Charlie Taylor

ctaylor27@csu.fullerton.edu

CPSC 335

6/23/2022

Project 2: Greedy and Exhaustive Algorithms

Greedy Algorithm Efficiency Analysis

```
 FoodVector per(foods); // n to copy elements from foods
 std::unique_ptr<FoodVector> results(new FoodVector()); // 1 to make empty object

 std::sort(per.begin(), per.end(),
   [](const std::shared_ptr<FoodItem> a, const std::shared_ptr<FoodItem> b)
   { return (a->foodCalories() / a->weight()) > (b->foodCalories() / b->weight()); });
// std::sort uses introsort which takes n log n steps
 double currWeight = 0; // 1
 for (auto &i : per) // n
 {
   double curr = i->weight(); // 1 for = and 1 for accessing weight

   if(currWeight + curr <= total_weight) // 2
   {
     currWeight += curr; // 1
     results->push_back(i); // 1
   }
 }

 return results; // 1
```

Overall, we get $n + 1 + n\,log(n) + 1 + n[2 + 2 + max(2, 0)] + 1$

Reduces to $nlog(n) + 7n + 3$ which belongs to O(n log(n))

$$\lim_{n\to\infty} \frac{nlog(n) + 7n + 3}{nlog(n)}$$

$$\lim_{n\to\infty} \frac{\frac{ln(x)+1}{ln(2)} + 7}{\frac{ln(x)+1}{ln(2)}}$$

$$\lim_{n\to\infty} \frac{\frac{1}{ln(2)x}}{\frac{1}{ln(2)x}}$$

$$\lim_{n\to\infty} \frac{ln(2)x}{ln(2)x} = 1$$

$L \neq \infty$ therefore $nlog(n) + 7n + 3$ belongs to O(n log(n))

Exhaustive Algorithm Efficiency Analysis

```
const uint64_t n = foods.size(); // 1 for = and 1 for size()

 std::unique_ptr<FoodVector> best(new FoodVector()); // 1 for empty object
 std::unique_ptr<FoodVector> candidate(new FoodVector()); // 1 for empty object
 double candidateCal = 0, candidateOz = 0, bestCal = 0; // 3

 for(uint64_t i = 0; i < pow(2, n); i++) // 0 to 2^n - 1 = 2^n
 {
   candidate->clear(); // n since max length of candidate could be n
   for(size_t j = 0; j < n; j++) // 0 to n-1 = n
   {
     if(((i >> j) & 1) == 1) // 3
     {
       candidate->push_back(foods[j]); // 2
     }
   }
   sum_food_vector(*candidate, candidateOz, candidateCal); // n
   if(candidateOz <= total_weight) // 1
   {
     if(best == NULL || candidateCal > bestCal) // 3
     {
       best.swap(candidate); // 1
       bestCal = candidateCal; // 1
     }
   }
 }
 return best; // 1
```

Overall we get

$$2 + 1 + 1 + 3 + 2^n[n + n(3 + max(2,0)) + n + 1 + max(3 + max(2,0)] + 1$$

Which reduces to $(7n + 6)2^n + 8$

$(7n + 6)2^n + 8$ belongs to $O(n*2^n)$

Find $c > 0$ and $n_0 >= 0$ such that $7n2^n + 6* 2^n + 8 <= c * 2^n$ for all $n >= n_0$

Good $c = 7 + 6 + 8 = 21$

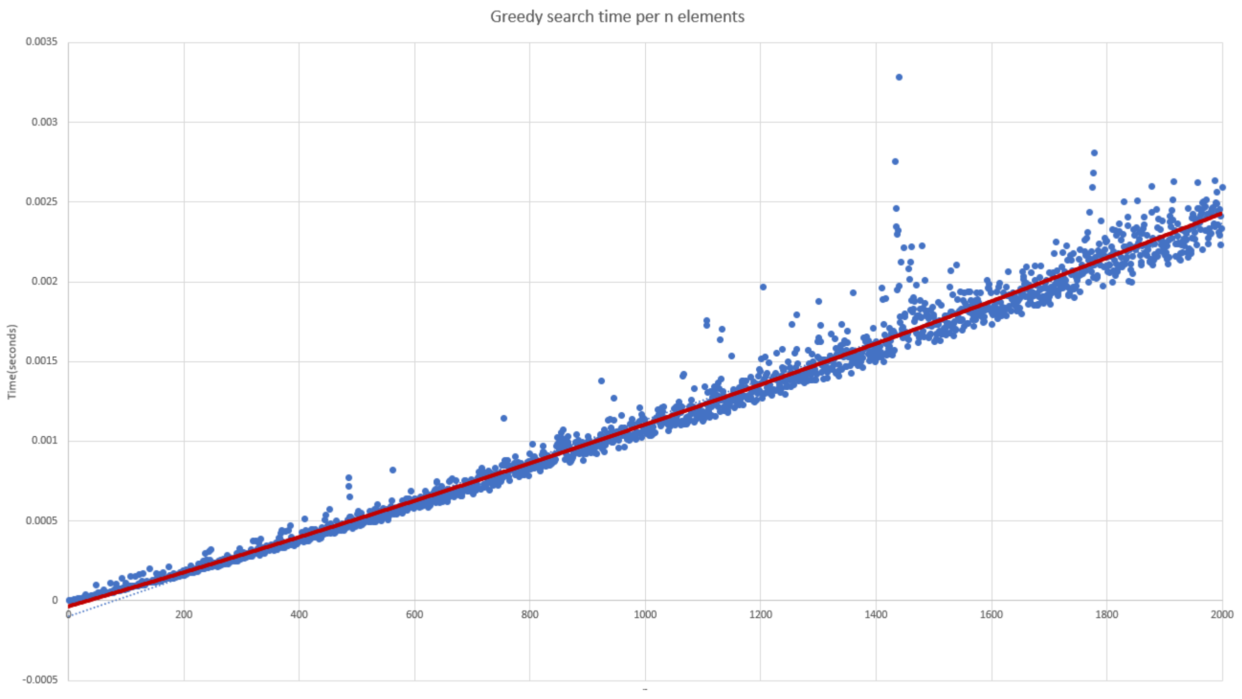
$(7 + 6 + 8)2^n - 7n2^n - 6*2^n - 8 <= 0$

