Project 2

CPSC 335 Roya Askari, Henry Nguyen
E-mail: royaskari@csu.fullerton.edu,
spanky14@csu.fullerton.edu

The Hypothesis

1. Exhaustive search algorithms are feasible to implement, and produce correct outputs.
2. Algorithms with exponential running times are extremely slow, probably too slow to be of practical use.

Problem:

1. **Analyze your greedy algorithm code mathematically to determine its big-O efficiency class, probably $O(n^2)$ or $O(n \log n)$.**

high-protein diet problem

input: a positive calorie budget C; and a vector V of food objects, where each food $f = (c, p)$ has an integer amount of calories $c > 0$ and protein $p \geq 0$
output: a vector K of food objects drawn from V, such that the sum of all calorie values is within the food budget i.e.

$\sum_{(c,p) \in V} c \leq C$; and the sum of all protein values $\sum_{(c,p)} p$ is maximized

```
greedy_max_protein(C, foods):
        todo = foods
        result = empty vector
        result_cal = 0
        while todo is not empty:
                Find the food f in todo of maximum protein.
                Remove f from todo.
                Let c be f's calories.
                if (result_cal + c) <= C:
                        result.add_back(f)
```

result_cal += c

return result

O(1)+O(1)+O(1)+O(n)*(o(n)+O(c)+O(1)+O(1)+O(1)+o(1))+O(1)

$=O(3+n^2+nc+3n+1)=> O(n^2)$

lemma: $4 + n^2 + nc + 4$

$\lim_{n\to\infty} (T(n)/f(n)) = \lim_{n\to\infty} ((4 + n^2 + nc + 4)/n^2)) = 1$

$4+ n^2 + nc + 4 \in O(n^2)$

2. **Gather empirical timing data for each of the two algorithms by running them for various values of** n.

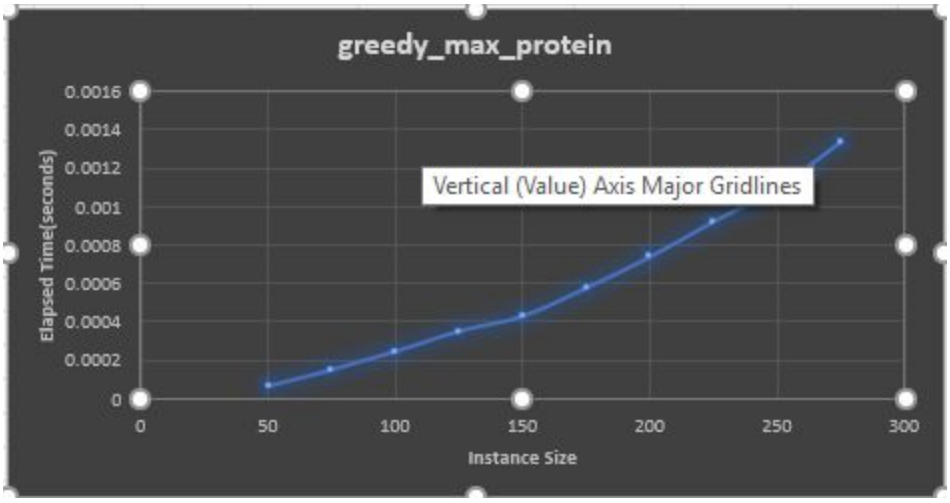
greedy _max_protein:

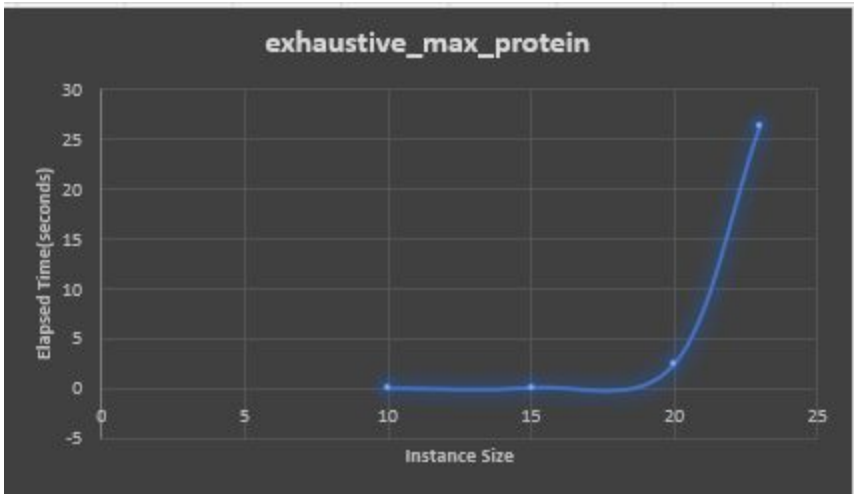
N times	Time(seconds)
50	0.0000656
75	0.000152
100	0.000247
125	0.000353
150	0.000431
175	0.000579
200	0.000740
225	0.000925
250	0.00110
275	0.00134

exhaustive _max_protein:

N times	Time(seconds)
10	0.00131
15	0.0608
20	2.396
23	26.295

3. Draw two scatter plots. Each plot should have the instance size n on the horizontal axis, elapsed time on the vertical axis, a fit line, title, and labels on both axes.





