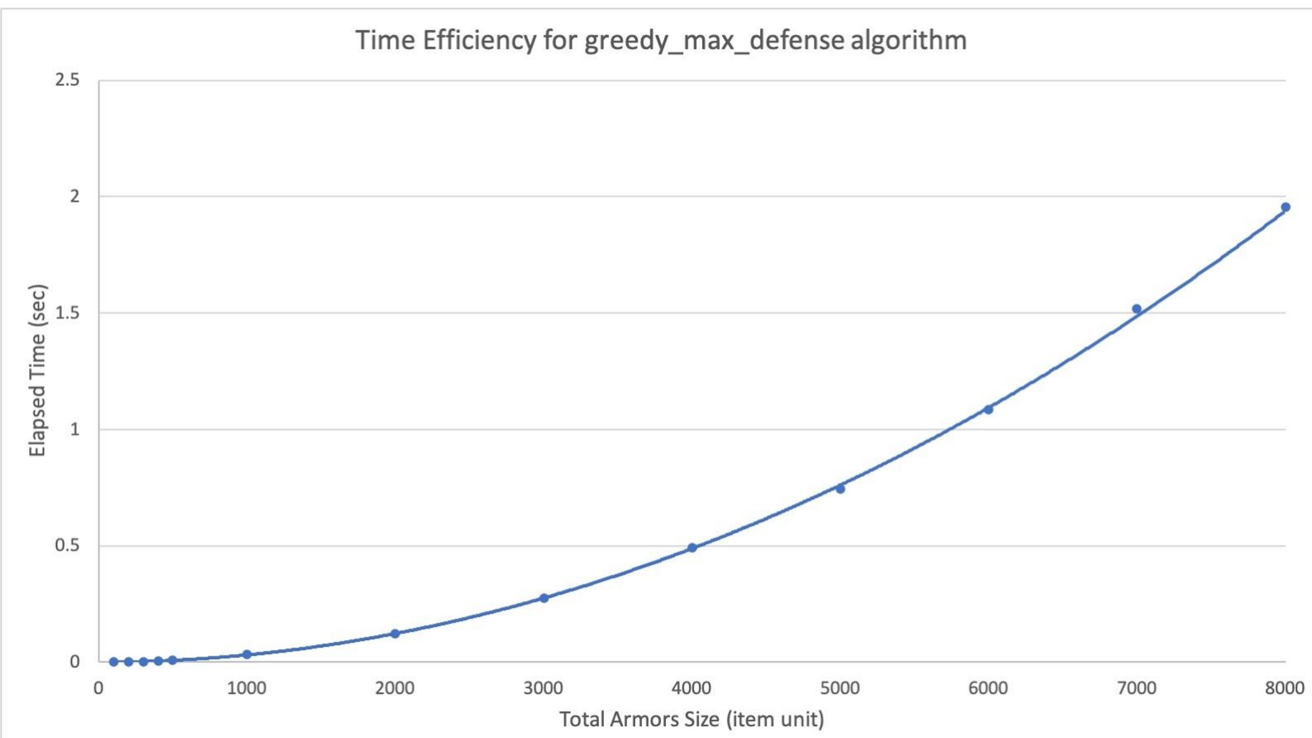
CPSC 335- Algorithm Engineering
Fall 2020
Instructor: Mike Peralta
Project 2: Team 3

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

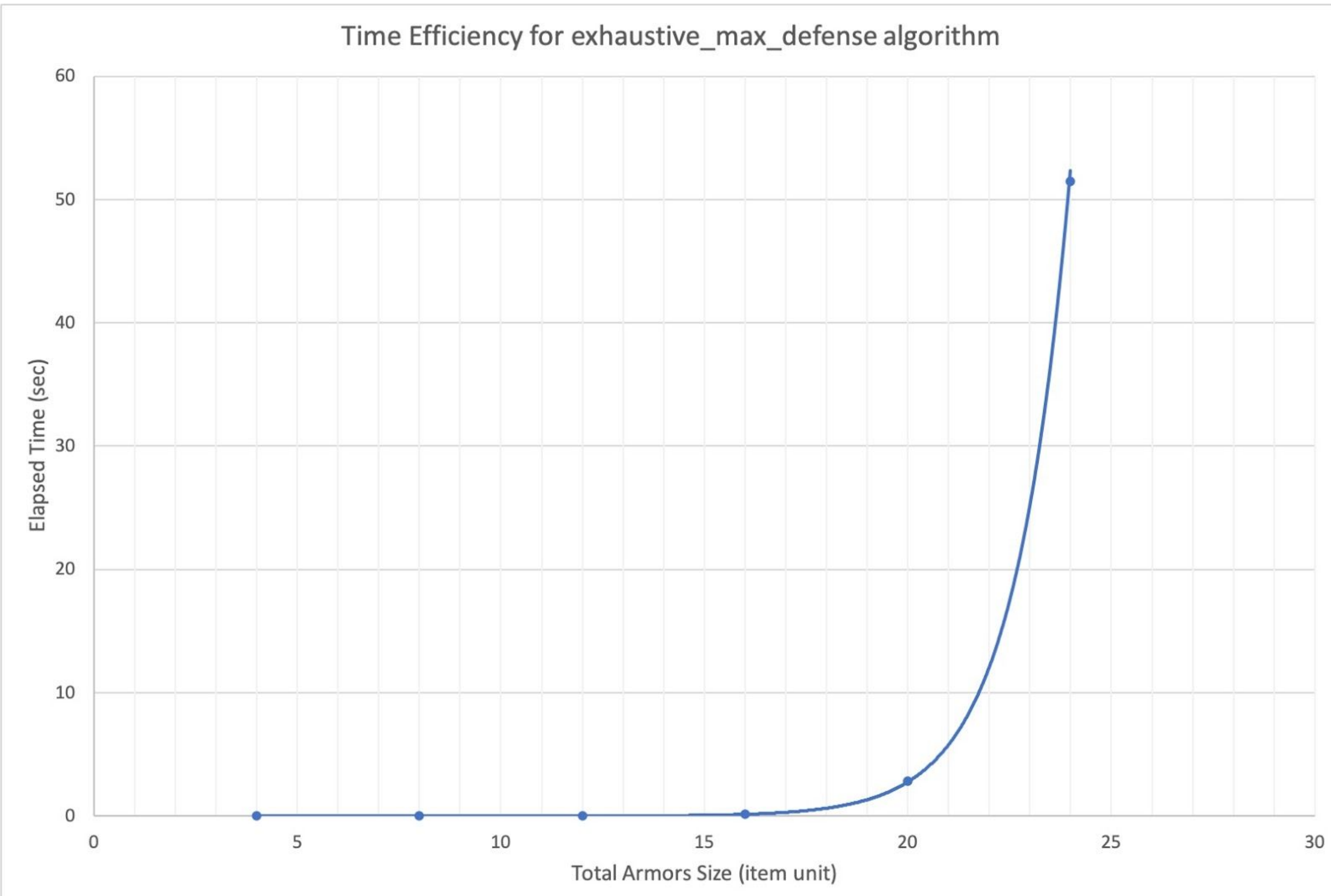
Scatter Plots:

