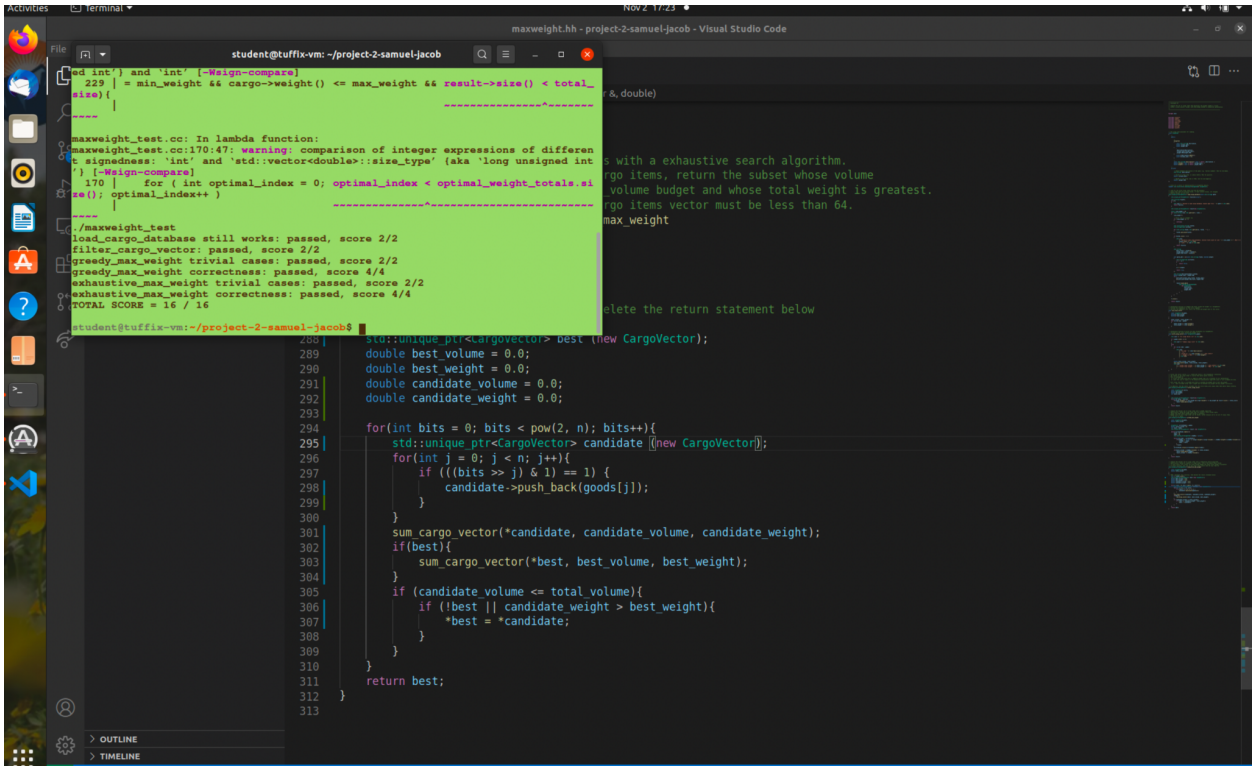
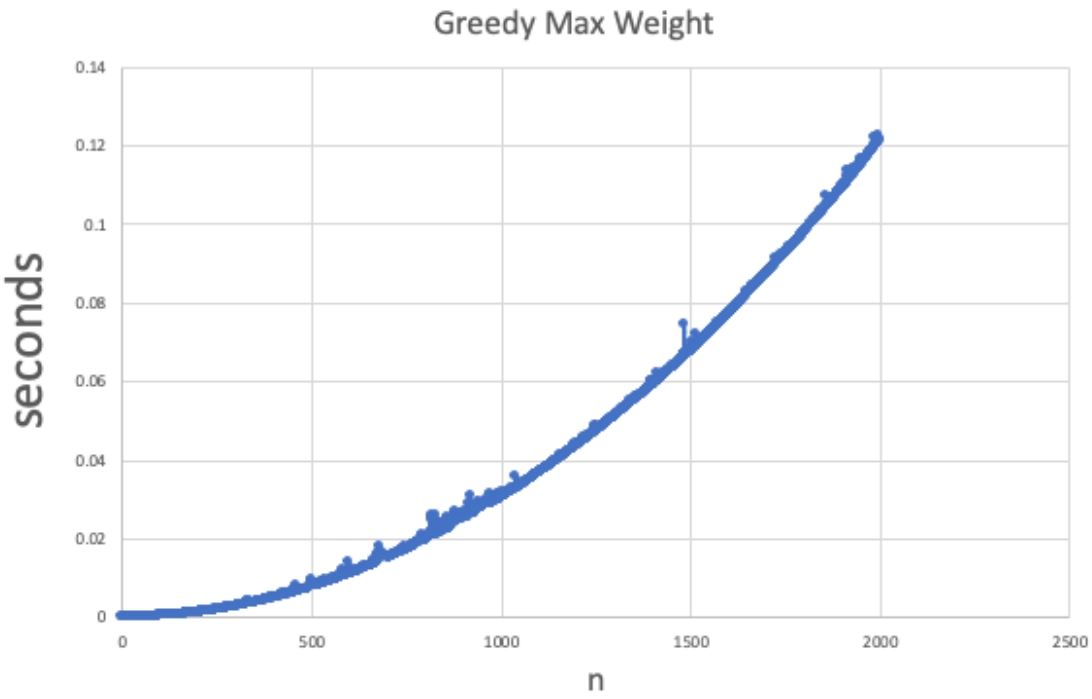
