Luke Makishima
LukeMakishima@csu.fullerton.edu
Carlos Felix
carlosfjr42@csu.fullerton.edu

CPSC 335 Project 2: Crane Unloading

Exhaustive Optimization Algorithm:
<u>Pseudocode:</u>

```
path crane_unloading_exhaustive(setting)
        assert(setting.rows > 0)
        assert(setting.columns > 0)
        size_t max_steps = setting.rows + setting.column - 2
        assert(max_steps < 64)
        path best(setting)

        for size_t steps = 0 in max_steps do
                msk = uint64(1)
                for uint64_t bits = 0 in msk do
                        path candidate(setting)
                        Bool valid = TRUE
                        for size_t i = 0 in steps do
                                Size_t bit = (bits >> i) & 1
                                if bit == 1 do
                                        if (candidate.is_step_valid(STEP_DIRECTION_EAST))
                                        do
                                                candidate.add_step(STEP_DIRECTION_EAST))
                                        else do
                                                valid = FALSE
                                         endif
                                else do
                                        if (candidate.is_step_valid(STEP_DIRECTION_SOUTH))
                                        do
                                                candidate.add_step(STEP_DIRECTION_SOUTH)
                                        elsedo
                                                valid = false;
                                        Endif
                                Endif
                        endfor
                        If (valid == true and (candidate.total_cranes > best.total_cranes) do
```

best = candidate
endif
endfor
Endfor
return best
DONE

Time Complexity Step Count:

SC = 3 + 3 + 5 + 2 + 1 + (**Σ**s = 1 to n) [**Σ**(b = 0 to 2^s-1)[1+ 1 + 1 + **Σ**(i = 0 to s-1)[3+3] + 5]]

SC = 14 + (**Σ**s = 1 to n) [**Σ**(b = 0 to 2^s-1)[3 + 6(s - 1 + 1) + 5]]

SC = 14 + (**Σ**s = 1 to n) [**Σ**(b = 0 to 2^s-1)[6s]] + (**Σ**s = 1 to n) [**Σ**(b = 0 to 2^s-1)[8]]

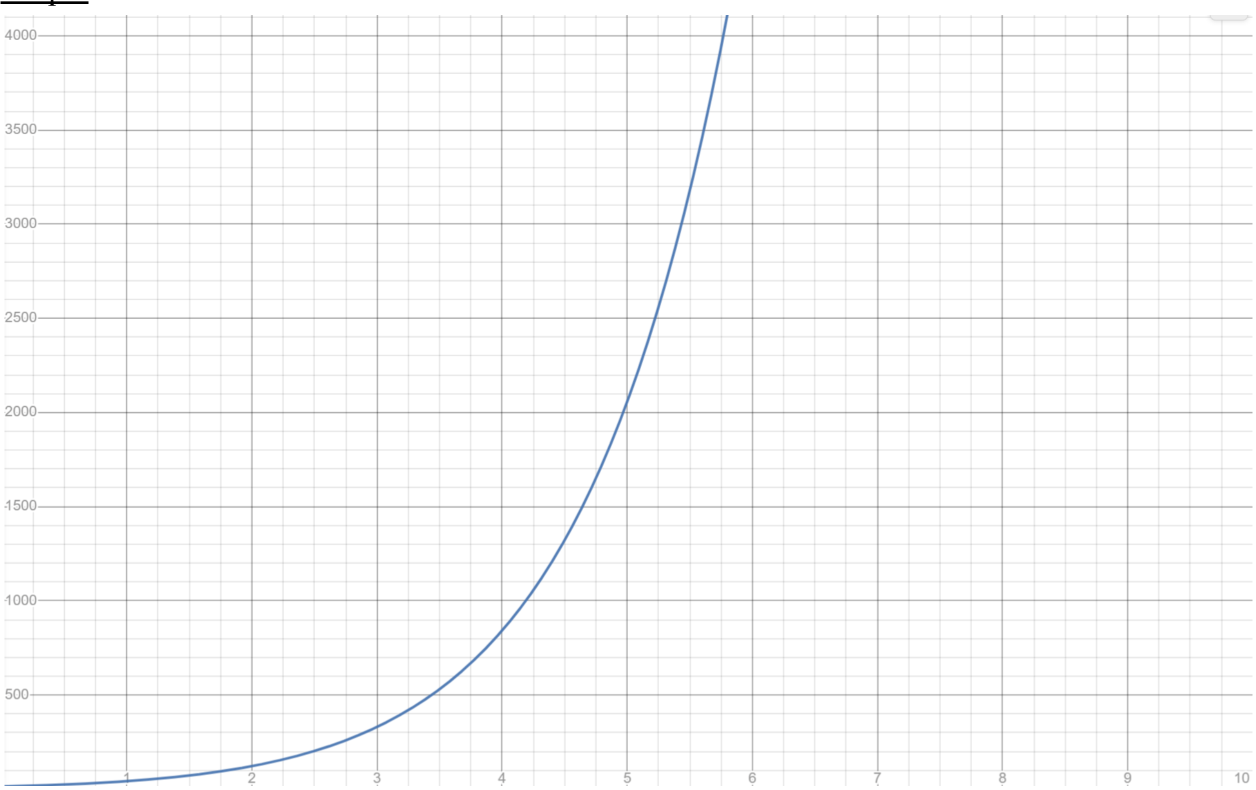
SC = 14 + (12(1 - 2^n + 2^nn)) + 16(2^n - 1)

