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

CPSC 335 Project 4

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.

Dynamic Programming Algorithm

PSEUDOCODE

```
dynamic_max_time(C, ride_items)
     i,j                                                          O(1)
     for i = 0, i <= rides.size(), i++:                           O(n)
          K.push_back                                             O(1)
          for j = 0, j <= total_cost, j++:                                 O(n)
               K.at(i).push_back(0.0)                             O(1)
     for i = 1, i <= rides.size(), i++:                           O(n)
          for j = 1, j <= total_cost, j++:                                 O(n)
               if (rides[i - 1]->cost() <= j)                     O(1)
                  K[i][j] = max(rides[i-1]->time() +
                    K[i-1][j - rides[i-1]->cost()], K[i-1][j]    O(1)
               else                                               O(1)
                  K[i][j] = K[i -1][j]                            O(1)
     res = K[rides.size()][total_cost]                            O(1)
     j = total_cost                                               O(1)
     for i = ride.size()                                          O(n)
          if(res == K[i - 1][j]                                   O(1)
               continue                                           O(1)
          else                                                    O(1)
```

```
                result->push_back(rides[i - 1]              O(1)
                res = K[i - 1][j - rides[i - 1]->cost()]    O(1)
                j = rides[i - 1]->cost()                    O(1)
    return result                                           O(1)
```

MATHEMATICAL ANALYSIS

O(1 + n(1 + n(1 + n(n(1 + 1 + 1 + 1 + 1 + 1 + n(1 + 1 + 1 + 1 + 1 + 1 + 1)))))

O(1 + n(1 + n(1 + n(n(6 + n(7))))

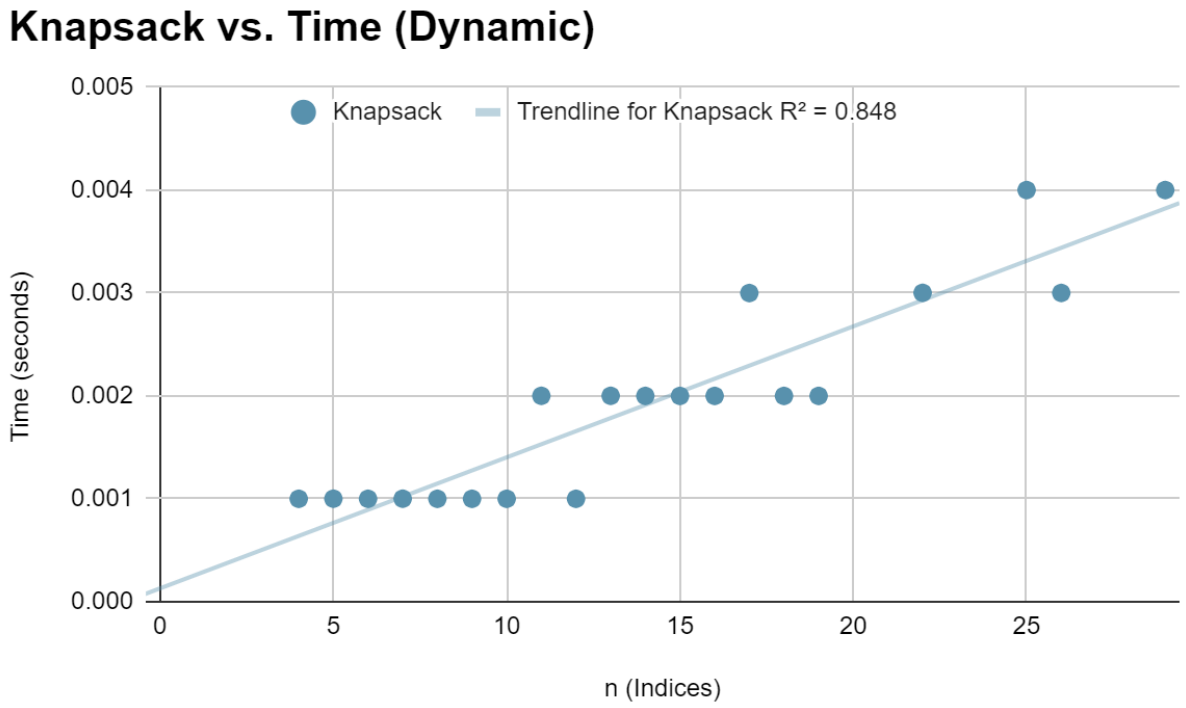
O(1 + n(1 + n(1 + $6n^2$ + $7n^3$)

