Project 2: Dynamic vs. Exhaustive - Crane unloading problem

CPSC 335 - Algorithm Engineering

Spring 2023

Instructor: Himani Tawade

Everette D. Webber 886894732

GitHub Repository: https://github.com/Digx7/Crane-Problem

Exhaustive Algorithm Solution Pseudocode

```
Given max_steps
Given empty vector allValidPaths
Given empty queue permutations
Given empty path start

Permutations.push(start)
While (permutations is not empty)
        Path current = path at front of permutations
Var length = number of steps of current
        If (length > 0)
                allValidPaths.add(current)
        if (length <= max_steps)
                if (current + one step right   IS VALID)
                        permutations.push( current + one step right )
                if (current + one step down   IS VALID)
                        permutations.push( current + one step down )

path best
for ( i = 0 to allValidPaths.size() )
        if ( allValidPaths[i].total_cranes > best.total_cranes )
                best = allValidPaths[i]
```

Time Analysis

The above code runs through two distinct loops. A While loop and a For loop. The while loop runs as long as the queue Permutations is not empty. This loop also sets the size of the vector allValidPaths. The For loop depends on the size of the vector allValidPaths. Together the combined time complexity of both loops makes the time complexity of the whole Exhaustive Algorithm Solution.

Exhaustive Algorithm Solution time = While loop + for loop

Where

For loop = O(s)

s = allValidPaths.size()

While loop = O(n)

n = number of all permutations of a set of 2 elements with repetition and variable length from 1 to “the maximum number of steps”

Number of permutations with repetition = x^r

x = number of elements in the set

r = length of the permutation

Number of permutations with repetition and variable length = The Sum of x^r with r = i to r = j when r increments by k

Therefore

n = The Sum of 2^r with r = 1 to r = “the maximum number of steps” when r increments by 1

Therefore

While loop = O(The Sum of 2^r with r = 1 to r = “the maximum number of steps” when r increments by 1)

This can be rewritten as

While loop = O(2^n)

n = “the maximum number of steps”

Therefore

Exhaustive Algorithm Solution time = O(2^n) + O(s)

= O(s + 2^n)

This gives us a worst case time complexity of

O(2 * 2^n)