SC = 14 + 12 - 12(2^n) + 12(2^n)(n) + 16(2^n) - 16

SC = 12(2^n)(n) + 4(2^n) + 10

Therefore, this algorithm falls under O(2^n * n^2) time complexity

Graph:



Dynamic Programming Algorithm:

Pseudocode:

path crane_unloading_dyn_prog (setting)

```
assert(setting.rows > 0)
assert(setting.columns > 0)
using cell_type = std::optional<path>

std::vector<std::vector<cell_type> > A(setting.rows,
                                       std::vector<cell_type>(setting.columns)

A[0][0] = path(setting)
assert(A[0][0].has_value)

for coordinate r = 0 in setting.rows do
        for coordinate c = 0 in setting.columns do
                if setting.get(r, c) not = CELL_BUILDING
                        cell_type from_above
                        cell_type from_left
                        if r > 0 and A[r - 1][c].has_value do
                                from_above = A[r - 1][c]
                                if from_above->is_step_valid
                                (STEP_DIRECTION_SOUTH) Do
                                from_above->add_step(STEP_DIRECTION_SOUTH)
                                endif
                        endif
                        If c > 0 and A[r][c-1].has_value do
                                From_left = A[r][c - 1]
                                If from_left->is_step_valid
                                (STEP__DIRECTION_EAST) do
                                        from_left->add_step(STEP_DIRECTION_EAST)
                                endif
                        endif
                        if (from_above.has_value and from_left.hasvalue do
                                if from_above->total_cranes > from_left->total_cranes do
                                        A[r][c] = from_above
                                else
                                        A[r][c] = from_left
                                endif
                        endif
                        if from_above.has_value and not from_left.hasvalue do
                                A[r][c] = from_above
                        endif
                        if from_left.has_value and not from_above.hasvalue do
```

```
                                        A[r][c] = from_left
                                endif
                        endif
                endfor
        endfor
cell_type best = &A[0][0]
assert(best->has_value)

for coordinate r = 0 in setting.rows do
        for coordinate c = 0 in setting columns do
                if (A[r][c].has_value and A[r][c]->total_cranes > (*best)->total_cranes do
                        best = &(A[r][c])
                endif
        endfor
Endfor

assert(best->has_value)

return **best
DONE
```

<u>Time Complexity:</u>

SC = 3 + 3 + 2 + 6 + 2 + 2 + n(n*(2 + max (1 + 1 + 8 + 8 + 7 + 5 + 5, 0)))) + 3 + 5n^2

