Project 2: Greedy vs. Exhaustive

by Dion Wayne Pieterse & Michael Lindwall

Mathematical Analysis – Greedy Algorithm

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 >= 0
Output: A vector K of food objects drawn from V, such that the sum of all calorie values is within the food budget.

Pseudo Code:

```
def greedy_max_protein(F, C):
      result = V[]                          (1)
      foods_cpy = [F]                       (n)
      max_index = 0                         (1)
      max_protein = 0                       (1)
      calories = 0                          (1)
      result_cal = 0                        (1)

      while(foods_cpy != empty):                          (1)
            max_protein = foods_cpy[0].protein_g()        (1)
            calories = foods_cpy[0].kcal()                (1)
            max_index = 0                                 (1)

            for j in |foods_cpy|:                                  (1)
                  if foods_cpy[j].protein_g() > max_protein:       (1)
                        max_protein = foods_cpy[j].protein_g()     (1)
                        max_index = j                              (1)
                        calories = foods_cpy[j].kcal()             (1)

            if result_cal + calories <= total_kcal:               (1)
                  result.push_back(foods_cpy[max_index])          (1)
                  result_cal += calories                          (1)
            foods_cpy.erase(foods_cpy[max_index])                 (1)

      return result                                               (1)
```

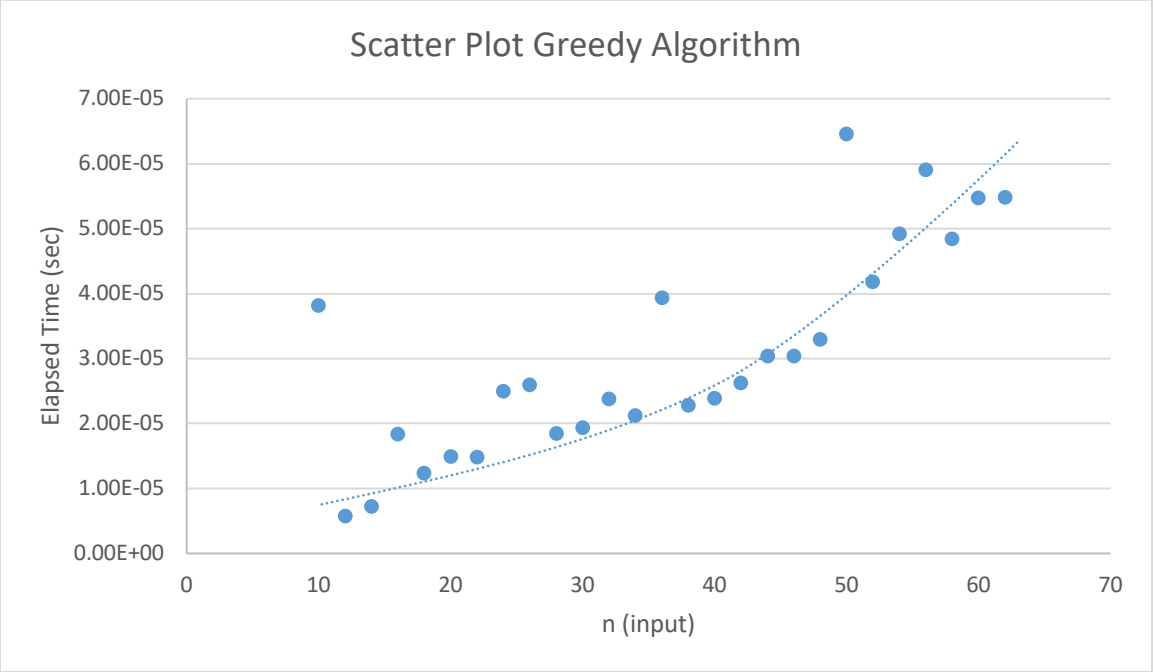
n iterations

n iterations

Efficiency = 1 + n + 1 + 1 + 1 + 1 + n(1 + 1 + 1 + 1 + n(1+ 1 + 1 + 1 + 1) + 1 + 1 + 1 + 1) + 1
= 5 + n + n(4 + n(5) + 4) + 1
= 5 + n + n(5n + 8) + 1
= 5 + n + $5n^2$ + 8n + 1
= $5n^2$ + 9n + 6

Proof:

$5n^2$ + 9n + 6 E O($5n^2$ + 9n + 6)	by Trivial Property Theorem
= O($5n^2$)	by Dominated Terms Theorem
= **O(n^2)**	by Constant Factor Theorem



Conclusion:
Yes, the empirically observed time efficiency data generated is consistent with the mathematically-derived big-O efficiency class that was determined.

