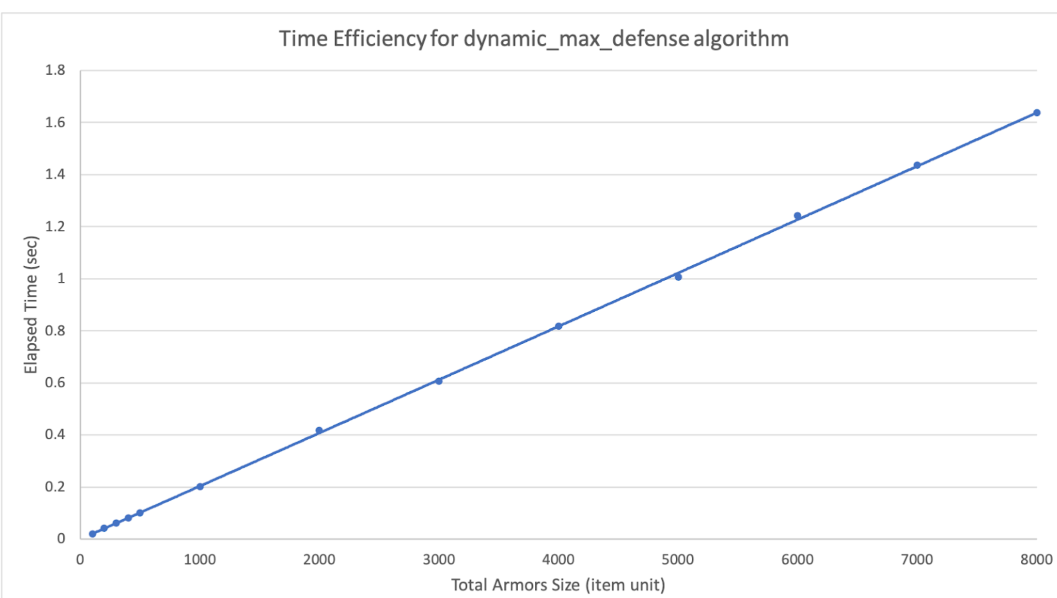
CPSC 335- Algorithm Engineering
Fall 2020
Instructor: Mike Peralta
Project 4: Team 3

