Project 4:
Exhaustive vs. Dynamic Programming

—

Charlie Taylor ctaylor27@csu.fullerton.edu
Matthew Butner mbutner@csu.fullerton.edu

Project 4

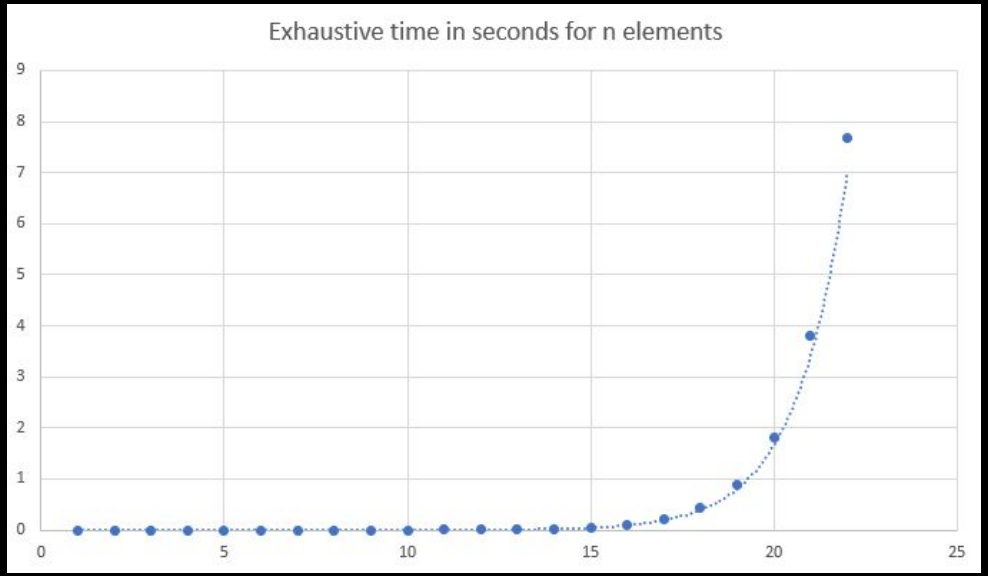
Group members:

Charlie Taylor ctaylor27@csu.fullerton.edu

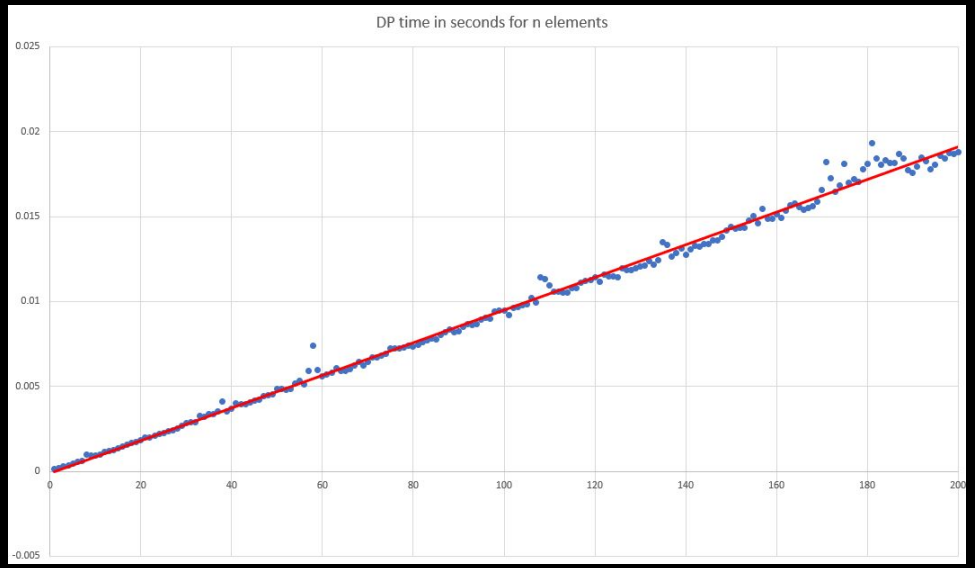
Matthew Butner mbutner@csu.fullerton.edu

```
csuftitan@LTMacAiM1165002 food-pantry---ead-charlieproj4 % make run_test
g++ -std=c++17 -Wall maxcalorie_test.cc -o maxcalorie_test
./maxcalorie_test
load_food_database still works: passed, score 2/2
filter_food_vector: passed, score 2/2
dynamic_max_calories trivial cases: passed, score 2/2
dynamic_max_calories correctness: passed, score 4/4
exhaustive_max_calories trivial cases: passed, score 2/2
exhaustive_max_calories correctness: passed, score 4/4
TOTAL SCORE = 16 / 16
```

Exhaustive Graph N<23 FoodObjects



DP Graph N<200 FoodObjects



Efficiency Proof: Exhaustive Algorithm

```
def exhaustive_max_Calories(FoodVector foods, double total_weight)
{
    for (uint64_t i=0; i < 2^n; i++) // 2^n times
    {
        FoodVector Candidate;
        for(int j=0; j<n; j++) // n
        {
            if(((i >> j) & 1) == 1) // 3
            {
                Candidate.push_back(foods[j]); // 1
            }
        }
        SumFoodVector(Candidate);  // n times
        if (weightSum <= total_weight) // 1
        {
            if (best->empty() || (calorieSum > bestTotalCalories)) // 3
            {
                *best = Candidate;  // n
                Sum_FoodVector(*best) // n times
            }
        }
    }
    return best;  // 1
}
```

