Question1

Algorithm 1 A Greedy algorithm to find maximum value on given grid

SOLUTION NAIVE(grid: Input)

$nDown \leftarrow 0$
$nRight \leftarrow 0$
$totalScore \leftarrow 0$
$maxRowNorColumn \leftarrow len(grid) - 1$
$totalScore \leftarrow totalScore + grid[nDown][nRight]$
while $nDown \neq maxRowNorColumn$ **or** $nRight \neq$maxRowNorColumn **do**
  if $nRight = maxRowNorColumn$ **then**
    $nDown \leftarrow nDown + 1$
  else if $nDown = maxRowNorColumn$ **then**
    $nRight \leftarrow nRight + 1$
  else if $grid[nDown + 1][nRight] > grid[nDown][nRight + 1]$ **then**
    $nDown \leftarrow nDown + 1$
  else
    $nRight \leftarrow nRight + 1$
  end if
  $totalScore \leftarrow totalScore + grid[nDown][nRight]$
end while
return totalScore, nRight, nDown

Algorithm 1 shows pseudocode of a greedy algorithm to find maximum value on given grid. It basically looks local best solution on possible movements. In any time only one path kept in memory and near possible directions selected which maximizes score. Firstly we take grid from user as input and set score as 0 since we start from 0. We will use nDown and nRight variables to keep track our position. I will use maxRowNorColumn variable to check possible movements and end condition. Before while loop we sum up start index score with the totalScore variable. If we reach end of row index, code shift on right of grid, if we reach end of column index code shift on down of grid. First if condition and second else if condition does this. In third else if we check local optimal solution since we have 2 possible movements we select bigger score index. At the end of if condition we sum current grid index score with totalScore variable. At the end of while condition we return totalScore, nRight, nDown variables.

QUESTION2

Algorithm 2 An Dynamic Programming approach to to find maximum value on given grid

SOLUTION OPTIMAL(grid: Input)
  $gridLen \leftarrow len(grid)$
  $sum2dArray \leftarrow array[gridLen + 1][gridLen + 1]$
  $i, j, nDown, nRight, totalScore \leftarrow 0, 0, 0, 0, 0$
  while $i \leq gridLen$ **do**
    while $j \leq gridLen$ **do**
      $sum2dArray[i][j] \leftarrow 0$
      $j \leftarrow j + 1$
    end while
    $j \leftarrow 0$
    $i \leftarrow i + 1$
  end while
  $i, j \leftarrow 1, 1$
  while $i \leq gridLen$ **do**
    $nRight \leftarrow nRight + 1$
    $nDown \leftarrow nDown + 1$
    while $j \leq gridLen$ **do**
      $sum2dArray[i][j] \leftarrow max(sum2dArray[i-1][j], sum2dArray[i][j-1]) + grid[i-1][j-1])$
      $j \leftarrow j + 1$
    end while
    $j \leftarrow 0$
    $i \leftarrow i + 1$
  end while
  $totalScore \leftarrow sum2dArray[nDown][nRight]$
  $nRight \leftarrow nRight - 1$
  $nDown \leftarrow nDown - 1$
  return totalScore, nRight, nDown

Algorithm 2 shows pseudocode of a dynamic programming approach to find maximum value on given grid Instead of Algorithm 1, it find optimal solution. Since we need to access bottom right cell in grid with maximum score, so let i = n-1 and j = n -1 which n is size of grid so that we need to find optimal score on grid[i − 1][j] or grid[i][j − 1] so that we can start from start point and calculate all optimal scores on each cell on given grid. So that in Algorithm 2 firstly we create an n+1xn+1 2D array and assign each cell as 0. We use totalScore variable to return score on bottom right cell maximum score and nRight and nDown variables to keep track on movements in grid. In our second nested loop we iterate over all cells to find maximum value on each cell will optimal path, to do that we shift each column and row with sum2dArray grid so we can use each cells maximum score. It simply start from [0][0] index and calculate each cell maximum score and look sum2dArray because we need to keep near indexes maximum score. At the end of while condition we assign totalScore as bottom right index of sum2dArray and subtract nRight and nDown variables w,th -1 since grid length was n+1 and return totalScore, nRight, nDown variables.

