Chanmolita Nuon
molitanuon36@csu.fullerton.edu

Project 2

Pseudocode for the greedy algorithm and Big-O Efficiency

```
def greedy_max_defense(armors, total_cost):
 todo = copy of vector armors                               1
result = empty vector                                       1
result_cost = 0                                             1
maxDefense = 0                                              1
defPerCost = 0                                              1
while(!todo->empty())                                       n-1
{
        maxDefense = 0.0;                                   1
        for int i = 0 to  n - 1 do                          n
        {
                defPerCost = armor at i defense/cost        1
                if(maxDefense < defPerCost )            1 + max(2, 0) = 3
                {
                        maxDefense = defPerCost;
                        save index
                }
        }

        choosenArmor = todo at index                                1
        if ( (result_cost + choosenArmor cost) <= total_cost) {     2 + max(2, 0) = 4
                result->push_back(choosenArmor);
                result_cost += choosesArmor cost();
        }
        todo->erase(armor at index);                                1
}
        return result;                                              1
}
```

SC : 5 + (n-1) [1 + n(4) + 5 + 1]+1 = 6 + (n-1)(4n + 7) = 6 + 4n^2 +3n - 7 = 4n^2 + 3n -1

Time complexity: O(n^2)

Proof that the step count belong to O(n^2) using limits

4n^2 + 3n -1 ∊ O(n^2)
lim n to inf 4n^2 + 3n -1/(n^2)

= lim n to inf (4n^2/n^2) + lim n to inf (3n/n^2) + lim n to inf (-1/n^2)

= lim n to inf (4) + lim n to inf (3/2n) + lim n to inf (-1/n^2)
= 4 >= 0 and a constant therefore 4n^2 + 3n -1 ∊ O(n^2)

Pseudocode for the exhaustive algorithm and Big-O Efficiency

```
def exhaustive_max_defense(armors, total_cost):
        cand_defense = 0;                                   1
        total_gold_cost = 0;                                1
        best_defense = 0;                                   1

        best = empty vector;                                1
        candidate = empty vector;                           1

        for int bits = 0 to 2^n - 1  do                     2^n
        {
                empty candidate                             1
                for int j = 0 to n - 1 do                   n
                {
                        if(((bits >> j) & 1) == 1)          3 + max(1, 0) = 4
                                add armor at j to candidate
                }

                sum_armor_vector(*candidate, total_gold_cost, cand_defense);        1

                if(budget <= total_cost)                                            1
                {
                        if (best == nullptr || candidate defense > best_defense)      3 + max(1,0)
                        {
                                best = candidate
                        }
                }
        }
        return best;                                                        1
}
```
SC : 5 + (2^n)[1+ n(4) + 1 + 4] + 1 = n*2^(2+n) + 6 *2^n + 6

Time complexity: O(2^n * n)

Proof that the step count belong to O(2^n * n) using limits

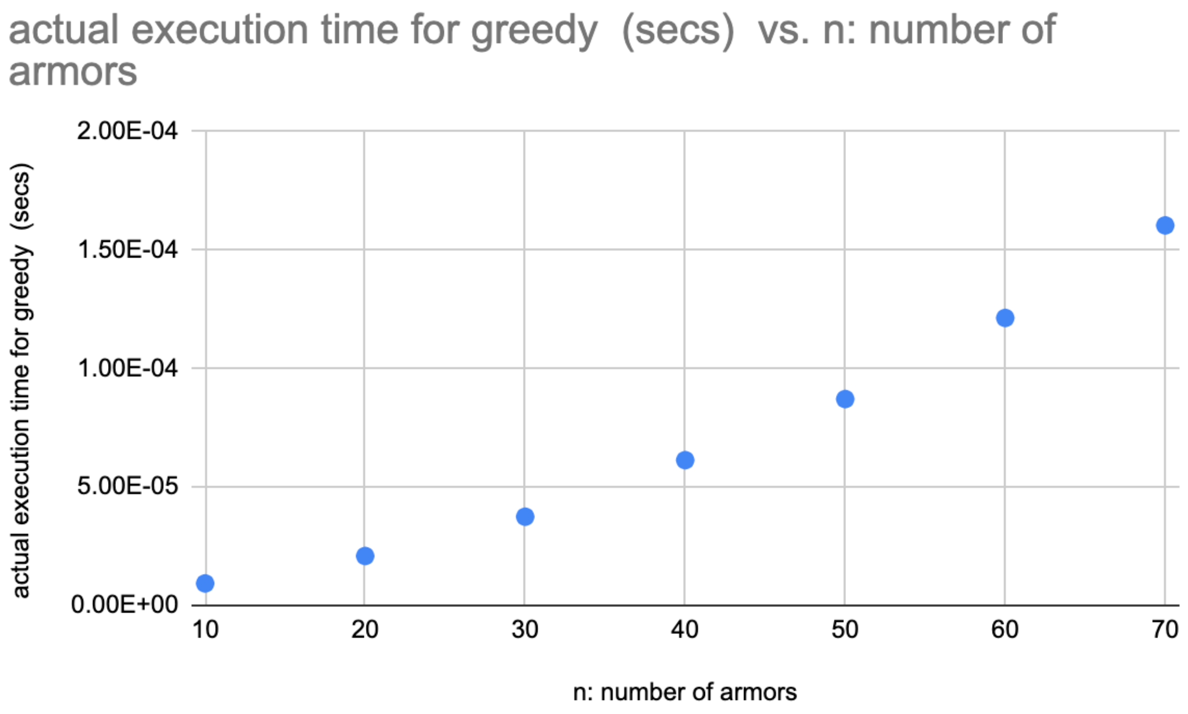
lim n to inf (n*2^(2+n) + 6 *2^n + 6 / (2^n*n))