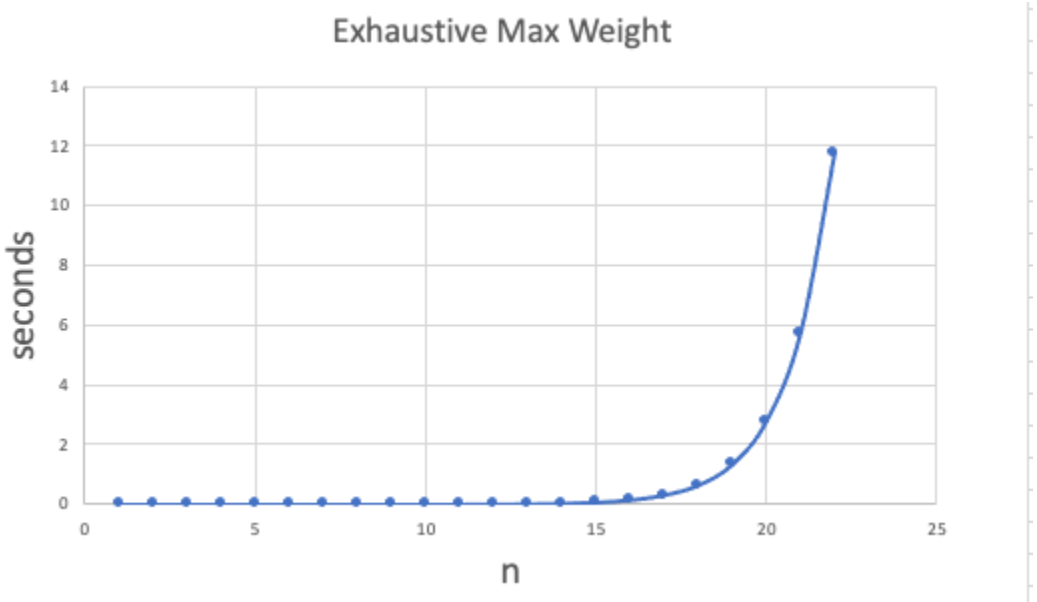
Project 2 Report