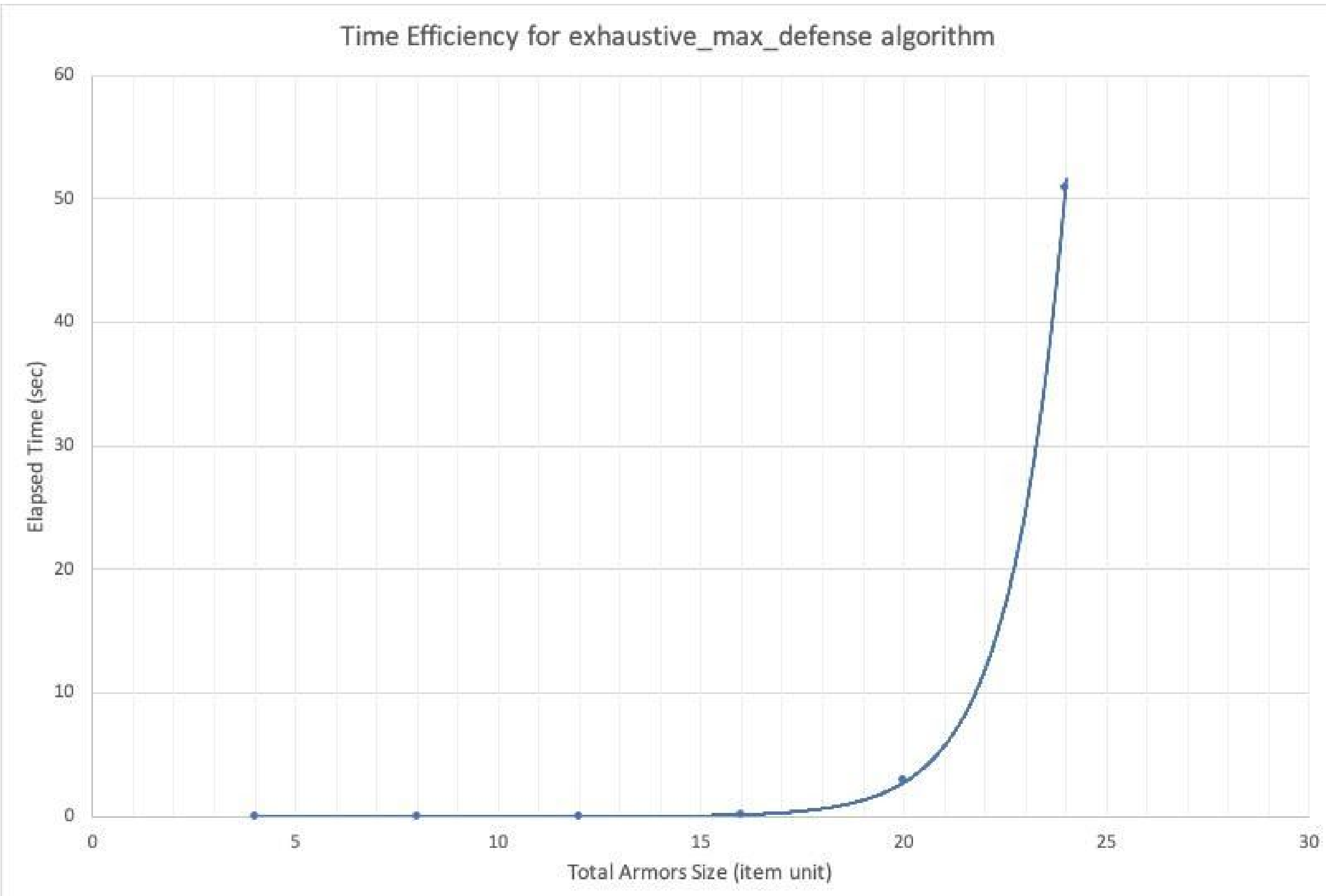
Pearl Law (pearl.law@csu.fullerton.edu)
Brandon Ruiz (bruiz19@csu.fullerton.edu)

Scatter Plots:

Dynamic Algorithm	
N	**Elapsed Time (s)**
100	0.0202088
200	0.0411272
300	0.0601195
400	0.081332
500	0.100019
1000	0.200081
2000	0.417311
3000	0.606565
4000	0.817612
5000	1.00757
6000	1.24319
7000	1.43541
8000	1.63788



Exhaustive Algorithm	
N	**Elapsed Time (s)**
4	2.45E-05
8	0.000378521
12	0.00777177
16	0.152135
20	2.85002
24	50.8085



This experiment will test the following hypotheses:

1. **Exhaustive search algorithms are feasible to implement, and produce correct outputs.**
2. **Algorithms with exponential running times are extremely slow, probably too slow to be of practical use.**

Questions:

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

   There is a noticeable difference in the performance of the two algorithms, for this particular algorithm solution it appears that the exhaustive optimization solution is of: O(2^n * n) and while the dynamic optimization solution is of O(n * W). Taking a look at the value used above for the graphs, as the value of n becomes larger, the dynamic solution becomes more notable when it comes to being more efficient.

b. Are your empirical analyses consistent with your mathematical analyses? Justify your answer.

   Our empirical and mathematical analysis are consistent for both algorithms. The dynamic_max_defense scatterplot matches that of O(n * W), which aligns with the mathematical analysis below:

   <u>Dynamic Algorithm Pseudocode:</u>

```
dynamic_max_defense(armor_items, total_cost)
{
    int items_length = length of armor_items

    // initialize 2d array to store values
    int A[items_length + 1][total_cost + 1]

    // build table to find maximized value
    for i = 0 to items_length // loop through rows
        for j = 0 to total_cost // loop through columns
            if (i = 0 or j = 0)
                A[i][j] = 0
             else if (armor_items[i – 1].cost <= j)
                A[i][j] = max(armor_items[i – 1].defense + A[i – 1][j –armor_items[i-1].cost], A[i-1][j])
             else
                A[i][j] = A[i – 1][j]

    // find optimized list of armor items with max defense within total cost
    int i = items_length
    int j = total_cost
    result = empty vector with int data type

    while (i > 0 and j > 0)
      if (A[i][j] > A[i – 1][j])
```

```
        insert armor_items[i-1] into result
            j = j - armor_items[i – 1].cost
    i = i - 1
```