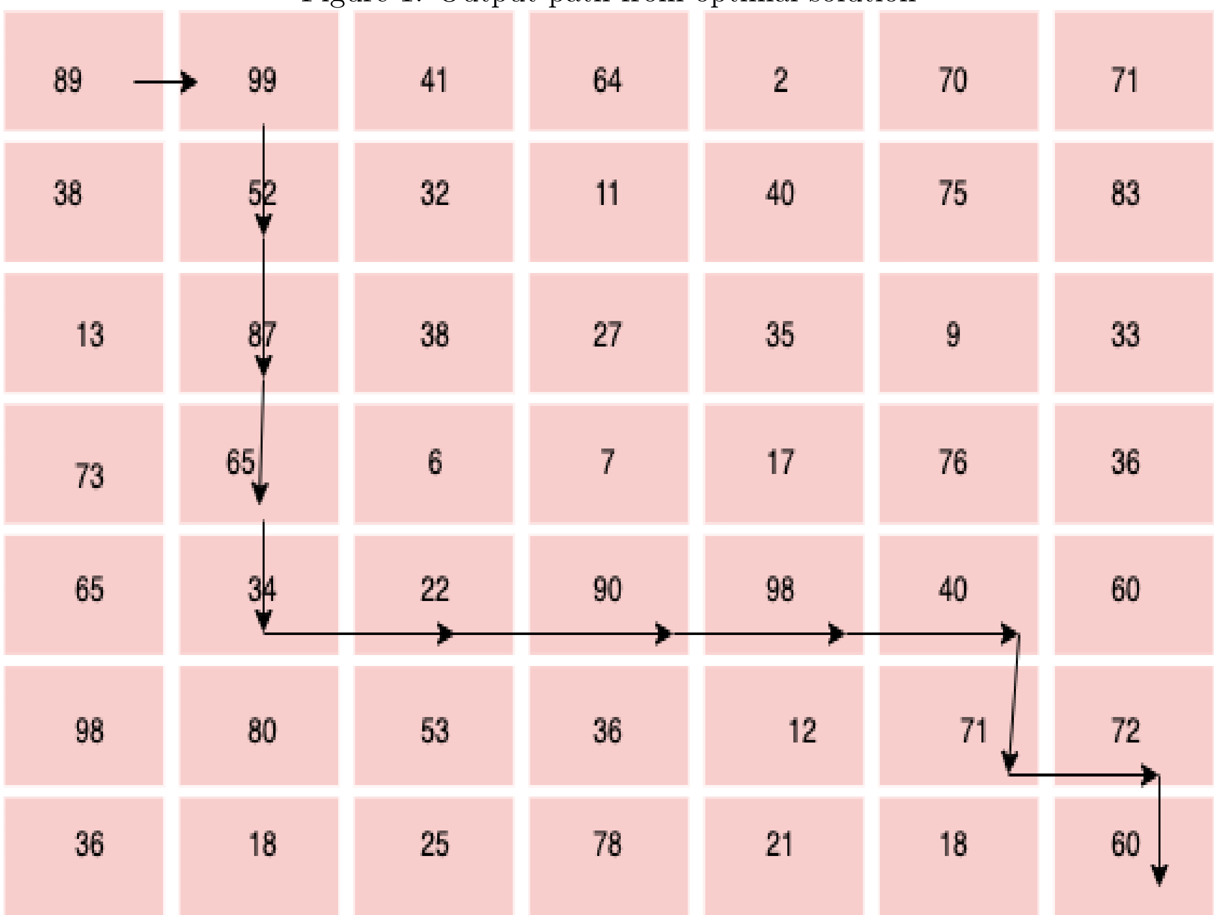
Question3

Algorithm 1 proposes a greedy algorithm which uses a for loop and look local optimal solution. There (n-1) movements for right and (n-1) movements for down which n is size of grid. So in for loop we sum these two movement types (n+n+(-1)+(-1)) = (2n-2) since we analyze worst case, we can remove constants then time complexity will be O(n). Since we don't use any data structure we just use local variables space complexity is O(1). This problem mainly contains overlapping subproblems. Algorithm 2 proposes a dynamic programming approach for these reason. Since we use two for loops nested and each loop iterate on n times which n is length of grid, time complexity will be O(n^2). We can memoize subproblems in a 2D matrix to avoid re-compute of same subproblems. Since we use n+1 x n+1 matrix to store maximum values, which n is size of given grid. In worst-case space complexity will be O(n^2). In terms of time complexity Algorithm 1. find solution on linear time but Algorithm 2 find solution on quadratic time so that in asymptotic analysis Algorithm 2 more slower than Algorithm 1 and also use extra space to keep all possible paths with scores. To find optimal solution algorithm 2 can be choosen but if we need faster solution we can use Algorithm 1 which find local optimal.

Question4

a) Optimal path on grid is like this 89 -> 99 -> 52 -> 87 -> 65 -> 34 -> 22 -> 90 -> 98 -> 40 -> 71 -> 72 -> 60

Figure 1: Output path from optimal solution



b) Total Score : 879, nRight : 6, nDown : 6