Project 2 Report

Team Members:

Ranny Khant Naing / Email: khant_naing@csu.fullerton.edu

Santiago Zavala / Email: arturo.santi015@csu.fullerton.edu

Ibrahim Israr / Email: misrar0@csu.fullerton.edu

Kyle Dang / Email: dangkyle@csu.fullerton.edu

I. Create a Document with the following

- Exhaustive Algorithm Solution Pseudocode and time analysis
- Plot a graph for time vs input size for the algorithm
- Dynamic Algorithm Solution Pseudocode and time analysis
- Plot a graph for time vs input size for the algorithm
- Answer the below question based on the algorithm run time observations

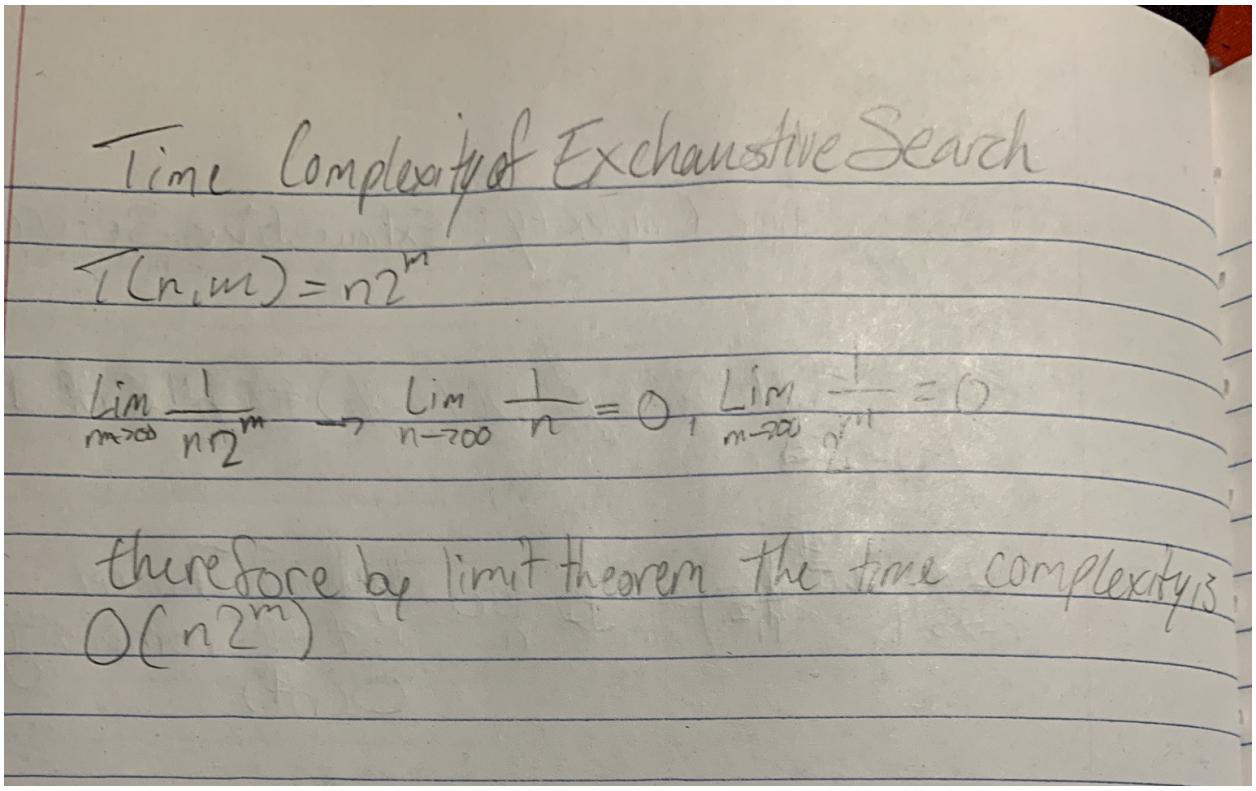
Exhaustive Search Step Count

```
//
for length = 0; length <= maximum_steps; length++                    n

   for bits = 0; bits <= 2^length  - 1; bits++                       2^n

         for k = 0; k < length; ++k                                  n

               //iterate through each bit generated from bits
               int bit;                                              1
               bit = (bits >> k) & 1                                 2

               //steps right
               if bit == 1                                           1
                     if the step is valid east                       1
```

```
                                    add step in the east                                    1

            //steps down
            else
                    else if step is valid south                                             1
                            add step in the south                                           1

        //comparing current versus the best
        if the last step of best == starting point OR                                       2
total cranes of candidate() > total cranes of candidate()
            best = candidate                                                                1

return best
```

