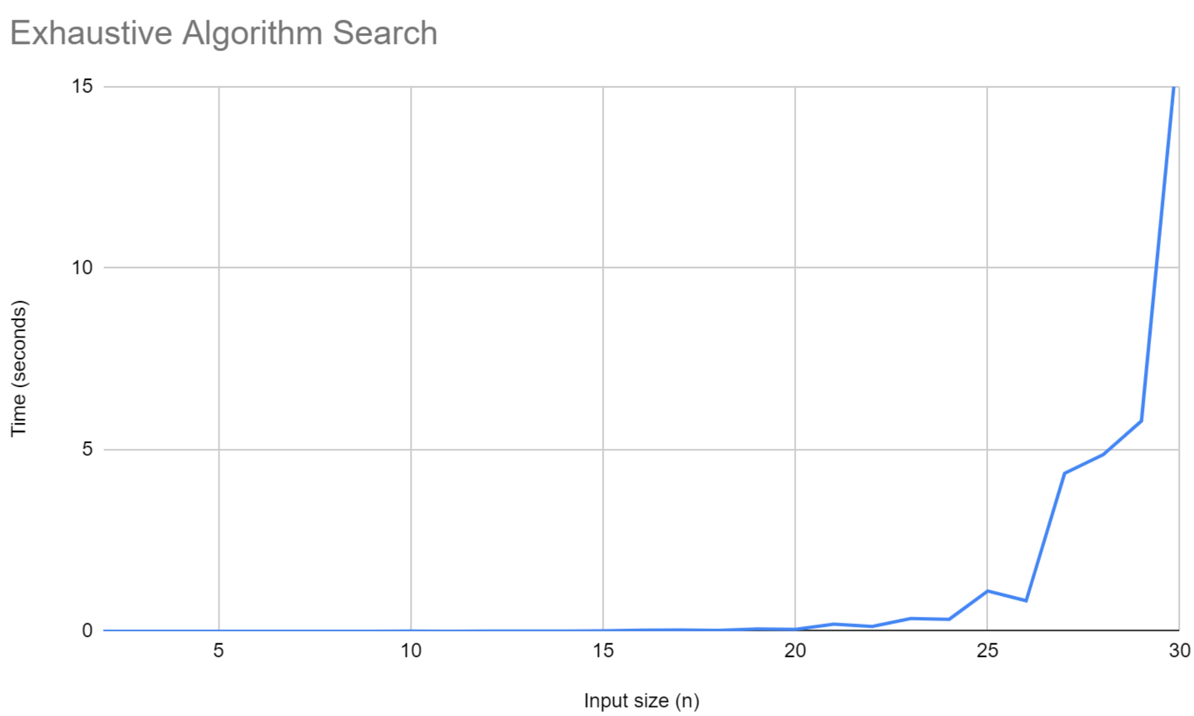
Exhaustive Algorithm Solution Pseudocode and time analysis

```
path crane_unloading_exhaustive(grid setting){

    max_steps = setting.rows + setting.columns - 2;                    -> 3 tu

    path best;
    path new_path;

    vector<path> all_paths;
    all_paths.append(new_path);                                         > 1 Tu

    for steps = 0 to max_steps:                                         ->  n loops
        if (best.final_row == setting.rows &&                           -> 1 Tu
            best.final_column == setting.columns):
            break;

        vector<path> new_paths = all_paths;                             -> 1 Tu
        all_paths.clear();

        while(!new_path.empty):                          (11 Tu per loop) = 11 * 2^n
            new_path = new_paths.back();                        -> 1Tu
            new_paths.pop_back();                               -> 1Tu
                                                         ( 1 + Max(2,8) ) = 9 Tu
            if (new_path.final_row == setting.rows &&           -> 1 Tu    (2 Tu)
                new_path.final_column == setting.columns):
                if (best.total_cranes < new_path.total_cranes):     -> 1Tu
                    best = new_path;                            -> 1Tu
            Else:                                                          (8 Tu)
                if (new_path.is_step_valid(EAST)):              -> 1Tu
                    path next_path = new_path;                  -> 1Tu
                    next_path.add_step(EAST);                   -> 1Tu
                    all_paths.append(next_path);                -> 1Tu
                if (new_path.is_step_valid(SOUTH)):             -> 1Tu
                    path next_path = new_path;                  -> 1Tu
                    next_path.add_step(SOUTH);                  -> 1Tu
                    all_path.append(next_path);                 -> 1Tu
    return best;
}
```

$$step\ count\ =\ 4\ +\ 11\sum_{i=0}^{n} 2^{i}\ tu$$

Plot a graph for time vs input size for the algorithm



Dynamic Algorithm Solution Pseudocode and time analysis

```
path crane_unloading_dyn_prog(grid setting){
   path best;
   int grid[setting.rows][setting.columns];                     ->  1 tu
   cell_kind current_cell;

   for i = 0 to setting.rows:                                   -> n/2 loops
      for j = 0 to setting.columns:                             -> n/2 loops
         if (i == 0 || j == 0):                                 -> 1tu + max( 1, 10) = 11 tu
            my_grid[i][j] = 0;                                  -> 1tu
         Else:                                                  -> 10tu
            if (i == 1 && j == 1):                              -> 1tu + max (1, 1) =  2 tu
               my_grid[i][j] = 1;                               -> 1tu
            else:
```

