For this assignment, I was tasked with implementing an algorithm to find the most efficient solution to a maze puzzle, traversing a grid cell by cell.

This solution is an implementation of a depth-first recursive backtracking approach, since that is the approach that I am most comfortable and familiar with implementing. In the **recursive case,** each next valid neighbor cell is appended to a temporary solution container, and the backtracking function calls itself to calculate the neighbors of that cell and so on. In the **first base case,** the destination is reached, and in the **second base case,** no valid neighbor cells remain (which will be reached in the event of a given path reaching a dead end).

To enable a depth-first search, the backtracking function saves copies of the memo and the stepwise cost and restores them after each backtracking call returns, so that the function can dive all the way into one potential path, reset, and go back and explore the others. Every time a destination is reached, the cost (in steps) is compared to our current "best solution" and updated if necessary. Once the puzzle has been fully explored, and the final best solution is returned at the end.

Since this approach solves the input puzzle using depth-first backtracking, the **time complexity** for my implementation is **O**(**n**+**m**) where **n** is the number of valid spots in the graph and **m** is the number of valid paths between spots.