**A.** A greedy algorithm is only optimal if the path taken is optimal against every other path. Hence the first choice may be optimal, however the second choice may not be the most optimal solution.

|  |  |  |
| --- | --- | --- |
| 4 | 9 | 4 |
| 7 | @ | 5 |
| 7 | 4 | 4 |

This example, the greedy algorithm would take the path: 9-4, whereas the actual optimal path is 7-7. Hence all paths must be examined as it is not only the first value that makes the path optimal.

**C.** The algorithm will complete in O(n2), where n would be the number of rows, rather than the middle position. As there are four quadrants, this number would multiply by four, resulting in an equation for worst case: f(n) = 4( (2n + 1) x (2n + 1 ) )2 where maximizing the size of n will result in the worst case, as the time is solely determined by the size of the grid: there are no other constraints.

|  |  |  |
| --- | --- | --- |
| 0,3 | 8,0 | 0,6 |
| 0,8 | 7,0 | 3,0 |
| @ | 0,3 | 0,9 |

8W -> 7F -> 3F -> 6W would be the path that Greedy follows, for example if we currently have minimum water, we then aim to collect the maximum food we can collect on our next move. That gives a result of 10F, 14W however the true optimal solution would be: 8W -> 7F -> 8F -> 6W which results in 15F and 14W, min(15,14) > min(10,14). Hence the greedy algorithm does not work, we must explore future paths.

**F.** The worst case would be when we maximize the amount of possibilities each cell can have: that is such that a cell cannot produce a tuple smaller than another cell. For example: (8,4) and (9,3): 8 < 9 and 4 > 3 hence this cannot be pruned, if this was the case for all cells then the number of possibilities is maximized.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | (6,0) | (0,9) | (8,0) |
|  |  | (0,8) | (5,0) | (4,0) |
|  |  | @ | (7,0) | (0,5) |
|  |  |  |  |  |
|  |  |  |  |  |

That being said, the worst case time complexity: O(n4) as this would be when the pruning fails.