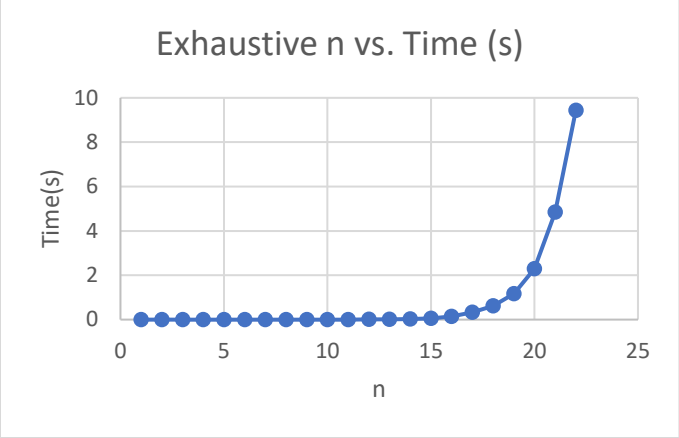
Moses Merugu

mmeru@csu.fullerton.edu

CPSC 335-01

Project 2: Greedy versus Exhaustive




```
std::unique_ptr<CargoVector> exhaustive_max_weight
(
        const CargoVector& goods,
        double total_volume
)
{
        const int n = goods.size();
        assert(n < 64);
        double candidate_volume, candidate_weight, best_volume, best_weight;
        std::unique_ptr<CargoVector> best (new CargoVector);

        for (uint64_t bits = 0; bits < pow(2, n); bits++) {
                std::unique_ptr<CargoVector> candidate (new CargoVector);

                for (int j = 0; j < n; j++) {
                        if (((bits >> j) & 1) == 1)
                                candidate->push_back(goods[j]);
                }

                sum_cargo_vector(*candidate, candidate_volume, candidate_weight);
                sum_cargo_vector(*best, best_volume, best_weight);

                if (candidate_volume <= total_volume)
                        if (best->empty() || candidate_weight > best_weight)
                                *best = *candidate;
        }
        return best;
}
```

Exhaustive mathematical analysis

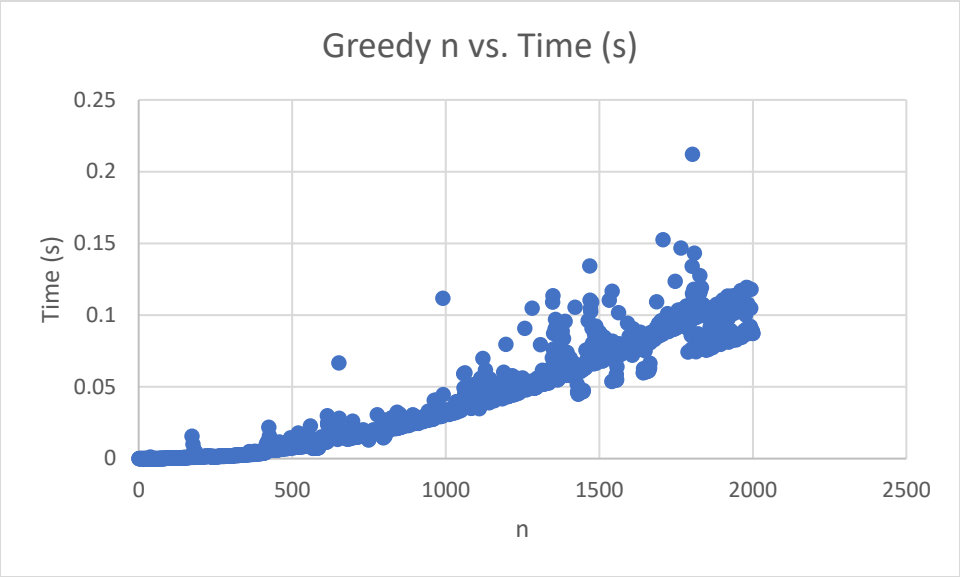
= O(7+2^n(1+n+2+n+n+3)+1)