S.C.(exhaustive Search) =

$$2^n * \left(n * \left(3 + \max(1,0) \right) + n + 1 + \max(3+2n, 0) \right) + 1$$

$$= 2^n * \left(4n + n + 1 + (3+2n) \right) + 1$$

$$= 2^n * \left(7n + 4 \right) + 1$$

$$= 7n \cdot 2^n + 4 \cdot 2^n + 1 = T(n)$$

$T(n) \in O(2^n \cdot n)$

$\frac{d}{dx}(2^n) = \frac{2^n}{\ln 2} = \frac{1}{\ln 2} \cdot 2^n$

$$\lim_{n\to\infty} \frac{7n2^n + 4\cdot 2^n + 1}{2^n \cdot n} \overset{\infty}{\underset{\infty}{=}} \lim_{n\to\infty} \frac{(7n)(2^n) + (2^n \cdot 7) + 4\cdot 2^n}{2^n + n \cdot 2^n}$$

$$= \lim_{n\to\infty} \frac{7n + 7 + 4}{1 + n} \overset{\infty}{\underset{\infty}{=}}$$

$$= \lim_{n\to\infty} \frac{7+0}{1} = 7 = L$$

Since $L > 0$ (and finite), $T(n) \in O(2^n \cdot n)$

Efficiency Proof: DP Algorithm

```
def dynamic_max_Calories(Vector Foods, total_weight)
  1 // Let W = total_weight (Let n be size of Foods)
  1 // Let T = dP matrix          (Let best = [])
  for (item = 1  to  n)  // n times
      for (weight = 1 to W)  // W times
          T[item][Weight] = -∞  //3

  for (i = 1  to n)   // n times
     for (j=0 to W)  // W+1 times
     {
        double default = T[i-1][j]  //3
        double  tentative = 0        //1
        int CurrentWeight = foods(i-1).weight()  //4
          if (j >= CurrentWeight)    // 1
          {
            tentative  = foods(i-1). Calories()      //4
            int remaining_Capacity = j-CurrentWeight //2
            tentative += T[i-1][remaining_Capacity]   //3
          }
        T[i][j] = max(default, tentative)    // 1
     }

  // Traceback:
  int i = n , int j = W      // 2
  While(i>0)               // n times
  {
    if (T[i][j] != T[i-1][j])  //5
    {
      j -= foods[i-1].weight()  //4
      best. append (foods(i-1)) //3
      i -= 1       // 1
    }
  }
  return best;  // 1
```

S.C. (dynamic_Programming) =

$$\underbrace{1+1+n*(W*(3))}_{\text{init matrix}} + n*\left((W+1)*\left((3+1)+4+1+\max(4+2+3,0)+1\right)\right) + 2 + n*\left(5+\max(4+3+1,0)\right)+1$$

$$= 2 + 3nW + n*((W+1)*19) + 2 + n*(13) + 1$$

$$= 3nW + (W+1)\cdot n + 19n + 13n + 5$$

$$= 3nW + nW + n + 19n + 13n + 5$$

$$= 4nW + 33n + 5 = T(n)$$

$$T(n) \in O(n*W)$$

$$\lim_{n\to\infty} \frac{4nW + 33n + 5}{n*W} \quad \frac{\infty}{\infty} \overset{?}{=} \lim_{n\to\infty} \frac{\frac{4n}{} + \frac{33n}{W} + \frac{5}{W}}{\frac{nW}{W}} \quad \frac{\infty}{\infty}$$

$$= \lim_{n\to\infty} \frac{4 + \frac{1}{W}(33) + 0}{1} = \frac{4 + \frac{33}{W}}{1} = \left(4 + \frac{33}{W}\right)$$

$$= 4 + \frac{33}{W} = L$$

Since $L > 0$ (and finite),
$T(n) \in O(n*W)$

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

A) There is a significant difference between the performance of the two algorithms. The Dynamic Method is far more feasible in terms of speed. The derivative to represent growth rate for the Dynamic Method (f(x)=nW => f'(x) = W. The derivative to represent growth rate for the Exhaustive Method
(g(x)=n2^n => g'(x) = n*2^n + 2^n)

This is not surprising, considering that the Dynamic algorithm follows a pseudo-polynomial rate, while the exhaustive method follows an exponential growth rate.

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

B)
The empirical analysis is consistent with the mathematical analysis. Based on our asymptotic analysis, we said the Dynamic algorithm belongs to the pseudo-polynomial class (nW), and the Exhaustive algorithm belongs to the exponential class (n*2^n).

n	Greedy	Exhaustive
5	3.494	160
10	10	10240
15	17.641	491520
20	26.02	20971520

The chart on the left outputs the comparison of the efficiency classes (the classes that the algorithms belong to). If we compare the rate of change between values of N to the empirical data recorded earlier, we can see a similar growth pattern as N moves toward infinity.

For exhaustive, we can confirm that anything beyond $N > 20$ will become computationally expensive to calculate (as displayed in our graph earlier).

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

C)

Hypothesis 1: *Exhaustive search algorithms are feasible to implement, and produce correct outputs.*

The evidence we have gathered is not exactly consistent with the hypothesis. The exhaustive search algorithm is feasible only under a certain size of input. If we want to find the optimal solution for a small set of items, we can use this method. If we prefer the optimal solution, and we happen to have a smaller set of items (say under 20), we could use this method to generate correct output.

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

C)
Hypothesis 2: *Algorithms with exponential running times are extremely slow, probably too slow to be of practical use.*

As described earlier, exhaustive search is only feasible if we're dealing with a set of smaller items. As displayed in the graph, when we reach values of N>27 , we're looking at runtimes of at least 200 seconds per iteration (keep in mind that this cost will grow exponentially as N increases). In conclusion, Exhaustive search is extremely slow and not practical if we are dealing with an input with cardinality greater than 20. We should only consider the exhaustive method if we want to calculate the optimal solution for a relatively small set of items.