Part B

Dynamic Time Warping (DTW) is an algorithm which finds the similarity between two sequences of data points arranged in time order. DTW accounts for differences in spacing between neighbouring elements and will perform 'stretching' and 'compressing' on the gaps between elements to ensure that similar features are aligned. Simpler sequence matching methods don't account for this and rather just find the direct distance between elements occurring at the same time. This means that variations in speed or length of the sequence are not accounted for. For instance, if we had two versions of the same song (same melody), but one was a faster, more upbeat cover, then DTW would be able to detect this similarity, whereas simple sequence matching would not.

In DTW, a matrix is used to represent the minimum cumulative distance. Each entry (i, j) represents the minimum cumulative distance if we aligned element *i* of sequence A with element *j* of sequence B. We first initialise this matrix with all entries equalling infinity. This prevents paths whos' distances we have not calculated yet from being used in our calculations. Entry (0,0) is initialised to 0 so that the first alignment occurs between the first elements of the two sequences. We then iterate through each element pairings, find the minimum cumulative distance out of the neighbouring pairs and add on the new, additional distance. This means that we are always finding the minimum distance, as we are taking the path with the smallest distance out of all the possible ones we have already calculated.

Part C

If we define $c_{i,j}$ to be the minimum cumulative distance found by aligning element i of sequence A with element j of sequence B, and $d_{i,j}$ to be the distance between elements i and j, the recurrence relation of DTW is;

$$c_{i,j} = d_{i,j} + \min(c_{i-1,j}, c_{i,j-1}, c_{i-1,j-1})$$

With the base cases $c_{0,0} = 0$, $c_{0,1} = \infty$ and $c_{1,0} = \infty$. This essentially means that we are taking the shortest option out of matching sequence A and B, skipping an element in sequence A, or skipping an element in sequence B and adding the extra distance.

This implementation of DTW is an instance of dynamic programming as we are maintaining a matrix of all minimum cumulative distances which we have already calculated. This means that for each sub-problem we do not have to unnecessarily repeat all of our past calculations, but instead we can just add one additional calculation to a value from our matrix.

Part E

If we define our basic operation as calculating the distance between two points, then our computational complexities $T_A(n, m)$ and $T_D(n, m, w)$ can be defined as;

$$T_A(n) \in \Theta(nm) \qquad \text{and} \qquad T_D \in \begin{cases} \Theta(wn), & n \geq m \\ \Theta(wm), & m < n \end{cases}$$

Where n is the length of sequence A, m is the length of sequence B and w is the window length. This means that when there is a large difference between the window length and the size of the sequences, the implementation is part D is much more efficient.

The recurrence relation for part D is the same as part A;

$$c_{i,j} = d_{i,j} + \min(c_{i-1,j}, c_{i,j-1}, c_{i-1,j-1})$$

Except with the added restriction that we only consider values of i and j such that $|i - j| \leq w$. This significantly reduces the number of calculations which must be performed, leading to the improved efficiency.

Part G

If we define our basic operation as calculating the distance between two points, then our computational complexities $T_A(n, m)$ and $T_F(n, m, l)$ can be defined as;

$$T_A(n) \in \Theta(nm) \qquad \text{and} \qquad T_F \in \Theta(lnm)$$

With n and m defined as in Part E, and l defined to be the maximum path length. This means that the implementation in Part F has a worse computational complexity, as it requires that we also keep track of the path length.

If we define $c_{i,j,k}$ to be the minimum cumulative distance found by aligning element i of sequence A with element j of sequence B along a path of length k, and $d_{i,j}$ to be the distance between elements i and j, the recurrence relation is;

$$c_{i,j,k} = d_{i,j} + \min(c_{i-1,j,k-1}, c_{i,j-1,k-1}, c_{i-1,j-1,k-1})$$

With the base cases $c_{0,0,0} = 0$, $c_{0,1,0} = \infty$ and $c_{1,0,0} = \infty$. This means that we are taking the shortest option out of matching sequence A and B, skipping an element in sequence A, or skipping an element in sequence B, all for a path one shorter and adding the extra distance. However, we only need to consider alignments which are possible given the path length, meaning that $i < k$, $j < k$ and $i + j \geq k - 1$.