Jessica Vargas
mvrgsj@csu.fullerton.edu
Dylan Nguyen
dylan-nguyen@csu.fullerton.edu
335-04

Team Yuh:
Project 4

Dynamic Max Defense

Pseudocode:

```
vector(maxsize, vector (total_cost, value of 0))

Def dynamic_defense(Armors)
        For i= 1 to maxsize
                For j = 1 to maxcost
                        T[i,j] = -infinity
                endfor
        Endfor

        For i =1 to max sizedo
                For j  =1 to maxcost do
                        T[i,j] = max(T[i-1,j], T[i-1,J-weight] + value]
                Endfor
        endfor
                Return T[n,W]
```

$$\sum_{i=1}^{n}(1+\sum_{j=1}^{m}2)$$

$$\sum_{i=1}^{n}(1+2(m-1+1))$$

$$\sum_{i=1}^{n}(2m-2+2+1)$$

$$\sum_{i=1}^{n}(2m+1)$$

$$2m(n-1+1)+1(n-1+1)$$

$$2m(n)+1(n)$$

2mn+n -> 2mn

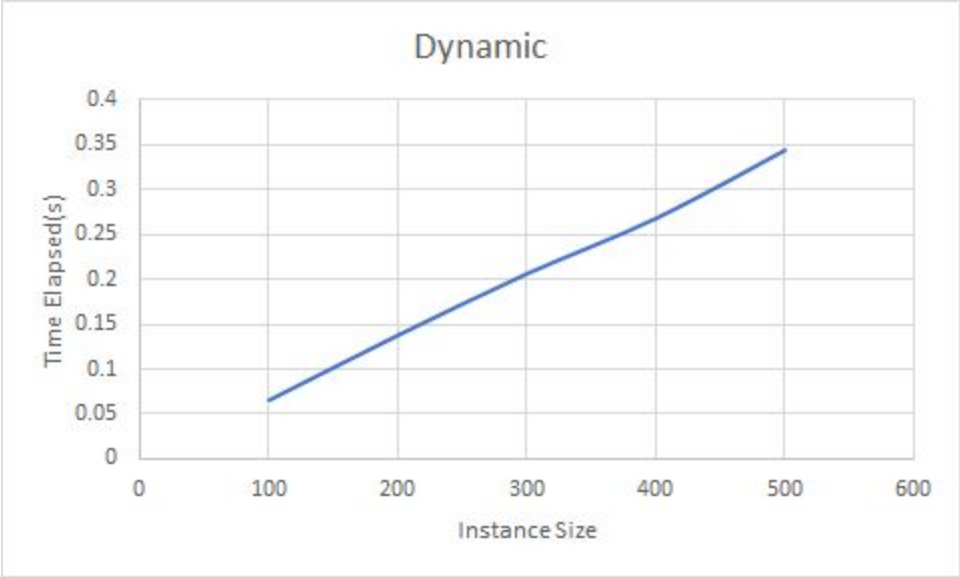
$$O(mn) = O(n^2)$$

Mathematical analysis:

For loop goes from 1 to n with one inner for-loops. The inner loop has a step count of 2. Evaluate the sigmas using C(Last +First +1) add 1 to the inner loop and get 2m+1. Evaluate outer sigma to 2mn+n. As m and n just represent a number for some value, they can be equated the same in this instance. Therefore 2mn is the same as 2n^2. Dropping to dominant terms we get 2n^2 and dropping the multiplicative constant of 2 we get n^2. Therefore the time complexity is O(n^2).