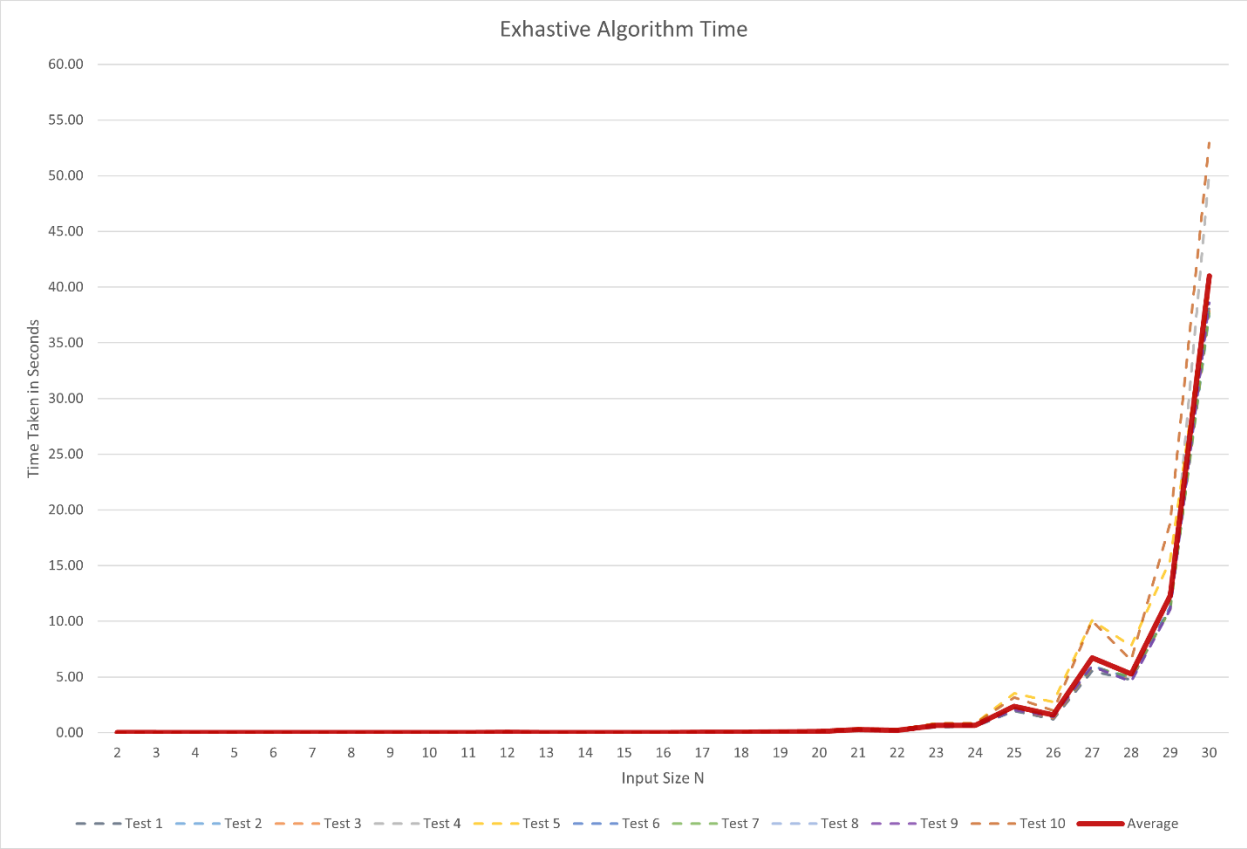
When s = 2^n

This gives us a best case time complexity of

O(2^n)

When s = 0

Graph for time vs. input size



Dynamic Algorithm Solution Pseudocode

Given a 2D vector called A of the same dimensions as a 2D grid G

A[0][0] = an empty path on G
Path best = an empty path on G

For (row = 0 to G.rows)
    For (column = 0 to G.columns)

        If (G[row][column] is a building)
            Continue

        Path from_above = an empty path

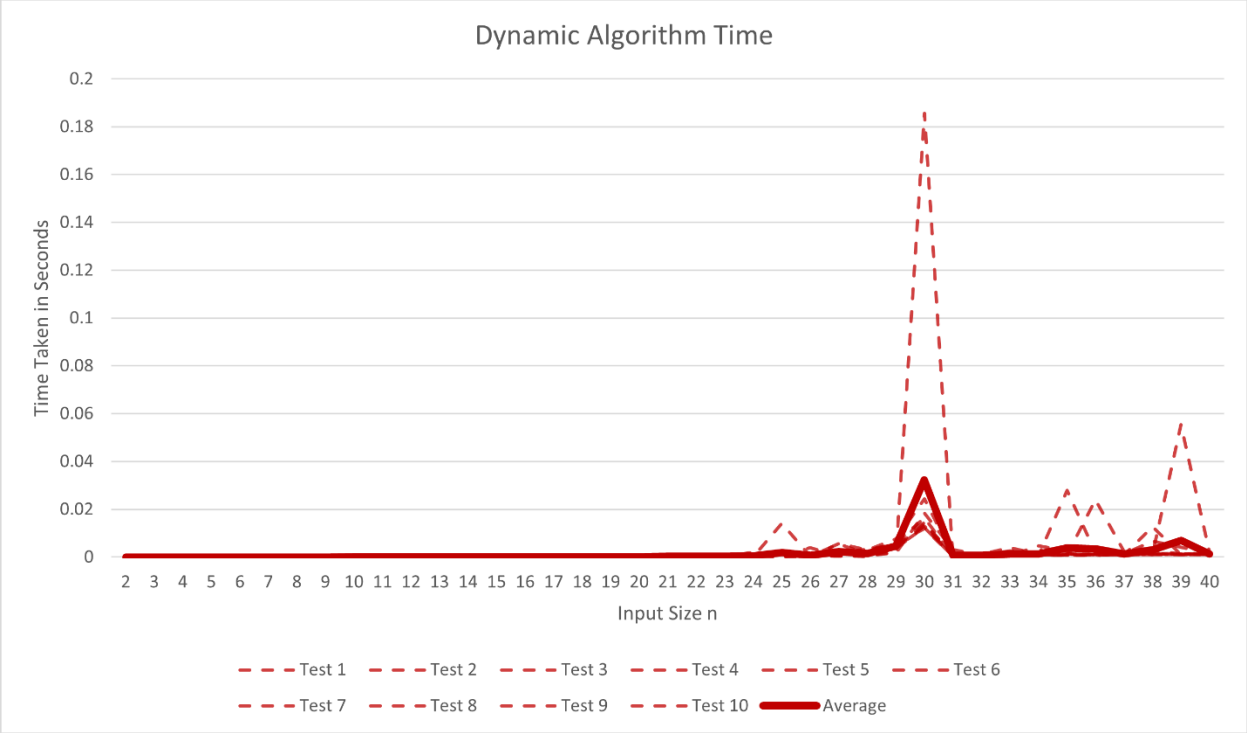
```
Path from_left = an empty path

If ( row != 0 AND G[row – 1][column] is NOT a building )
        From_above = A[row-1][column] + one step down

If ( column != 0 AND G[row][column - 1] is NOT a building )
        From_left = A[row][column - 1] + one step right

If (from_above OR from_left are NOT empty paths)
        A[row][column] = from_above or from_left whichever has more total cranes

        If (A[row][column] total cranes > best total cranes )
                Best = A[row][column]
```

