Problem 1
Do problems 6.30 and 6.40 in the book.

Solution:

Problem 6.30: First, we can find the alignment that corresponds to the smallest edit distance between v₁ and v₂. This can be done by setting the score of a match to 0, and the score a mismatch or a gap to -1. Then maximizing the score is equivalent to finding the edit distance (minimize mismatches and gaps).

Then, we can cut the alignment we obtain at the point where the score is about 1/2 the total score. This cut defines a prefix of v₁ and a suffix of v₂. We make w the concatenation of the prefix and the suffix. It is clear that w will a balanced edit distance from v₁ and v₂. In addition, d(v₁, w) + d(v₂, w) = d(v₁, v₂).

Problem 6.40: There are many ways we can solve this problem. One possibility is to adapt the algorithm for a general gap penalty function. This runs if O(n³) and gives the correct answer provided that γ(x₁ + x + 2) ≤ γ(x₁) + γ(x₂), which is satisfied in our case. For instance, a gap of length 2 spanning the same character costs only 1, whereas two gaps of length 1 will cost 2. Using techniques similar to the one we saw in class, we can further improve the running time to O(n²) by using multiple matrices and distinguishing whether alignments end in a gap or not.

Here’s another approach. We can compress each string by removing repeated characters. For example, acctttgaa will be compressed as actga. Once we have done that for both strings, we can align them in the best way we can; for instance, we can minimize the number of gaps (mismatches not allowed), i.e. matches cost 0, gaps cost 1, and mismatches cost ∞. Finally, we “insert” back the characters we removed during compression. If a sequence of characters has already been associated with a gap, nothing is to be done. Otherwise, we add a cost of 1.

Problem 2
Do problem 7.7 and 7.10 in the book. For problem 7.7, if we break the mn size problem into two ij and (m − i)(n − j) size problems, we need ij ≈ (m − i)(n − j).
Find a curve describing the relation between i and j, the path of the optimal alignment must cross that curve.
Solution:

Problem 7.7: Assume without loss of generality that $m < n$. Solving for $ij = (m - i)(n - j)$ we have $i = m - (m/n)j$. Since $i$ and $j$ are integers, this means if $j$ increases by 1, $i$ may or may not increase. This gives us a “continuous” staircase from $(i, j) = (m, 0)$ to $(i, j) = (0, n)$. We can also “fill” this staircase with additional entries so that every path from $(i, j) = (0, 0)$ to $(i, j) = (m, n)$ will have to intersect with one entry. Since all these entries approximately define balanced sub-problems, and there are $O(n)$ of them, we can solve the problem using the divide and conquer technique described in class by checking the best entry that partitions the problem, and each time we get a balanced partition.

Problem 7.10: This is a generalization of the 2D case. Instead of saying $x_{mid}$ will map to one of $O(n)$ positions, we now have $x_{mid}$ map to one of $O(n^2)$ positions, $O(n)$ in each of the two other strings.

Problem 3
For the spliced alignment problem, improve the running time of the algorithm by using the idea of sorting the end points of intervals, and scanning those points to detect when an interval is entered and when an interval is exited.

I will not write the solution of this problem, but the idea is similar to the one described for the 1-dimensional segments problem.