with n+1 amount of value 0                                         1
        k = 0                                                                           1
        -----------------The part while is the loop depend on the size n, so this loop is 2^n----------
        while true do                                                                   1
          If stack[k] < n                                                               1 + max(3,3) = 4
                    Stack[k+1] = stack[k] + 1
                    k = k +1
          else
                    stack[k-1] = stack[k-1]  + 1
                    k = k -1
          If k is equal to 0                                                           1 + max(1,0) = 2
                    break loop
        creating new sequence candidate                                                 1
        for i = 1 to k do                                                               n
          candidate.push_back(A[stack[i]-1])
        if candidate is nondecreasing and candidate size > best size                    2 + max(1,0) = 3
          best = candidate

        return best                                                                     1
}
```

---------- outside loop -> 1 + 1 + 1 + 1 + 1 = 5 ----------

(n + 10) * 2^n + 5 = **O((2^n)*n)**

- Gather empirical timing data by running your implementation for various values of n. As discussed in class, you need enough data points to establish the shape of the best-fit curve (at least 5 data points, maybe more), and you should use n sizes that are large enough to produce large time values (ideally multiple seconds or even minutes) that minimize instrumental error

Timing data

n = 2
[684, 559]
---
end to beginning
output = [684, 559]
of length = 2
elapsed time=1.077e-05 seconds
---
powerset
output = [684, 559]
of length = 2
elapsed time=9.609e-06 seconds

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=2.841e-05 seconds
---
powerset
output = [835, 763, 707, 599, 472, 396, 314]
of length = 7
elapsed time=3.83667 seconds

n = 18
[684, 559, 629, 192, 835, 763, 707, 359, 9, 723, 277, 754, 804, 599, 70, 472, 600, 396]
---
end to beginning
output = [835, 763, 707, 359, 277, 70]
of length = 6
elapsed time=2.4242e-05 seconds
---
powerset
output = [835, 763, 707, 359, 277, 70]
of length = 6
elapsed time=0.877588 seconds

n = 16
[684, 559, 629, 192, 835, 763, 707, 359, 9, 723, 277, 754, 804, 599, 70, 472]
---
end to beginning
output = [835, 763, 707, 359, 277, 70]
of length = 6
elapsed time=1.7151e-05 seconds
---
powerset
output = [835, 763, 707, 359, 277, 70]
of length = 6
elapsed time=0.222631 seconds

n = 12
[684, 559, 629, 192, 835, 763, 707, 359, 9, 723, 277, 754]
---
end to beginning
output = [835, 763, 707, 359, 9]
of length = 5
elapsed time=1.623e-05 seconds
---
powerset
