Longest\_nondecreasing\_end\_to\_beginning(vector A)

n = max size of A 1

Create vector H with n values being 0 1

For i = 0 to n – 2 n

For j = i + 1 to n n

If A[i] <= A[j] 1 + max(3, 0) = 4

If H[j] >= H[i] 1 + max(2,0) = 3

H[i] = H[j] + 1

End if

End if

End for

End for == 1 + 1 +(n\*n\* 4) + BOTTOM HALF

max = size of vector H + 1 2

Create vector R with max values n

J = 0 && index = max – 1 3

For I = 0 to n n

If H[i] equals index 1 + max(3,0) = 4

R[j] = A[i]

index - -

j++

end if

end for n \* 4 = 4n

TOP HALF + 2 + n + 3 + 4n

return R

1 + 1 +(n\*n\* 4) + 2 + n + 3 + 4n

= 4n2 + 5n + 7

= O(n2)

After calculating the step count for my pseudocode algorithm for end to beginning we can see a double nested for loop in the beginning of the algorithm, which can we see will be of n2 value. The final product of the step count will be 4n2 + 5n + 7 which once simplified down for big-Oh is O(n2).

longest\_nondecresing\_powerset(sequence A)

N = A.size()

Sequence best;

vector(n+1, 0)

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 1 to k

candidate.push\_back(A[stack[i]-1])

If candidate is nondecreasing and candidate size > best size

Best = candidate

Return best

Time Complexity:

longest\_nondecresing\_powerset(sequence A)

N = A.size() - 1

Sequence best; - 1

vector(n+1, 0) - 1

K = 0 - 1

while(true) -2k

If stack[k] < n - 1

Stack[k+1] = stack[k] + 1 - 1

++k - 1

Else

Stack[k-1]++ - 1

K-- - 1

If k == 0 - 1

Break - 1

Sequence candidate - 1

For 1 to k - k

candidate.push\_back(A[stack[i]-1]) - 2

If candidate is nondecreasing and candidate size > best size - 3

Best = candidate - 1

Return best - 1

7 + 1 + 4 + (2n) \* (n \* 2) + 5 = 10 + 7n + 7 + 2n =2n\*9n+ 17 = O()

is\_nondecreasing(sequence A)

For i in A:

If A[i-1] > A[i]

Return false

Return true

Time Complexity:

is\_nondecreasing(sequence A)

For 1 in A.size(): - n-1

If A[i-1] > A[i] - 1

Return false - 1

Return true - 1

(n-1) \* (1+1+1) = 3n-3 = O(n)

According to the step count, the resulting equation is 2n-2 which when simplified is O(n).

TIME ANALYSIS:

N = 15

End to beginning = 5.55 e -06 or 0.00000555

Power set = .08

N = 18

End to beginning = 4.48 e -05 or 0.0000448

Power set = .068

N = 20

End to beginning = 6.59 e -06 or 0.00000659

Power set = 3.24

N = 23

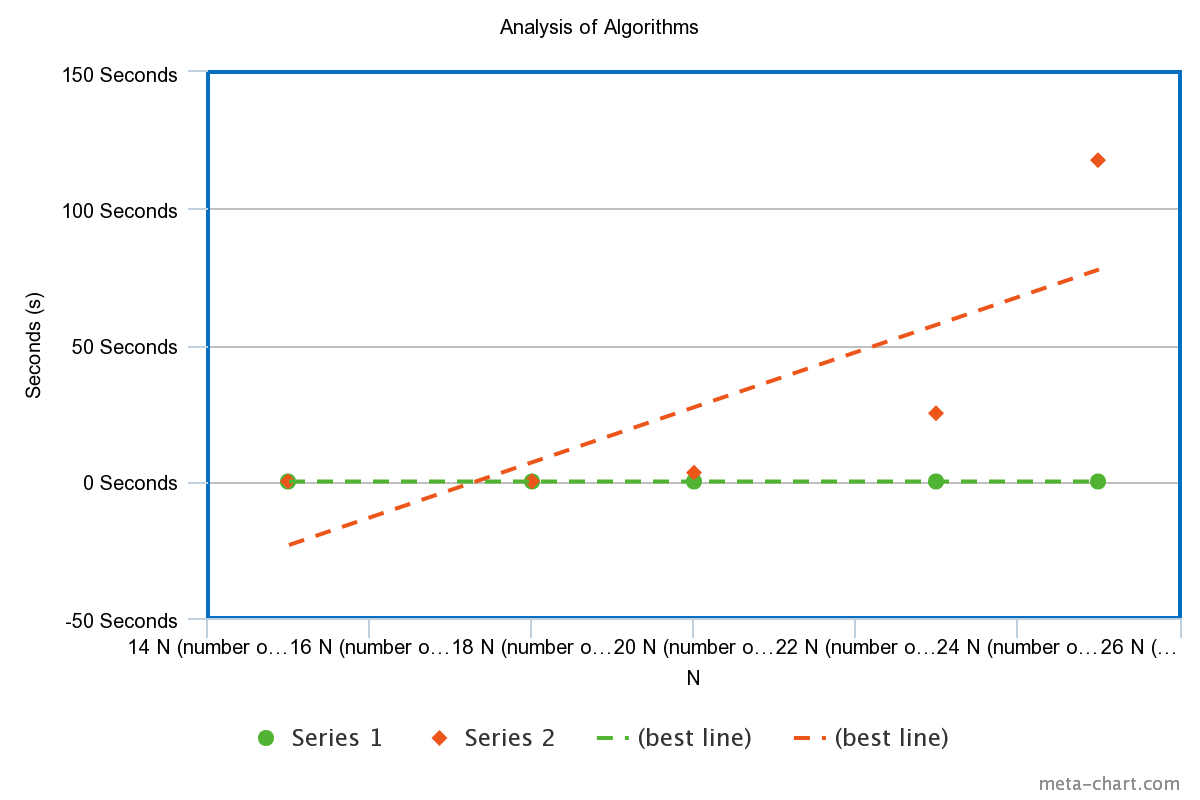
End to beginning = 8.48 e -06 or 0.00000848

Power set = 25 seconds

N = 25

End to beginning = 9.34 e -06 or 0.00000934

Power set = 117.719



Series 1 = End to Beginning Series 2 = Power Set

We can see from the graph and the best fit line that the first series, the end to beginning is a much more efficient algorithm compared to the power set algorithm. It remains at a fairly constant rate while the power set seems to be increasing as the number of inputs are increasing. When I made our n value to be 28 it never finished, and I lost patience and lowered it to 25. We can say that the empirically observed data is consistent with the Big Oh notation.