Miles McCloskey miles.mccloskey@csu.fullerton.edu

Chase Moynihan chasemoy@csu.fullerton.edu

Diego Franchi diegofranchi@csu.fullerton.edu

CPSC 335

PROJECT 2 REPORT

```
greedy_max_protein(C, foods)
    todo = foods                                    O(1)
    result = empty vector                           O(1)
    result_cal = 0                                  O(1)
    while todo is not empty:                        O(n)
        find the food in f in todo of maximum protein  O(n)
        remove f from todo                          O(c)
        c ← f's calories                            O(1)
        if(result_cal + c) <= C                     O(1)
            result.add_back(f)                      O(1)
            result_cal += c                         O(1)
    return result                                   O(1)
```

Mathamatical analysis

= O(1+1+1+n(n + c + 1 + 1 +1 +1)+1)

= O(4 + n(n + c + 4))

=O(4 + n^2 + nc + 4)

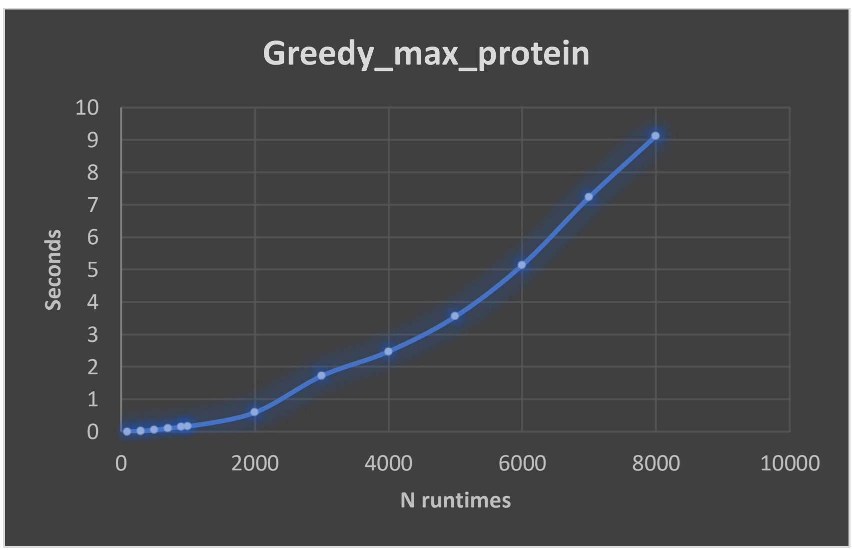
=O(n^2)

lemma: 4 + n^2 + nc + 4

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

∴ 4+ n^2 + nc + 4 ∈ O(n^2)

N times	Seconds
100	0.00393837
300	0.0186959
500	0.0536141
700	0.0987309
900	0.14313
1000	0.157881
2000	0.59032
3000	1.72011
4000	2.47075
5000	3.55241
6000	5.13744
7000	7.22963
8000	9.12386



```
exhaustive_max_protein(C, foods):
    n = |C|                                                    O(1)
    best = None                                                O(1)
    for bits from 0 to (2^n -1):                               O(2^n)
        candidate = empty vector                               O(1)
        for j from 0 to n – 1                                  O(n)
            if((bits >> j) & 1) == 1:                          O(c)
                candidate.add_back(foods[j])                   O(1)
            if total_calories(candidate) <= C:                 O(1)
        if best is None or                                     O(1)
            total_protein(candidate) > total_protein(best):    O(1)
            best = candidate                                   O(1)
return best                                                    O(1)
```

Mathamatical analysis

= $O(1+1 + 2^n(1 + n(c + 1 + 1) + 1 + 1 + 1)+1)$

= $O(3 + 2^n(4 + n(c + 2))$

~remove insignificant terms~

=$O(2n(n(c)))$

=$O(3 + 2^n(4 + nc + 2n)$

=$O(8^n +2^n nc + 4^n n + 3)$

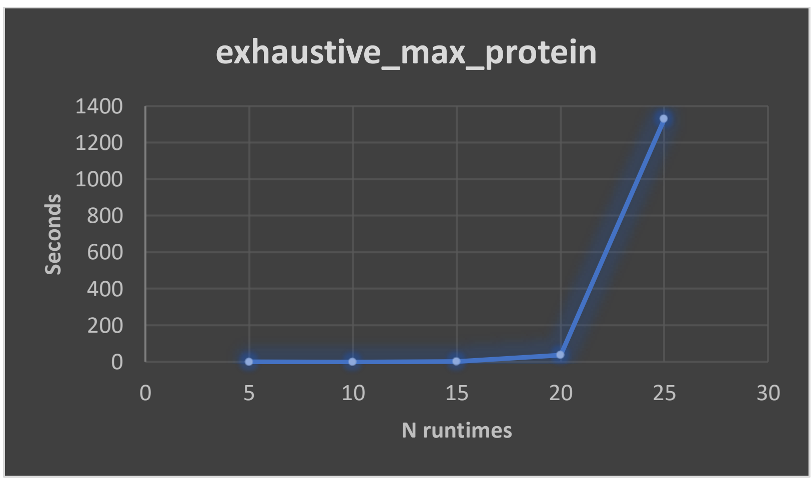
= $O(2^n * n)$

lemma: 3 + 2^n(4 + nc + 2n)

$$\lim_{n\to\infty}\left(\frac{T(n)}{f(n)}\right) \quad = \lim_{n\to\infty}\left(\frac{3 + 2^\wedge\mathrm{n}(4 + \mathrm{nc} + 2\mathrm{n})}{2^n n}\right) = 1$$

$\therefore$ 3 + 2^n(4 + nc + 2n) $\in O(2^n * n)$

N times	Seconds
5	0.000370568
10	0.0225205
15	0.916211
20	35.7246
25	1327.93



Conclusion

Is there a noticable difference in the performance of the two algorithms? Which is faster, and by how much?

There is a considerable difference between the greedy and exhaustive runtimes. Moreover, the greedy algorithm runs sufficiently faster than the exhaustive algorithm by

Does this surprise you?

The results were surprising. We didn't realize how fast the exhaustive algorithm would explode in time conplexity. A jump from 20 to 25 resulted in an increase from 35 seconds to 1327 seconds. This was a huge increase from such a small change in the value of n runtimes. We were expecting it to be more gradual and not really begin to consume vast amount of time resources until the value of n was a lot larger.

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

Yes our empirical analysis is consistent with out mathematical analysis because within our mathematical analysis we concluded that the runtime of the greedy algorithm is n^2 while our exhaustive algorithm's runtime complexity was 2^n*n which is much slower, and the differences in the empirical analysis were much slower in the exhaustives chart. Which makes the two forms of analysis entirely consistent.

Is this evidence consisten or inconsisten with hypothesis 1? Justify.

Yes, exhaustive search algorithms can produce correct outputs, but are not entirely feasible to implement, due to the exponential increase in runtime when higher values of n are added.

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

Yes, exponential algorithms are extremely slow, and are not enitrely practical to use because of the n runtimes behavior when higher values of n are input into the original function. The increases were so much, that the amount of time it takes waiting for it to run make running it basically infeasible, or pointless.