- Exhaustive Algorithm Solution Pseudocode and time analysis

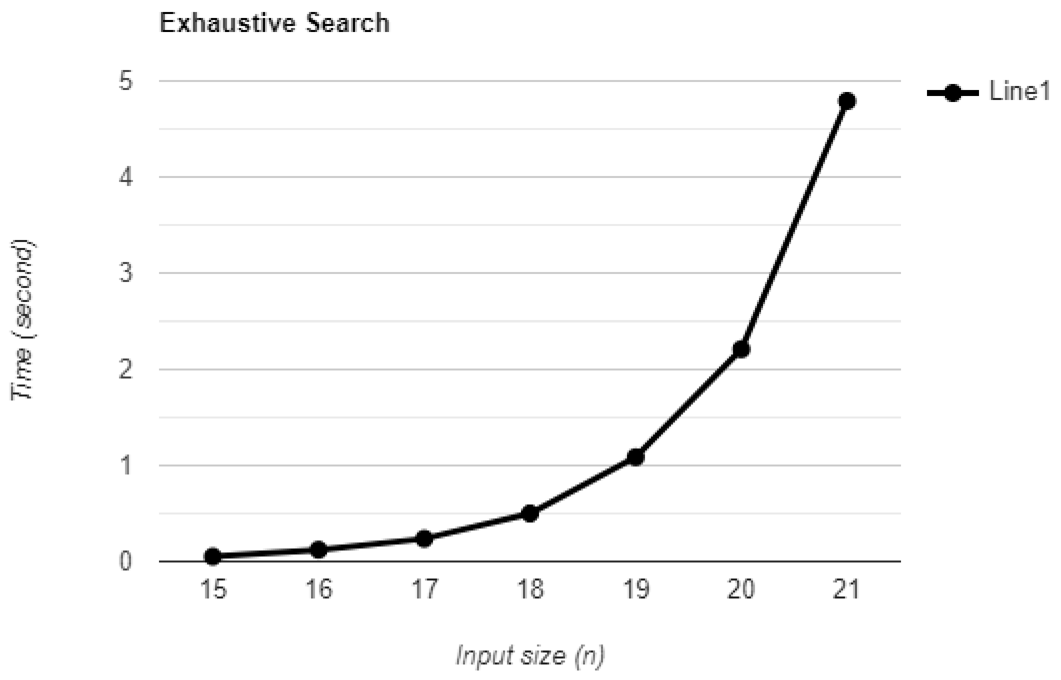
```
function crane_unloading_exhaustive(setting)
set grid must be non-empty
assert(setting.rows() > 0)
assert(setting.columns() > 0)

size_t max steps = setting.row() + setting.column() - 2
assert(max steps < 64) must be small enough to fit in a 64-bit int

path best(setting)
for steps = 0 to max steps do
    for row = 0 to pow(2, steps) do
        Path candidate(setting)
        for col = 0 to steps do
        auto rows = (row >> col) & 1  (iterate through each rows)
        If (rows ==1)
            check if candidate step is valid from East
            then add step to candidate from East
        else
            Check if candidate step is valid from South
            then add step to candidate from South
        if candidate number of total crans is greater than best
            set new best = candidate
        endfor
    endfor
endfor
return best
end
```

Step Count = n * 2^n * n (2 + 1 + 1 + max(2,2) + 2)

= ~~n^2~~ * 2^n * ~~8~~

Time complexity = O(2 ^ n)