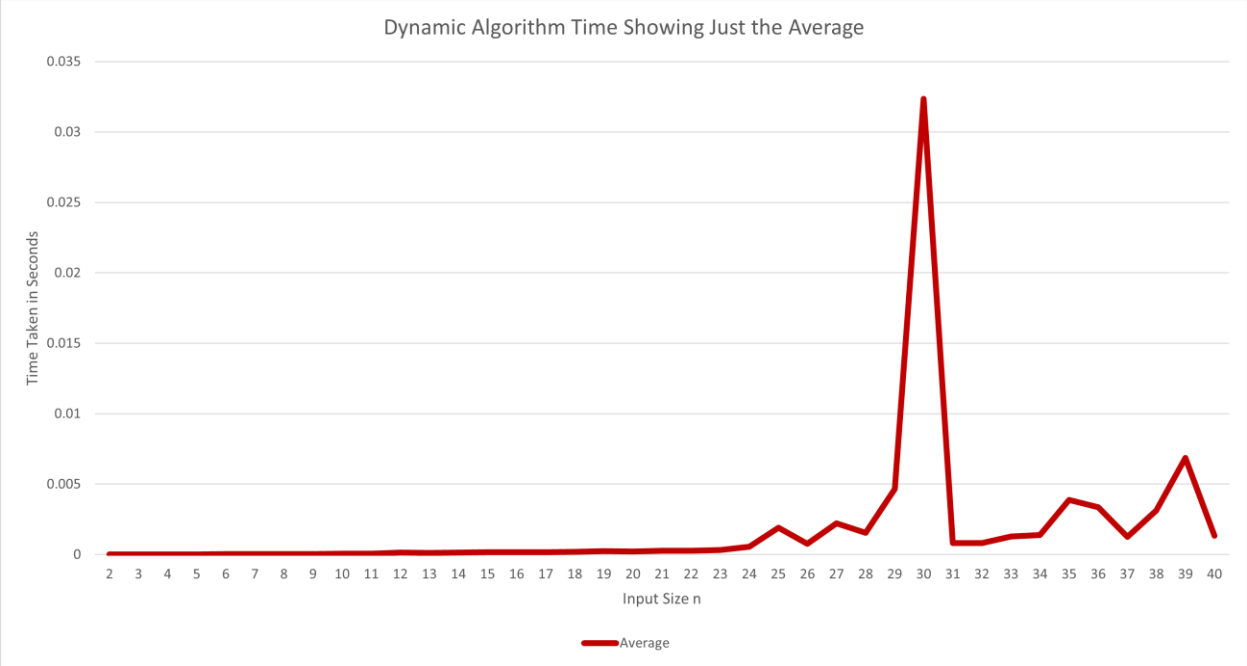
Time Analysis

Given that the entire algorithm is in a nested for loop we get a time complexity of O(n^2) where n = the length of the side of the grid.