Samuel Sandoval, sjsandoval@csu.fullerton.edu
Jacob Nguyen, jn42@csu.fullerton.edu



Scatterplots





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's a noticeable difference in the performance of both algorithms. It seems that Greedy is much faster compared to exhaustive. Greedy is faster by 12 seconds, this is demonstrated by the scatter plots above.

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

Yes because the exhaustive csv generation took a lot longer than the greedy algorithm to generate and their Big-O reflects this as well.

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

It is consistent with hypothesis 1 because we were able to generate the solutions using each pattern.

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

Yes, the exhaustive optimization was way too slow and took a long time to generate the CSV

Notes

Requirements:

- Analyze your greedy algorithm code mathematically to determine its big-O efficiency class, probably $O(n^2)$ or $O(n \log n)$.
- Analyze your exhaustive optimization algorithm code mathematically to determine its big-O efficiency class, probably $O(2^n \cdot n)$.
- Gather empirical timing data by running your implementations for various values of n.
- Draw a scatter plot for each algorithm and fit line for your timing data. The instance size $n$should be on the horizontal axis and elapsed time should be on the vertical axis. Each plot should have a title; and each axis should have a label and units of measure.
- Conclude whether or not your empirically-observed time efficiency data is consistent, or inconsistent, with your mathematically-derived big-O efficiency class for each algorithm.

Scatter plot run: `g++ -std=c++17 maxweight_scatterplot.cc`

- Two scatter plots meeting the requirements stated above.
- Answers to the following questions, using complete sentences.

```
std::unique_ptr<CargoVector> greedy_max_weight
(
   const CargoVector& goods,
   double total_volume
)
{
   CargoVector currentGoods = goods; //1 tu
   double result_volume = 0; // 1 tu
   int count, index;
   std::unique_ptr<CargoVector> result (new CargoVector); // 1 tu

   while(!currentGoods.empty()){ //n
       count = 0; //1 tu
       index = 0; //1 tu
       std::shared_ptr<CargoItem> itemObj = nullptr; //1 tu
```

```
        for(auto& cargo : currentGoods){ //n
            if(itemObj == nullptr || (cargo->weight()/cargo->volume() >
itemObj->weight()/itemObj->volume())){ //5 tu
                itemObj = cargo; //1 tu
                index = count; //1 tu
            }
            ++count; //1 tu
        }
        currentGoods.erase(currentGoods.begin()+index); //1 tu

        if(result_volume + itemObj->volume() <= total_volume){ //2 tu
            result->push_back(itemObj); //1 tu
            result_volume += itemObj->volume();//1 tu
        }
    }
    return result;
}
```

$3 + n(3 + 9n) + 2 + 5 = 3 + 3n + 9n^2 + 7 = 9n^2 + 3n + 10$ tu

```
std::unique_ptr<CargoVector> exhaustive_max_weight
(
    const CargoVector& goods,
    double total_volume
)
{
// TODO: implement this function, then delete the return statement below
    int n = goods.size(); // 1 tu
    std::unique_ptr<CargoVector> best (new CargoVector);
    double best_volume = 0.0; //1 tu
    double best_weight = 0.0; //1 tu
    double candidate_volume = 0.0; //1 tu
    double candidate_weight = 0.0; //1 tu

    for(int bits = 0; bits < pow(2, n); bits++){ // 2^n
        std::unique_ptr<CargoVector> candidate (new CargoVector);
        for(int j = 0; j < n; j++){ // n
            if (((bits >> j) & 1) == 1) { //3 tu
                candidate->push_back(goods[j]);//1 tu
            }
```

```
        }
        sum_cargo_vector(*candidate, candidate_volume, candidate_weight);//n
        if(best){
            sum_cargo_vector(*best, best_volume, best_weight); //n
        }
        if (candidate_volume <= total_volume){ // 1 tu
            if (!best || candidate_weight > best_weight){ //2 tu
                *best = *candidate; //1 tu
            }
        }
    }
    return best;
}
```

5 + 2^n(n(3+1+max(0,n)+max(3, 4))) = 5 + 2^n(n(4 + 4 + n + n)) = 5 + 2^n(n(2n+8))= 2^n(2n^2 + 8n) + 5