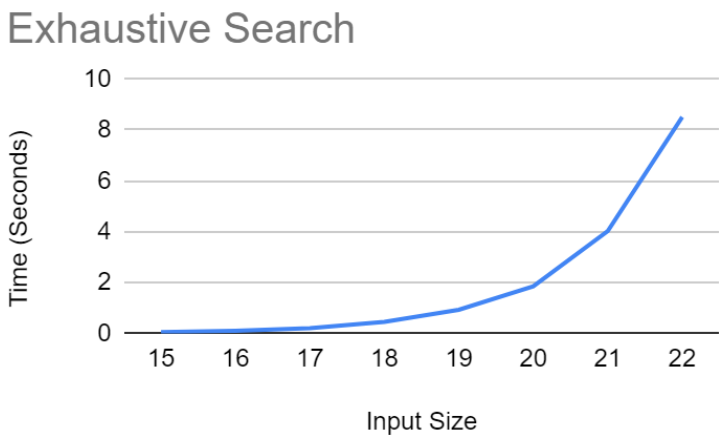
Time complexity

Step count = n * 2^n * n (1 + 2 +1 + max(2 , 2) + 3)
= 2^n * n ^ 2 * 9
Time complexity = O(2 ^ n)



Time analysis of Exhaustive Search

Exhaustive Search	
Input Size	Time
15	0.0469457
16	0.0989363
17	0.210667
18	0.455132
19	0.930642
20	1.85593
21	4.02677
22	8.50482



Dynamic Programming

Dynamic Pseudocode & Step Count

```
function crane_unloading_dyn_prog(grid setting):

    assert(settings.rows() > 0 and settings.columns() > 0)
    A[0][0] = new path(setting)                                   1 tu

    for each row r in setting.rows():                             n tu
        for each column c in setting.columns():                   n tu

            if settings.get(r,c) = Cell_Building:                 1 tu
                A[r][c] = null                                    1 tu
                continue                                          1 tu

            path from_above = null                                1 tu
            path from_left = null                                 1 tu

            if r > 0 and A[r-1][c] not null:                      2 tu
                from_above = new path(A[r-1][c])                  n tu

            if c > 0 and A[r][c-1] not null:                      2 tu
                from_left = new path(A[r][c-1])                   n tu

            if from_above is null and left is null:               2 tu
                continue                                          1 tu
            if from_above is null or (left is not null and left.total_cranes() > above.total_cranes()):
                A[r][c] = from_left
                A[r][c].add_step(direction_East)                  4 tu

    else:
        A[r][c] = from_above                                      1 tu
        A[r][c].add_step(step_Direction_South                     1 tu
    best = A[setting.rows()-1][setting.columns()-1]               1 tu
    return best                                                   1 tu
```

Step count = $n \cdot n \cdot n \cdot 18 + 1 + u + 1 + y$

SC = For loop$_1$ · For loop$_2$ · For loop block

$\boxed{SC = 18n^3 + 2u}$

Time Complexity

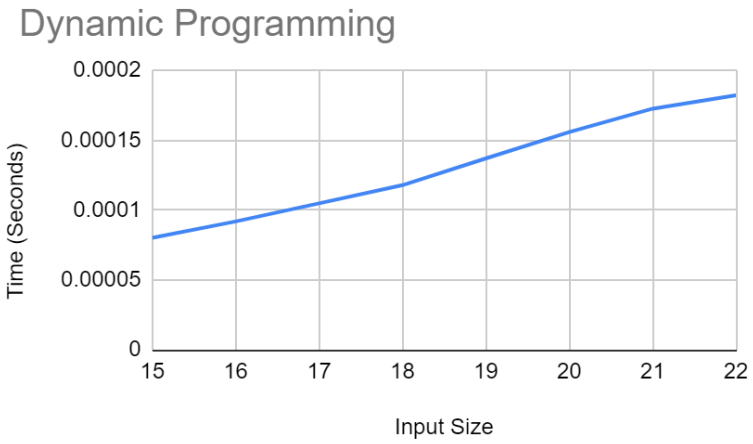
$T(n) = n \cdot n \cdot n = n^3$

$\lim_{n \to \infty} \frac{1}{n} = 0$ ∴ $\lim_{n \to \infty} \frac{1}{n} = 0$ ∴ $\lim_{n \to \infty} \frac{1}{n} = 0$

Therefore by the limit theorem, the time complexity is $O(n^3)$.

Time analysis for Dynamic Programming

Dynamic Programming	
Input Size	Time
15	8.02E-05
16	9.19E-05
17	0.000104837
18	0.000117842
19	0.00013699
20	0.00015584
21	0.00017259
22	0.00018214



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?

There is a significant difference in the performance of the two algorithms. The dynamic programming algorithm is faster because its time complexity is polynomial (O(n^3)), whereas the exhaustive algorithm has an exponential time complexity (O(2^n)). No, it does not surprise me, as dynamic programming is usually used to optimize algorithms by reusing previous computed results.

2. Are your empirical analysis consistent with your mathematical analyses? Justify your answer.
   Yes, the empirical analysis is consistent with the mathematical analyses. Both analyses indicate that the dynamic programming algorithm would be more efficient, especially for larger inputs.

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

   The evidence is consistent with the hypothesis. The hypothesis stated that "Polynomial-time dynamic programming algorithms are more efficient than exponential-time exhaustive search algorithms that solve the same problem." Given the time complexity of both algorithms, it can be concluded that the dynamic programming algorithm is indeed more efficient than the exhaustive search algorithm.