The README for this assignment implies that we should get a time complexity of O(n^3). This seems to be because the README suggests that once you fill out the 2D Vector A with all the paths you should loop through that again and find the best path. However, we can instead check for the best path at the same time we are filling out the 2D Vector A. Thus giving us the improved time complexity of O(n^2).

Graph for time vs. input size





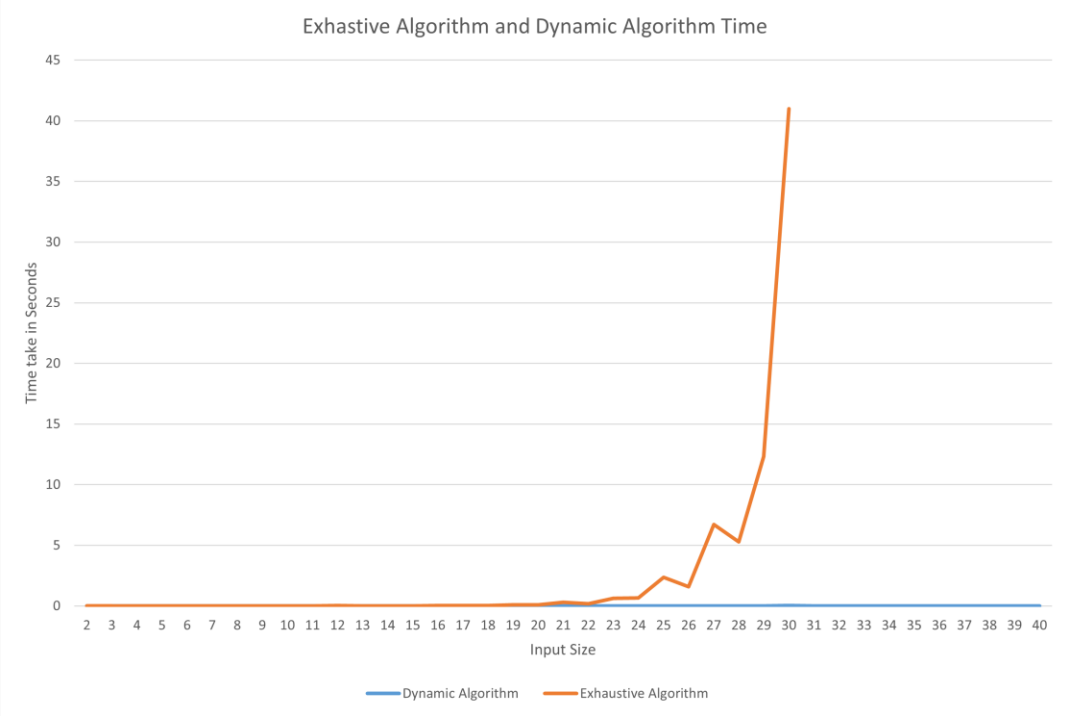
Questions

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

Answer: There is absolutely a noticeable difference between the performance of the two algorithms. The Dynamic Algorithm is much faster than the Exhaustive Algorithm at larger input sizes. This does not surprise me at all given the Big-O notations of both algorithms.

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

   Answer: In general my empirical analyses are consistent with my mathematical analysis. In so much as the fact that the Exhaustive Algorithm is much slower than the Dynamic Algorithm at scale. In fact beyond an input size of 30 the Exhaustive Algorithm is not really practical anymore as it starts to take 40+ seconds to run.



   I will note however that I had a hard time finding the exact increase in time for the Dynamic Algorithm. Meaning I am uncertain if my mathematical analysis got the exact time complexity for each. But in general, the expected relationship between the two algorithms seems to be true.

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

   Hypothesis 1: Polynomial-time dynamic programming algorithms are more efficient than exponential-time exhaustive search algorithms that solve the same problem.

   Answer: Yes this evidence is consistent with hypothesis 1. The Polynomial-time dynamic algorithm with a time complexity of O(n^2) is much more efficient than the exhaustive search algorithm with time complexity of O(m + 2^n)

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

Hypothesis 2: ???

Answer: On the README for this assignment only one hypothesis is listed. Therefore I am uncertain what hypothesis 2 is and cannot say weather my evidence is consistent or not with it.