Exhaustive Algorithm Solution Pseudocode and Time Analysis

```
function crane_unloading_exhaustive(setting):
    assert setting.rows() > 0 —-------> 1 tu
    assert setting.columns() > 0 —-------> 1tu

    max_steps = setting.rows() + setting.columns() - 2 —-------->1tu
    assert max_steps < 64 —-------> 1 tu

best = initialize_best_path(setting) —-------> 1tu

for steps = 0 to max_steps:
    for bits = 0 to pow(2, steps) - 1:
        nextStep = initialize_path_with_setting(setting)  —-------> 1tu
        valid = true  —-------> 1tu

        for k = 0 to steps - 1:
            bit = (bits >> k) & 1
            if bit == 1 —-------> 1tu
                step_direction = go_right —-------->1tu
            else
                step_direction = go_down—-------->1tu

            if nextStep.is_step_valid(step_direction) —-------->1tu
                nextStep.add_step(step_direction) —-------->1tu
            else
                valid = false —-------> 1tu
                break
         endfor

        if valid and nextStep.total_cranes() > best.total_cranes() —-------> 1tu
            best = nextStep —-------> 1tu
     endfor
endfor
return best
```

$$5 + \sum_{s=0}^{max} \sum_{b=0}^{pow(2,s)} 4 \sum_{k=0}^{s-1} 6$$

Since there’s a power function, then the Exhaustive Algorithm will end up having an exponential time complexity.

Dynamic Algorithm Solution Pseudocode and Time Analysis

```
function crane_unloading_dyn_prog(setting):

    assert setting.rows() > 0  —------> 1 tu
    assert setting.columns() > 0 —------> 1 tu

    cell_type = optional<path> —------> 1 tu

    A = create_2d_vector(setting.rows(), setting.columns()) —------> r * c

    A[0][0] = create_new_path(setting) —------> 1 tu
    assert A[0][0].has_value() —------> 1 tu

    for r = 0 to setting.rows() - 1
        for c = 0 to setting.columns() - 1
            if setting.get(r, c) == CELL_BUILDING —------> 1 tu
                A[r][c].reset() —------> 1 tu
                continue

            from_above = null —------> 1 tu
            from_left = null —------> 1 tu

            if r > 0 and A[r - 1][c].has_value()
                from_above = A[r - 1][c] —------> 1 tu
                if from_above.is_step_valid(go_down)
                    from_above.add_step(go_down) —------> 1 tu

            if c > 0 and A[r][c - 1].has_value() —------> 1 tu
                from_left = A[r][c - 1] —------> 1 tu
                if from_left.is_step_valid(go_right) —------> 1 tu
                    from_left.add_step(go_right) —------> 1 tu

            if from_above has value and from_left has value
                if from_above.total_cranes() > from_left.total_cranes() —------> 1 tu
                    A[r][c] = from_above —------> 1 tu
                else
                    A[r][c] = from_left —------> 1 tu
            else if from_left has value
                A[r][c] = from_left —------> 1 tu
            else if from_above has value
```

```
            A[r][c] = from_above —-------> 1 tu
        endfor
    endfor
   best = A[0][0] —-------> 1 tu

   for r = 0 to setting.rows() - 1
      for c = 0 to setting.columns() - 1
         if A[r][c].has_value() and A[r][c].total_cranes() > best.total_cranes() —-------> 1 tu or
r*c
            best = A[r][c] —-------> 1 tu
      endfor
    endfor

   assert best.has_value()
   return best
```

$$\sum_{r=0}^{r-1}\sum_{c=0}^{c-1} 17 + 1 + \sum_{r=0}^{r-1}\sum_{c=0}^{c-1} 2$$

Since the Dynamic Algorithm appears to have a r*c, it would indicate that it has a polynomial time complexity which would mean it is faster than the exhaustive 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?
    a. After running the code, it seems like that Dynamic Algorithm is the faster one of the two algorithms by a lot. This does not surprise me because of the nature of how Exhaustive search has to explore all the potential paths to decide which is the best. Exhaustive searches typically have exponential time complexities while Dynamic Algorithms can have polynomial time complexities.
2. Are your empirical analyses consistent with your mathematical analyses? Justify your answer.
    a. An empirical analysis should be consistent with a mathematical analysis, depending on a problem's time complexities. Since an expected time complexity for the Dynamic is a polynomial and exhaustive is exponential, if they were to change, then it would not be consistent with a mathematical analysis.
3. Is this evidence consistent or inconsistent with hypothesis 1? Justify your answer.
    a. The hypothesis: “This experiment will test the following hypothesis: Polynomial-time dynamic programming algorithms are more efficient than exponential-time exhaustive search algorithms that solve the same problem.”

   b. Based on this hypothesis, the evidence would be considered consistent. This is because when running the code, while both are able to return 4 cranes, the elapsed time for exhaustive optimization is around 0.122773 and dynamic is around 0.0003671. While these numbers do change, the dynamic algorithm demonstrates a faster time every time.

4. Is this evidence consistent or inconsistent with hypothesis 2? Justify your answer.
   a. Hypothesis 2 would mean that exponential search is more efficient than polynomial.
   b. Based on this hypothesis, the evidence would be inconsistent because in this case, the exhaustive search has an exponential time complexity as it has to go through every possible path first before deciding which would be the best. This is in contrast to dynamic which has completed the same problem in less time.