Names:
Jose Arce josearce@csu.fullerton.edu
Rainier Sarmiento rensarmie@csu.fullerton.edu
CPSC 335-02
Professor Doina Bein

CPSC 335 Project 2

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.

Greedy Pattern Algorithm

PSEUDOCODE

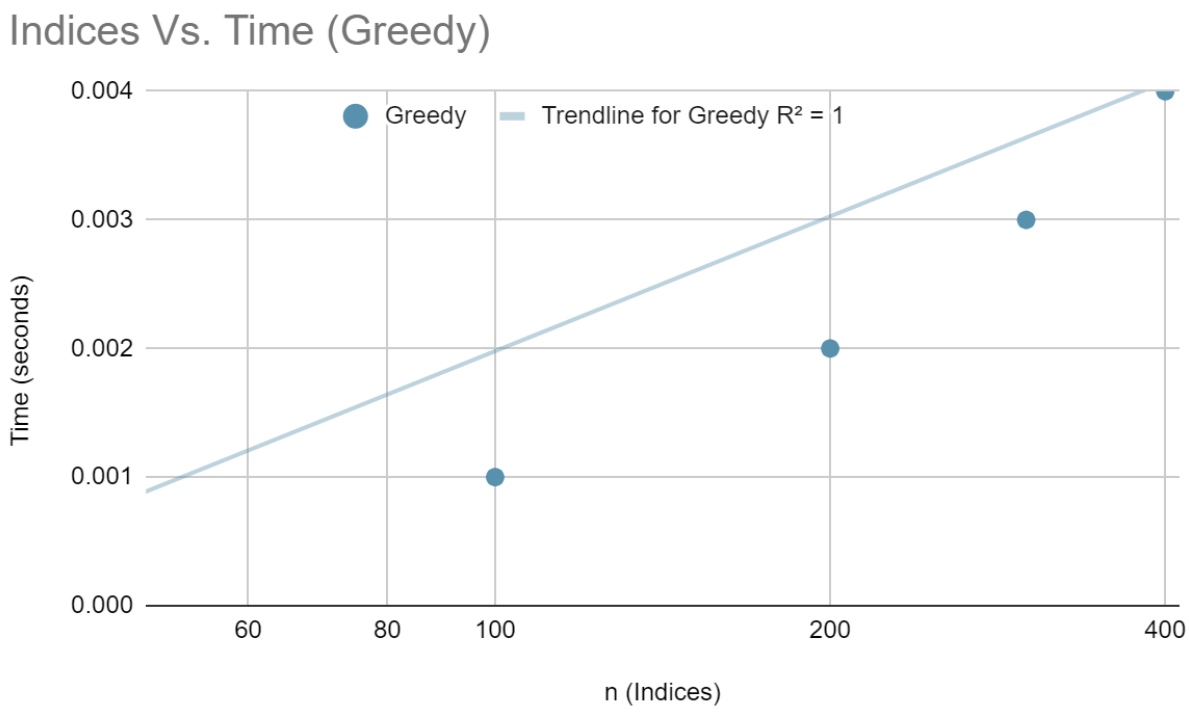
```
greedy_max_time(C, ride_items):
      todo = ride_items                                  O(1)
      result = empty vector                              O(1)
      result_cost = 0                                    O(1)
      while todo is not empty:                           O(n)
        Find the ride item "a" in todo of maximum time per its cost
O(n)
            Remove "a" from todo                         O(c)
            Let c be a's cost in dollars                 O(1)
            if (result_cost + c) <= C:                   O(1)
                  result.add_back(a)                     O(1)
                  result_cost += c                             O(1)
      return result                                      O(1)
```

MATHEMATICAL ANALYSIS

O(1 + 1 + 1 + n (n + c + 1 + 1 + 1 + 1) + 1)
O(4 + n(n + c + 4))
O(4 + n^2 + nc + 4n)
O(n^2 + 4n + nc + 4)

= $O(n^2)$

SCATTERPLOT



Exhaustive Optimization Algorithm

PSEUDOCODE

```
exhaustive_max_time(C, ride_items):
     n = |ride_items|                                                    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(ride_items[j])             O(1)

          if total_cost(candidate) <= C:                          O(1)
               if best is None or                                 O(1)
                  total_time(candidate) > total_time(best):               O(1)
                    best = candidate                                      O(1)
     return best                                                  O(1)
```

