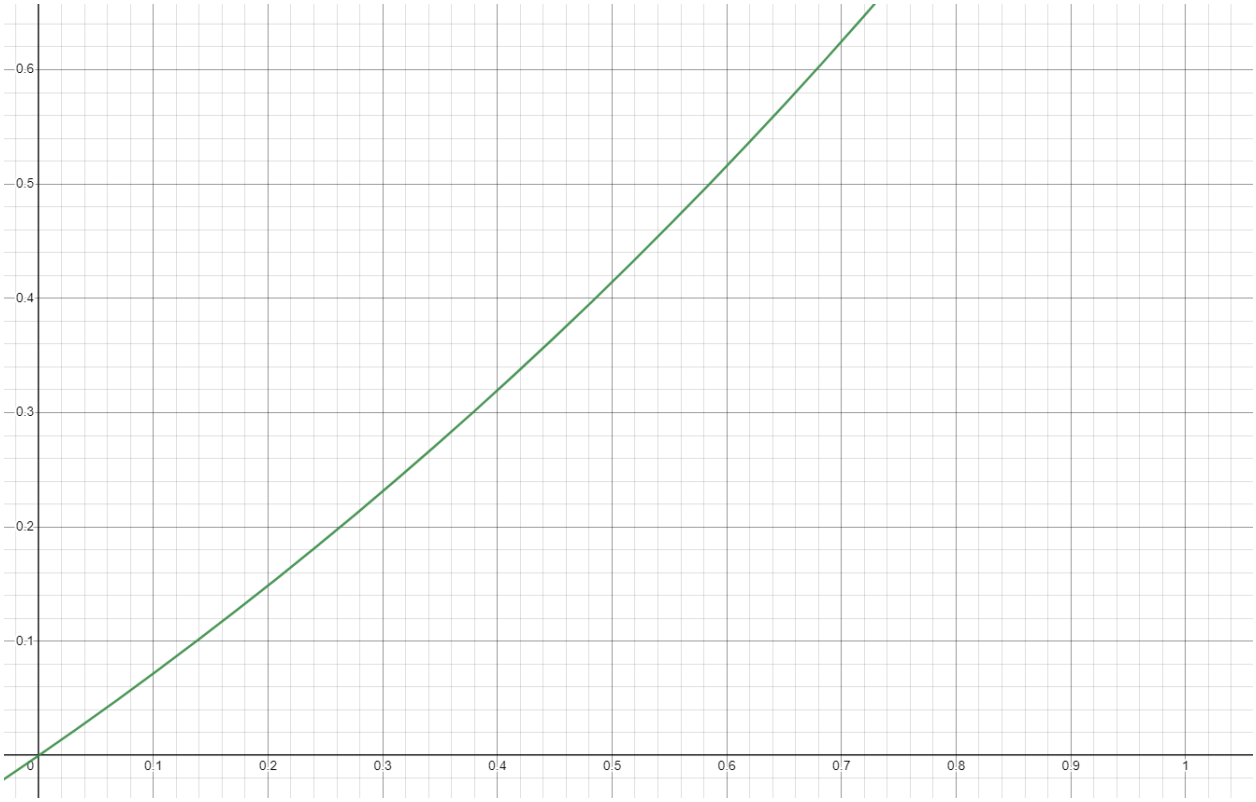
Saul Ruiz
Saul385@csu.fullerton.edu
Project 2 335
Crane Problem
Himani Tawade
5/14/23
https://github.com/Saul385/Crane-Problem/tree/main