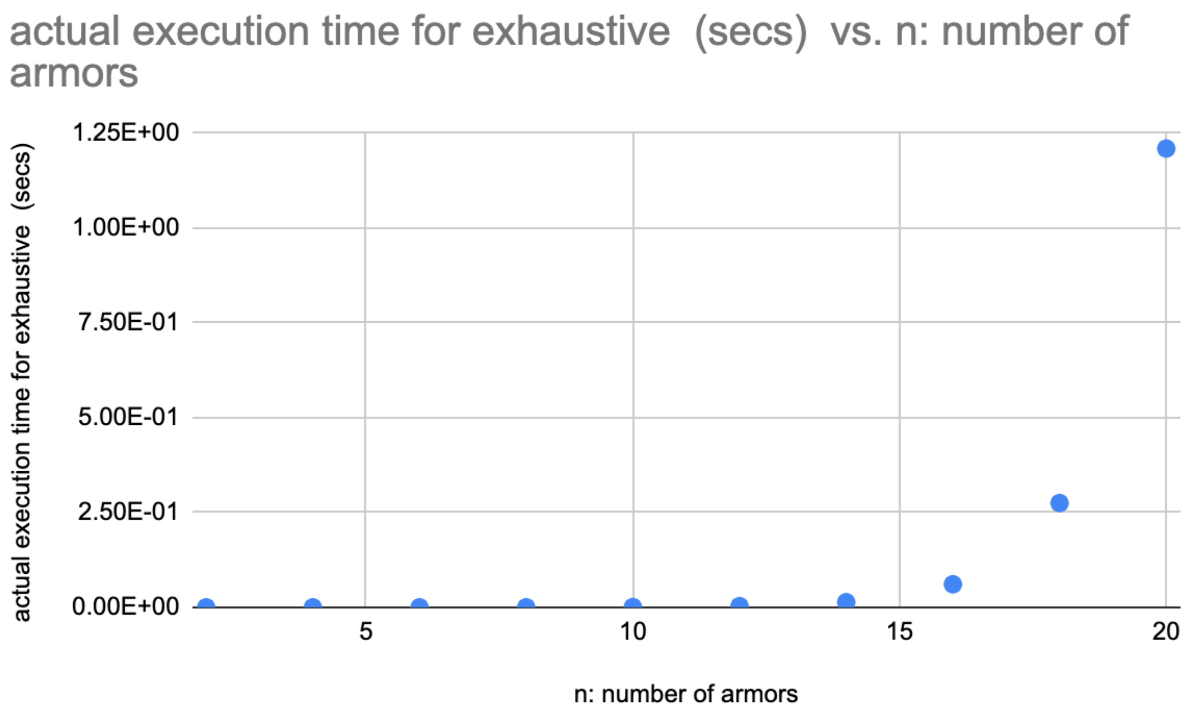
= lim n to inf(n*2^(2+n) / 2^n*n) + lim n to inf (6 *2^n / 2^n*n) + lim n to inf (6/ 2^n*n)

= lim n to inf(4) + lim n to inf (6/n) + lim n to inf (3* 2^(1-n)/n)

= 4 >= 0 and a constant therefore n*2^(2+n) + 6 *2^n + 6 / (2^n*n)

Scatter Plot





Question:

- 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, the greedy algorithm is increasing n^2 time while the exhaustive algorithm is trending like an 2^n*n. The greedy algorithm is faster by 9.65*10^-3 seconds. No, this does not surprise me.

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

My empirical analyses are consistent with my mathematical analyses because the graph for greedy is trending quadratically while the exhaustive algorithm is behaving like an exponent function.

- Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.
  This evidence is consistent with hypothesis 1 because the exhaustive search algorithm for this project was feasible to implement and does produce correct outputs, it just took longer to the point that I would not get an execution time from the experiment. My machine was unable to execute a time for armors greater than 20.

- Is this evidence consistent or inconsistent with hypothesis 2?Justify your answer.
  This evidence is consistent with hypothesis 2 because algorithms with exponential are extremely slow to the point where it is of no use. As proven in my graph, the exponential running time was slower than the quadratic algorithm.