California State University-Fullerton Spring 2024

# Project 2: Greedy and Exhaustive

Author: Diego Vela

e-mail: dvela2@csu.fullerton.edu

Instructor: Zhiyuan Yang

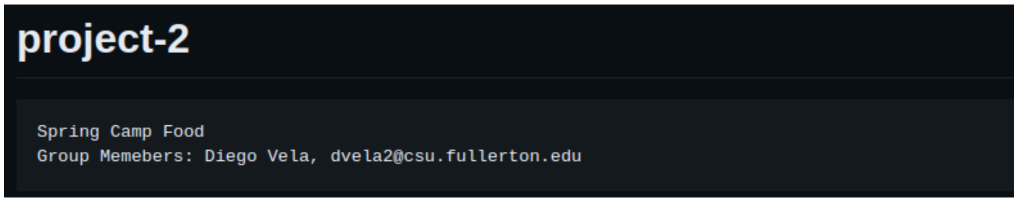
March 16, 2024

## INTRODUCTION

A summer camp is expecting lots of students over the summer and needs to order canned, fresh, or frozen items. Each product has a certain weight in ounces and a total calorie content. We are given a set of these products and are asked to design algorithms to pick a subset of food products that fit within a given the number of calories one can obtain by eating them while maximizing total weight.

The objective of the report is to prove that exhaustive algorithms are feasible to implement and produce correct outputs and that algorithms with exponential running times are too slow to be practical. A greedy and exhaustive algorithm will be compared and analyzed in their step counts and time complexity using visual and mathematical data to prove or disprove the hypotheses.

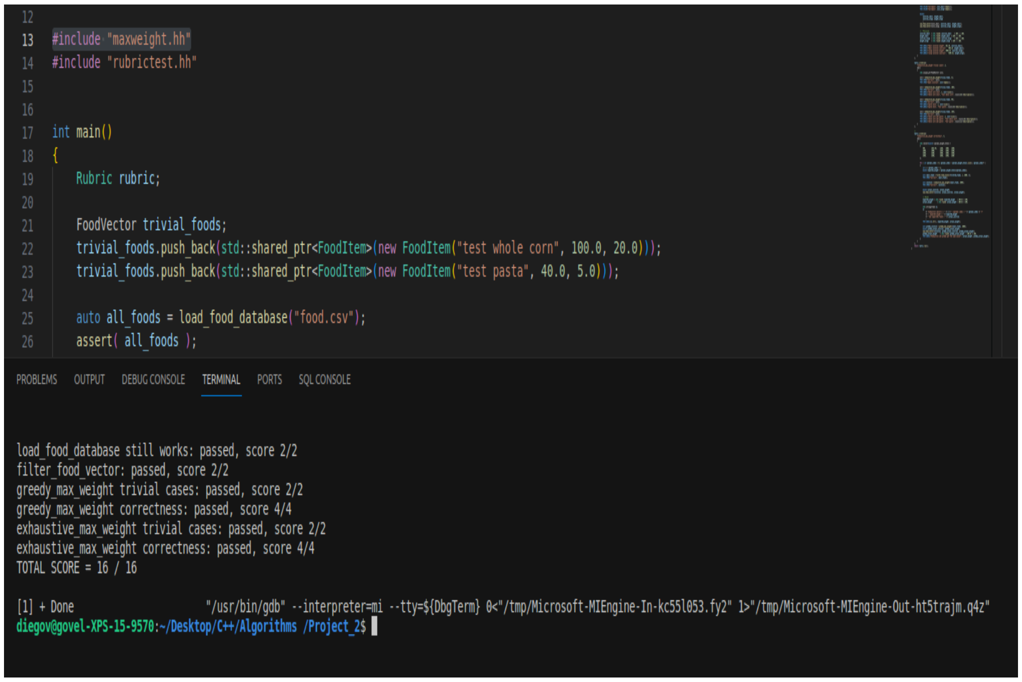
## README.MD



**project-2**

```
Spring Camp Food
Group Memebers: Diego Vela, dvela2@csu.fullerton.edu
```

## CODE EXECUTION



```cpp

#include "maxweight.hh"
#include "rubrictest.hh"


int main()
{
    Rubric rubric;

    FoodVector trivial_foods;
    trivial_foods.push_back(std::shared_ptr<FoodItem>(new FoodItem("test whole corn", 100.0, 20.0)));
    trivial_foods.push_back(std::shared_ptr<FoodItem>(new FoodItem("test pasta", 40.0, 5.0)));

    auto all_foods = load_food_database("food.csv");
    assert( all_foods );
```