Exhaustive Pseudo:

```
function crane_unloading_exhaustive(setting)
    assert setting.rows() > 0 and setting.columns() > 0  // Grid
non-empty                2tu
    max_steps <- setting.rows() + setting.columns() - 2          3tu
    assert max_steps < 64            1tu
    best <- create_path(setting)    1tu
    new_path <- create_path(setting)    1tu

    all_paths <- [new_path]          1tu

    for steps in range(0, max_steps+1)         2tu
        if best.final_row() + 1 = setting.rows() and best.final_column() + 1
= setting.columns()             4tu
            break

        new_paths <- []  1tu
        swap(new_paths, all_paths)        1tu

        while not new_paths.empty()
            current_path <- move_last_element(new_paths)       1tu

            if current_path.final_row() + 1 = setting.rows() and
current_path.final_column() + 1 = setting.columns()              1tu
```

```
            if best.total_cranes() < current_path.total_cranes() 1tu
                best <- current_path                1tu

        else
            if current_path.is_step_valid(STEP_DIRECTION_EAST) 1tu
                next_path <- copy_path(current_path)            1tu
                next_path.add_step(STEP_DIRECTION_EAST)     1tu
                all_paths.push_back(next_path)      1tu
            if current_path.is_step_valid(STEP_DIRECTION_SOUTH)
1tu
                next_path <- copy_path(current_path)    1tu
                next_path.add_step(STEP_DIRECTION_SOUTH)    1tu
                all_paths.push_back(next_path)      1tu

    return best
end function
```

For loop and while loop
SC = (n * 6tu) + (2^n) = max(7n + 2^n) =
O(2^n)



