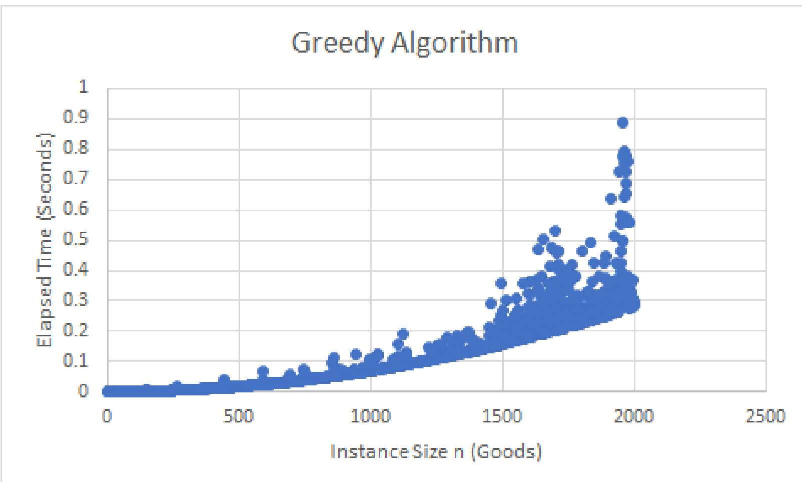
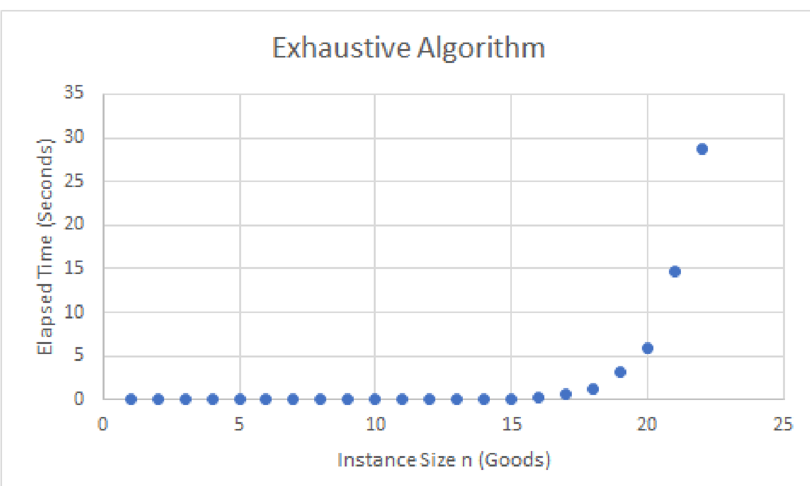
CPSC 335 Project 2 Submission

Names/Email:

Justin Bui: Justin_Bui12@csu.fullerton.edu
Benson Lee: blee71@csu.fullerton.edu

Scatterplots





Pseudocode

Filter Cargo Vector:

```
def filter_cargo_vector (source, min_weight, max_weight, total_size):
        filtered_vector = new CargoVector
        n = source.size()

        for i = 0 to n - 1:
                cargoVecSize = filtered_vector.size()
                if (cargoVecSize < total_size):
                        if (source[i].weight() >= min_weight && source[i].weight() <= max_weight):
                                filtered_vector.push_back(source[i])


        return filtered_vector
```

Greedy Max Weight:

```
def greedy_max_weight (goods, total_volume):
        before = new CargoVector(goods)
        after = new CargoVector

        result_vol = 0
        token = 0
        n = before.size()

        while (!before.empty()):
                maxWeightPerVolume = 0

                for i = 0 to n - 1:
                        if ((before[i].weight() / before[i].volume()) > maxWeightPerVolume ):
                                token = i
                                maxWeightPerVolume = before[i].weight() / before[i].volume()

                v = before[token].volume()
                if ((result_vol + v) <= total_volume:
                        after.push_back(before[token])
                        result_vol += v

                before.erase(before.begin() + token)

        return after
```

Exhaustive Max Weight:

```
def exhaustive_max_weight (goods, total_volume):
        best = new CargoVector
        n = goods.size()

        for i = 0 to 2^n - 1:
                candidate = new CargoVector
                candidate_volume = 0
                candidate_weight = 0

                for j = 0 to n - 1:
                        if (((i >> j) & 1) == 1):
                            candidate.push_back(goods[j])
                            sum_cargo_vector (candidate, candidate_volume, candidate_weight)

                if (candidate_volume <= total_volume)
                        if (best->empty() || candidate_weight > best_weight):
                                best_weight = candidate_weight
                                best->clear()
                                best = candidate

        return best
```

Sum Cargo Vector (Used by Exhaustive)

```
def sum_cargo_vector (goods, total_volume, total_weight):
        total_volume = total_weight = 0
        n = goods.size()

        for i = 0 to n - 1:
               total_volume += goods[i].volume()
               total_weight += goods[i].weight()
```

Step Count/Time Complexity Proofs

Fitler Cargo Vector:

```
filtered_vector = new CargoVector (1)
n = source.size()  (2)

for i = 0 to n - 1: ((n - 1) - 0 + 1) = n
    cargoVecSize = filtered_vector.size() (2)
    if (cargoVecSize < total_size): (1)
        if (source[i].weight() >= min_weight && source[i].weight() <= max_weight): (5)
            filtered_vector.push_back(source[i]) (1)


return filtered_vector (0)
```

$$1 + 2 + n\left[2 + 1 + \max(5 + \max(1,0), 0)\right]$$
$$= 3 + n\left[3 + 6\right]$$
$$= 3 + n\left[9\right]$$
$$= 9n + 3$$