$(7n*2^n - 7n2^n) + (6*n2^n - 6*n2^n) + (8*n2^n - 8) <= 0$

n=1	n=0	n=1

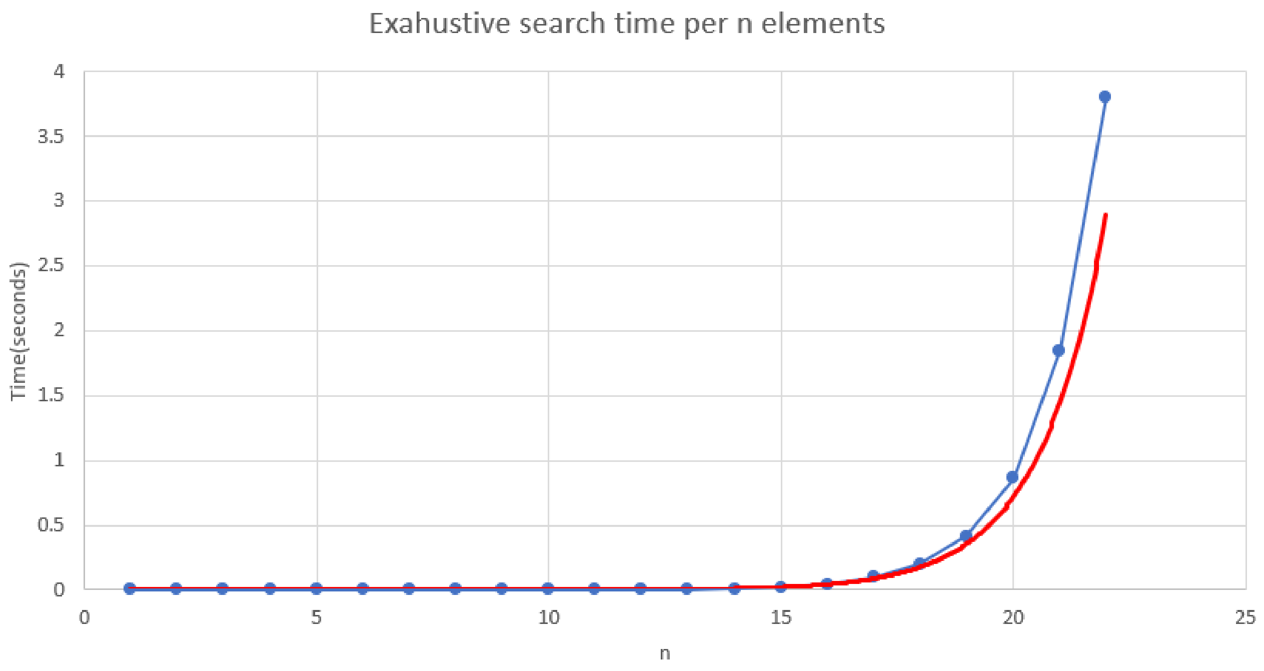
$n_0 = 1$

Greedy run time in seconds(y) over n elements(x)



This appears to match the expected efficiency class O(n log(n))that was calculated earlier.

Exhaustive run time in seconds(y) over n elements(x)



This also appears to match the expected efficiency class of O(n*2^n) that was previously calculated.

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 massive difference between the two algorithms, the greedy algorithm only takes about 0.0025 seconds to run with 2000 elements while the exhaustive algorithm takes almost 4 whole seconds to do only 22 elements. The greedy algorithm is many powers of 10 faster than the exhaustive algorithm, as it is only n log(n) while exhaustive is exponential. This does not surprise me because from the analysis of the algorithms I knew that the exhaustive algorithm would be much much slower.

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

> Yes, making a line of best fit for each it is easy to see that they follow the general pattern of the efficiency classes that were calculated.

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

> The evidence is consistent with hypothesis 1 because I was able to implement the algorithm and it gave correct results.

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

> The evidence also lines up with hypothesis 2 stating that this is an impractical way of solving the problem because it takes almost 4 seconds to do 22 elements, whereas a greedy implementation can do 2000 in about 0.0025 seconds.