CPSC335 Project2 Report

Jorge Diaz jorgediaz@csu.fullerton.edu

Jacqueline Isabel Cardenas jacisac@csu.fullerton.edu

End-to -beginning Algorithm:

//Longest_nonincreasing_end_to_beginnning

n=A.size()

create vector H with 0 values

for i= n-2 to 0

for j = i+1 to n

if A[i] greater than or equal to A[j] and H[i] is less than H[j]+1

do

H[i] = H[j]+1

//calculate max length of longest nonincreasing subsequence

max= (max Value in H)+1

//allocate space for subsequence

create vector R

index=max-1, j=0

for i=0 to n

if H[i] == index

do

R[j] = A[i]

Index - -

j++

return sequence R

End-to-beginning Proof:

n=A.size --------------------------------2

vector H --------------------------------1

n-2-0+1 =n-1

i+1-n+1=i-n+2

if condition – 2+2=4

Σ()=4(n-1)=4n-4i -n

4n(0-n+3)-4(0*1/2)=-----------------4n^2+12n

max=(max value of H)+1------------2

n-0+1= n+1

vector R --------------------------------1

index=max-1 --------------------------2

j=0 --------------------------------------1

if condition - 1+3=4

4 (n+1) =---------------------------------4n+4

return sequence R -------------------1

Sc = 4n^2 +12n +4n+4+10

Efficiency= $O(n^2)$

CPSC335 Project2 Report

Exhaustive Algorithm:

//longest_nonincreasing_powerset

n=A.size

best sequence creation

stack creation

k=0

while

if stack[k] less than n

do

stack[k+1] = stack[k]+1

++k

Else

Stack[k-1]++

k- -

If k==0

Break;

Candidate sequence creation

For i=1 to k

do

candidate.push_back

if candidate.size() less than best.size and is_nonincreasing(candidate)

then

best=candidate

return best

Exhaustive Algorithm Proof:

n=A.size-------------------2

best sequence-----------1

stack-----------------------1

CPSC335 Project2 Report

k=0-------------------------1

while loop--------2^n

if condition-1+3=-4

else-1+2=----------3

if condition-1+1=2

for- 1k--------------k

if condition-2+1--3

return best--------------1

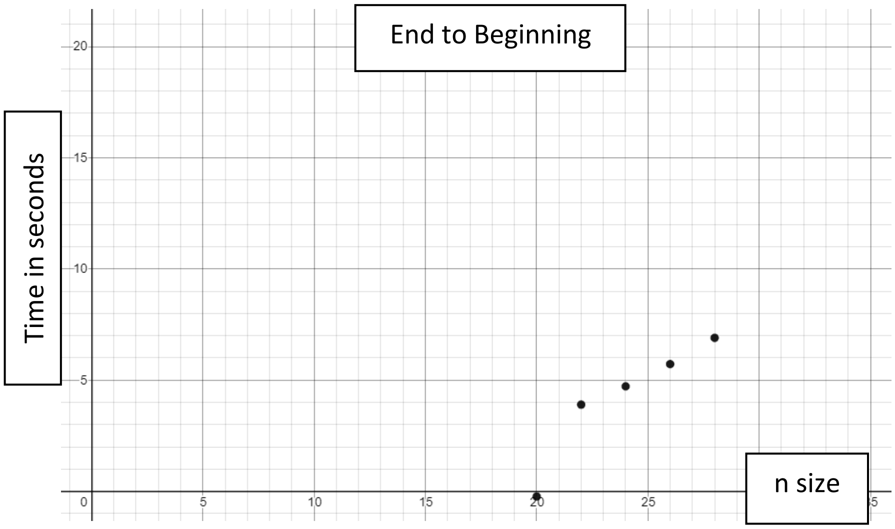
4+3+2+k+3=12+k

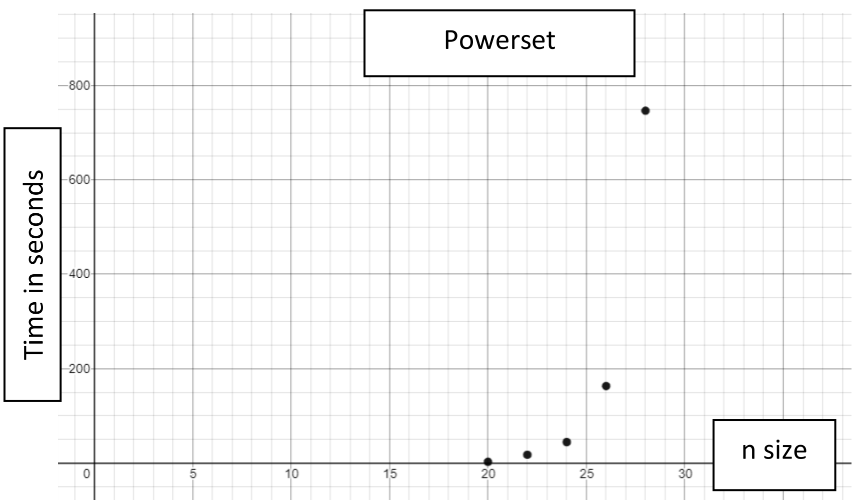
12+k*(2^n)=n+12*2^n

N+12*2^n+6

Efficiency= $O(n*2^n)$

CPSC335 Project2 Report





3b. The greedy algorithm complexity is $O(n^2)$ which is faster than the complexity of the exhaustive algorithm $O(n*2^n)$.

3c. The algorithm that is faster is that of the end-to-beginning. This is expected since this is a greedy algorithm that examines data only until it finds a candidate with satisfying requirements. On the other hand, the powerset algorithm which is exhaustive will examine all the data. Between these two algorithms there is a significant difference of $2^n/n$.

3d. The scatter plot graph shows how the greedy algorithm is much faster than the exhaustive algorithm. These graph lines are consistent with both the efficiency classes of the greedy algorithm, $O(n^2)$, and the efficiency of the exhaustive algorithm, $O(n*2^n)$, which is way slower. It can be seen just how the greedy algorithm shows a linear increasement, while the exhaustive algorithm just keeps getting longer and longer as inputs went up since it tests all values of the data.

3e. With all these evidence it just proves how all this data is consistent with the hypothesis stated on the first page that exhaustive search algorithms are feasible to implement, while producing correct outputs and that algorithms with exponential/ factorial running times are extremely slow for practical use. The correct output of an exhaustive algorithm is further supported by the program passing of the tests. While the slow execution of the exhaustive algorithm just proves how factorial and exponential running times are not of a practical use.