Greedy Algorithm

N	Elapsed Time (s)
100	0.000332448
200	0.00123674
300	0.00275334
400	0.00480554
500	0.00757559
1000	0.0322792
2000	0.121298
3000	0.275597
4000	0.492779
5000	0.744627
6000	1.08581
7000	1.51913
8000	1.95448



Exhuastive Algorithm	
N	**Elapsed Time (s)**
4	2.21E-05
8	0.000375421
12	0.00768081
16	0.148741
20	2.80448
24	51.4657



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 significant difference in the performance of the two algorithms in which one is extremely faster than the other. The greedy algorithm is much faster than the exhaustive algorithm by almost double the amount. It didn't really surprise us since, as opposed to the greedy algorithm, the exhaustive algorithm has the restriction of n < 64 in the requirements, making the process by which the algorithm would be carried out to be less efficient than the greedy algorithm.

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

   Our empirical and mathematical analysis are consistent for both algorithms. The greedy_max_defense scatterplot matches that of O(n^2), which aligns with the mathematical analysis below:

   Greedy Algorithm Pseudocode:

```
greedy_max_defense(G, armor_items):
      todo = armor_items
      result = empty vector
      result_cost = 0
      while todo is not empty:    // for i = todo.length (or n) - 1 to 1
            max_item_index = 0
            for (i = 1 to todo.length - 1)
                  current_item = todo[i]
                  if (todo[max_item_index] < current_item)
                        max_item_index = i
                        todo[max_item_index] = current_item
            todo.remove(a)   // todo length -= 1
            g = a's cost
            if (result_cost + g) <= G:
                  result.add_back(a)
                  result_cost += g
      return result
```

   Greedy Max Defense Step Count:

   for loop inside while loop:

   # loops = = n – 1 iterations

   s.c = initialize current_item (1 step) + inner if block (3 steps) = 4 steps

   if block inside for loop:

analyze if statement = 1 step
then = 2 steps
else = 0 steps
$s.c_{\text{if statement}} = 1 + \max(2, 0) = 3$ steps

$s.c_{\text{for loop}}$ = analyze for statement (1 step) + (# loops * s.c) = 1 + (4 * (n – 1)) = **4n – 3 steps**

<u>if block inside while loop:</u>
analyze if statement = 1 step
then = 2 steps
else = 0 steps
$s.c_{\text{if block}} = 1 + \max(2, 0) =$ **3 steps**

<u>while loop:</u>
loops = ((1 – (n – 1))/1) + 1 = 3 – n iterations
initialize max_item_index, g, and remove() function call = 3 steps
$s.c_{\text{while block}} = 3 + s.c_{\text{for loop}} + s.c_{\text{if block}} = 3 + (4n - 3) + 3 =$ **4n + 3 steps**
$s.c_{\text{while loop}} = \# \text{ loops} * s.c_{\text{while block}} = (3 - n) * (4n + 3) = |12n + 9 - 4n^2 - 3n| =$ **$4n^2 + 9n + 9$ steps**

<u>Entire algorithm step count:</u>
initialize todo, result, result_cost, and return statement call = 4 steps
$4 + s.c_{\text{while loop}} = 4 + 4n^2 + 9n + 9 =$ **$4n^2 + 9n + 13$ steps**

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

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:

<u>Exhaustive Optimization Pseudocode:</u>

```
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.

Hypothesis 1 states that exhaustive search algorithms are feasible to implement and produce the correct output. Our evidence does not support this hypothesis, and instead shows that the greedy algorithm was more feasible and efficient than the exhaustive algorithm. Additionally, we were able to visually see that the exhaustive scatter plot had a steeper, rapidly growing slope compared to the greedy algorithm. This indicates that, despite producing a correct output, the feasibility of the exhaustive algorithm, given large n values, becomes compromised.

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

Our evidence, as shown by the exhaustive_max_defense graph, is consistent with hypothesis 2, which states that algorithms with exponential running times are extremely slow and too slow for practical use. We can see that the running times for the first three n values are not significantly different. However, when n = 20, it takes almost 3 seconds to run and even more so when n = 24 (51.4 seconds). The exponential difference in time (51.4 - 2.8 = 48.6 second difference) observed by adding 4 more n items demonstrates how this algorithm will fail to perform efficiently in practice when given much larger n values.