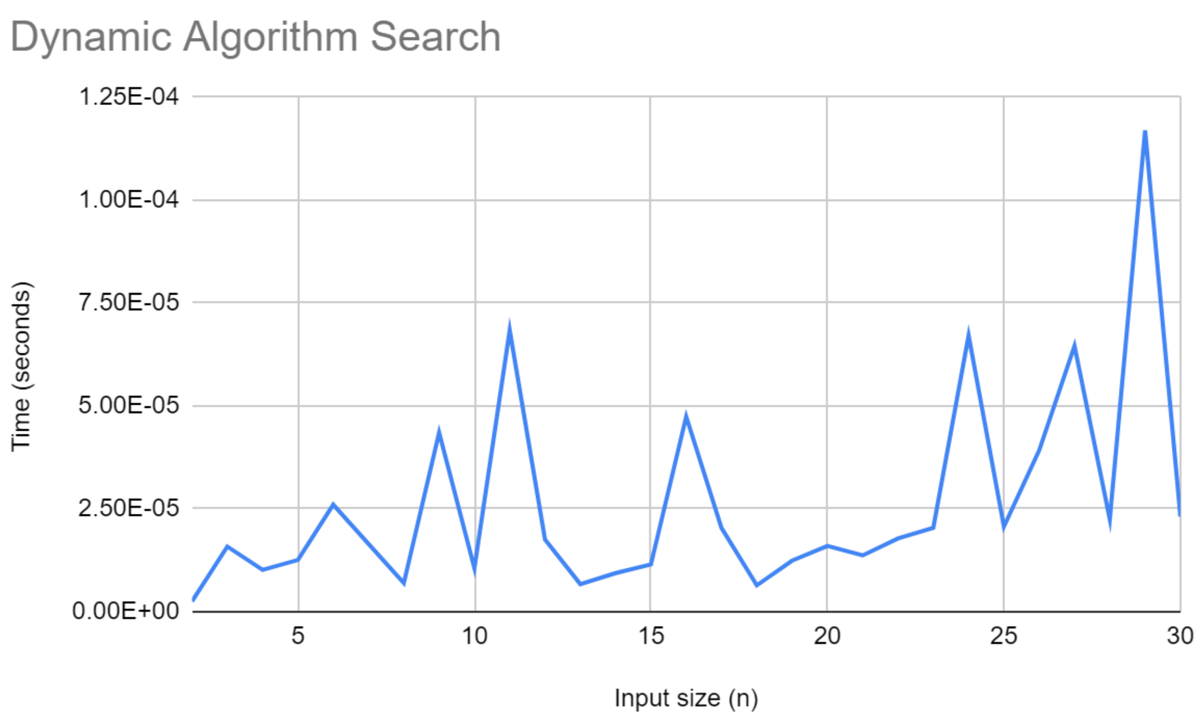
```
                my_grid[i][j] = 0;                                    -> 1tu
            current_cell = setting.get(i-1,j-1);                      -> 1tu
            if (current_cell == BUILDING):                            -> 1tu + (max ( 1, 6)) = 7 tu
                my_grid[i][j] = -1;
            Else:                                                     -> 6 tu
                if (current_cell == CRANE):                           -> 1tu + (max ( 1, 0) ) = 2 tu
                    my_grid[i][j] += 1;                               -> 1tu
                int max;                                              -> 1tu
                if (my_grid[i-1][j] > my_grid[i][j-1]):               -> 1tu + (max ( 1, 1) = 2tu
                    max = my_grid[i-1][j];                            -> 1tu
                else:
                    max = my_grid[i][j-1];                            -> 1tu
                my_grid[i][j] += max;                                 -> 1tu

max_steps = setting.rows + setting.columns - 2;                       -> 3 tu
int x = setting.rows;                                                 -> 1 tu
int y = setting.columns;                                              -> 1 tu
vector<step_directon> directions;

for i = 0 to max_steps:                                               -> m loops = m * 9 tu
    if ((my_grid[x-1][y] == -1) && (my_grid[x][y-1] == -1)):          -> 2 tu + (max ( 2, 7)) = 9 tu
        --x;                                                          -> 1 tu
        --y;                                                          -> 1 tu
    else if (my_grid[x][y] == -1):                                    -> 1 tu + (max (1, 6)) = 7 tu
        --y;                                                          -> 1 tu
        break;
    else if ((my_grid[x-1][y] >= my_grid[x][y-1]) && x != 1):         -> 2 tu + (max ( 2, 4)) = 6 tu
        directions.append(SOUTH);                                     -> 1 tu
        --x;                                                          -> 1 tu
    else if (y != 1):                                                 -> 2 tu + (max (2, 0 ) = 4 tu
        directions.append(EAST);                                      -> 1 tu
        --y;                                                          -> 1 tu

step_direction current_direction;
for i = 0 to directions.size():                                       -> m loops =  m * 4 tu
    current_direction = direction.back;                               -> 1 tu
    if (best.is_step_valid(current_direction)):                       -> 1tu + max ( 1, 0) = 2 tu
        best.add_step(current_direction);                             -> 1tu
    directions.pop_back();                                            -> 1 tu
return best;
```

}

$$step\ count\ =\ 1\ +\ (n/2)^2(11)\ +\ 13m\ \ tu$$

$$step\ count\ =\ 1\ +\ 11(n^2/4\ )\ +\ 13m\ \ tu$$

Plot a graph for time vs input size for the algorithm



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 noticeable difference between both algorithms; dynamic looks more messy but it is in milliseconds. In conclusion, Dynamic search is much faster than Exhaustive search. The results are not surprising at all.
2. Are your empirical analyses consistent with your mathematical analyses? Justify your answer.
   - Yes, we thought the Dynamic would outperform exhaustive search by a landslide, and our tests were consistent.

3. Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.
    - This evidence is consistent with hypothesis 1 because the time to execute for both algorithms with the same problem, Dynamic programming outshined the exhaustive algorithm.
4. Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.