O(1 + n(1 + n + $6n^3$ + $7n^4$)

O(1 + n + n^2 + $6n^4$ + $7n^5$)

= 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

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

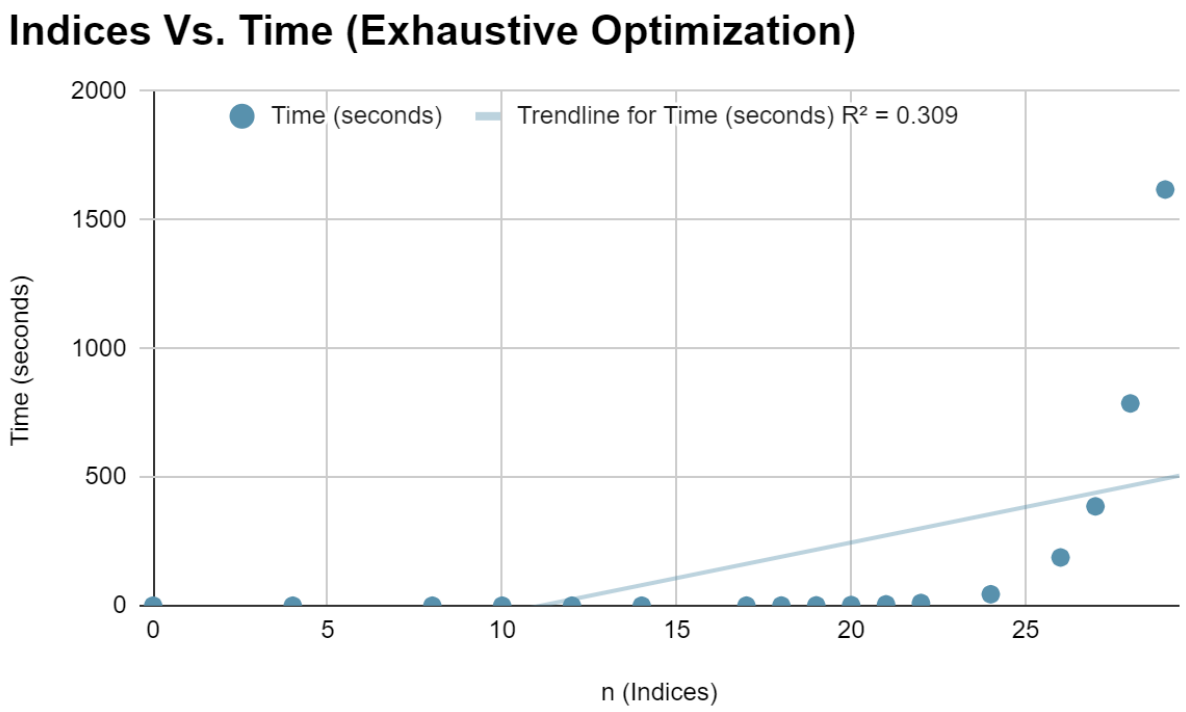
$O(3 + 2^n(4 + n(c + 2))$

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

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

$= O(2^n * n)$

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: There is a noticeable difference in the performance of the two algorithms. The dynamic knapsack approach is by far better. This does not surprise us, because exhaustive has an exponential time complexity, and would only make sense to take much more time than the dynamic approach. To put into perspective, the exhaustive algorithm needed 26 minutes to calculate 29 indices. This is much greater than the dynamic algorithm which spent 0.004001 seconds to complete 29 indices.

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

A: Our empirical analyses are consistent with our mathematical analyses. Using step-count, this provides a $O(2^n * n)$ time complexity. From the trend of the line, that remains consistent with the empirical analyses. Our empirical analyses for the dynamic algorithm are also correct. The time

complexity of the knapsack algorithm is O(N*W) where N is the weight element and W is the capacity.

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 evidence is consistent with hypothesis 2 from the plot above. Exponential running times are extremely slow and thus are not valid for practical use.