Mathematical Analysis – Exhaustive Search Algorithm

Pseudo Code:

```
def exhaustive_max_protein(F, C):
     n = |F|                                                    (1)
     MAX_POSSIBLE_SUBSETS = pow(2, n)                           (1)
     best = []                                                  (1)

     for subset in range(0, MAX_POSSIBLE_SUBSETS):              (1)
          candidate = []                                        (1)
          total_calories_can = 0                                (1)
          total_protein_can = 0                                 (1)
          total_calories_best = 0                               (1)
          total_protein_best = 0                                (1)

          for shift_num in range(0, n):                         (1)
               if subset >> shift_num & 1 == 1:                 (1)
                    candidate.push_back(F[shift_num])           (1)


          sum_food_vector(total_calories_can, total_protein_can,
          candidate)                (n)

          sum_food_vector(total_calories_best,
          total_protein_best, best)                             (n)

          if total_calories_can <= total_kcal:                  (1)
               if best.size() == 0 || (total_protein_can >
               total_protein_best):                             (1)
                    best = candidate                            (n)
     ptr_best[best]                                             (n)
     return ptr_best                                            (1)
```

2^n iterations

n iterations

Efficiency = $1 + 1 + 1 + 2^n[\ 1 + 1 + 1 + 1 + 1 + 1 + n(1 + 1 + 1) + n + n + 1 + 1 + n] + n + 1$

$= 3 + 2^n\ [\ 6 + 3n + 2n + 2 + n\ ] + n + 1$

$= 3 + 2^n\ [\ 6n + 8\ ] + n + 1$

$= 3 + (6n)(2^n) + 8(2^n) + n + 1$

$= 4 + 6n(2^n) + 8(2^n) + n$

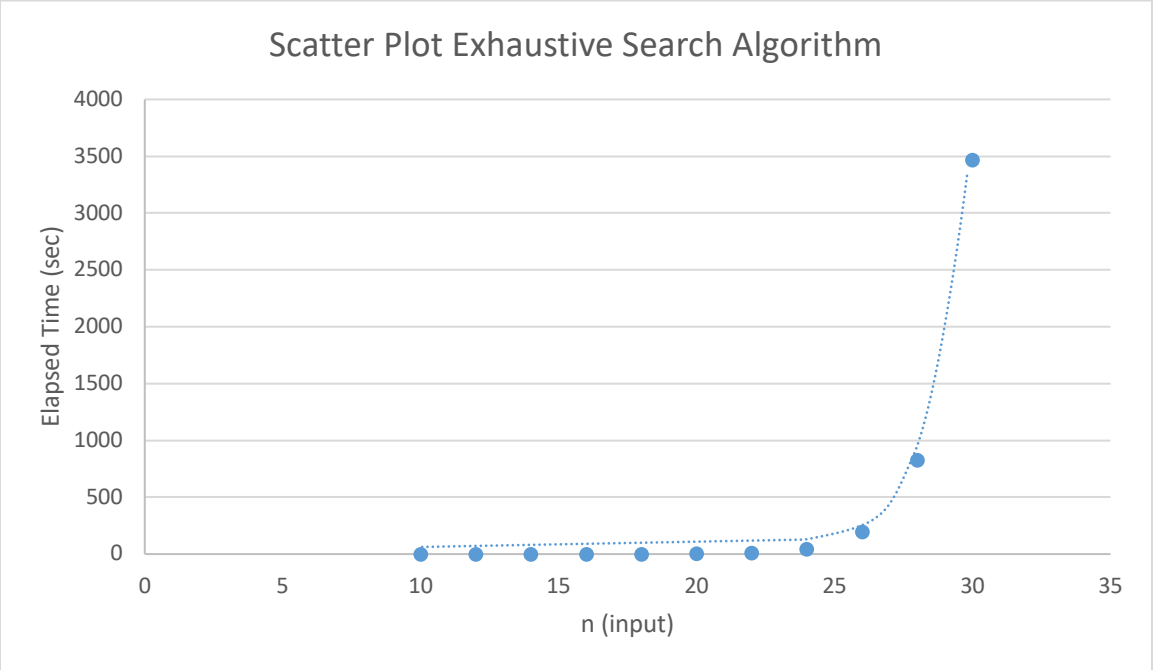
$= 6n(2^n) + 8(2^n) + n + 4$

Proof:

$6n(2^n) + 8(2^n) + n + 4 \in O(6n(2^n) + 8(2^n) + n + 4)$ by Trivial Property Theorem

$= O(6n(2^n))$ by Dominated Terms Theorem

$= \mathbf{O(n(2^n))}$ by Constant Factor Theorem



Conclusion:

Yes, the empirically observed time efficiency data generated is consistent with the mathematically-derived big-O efficiency class that was reached.

Analysis and Comparison of Algorithm Performance:

- **Is there a noticeable difference in the performance of the two algorithms? Which is faster, and by how much? Does this surprise you?**
  Yes, the greedy algorithm performs much faster than the exhaustive search. The difference is extremely noticeable. No this does not surprise us because the exhaustive search has to cycle through all subsets in the outer loop (2^n), which is significantly larger than just n for the outer while loop for the greedy algorithm.

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

  Yes, our empirical analyses are consistent with our mathematical analyses. All the work is stated in the above pages (including mathematical analysis breakdown and proofs for each algorithm). They are restated below:

  Greedy Algorithm:
  Efficiency Class: $O(n^2)$

  Exhaustive Search Algorithm:
  Efficiency Class: $O(n2^n)$

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

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

  Yes, the exhaustive search algorithm was indeed quite straight forward to implement and yes the output produced by the search was indeed correct. This is verified by the fact that all tests were passed for the exhaustive search.

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

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

  Yes, the exhaustive search algorithm became unbearably slow as the value of n went past 30. We had to redefine the data set up until 30 because anything after n = 30 caused us to wait for extremely long periods of time for results to come back.