Complexity Order

For our project we decided to use a greedy DFS recursive search to find the best path from the source to the sink. Given that the function is recursive you would think you would be able to apply master's theorem to it, but since it does not fulfill the constraint of a >=1, **b > 1**, and d > 0 (fails on b > 1) we are unable to apply it.

Taking another approach, we can look at the constraint of the number of edges we can use. Since the max number of edges we can use is 15-30 we can just call that N as the number of edges to be used. Next taking a look at the number of moves that can be made from a node is 7 (no revisiting the node we just came from) we can consider the maximum number of comparisons (our main operation) to be made as O(7N). Since the 7 is a constant we can drop it and we end up with O(N) as our complexity for our algorithm.

This is not like other DFS algorithms that consider the other paths it can take because once it chooses its next node it discard all other options. We originally implemented it to consider all other options by backtracking but we ended up with a stack overflow since the grid is way too big and there are just too many options it can go to.

Since we implanted it this way, increasing the input size, graph size, or even making a more complex graph our algorithm will still hold true because it checks if the node goes out of bound. It only considers valid nodes and chooses the highest possible value every time so it should have no problem with any of these changes.