SC = 9n + 3

Greedy Max Weight:

```
before = new CargoVector(goods) (1)
after = new CargoVector (1)

result_vol = 0 (1)
token = 0 (1)
n = before.size() (2)

while (!before.empty()): (n step counts since it loops through each good in before once & deletes it using erase())
    maxWeightPerVolume = 0 (1)

    for i = 0 to n - 1: ((n - 1) - 0 + 1) = n
        if ((before[i].weight() / before[i].volume()) > maxWeightPerVolume ): (4)
            token = i (1)
            maxWeightPerVolume = before[i].weight() / before[i].volume() (4)

    v = before[token].volume() (2)
    if ((result_vol + v) <= total_volume: (2)
        after.push_back(before[token]) (1)
        result_vol += v (1)

    before.erase(before.begin() + token) (3)
```

return after

$$1+1+1+1+2+n\left[1+n\left(4+\max(1+4,0)\right)+2+2+\max(1+1,0)+3\right]$$
$$+0$$
$$=6+n\left[1+9n+4+2+3\right]$$
$$=6+n\left[9n+10\right]$$
$$=9n^2+10n+6$$

SC = 9n^2 + 10n + 6

Proof for Greedy:

Proof by Definition:

$f(n) \in O(g(n))$ for constants $c>0$ and $n_0 \geq 0$

such that $f(n) \leq c \cdot g(n)$, $n \geq n_0$

Let $f(n)=9n^2+10n+6$ and $g(n)=n^2$

$9n^2+10n+6 \leq c \cdot n^2$, $n \geq n_0$

Let $c=25$ and $n_0=1$

$\hookrightarrow 9n^2+10n+6 \leq 25n^2$, $n \geq 1$

$= 9n^2+10n+6 \leq 9n^2+10n^2+6n^2$, $n \geq 1$

This is trivially true for all n, therefore, greedy algorithm $\in O(n^2)$ ■

Proof by limit

$f(n) \in O(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} =$ constant or 0.

$f(n)=9n^2+10n+6$ and $g(n)=n^2$

$$\lim_{n\to\infty} \frac{9n^2+10n+6}{n^2} = \lim_{n\to\infty}\left(\frac{9n^2}{n^2}+\frac{10n}{n^2}+\frac{6}{n^2}\right)$$
$$= 9+\frac{10}{\infty}+\frac{6}{\infty} = 9 \ ■$$

Exhaustive Max Weight:

```
candidate = new CargoVector (1)
candidate_volume = 0 (1)
candidate_weight = 0 (1)

for j = 0 to n - 1:  ((n - 1) - 0 + 1) = n
        if (((i >> j) & 1) == 1):  (3)
                candidate.push_back(goods[i])   (1)
                sum_cargo_vector(candidate, candidate_volume, candidate_weight) (4n + 4)

if (candidate_volume <= total_volume) (2)
        if (best->empty() || candidate_weight > best_weight):  (3)
                best_weight = candidate_weight (1)
                best->clear()  (1)
                best = candidate  (1)
```

$$1 + 1 + 1 + n\left[3 + \max((4n+4)+1, 0)\right] + 2 + \max(3 + \max(3,0), 0)$$

$$= 3 + n\left[3 + 4n + 5\right] + 2 + 6$$

$$= 11 + 3n + 4n^2 + 5n$$

$$= 4n^2 + 8n + 11$$

SC = 4n^2 + 8n + 11

** Note this is just the FOR LOOP block calculated for exhaustive. Not entire algorithm**

Sum Cargo Vector (Used by Exhaustive Max Weight):

```
total_volume = total_weight = 0   (2)
n = goods.size()   (2)

for i = 0 to n - 1:  ((n - 1) - 0 + 1) = n
        total_volume += goods[i].volume()  (2)
        total_weight += goods[i].weight()    (2)
```

Step Count: 2 + 2 + n(2 + 2) = 4n + 4

Hypothesis 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?

Yes, there is a noticeable difference in the performance of the two algorithms. The greedy algorithm seems to be about 30 times faster as each instance size takes less than a second to run, while the exhaustive algorithm exponentially increases up to about 30 seconds as n increases. This does not surprise us as the exhaustive algorithm creates 2^n - 1 candidates, which is extremely slow compared to greedy, which takes O(n^2) time to run this function.

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

Yes, it is. In our mathematical analyses, we found that the greedy algorithm step count belongs to O(n^2) whereas the exhaustive algorithm is much slower as it belongs to O(2^n * n). Therefore, the greedy algorithm has a faster time complexity than the exhaustive algorithm.

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

Exhaustive search algorithms are not feasible to implement, because as the instance size n increases by 1, the time to perform this algorithm increases exponentially by 2^n time units. However, since the algorithm produces ALL possible candidates, it does produce correct outputs, as long as the candidate is verified.

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

The evidence is consistent with hypothesis 2 because the exhaustive algorithm (Which has an exponential running time) in this project almost takes 30 seconds with an instance size of 20-23. This would be extremely impractical in the real world as instance sizes can be in the range of thousands to millions. With an exponential running time algorithm, this would take extremely long!