Dynamic Max Defense Time Complexity:

1) Dependent nested for loop:

    Inner for loop steps:
    for statement – 1 step
    if statements: 1 + max(1, 2, 3) = 4 steps
    Step count = 5 steps

    Outer for loop steps:
    Let n = items_length, W = total_cost
    Step count =

    Total steps nested for loops = **5 + (5W + 5) * (n + 1)**

2) While loop:
    while (i > 0 and j > 0) equivalent to: for i & j to 1
    # loops = (n – 1)/1 + 1 = n times
    Loop block step count = if statement + decrement i = 1 + max(2, 0) + 1 = 4 steps

    Total steps = n * 4 = 4n steps

3) Initializations/Return:
    int items_length – 1 step
    int A[][] – 1 step
    int i – 1 step
    int j – 1 step
    result – 1 step
    return (result) statement – 1 step

    Total steps = **6 steps**

Time Complexity of Dynamic Algorithm:
Nested for loops + while loop + initialization/return
= 5 + (5W + 5) * (n + 1) + 4n + 6
= **(5W + 5) * (n + 1) + 4n + 11**

After dropping constants, dominated terms, and multiplicative constants, the step count for the entire algorithm is W * n. Thus, the time complexity efficiency class the dynamic algorithm belongs to is O(W * n).

The exhaustive_max_defense scatterplot is consistent with the mathematical analysis. The mathematical analysis below shows that this algorithm would be significantly slower and take a longer period of time to complete a search than the greedy algorithm would have done in double the amount of time:

Exhaustive Optimization Pseudocode:

```
exhaustive_max_defense(G, armor_items):
        n = |armor_items|
        best = None
                for bits from 0 to (2n -1):
                candidate = empty vector
                for j from 0 to n-1:
                if ((bits >> j) & 1) == 1:
                candidate.add_back(armor_items[j])
if total_gold_cost(candidate) <= G:
        if best is None or
        total_defense(candidate) > total_defense(best):
best = candidate
return best
```

Exhaustive Max Defense Time Complexity:

exhaustive_max_defense(G , armor_items):
//size of |armor_items|

n = |armor_items|
best = None
// loop for (O to 2^n - 1) : Time complexity of this loop = O(2^n)
// Time complexity of function = O(2^n)
for bits from 0 to (2^n-1):
candidate =empty vector
// loop form 0 to n-1: Time Complexity of this loop = O(n)

// Time Complexity of Function : O(2^n *n)
for j from 0 to n-1:
// Right Shift of Bits by j = Bits/(2^j) : Time Complexity = O(1)
if ((bits >> j) & 1) == 1:
candidate.add_back(armor_items[j])
// Time Complexity of Function : O(2^n*n*1)
//Time Complexity of below If Else if O(1)

if total_gold_cost(candidate) <= G:
if best is None or
total_defense(candidate) > total_defense(best):
best = candidate
return best
Time Complexity of Function : O(2^n * n * 1) = O(2^{n} * n).

After dropping constants, dominated terms, and multiplicative constants, the step count for the entire algorithm is 2^n * n. Thus, the exhaustive optimization algorithm belongs to O(2^n * n) efficiency class.

c. Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.

   The evidence is in fact consistent with our hypothesis 1 in which the dynamic programming algorithm is more efficient than the exponential exhaustive search algorithm for the same problem. Taking a look at the created graph a more visual approach to this evidence is seen when both algorithm approaches are compared.

d. Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.

   The evidence is also consistent with our hypothesis 2, however, the implementation of the dynamic algorithm was a bit more complex in which the minor details on how the algorithm was implemented needed to be correct in what exactly was being stored, if not then minor errors will begin to occur. Although the exhaustive optimization algorithm was shorter and easier to implement making it more of a viable choice, the dynamic algorithm's efficiency was superior to that of the exhaustive in time complexity making it the ideal choice regardless of its lengthy implementation compared to the exhaustive optimization.