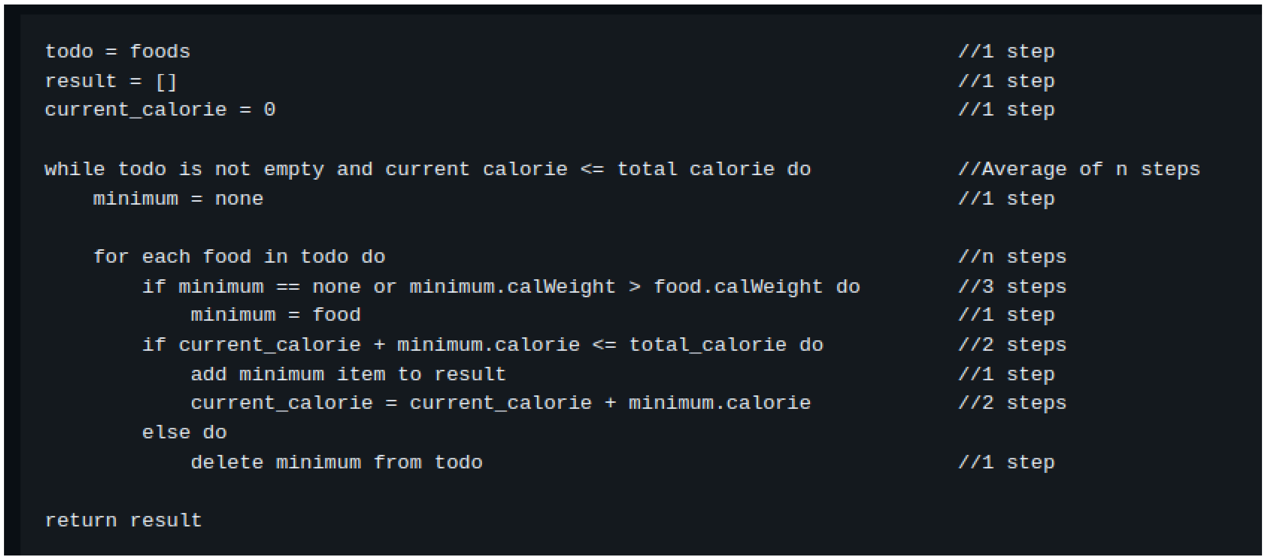
```
load_food_database still works: passed, score 2/2
filter_food_vector: passed, score 2/2
greedy_max_weight trivial cases: passed, score 2/2
greedy_max_weight correctness: passed, score 4/4
exhaustive_max_weight trivial cases: passed, score 2/2
exhaustive_max_weight correctness: passed, score 4/4
TOTAL SCORE = 16 / 16

[1] + Done                    "/usr/bin/gdb" --interpreter=mi --tty=${DbgTerm} 0<"/tmp/Microsoft-MIEngine-In-kc55l053.fy2" 1>"/tmp/Microsoft-MIEngine-Out-ht5trajm.q4z"
diegov@govel-XPS-15-9570:~/Desktop/C++/Algorithms /Project_2$
```

MATHEMATICAL ANALYSIS

## Assumptions

- The sum_food vector function will be counted as 1 step
- A food object will have an added attribute “calWeight” which is the ratio of calories to weight.

## Greedy Algorithm: PseudoCode



```
todo = foods                                                            //1 step
result = []                                                             //1 step
current_calorie = 0                                                     //1 step

while todo is not empty and current calorie <= total calorie do         //Average of n steps
    minimum = none                                                      //1 step

    for each food in todo do                                            //n steps
        if minimum == none or minimum.calWeight > food.calWeight do     //3 steps
            minimum = food                                              //1 step
        if current_calorie + minimum.calorie <= total_calorie do        //2 steps
            add minimum item to result                                  //1 step
            current_calorie = current_calorie + minimum.calorie         //2 steps
        else do
            delete minimum from todo                                    //1 step

return result
```