output = [835, 763, 707, 359, 9]
of length = 5
elapsed time=0.0112568 seconds

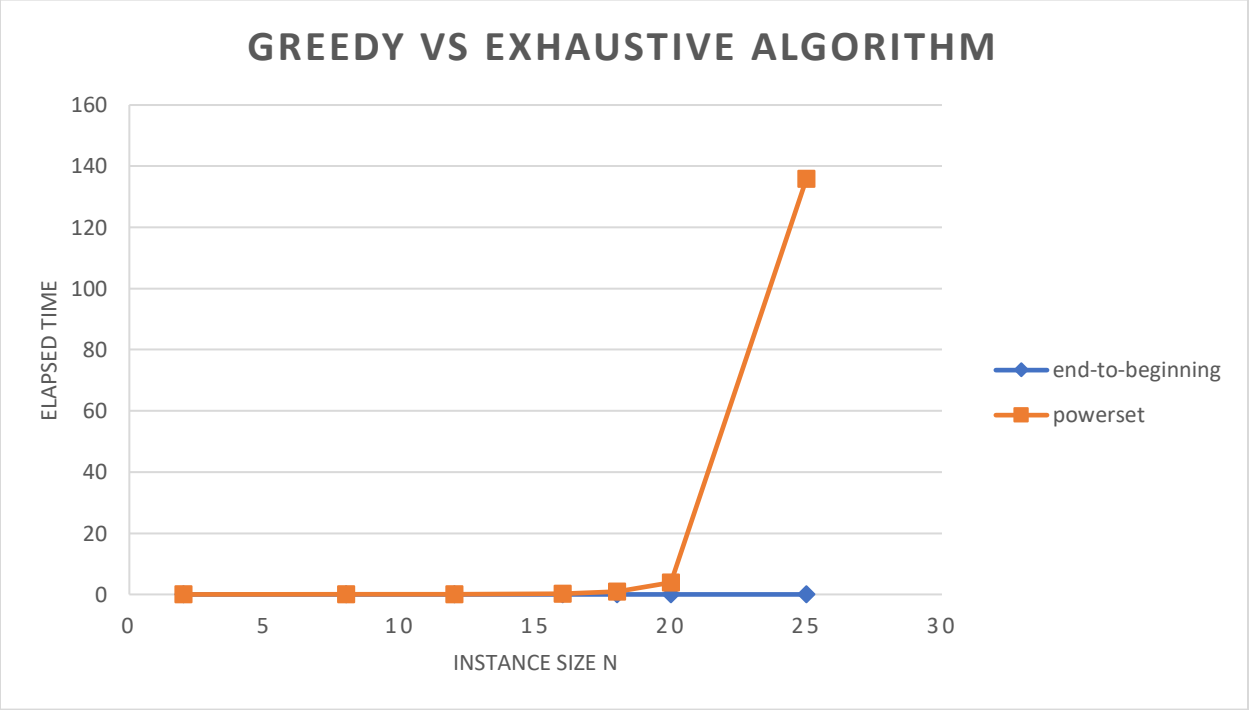
n = 8
[684, 559, 629, 192, 835, 763, 707, 359]
---
end to beginning
output = [835, 763, 707, 359]
of length = 4
elapsed time=8.704e-06 seconds
---
powerset
output = [835, 763, 707, 359]
of length = 4
elapsed time=0.000549804 seconds

n = 25
[684, 559, 629, 192, 835, 763, 707, 359, 9, 723, 277, 754, 804, 599, 70, 472, 600, 396, 314, 705, 486, 551, 87, 174, 600]

```
end to beginning
output = [835, 763, 707, 599, 472, 396, 314, 87]
of length = 8
elapsed time=1.6492e-05 seconds
----------------------------------------------------------------------------------
powerset
output = [835, 763, 707, 599, 472, 396, 314, 87]
of length = 8
elapsed time=135.917 seconds
```

.

- Draw a scatter plot and fit line for your timing data. The instance size n should be on the horizontal axis and elapsed time should be on the vertical axis. Your plot should have a title; and each axis should have a label and units of measure.



- Conclude whether or not your empirically-observed time efficiency data is consistent, or inconsistent, with your mathematically-derived big-O efficiency class.

From the graph we can tell that while end to beginning to remain consistent efficiency, powerset's graph show that time has rapidly increase with size of N while end to beginning stays the same running time.

Answers to the following questions, using complete sentences.

a. Provide pseudocode for your two algorithms. **(above)**
b. 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 longest nonincreasing end to beginning which is O(n^2) and in the class of nonincreasing powerset which is O(2^n), thus the efficiency class of longest nonincreasing end to beginning is faster than nonincreasing powerset.

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

The running speed is really noticeable difference, in the end to beginning algorithms time is O(n^2) is faster than the powerset algorithms O(2^n). Between these two algorithms running speed not only difference in small number when the array has more elements that will become a large difference for the running speed. And it might not surprise me because the powerset algorithms in my imagine is not a fast algorithm.

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

The fit lines is consistent because we can see from the graph that powerset graph increased with N, and end beginning to end stays the same.

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

The evidence is consistent with the hypothesis, the program does work correctly and produce correct output, also algorithms with exponential or factorial running times are extremely slow, in my test, when n = 25, is took around 2.3 minutes to run, so we can prove that it is too slow to use.