MATHEMATICAL ANALYSIS

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

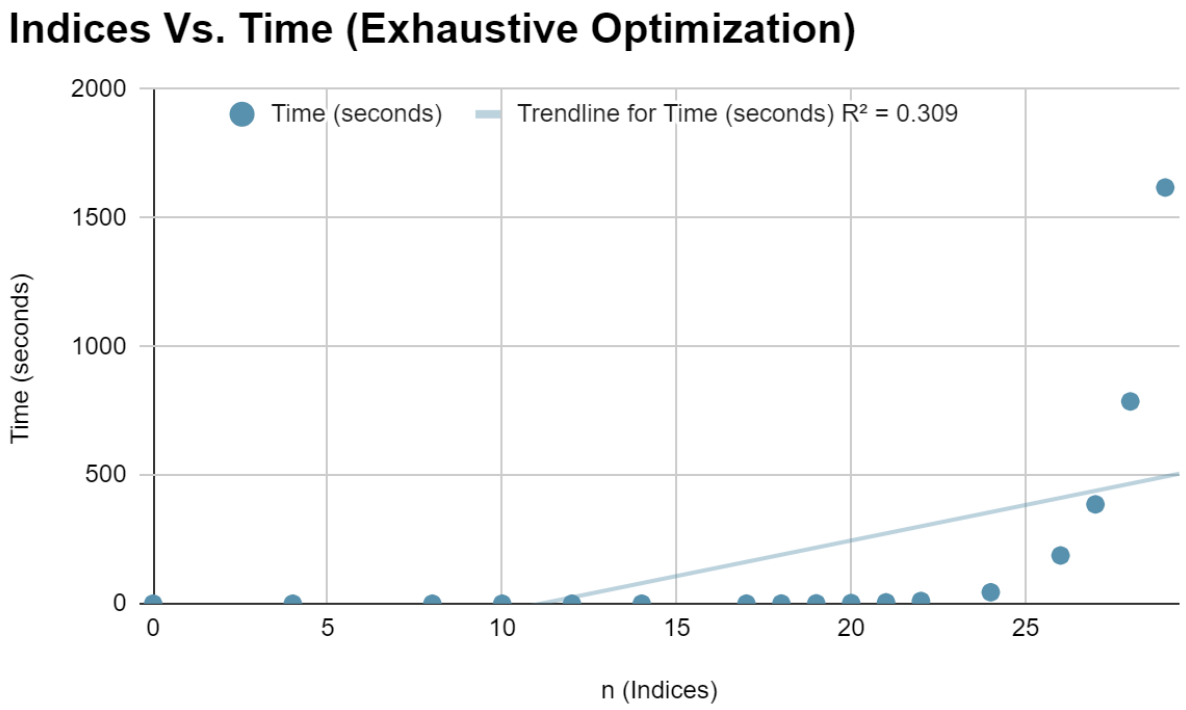
$O(3 + 2^n(4 + n(c + 2))$

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

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

$= O(2^n * n)$

SCATTERPLOT



Following Questions

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

A: The performance of the two algorithms is very drastic. This is not a surprise due to understanding the theory behind both algorithms. Greedy will provide a locally optimal solution, while Exhaustive Optimization will always give the best result.

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

A: Our empirical analysis is consistent with our mathematical analysis. For Greedy, we receive a scatterplot with plots resembling an $O(n^2)$ graph. Our Exhaustive Optimization scatterplot resembles a $O(2^n)$ graph.

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

A: The evidence is consistent with hypothesis 1. Exhaustive Algorithms are feasible to implement and output the correct values. The implementation was simple enough, however the true problem is the resources needed to calculate the outputs.

Q: Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.
A: The feasibility of the implementation is offset by the resources needed to calculate the outputs. From the scatter plots above, the Exhaustive Optimization justifies a Big O notation of $O(2^n)$. To calculate a vector of size n = 29, it takes about 786 seconds or 13 minutes. At n=30, the time to complete was 1618 seconds or about 27 minutes, which is consistent with the Big O notation. In conclusion, for such a small index, it would not be practical to implement Exhaustive Algorithms.