SC = 21 + n(n*(2+35)) + 5n^2

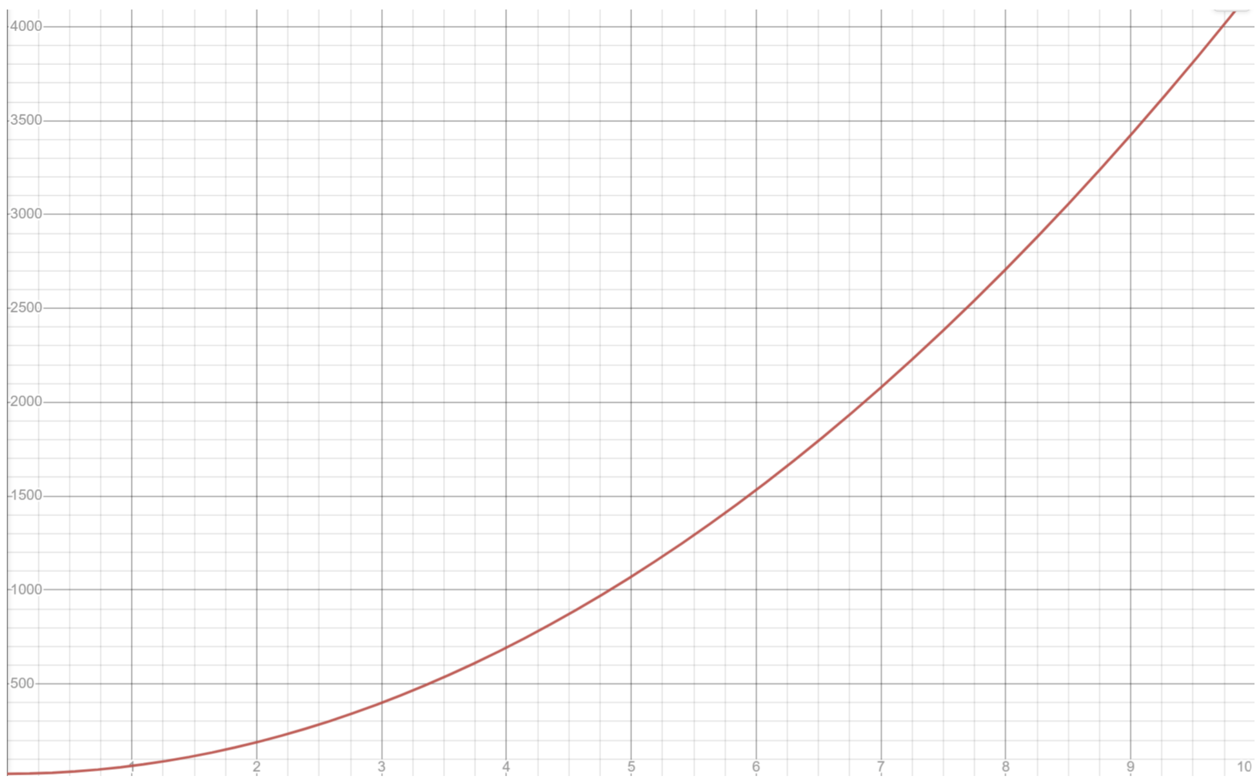
SC = 21 + 37n^2 + 5n^2

SC = 21 + 42n^2

SC = 42n^2 + 21

Therefore, this algorithm falls under O(n^2) time complexity

<u>Graph:</u>



QUESTIONS:

1. There is a noticeable difference in the performance of the two algorithms. The dynamic programming approach is faster by quite a significant margin. This answer does not surprise me as the brute force method of the exhaustive optimization algorithm is not very efficient at all.
2. Our empirical analyses do match with our mathematical analyses. We mathematically proved that the exhaustive optimization algorithm has a time complexity of O(2^n * n^2), making the algorithm exponential time which is very slow. We also mathematically proved that the dynamic programming algorithm belongs in O(n^2) time, which is quadratic or polynomial, significantly faster than the exponential time the exhaustive optimization runs at.
3. Our data is consistent with hypothesis 1. As stated in question 2, the polynomial time complexity of the dynamic programming algorithm is significantly faster than the exponential time complexity of the exhaustive optimization algorithm.

Demonstration

https://drive.google.com/file/d/1Nx7dJJ5_lxMKQzDZnBoL1vLKTcdFf-ND/view?usp=sharing