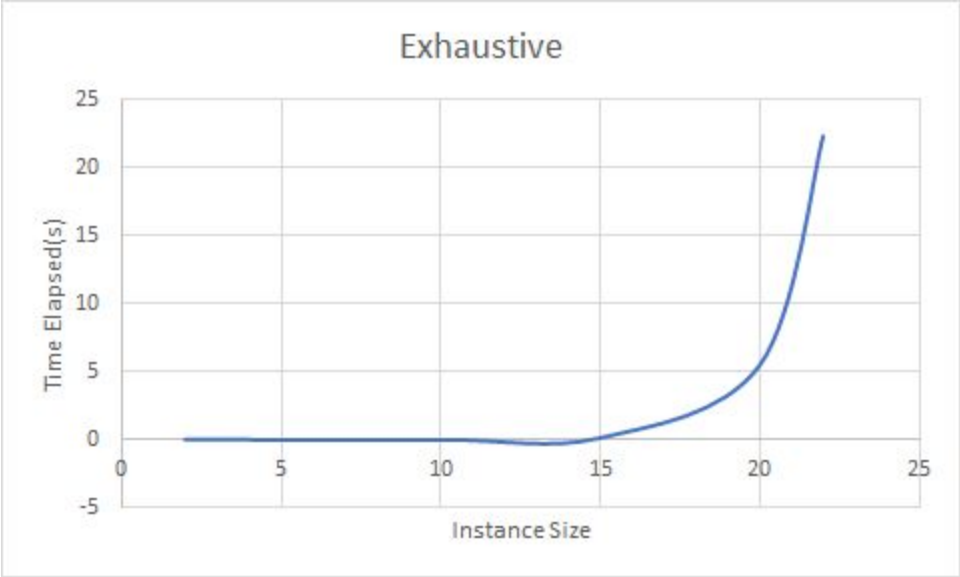
Dynamic Graph

item	time
100	0.064649
200	0.137121
300	0.205689
400	0.267398
500	0.343345



Exhaustive Graph

items	time
2	0
10	0.002939
15	0.133212
20	5.46038
22	22.3914



Exhaustive Max Defense

Pseudocode:

```
# greedy
bestset = None |1
candidate = None |1
for x in subSetAmt: |1

Set candidate = None |1
  for armor in armory:  |1

            if index & 1 << armor|1
              candidate = armor|1
            endif
endfor
            sum_armor_vector(candidate)|1
  if defense > best && budget < total_cost:|1
            best = candidate|1
            endif
endfor
            return best|1
```

SC= Sigma 0 to n (4+ Sigma 0 to m (2(m-0+1)))

3+ (n-0 (4+ 2m+1))= 3+4n+2mn +n-> 2mn + 5n +3 ->n = 2^n

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

Mathematical Analysis:

For the loop 0 to N, there is another nested “for-loop” for 0 to m. After calculating the inner “for-loop” steps to be 2m+1, we add that to the steps of the outer “for-loop” and come out with a subvalue of 2mN +5N + 3. If we take a look at n represents, N is the amount of subsets that are being made for the list. This value will always be 2^the amount of items in the list, or for short 2^n. M simply represents the amount of items in a list. Thus we can convert 2mN +5N + 3 to 2(2^n)m + 5(2^n) + 3. Dropping to the dominant term we get 2^n * m, or rather O(n * 2^n) as n and m are equivalent in this instance.

Conclusion

a. Is there a noticeable difference in the performance of the two algorithms? Which is faster, and by how much? Does this surprise you?
   Yes there is a noticeable difference in the performance of our two algorithms. For all sizes, dynamic tends to be the faster executing algorithm, by a significant amount. No, because the time complexity of our dynamic algorithm is O(n^2), while our exhaustive is O(n*2^n). We used the complexities to predict that dynamic would generally be faster and it has proven true.
b. Are your empirical analyses consistent with your mathematical analyses? Justify your answer.
   As stated briefly above, and with our graphs, the data collected is consistent with our predictions on speed and with our mathematical analysis. The best fit lines are consistent with the time complexities that were found for each algorithm, that of which being O(n^2) and O(n*2^n). The best fit of the exhaustive follows the expectations that it will be exponential, while the best fit of the dynamic follows a linear path.
c. Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.
   We have created an exhaustive search algorithm that is implementable and is able to create correct outputs. So the evidence is consistent with hypothesis 1. Our exhaustive search algorithm goes through every possible set of a given power set and chooses the optimal subset available.
d. Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.
   Hypothesis 2 states that algorithms with exponential run times are too slow for practical use. This hypothesis is mostly true. Using exponential run time algorithms are still practical for small sets of inputs, but for greater sizes it is

inefficient. One can assume an algorithm to be of “practical use,” meaning all inputs will be run efficiently, so yes, the evidence is consistent that exponential run time algorithms are too slow for full practical use.