- Plot a graph for time vs input size for the algorithm



- Dynamic Algorithm Solution Pseudocode and time analysis

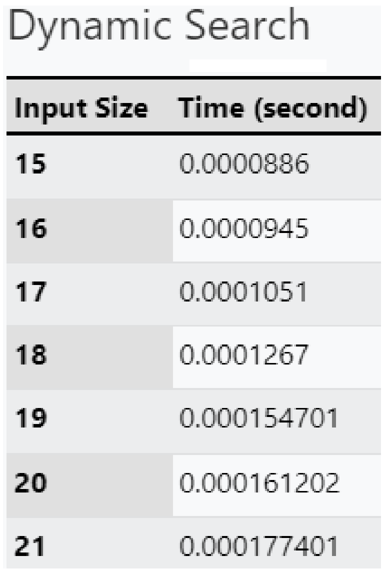
```
function crane_unloading_dyn_prog(setting)
set grid must be non-empty
assert(setting.row() > 0)
assert(setting.column() > 0)

create 2D array of size setting.row() * setting.column()
set A[0][0] to path(setting) initialize the start
assert(A[0][0].has_value())

for r = 0 to setting.rows()
    for c = 0 to setting.column()
        check if path at (r,c) is at cell_building
        then reset A[r][c]
        continue
        initialize cell_type from_above & from_left
        if r is greater than 0 and A[r-1][c] has value
        then from_above = A[r-1][c]
            check if from_above is valid step from South
            then add step from_above from South

        if c is greater than 0 and A[r][c-1] has value
        then from_above = A[r][c-1]
            check if from_left is valid step from East
            then add step from_left from East

        Check if from_above has value & from left has value
        If from_above has total_cranes then A[r][c] = from_above
        else from_left has total_cranes then A[r][c] = from_left
        Check if from_above has value & from_left does not has value
        Then set A[r][c] = from_above
        Check if from_above does not has value & from_left has value
        Then set A[r][c] = from_left
    endfor
endfor

set cell_type has pointer value of best to address of A[0][0]
for r = 0 to setting.rows()
    for c = 0 to setting.columns()
        Check if A[r][c] has value & A[r][c] total crane is greater than
        best total crane
        Then set best = address at A[r][c]
    endfor
endfor
assert(best->has_value())

return **best
```

Step Count = max(n * m * 15 + 2, n * m + 3 + 1)

= ~~15~~ nm + ~~2 , nm +4~~

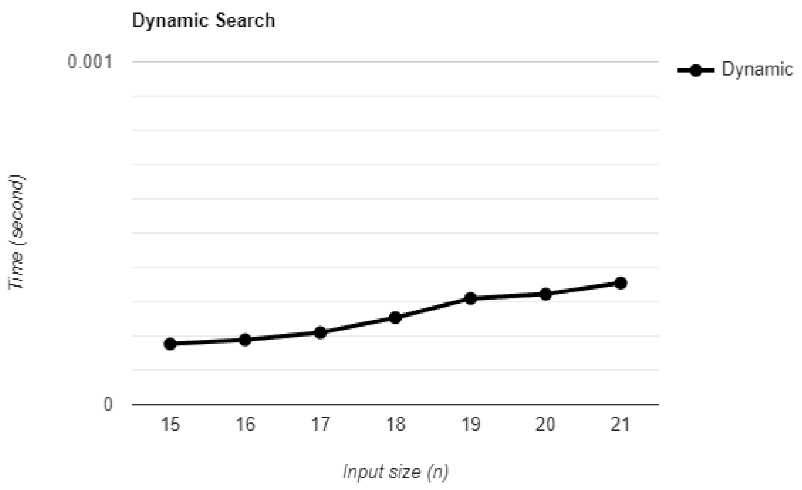
Time Complexity = O(nm)

- Plot a graph for time vs input size for the algorithm



Dynamic Search

Input Size	Time (second)
15	0.0000886
16	0.0000945
17	0.0001051
18	0.0001267
19	0.000154701
20	0.000161202
21	0.000177401



- Answer the below question based on the algorithm run time observations

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?

Even from the initial input size of n = 15, between the two search algorithms there is a noticeable difference about two decimal points. Without a doubt, dynamic programming is much faster and consistent. Originally, the input size test was going to be tested up to 22, but once n = 22 the elapsed time for the exhaustive search came close to 10 seconds which started to become a huge difference between all inputs. So input size was tested for 15 to 21. Even with testing up to 21, as you can see from the line graph of the dynamic search, the y-axis of time is only shown from 0 to 0.001 which is still too small for the graph. For exhaustive search, the graph becomes quite noticeable even after the 4th input. These results did not surprise me because I know the time complexity between the two searches is noticeably big.

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

Yes, as examined the empirical analyses are consistent with the mathematical analyses. The exhaust search requires much more iterations to complete since the worst-case time complexity is at O(2^n). As for the dynamic program, iterating through the grid took much less iterations, especially since the worst-case time complexity is at O(nm). Since the time complexity of exhaust search is exponential, time for input size getting larger, the system will eventually won’t be able to comply with the program.

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

Hypothesis 1 states that “Polynomial-time dynamic programming algorithms are more efficient than exponential-time exhaustive search algorithms that solve the same problem”. After going through both of the searches, we can conclude that the exhaust search which is exponential-time search is definitely less efficient.

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

Since the exhaust algorithm grows exponentially over the size of the grid, it is true that the dynamic programming results are more efficient. Looking at the two graphs again, the input of dynamic search does not even exceed the time of exhaust search at its first input. This concludes the evidence to support the 2nd hypothesis.