Conclusion:

3. **A. 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 very noticeable difference between the performance of greedy vs exhaustive. The exhaustive algorithm takes significantly more time. This isn't really surprising since the theory is that it takes $O(2^N * n)$ time and the scatter plot shows an exponential growth.

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

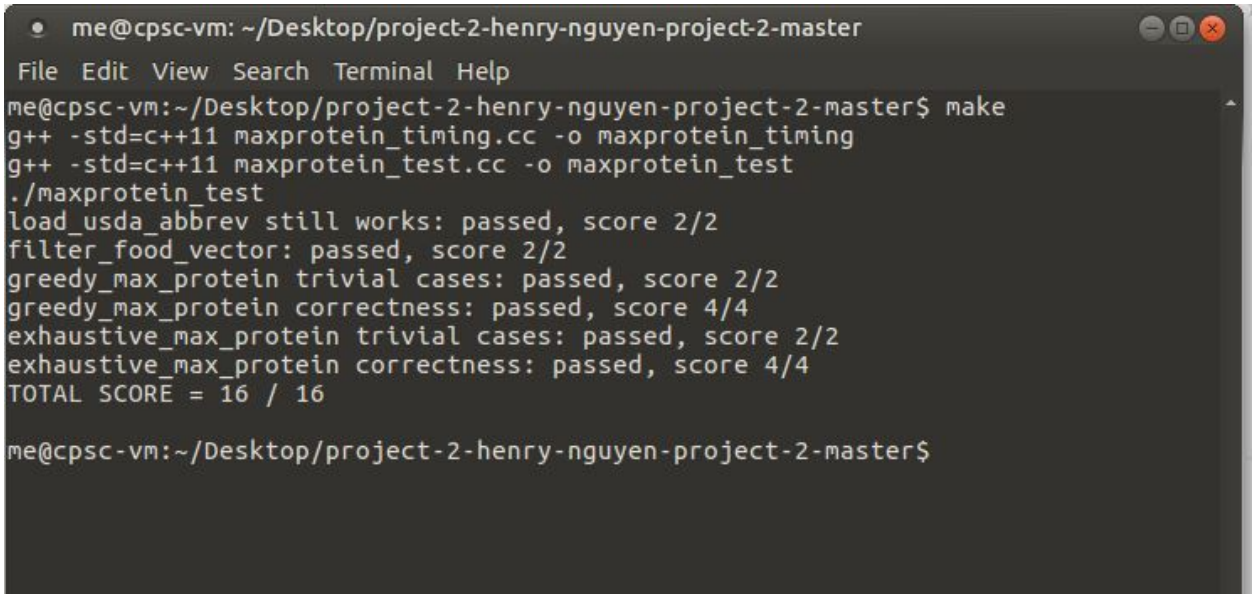
   They are consistent. Mathematical analysis shows that the greedy algorithm takes $O(n^2)$ time. The scatter plot that is depicted above clearly shows a line that behaves similarly to $O(n^2)$ as it curves upward.

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

   This evidence is consistent with hypothesis 1. The exhaustive algorithm is easy to implement and will explore all possible options to choose the best solution. Therefore, it will always produce the correct output even though the run time may be long.

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

   This evidence is consistent with hypothesis 2. The Exhaustive algorithm which is exponential, is far too slow to be of practical use. As you can see on the scatter plot, the line shoots upwards even though it has a relatively small n size.


me@cpsc-vm: ~/Desktop/project-2-henry-nguyen-project-2-master
File Edit View Search Terminal Help
me@cpsc-vm:~/Desktop/project-2-henry-nguyen-project-2-master$ make
g++ -std=c++11 maxprotein_timing.cc -o maxprotein_timing
g++ -std=c++11 maxprotein_test.cc -o maxprotein_test
./maxprotein_test
load_usda_abbrev still works: passed, score 2/2
filter_food_vector: passed, score 2/2
greedy_max_protein trivial cases: passed, score 2/2
greedy_max_protein correctness: passed, score 4/4
exhaustive_max_protein trivial cases: passed, score 2/2
exhaustive_max_protein correctness: passed, score 4/4
TOTAL SCORE = 16 / 16
me@cpsc-vm:~/Desktop/project-2-henry-nguyen-project-2-master$