## Greedy Algorithm: Mathematical Analysis

$$BigO = 1 + 1 + 1 + \sum_{1}^{n} 1 + \sum_{1}^{n} 3 + 1 + 2 + max(3, 1)$$

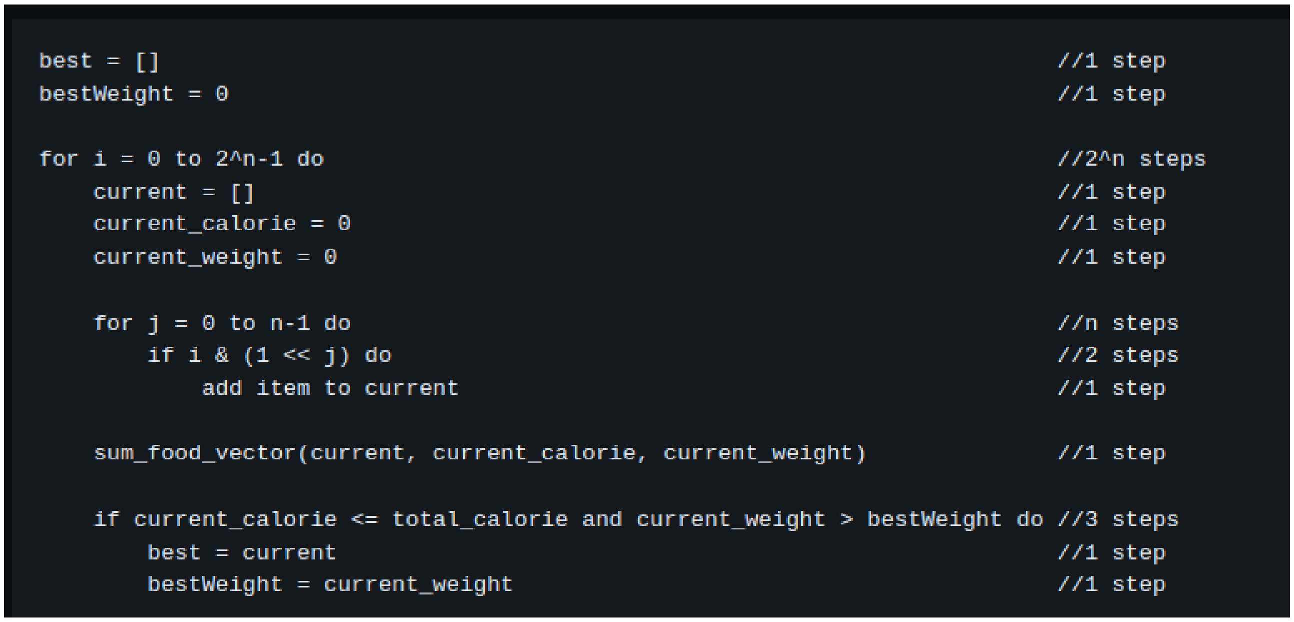
$$=> 3 + \sum_{1}^{n} 1 + \sum_{1}^{n} 9$$

$$=> 3 + \sum_{1}^{n} 1 + 9n$$

$$=> 3 + n + 9n^2$$

$$9n^2 + n + 3 = O(n^2)$$

## Exhaustive Algorithm: PseudoCode



```
best = []                                                               //1 step
bestWeight = 0                                                          //1 step

for i = 0 to 2^n-1 do                                                  //2^n steps
    current = []                                                        //1 step
    current_calorie = 0                                                 //1 step
    current_weight = 0                                                  //1 step

    for j = 0 to n-1 do                                                 //n steps
        if i & (1 << j) do                                              //2 steps
            add item to current                                         //1 step

    sum_food_vector(current, current_calorie, current_weight)           //1 step

    if current_calorie <= total_calorie and current_weight > bestWeight do //3 steps
        best = current                                                  //1 step
        bestWeight = current_weight                                     //1 step
```

## Exhaustive Algorithm: Mathematical Analysis

$$BigO \ = \ 1 + 1 + \sum_{1}^{2^n} 1 + 1 + 1 + (\sum_{1}^{n} 2 + 1) + 1 + 3 + 1 + 1$$

$$=> \ 2 + \sum_{1}^{2^n} 9 + \sum_{1}^{n} 3$$

$$=> 2 + \sum_{1}^{2^n} 9 + 3n$$

$$=> 2 + 9 * 2^n + 3n * 2^n$$

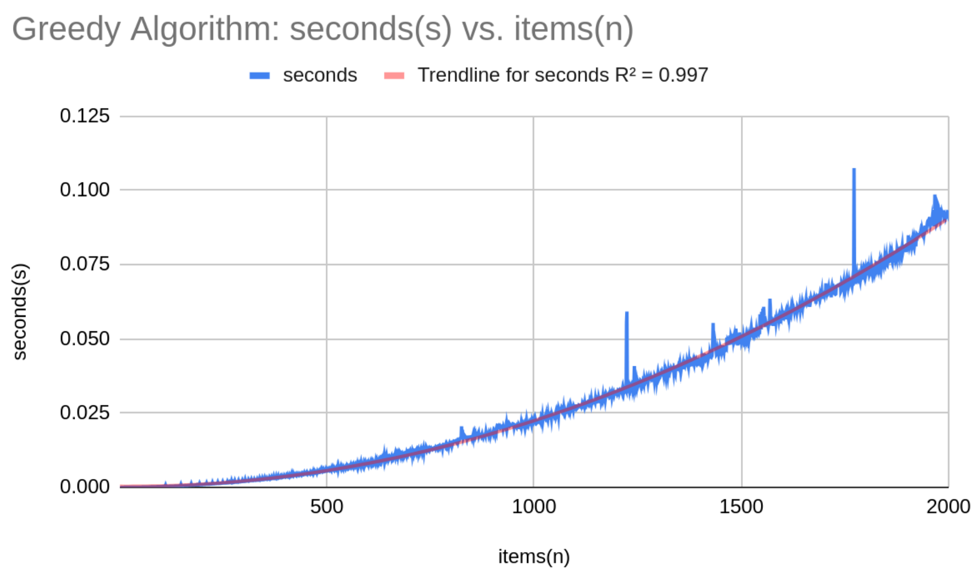
$$3n * 2^n + 9 * 2^n + 2 = \ O(2^n * n)$$

