Computational complexity: Discuss the computational complexity of the algorithm implemented in Part D (i.e., DTW with boundary constraints). Is it different from the algorithm in Part A (i.e., standard DTW)? State and explain the recurrence relation.

In part A, if n is the size of time sequence A and m is the size of time sequence B, the time complexity of initialising the cost matrix is (n+1)*(m+1) = Θ (n*m), and the time complexity of computing cost matrix in an unconstrained DTW is Θ (n*m) since we visit every cell in the matrix except the first row and column.

Whereas in part D, with the constraint of size c implemented, the time complexity of initialising the cost matrix is still Θ (n*m) as the size of the cost matrix does not change, but the time complexity of computing cost matrix in a constrained DTW will change to n*(2c + 1) = Θ(n*c) since for each row in the matrix except the first row, there are maximum of 2c+1 columns will be filled. Therefore, the constrained DTW algorithm in part d would be faster than the DTW algorithm in part a if value of c is less than m.

The space complexity for both algorithm in part A and D remains the same: (n+1)*(m+1) = Θ (n*m)

In more detail, The recurrence relation formula used in constrained DTW algorithm does not change, but the range of j changed from [1,m] to [max(1, i - c), min(m, i + c)], where range of i = [1, n]

costMatrix[i][j] = costOf(A[i-1], B[j-1]) + min(costMatrix[i -1][j], costMatrix[i, j-1], costMatrix[i - 1][j - 1])

This recurrence relation calculates the minimum cumulative cost for the cell at ith row and jth column of the cost matrix, and the minimum cumulative cost is calculated by first finding the difference between (i-1)th element in sequence A with (j-1)th element in sequence B, and then add it with the minimum cumulative costs of all possible pathways that can reach to the cell (i, j), but since we restricted the range of j, this recurrence relation will only apply to cell at row 1 to n and column max(1, i - c) to min(m, i + c) leading to the decrease in time complexity from Θ(n*m) in part 1a to Θ(n*c) in part 1d, given that c is less than m.

This means that part1d algorithm may be superior than the part1a algorithm as it takes less time complexity, however, in restricting the path option in part 1d by adding a window size of c, we may accidentally rejected the optimal wrapping path as well. Therefore, we need to be careful when choosing the window size so that we can decrease time complexity, and at the same time, finding the optimal path.