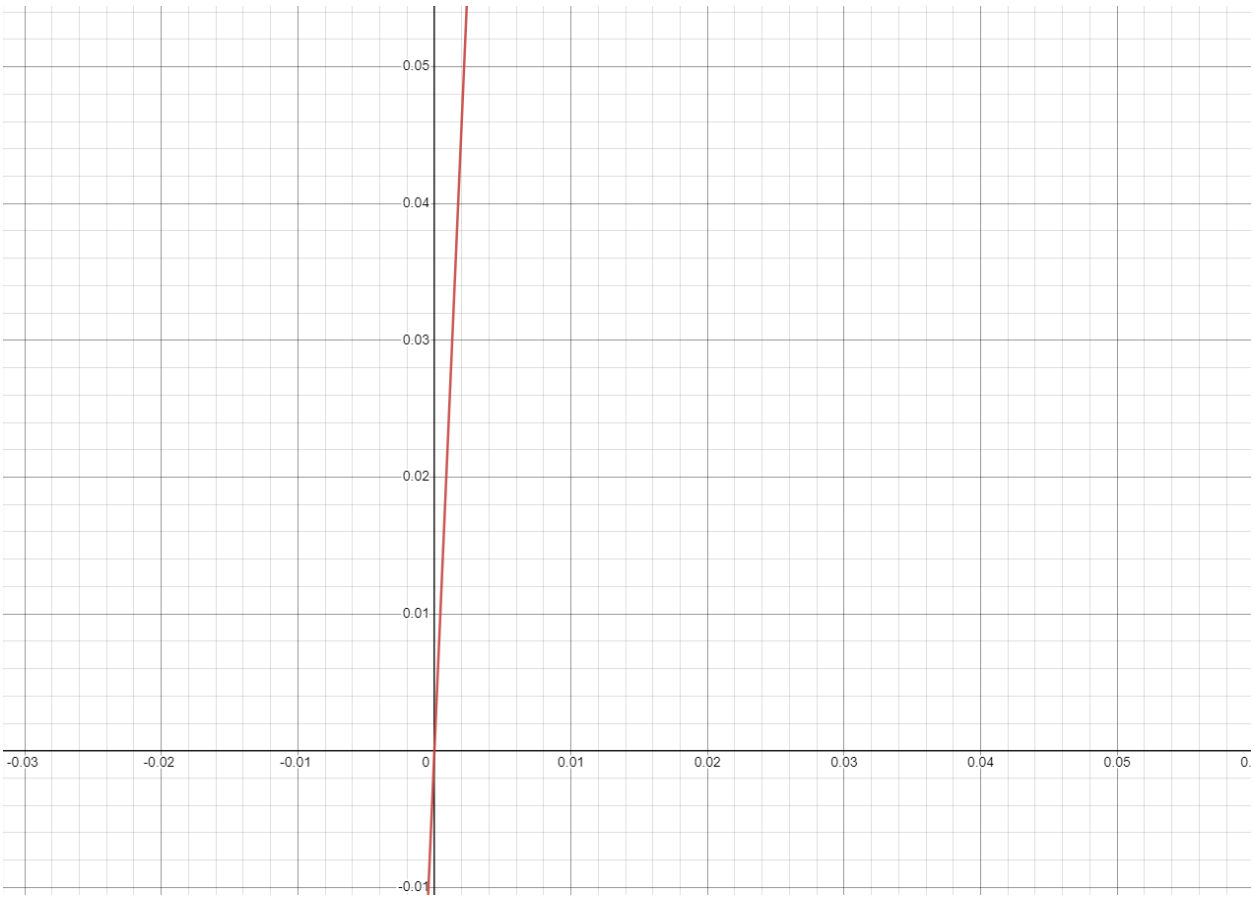
Dyn Algo Pseudo:

```
function crane_unloading_dyn_prog(setting)
    best <- create_path(setting)    1tu
    assert setting.rows() > 0         1tu
    assert setting.columns() > 0            1tu

    my_grid <- create_2d_array(setting.rows() + 1, setting.columns() +
1)
    current_cell <- undefined 1tu

    for i in range(0, setting.rows() + 1)   n
        for j in range(0, setting.columns() + 1)    m
            if i = 0 or j = 0         2tu
                my_grid[i][j] <- 0  1tu
            else
                if i = 1 and j = 1          2tu
```

```
            my_grid[i][j] <- 1      1tu
        else
            my_grid[i][j] <- 0      1tu

        current_cell <- setting.get(i - 1, j - 1)    1tu
        if current_cell = CELL_BUILDING       1tu
            my_grid[i][j] <- -1     1tu
        else
            if current_cell = CELL_CRANE      1tu
                my_grid[i][j] <- my_grid[i][j] + 1     2tu

            max <- max(my_grid[i - 1][j], my_grid[i][j - 1]) 1tu
            my_grid[i][j] <- my_grid[i][j] + max

max_steps <- setting.rows() + setting.columns() - 2     2tu
x <- setting.rows()
y <- setting.columns()

directions <- []

for i in range(0, max_steps) n
    if my_grid[x - 1][y] = -1 and my_grid[x][y - 1] = -1          4tu
        y <- y - 1        2tu
        x <- x - 1        2tu
    else if my_grid[x][y] = -1
        y <- y - 1        2tu
        break
    else if my_grid[x - 1][y] >= my_grid[x][y - 1] and x != 1   5tu
        directions.append(STEP_DIRECTION_SOUTH)
        x <- x - 1        2tu
    else if y != 1        1tu
        directions.append(STEP_DIRECTION_EAST)
```

```
            y <- y - 1

    size <- directions.length()

    for i in range(size, 0, -1)          n
        current_direction <- directions.back()

        if best.is_step_valid(current_direction)         tu
            best.add_step(current_direction)     tu
        directions.pop_back()          tu
    return best
end function
```

for(for) + for + for
SC: (n + 9tu, n + 8tu, n + 3tu) = 3^N +21tu
Time:O(3^n)



Question

1. Is there a noticeable difference in the performance of the two algorithms? Which is faster and by how much? Does this surprise you?
   Yes there is a difference between the 2 where the dynamic algo is faster than the Exhaustive Algo. The big O notations support this.
2. Are your empirical analyses consistent with your mathematical analyses? Justify your answer?
   Yes it is consistent with my mathematical analyses considering how the graphs are and which one is actually faster.
3. Is this evidence consistent or inconsistent with hypothesis 1 ? Justify your answer.
   Yes the evidence is consistent with hypothesis 1. The exhaustive search algo isnt as fast the polynomial-time dynamic algo with $O(n^2) < O(m + 2^n)$.
4. Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.
   ?