## Summary

The greedy algorithm has a lower time complexity than the exhaustive algorithm. The exhaustive algorithm is expected to run at an exponential growth rate while the greedy algorithm is expected to run at a polynomial growth rate of n^2.

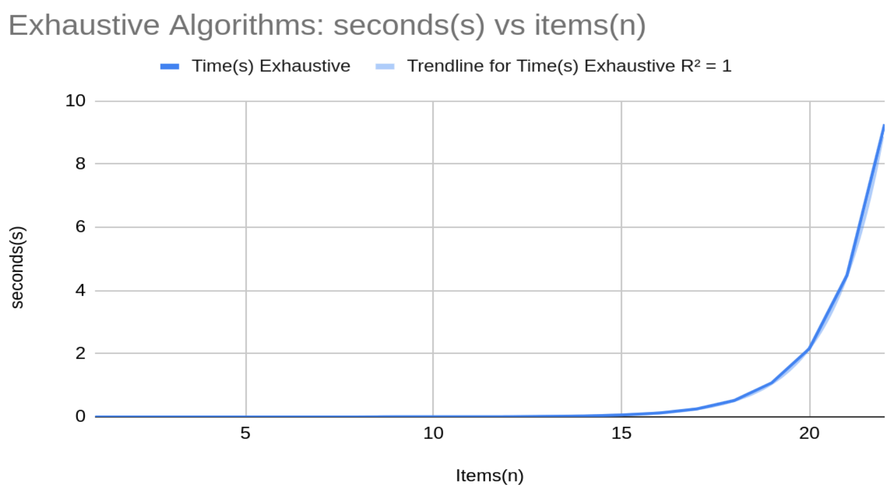
SCATTERPLOTS

## Greedy Scatterplot



- The trendline that best fit was a polynomial trendline which is consistent with the time complexity analysis. The coefficient of determination is 0.997. The data is noisy and has a few outliers due to computational limitations.

## Exhaustive Scatterplot



- The trendline that best fit was an exponential trendline which is consistent with the time complexity analysis. The coefficient of determination is 1.

QUESTIONS

**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 large difference between the two algorithms. By simply looking at the scatterplot we can see that at about 20 items, the exhaustive algorithm is nearing 10 seconds to finish. When we compare this to the greedy algorithm, we see that at 2000 items, the computational time is under a tenth of a second. The overall result was not surprising since we can see the time efficiency were of two distinct classes but the actual difference in computation was surprising.

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

My empirical analyses were consistent with my mathematical analyses. According to my empirical analysis, the best fit trendline for the greedy algorithm was a polynomial trendline which is consistent with the mathematical analysis of $O(n^2)$. For the exhaustive algorithm, the best fit trendline was the exponential trendline which is consistent with the mathematical analysis of $O(2^n)$.

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

This evidence is consistent with hypothesis 1 because the implementation of an exhaustive algorithm is to simply check every possible combination and it will always produce the best result.

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

This evidence is consistent with hypothesis 2 because the scatterplots show that an exponential growth will quickly skyrocket the computational time. On the contrary, a greedy algorithm was able to compute up to 2000 items before an exhaustive algorithm can compute 13 items.

## CONCLUSION

In this project, the time complexity of an exhaustive algorithm was compared to a greedy algorithm in a real world problem. Given a set of food items with a calorie and weight value, the algorithms found the subset to maximize the weight given a max amount of calories. Using this problem, an exhaustive algorithm was proven to be accurate and simple in terms of generating the best results and in implementation respectively. It was also proven that although an exhaustive algorithm can be accurate, the computational time it takes to generate outputs is too slow to be feasible in large sets of data. These results showcase the importance of time complexity when designing algorithms and how the difference in efficiency makes large impacts as items increase.