= O(3n*2^n+6*2^n+8)

= O(2^n*n)

$$\lim_{n\to\infty}\left(\frac{3n\cdot\ 2^n+6\cdot\ 2^n+8}{2^n\cdot\ n}\right)=3$$

∴ 3n*2^n+6*2^n+8 ∈ O(2^n*n)




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

        while (!todo.empty()) {
                count = 0;
                std::shared_ptr<CargoItem> best (new CargoItem(" ",1,0));

                for (auto& item : todo) {
                        // Compare ratio, update best accordingly
                        if ((item->volume() / item->weight()) < (best->volume() / best->weight())) {
                                best = item;
                                index = count;
                        }
                        count++;
                }

                todo.erase(todo.begin() + index);

                if (result_volume + best->volume() <= total_volume) {
                        result->push_back(best);
                        result_volume += best->volume();
                }
        }
        return result;
}
```

Greedy mathematical analysis

= O(5+n(2+n+4+n+3)+1)

= O(2n^2+9n+6)

= O(n^2)

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

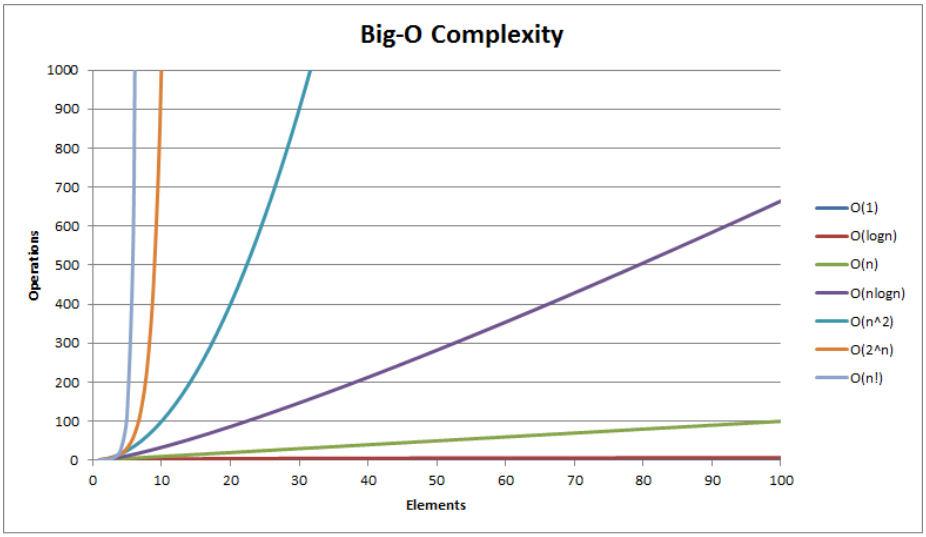
∴ 2n^2+9n+6 ∈ O(n^2)

Questions

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

The difference of runtime between the greedy and exhaustive algorithms is quite clear. The greedy method is much faster as it took under .25 seconds to run 2000 times while exhaustive method took almost 10 seconds to run 22 times. This is not surprising because exhaustive is a brute-force method.

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



I believe the empirical analysis of the exhaustive algorithm is consistent with my mathematical analysis as my exhaustive graph correlates with the Big-O Complexity graph of O(2^n). My greedy method graph on the other hand correlates more with the graph of O(nlogn) rather than O(n^2). This may mean there was an error in my interpretation of the step count.

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

This evidence is consistent with hypothesis 1 as far as correctness goes but the exhaustive method may not always be the most feasible route to go as it has a lengthy runtime and can easily be outran by other algorithms.

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

This evidence is consistent with hypothesis 2 as exponential algorithms become very slow when the size of n increases which is shown with the exhaustive method. Exponential algorithms are not ideal for practical usage as users should not have to wait longer than they need to when there are more efficient options available.