**Prove the correctness of the algorithm that you wrote to verify the solution.**

def is\_valid\_sudoku(board\_to\_test):

temp\_board = list(board\_to\_test)

for row in range(9):

for col in range(9):

if temp\_board[row][col] == 0:

return {'is\_valid': False, 'invalid\_row': row, 'invalid\_column': col, 'invalid\_digit': 0}

temp = temp\_board[row][col]

temp\_board[row][col] = 0

if not is\_placement\_possible(y=row, x=col, n=temp, board=temp\_board):

return {'is\_valid': False, 'invalid\_row': row, 'invalid\_column': col, 'invalid\_digit': temp}

temp\_board[row][col] = temp

return {'is\_valid': True, 'invalid\_row': None, 'invalid\_column': None, 'invalid\_digit': None}

def is\_placement\_possible(y, x, n, board):

copied\_board=board.copy()

for i in range(0, 9):

if copied\_board[y][i] == n:

return False

if copied\_board[i][x] == n:

return False

subsquare\_x = (x//3)\*3

subsquare\_y = (y//3)\*3

for i in range(3):

for j in range(3):

if copied\_board[subsquare\_y+i][subsquare\_x+j] == n:

return False

return True

Proof of Correctness

1. Prove the algorithm works for one particular input.
2. Prove that for all inputs, the algorithm correctly determines whether the input is valid or not.
   1. Prove the algorithm terminates: The algorithm terminates in three different ways:
      1. If the value of any particular cell is 0, then the algorithm will terminate. This is due to the fact that no valid Sudoku board contains a zero. In the algorithm, it can be seen that if this is the case, the algorithm returns that the ‘is\_valid’ value is False.
      2. If the value placed in any particular cell is not a possible input (the value is already found in the row of that cell, the column of that cell, or the subsquare of that cell), then the algorithm will terminate. This is due to the fact that no valid Sudoku board can contain a value that is repeated in the same row, column or subsquare. In the algorithm, it can be seen that if a repeated value is encountered, the algotihm returns that the ‘is\_valid’ value is False
      3. If the algorithm has investigated all possible cells on the board, and determined every cell to be valid, then thee entire board has been determined to be valid. This can be seen since the end of the ‘is\_valid\_sudoku’ function returns an ‘is\_valid’ value of True, and this return statement is only accessible after all cells of a provided board have been examined and determined to be valid. Thus, the algorithm will always stop after completing x loops over the rows and y loops over the columns.
   2. Prove that the algorithm determines a valid solution to each posed board. This is proven using the loop invariant (the property of the loop that should always be true). In this case, the loop invariant is that after looping over x rows and y columns, the ‘board’ that contains cells with rows <= x and column values <= y will only contain valid inputs (if the algorithm has not terminated into one of the above-mentioned base cases). Put another way, the loop invariant is that if each cell that has been inspected up to the current cell have been valid, then all cells up the current cell are valid.
      1. Prove the base case that if the value of any particular cell is 0, then the algorithm will terminate. Given a provided board where the top left cell is 0 (row 0, column 0), we can walk-through the algorithm to determine the result.
3. def is\_valid\_sudoku(board\_to\_test):
4. temp\_board = list(board\_to\_test)
5. for row in range(9):
6. for col in range(9):
7. if temp\_board[row][col] == 0:
8. return {'is\_valid': False, 'invalid\_row': row, 'invalid\_column': col, 'invalid\_digit': 0}

In this part of the algorithm, we can see that if the board\_to\_test contains the topleft cell as zero, then a temporary duplicate of the provided board\_to\_test array is created. No differences have been incorporated into the duplicate. Then, the outer loop will enter at index 0 for the row range and 0 for the index of the column value. Then the validation email goes thourgh an d

**Discuss the time complexity of the verification algorithm?**