Winning the coin taking game

For this task I have chosen a dynamic approach where the sub-problem is the optimum total for each combination of piles.
We first build the Memo array for memoization purposes.

This is done with time complexity O(NM)
The Space complexity of O(NM)
N is the number of elements in pile1
M is the number of elements in pile2

We then proceed to filling the first row and first column of the memo array which contains the optimum values for each combination of pile, given both players pick from pile1 and pile2 respectively. This is the base case.

The subproblem is the optimum value for each combination of values of piles.
Therefore, the elements of the memoization (memo array) will contain the optimum values for each combination of piles (and coin values in piles).
Therefore, any given row and column in the memo will contain the optimal value to be obtained by player (max value) for that combination of coins in the 2 piles.

The optimal substructure is the dynamic programming solution taken where, to find the optimum value at a specific position in the memo, considering the coin values for both piles at that position O(NM), we first find the sum of coins both piles at that relevant position in the memo. This sum is found to determine the maximum coin value (of both piles) at that stage, from this sum we have to subtract the minimum adjacent optimum value.
This is done because, in keeping with the fact that both players are thinking optimally**,** the only way for one player to supersede the other is to obtain the difference between the sum of the 2 piles (at that point) and subtraction of the minimum adjacent optimum value.
In this manner we find the optimum values for all combinations of piles (considering coin values of piles).

This is done in O(NM) time complexity where N is the number of elements in pile 1 and M is the number of elements in pile 2.
The Space complexity is O(NM)

In order to find the optimal choices made by both players we initiate a pointer at the greatest optimum value (found at the last element of Memo) considering the given combination of pile and coin values.
We then retrace the decisions made by the player by looking at the adjacent minimum optimum values and moving the pointer. Based on the direction of the adjacent minimum optimum value we create the list of choices made by both players. (More details in comments of code)
The backtracking is done in time complexity O(N+M) where N is the number of elements in pile 1 and M is the number of elements in pile 2.

Finding Snakey words

For this this task I have chosen a dynamic programming approach where the sub-problem is the combination of positions (of the grid) found in sequence for each substring of the word. This is a path to each substring of the given word.

The memorization array is a list containing K number of lists, where K is the length of the word. K also represents the number of substrings for that word, because a word of length K contains K substrings. This is done in O(K) time complexity.

The Base Case:
Consider the base substring which is the first character of the word.
The Algorithm fulfils the base case by iterating over all characters in the grid and finding the coordinates of characters that equivalent to the first character of word. These coordinates are added as a list of tuples as the first element of the Memo.
This is done in O(N^2) time complexity.

The memoization will contain a list of lists, where each list contains the combination of paths for each substring of the word.
A Path is a combination of tuples which represents the positions of the grid found in sequence for each substring of the word.
This path of each substring, being stored in each element of memo, is the subproblem.

The optimal substructure:
The optimal substructure is the dynamic programming solution taken where for each list of the memo list [O(K) where K is the length of Memo] , we iterate over the contents of that list [O(N^2) because we iterate over a list of lists] which contain combinations of sequence of tuples, (representing the positions of the grid found in sequence) for the substring of the word. For each combination of tuples we access the last tuple (position of last character in that substring) and attempt to find the position of next adjacent element in the next substring. This new path is then added into the relevant position in the array.
In this manner we find the path to each substring of the word.

This runs in O(K x N^2) time complexity.
K is the number of elements in the Memo
N^2 is due to iteration of list of lists in each element of Memo.

Space Complexity: O(KN^2)
K is the number of lists in the Memo
N^2 is the lists in each list