Assignment 8 Solutions

Consider the Longest Increasing Subsequence Problem from class. In this problem, we are given a list of numbers $x_1, ..., x_n$ and want to find the longest increasing subsequence (not necessarily consecutive) within this list. In class, we first considered the tree of possible solutions:

![Tree Diagram]

We argued that the optimal solution would be somewhere at the bottom level of this tree, but that there would be $2^n$ nodes at the bottom and thus it would be infeasible to check all of them. So we came up with a few pruning rules that would reduce the number of solutions we’d have to feasibly check:

1. Convince me that each of the pruning rules are valid - that is, even after pruning off the described subtrees at every level, we will still have the optimal solution somewhere on the bottom level of the remaining subtrees.

**Solution:** We can definitely prune any sequence that isn’t increasing - adding more elements to the end of it isn’t going to make it increasing, so it won’t lead to an optimal solution.

If two sequences at the same depth have the same length, we can prune the one that ends in a larger number: let $s_1$ and $s_2$ denote these two sequences, where $s_2$ is the one ending in a larger number. Suppose there is an optimal solution in the subtree of $s_2$, that is a sequence $s_3$ such that the combination $s_2 \cdot s_3$ is an optimal solution.
Then \( s_1 \cdot s_3 \) is the same length, but is also increasing: \( s_1 \) is increasing by virtue of it not being pruned by the first rule, \( s_3 \) is increasing by virtue of it being a part of an optimal solution, and the last number of \( s_1 \) is smaller than the last number of \( s_2 \) which is smaller than the first number of \( s_3 \). Thus we can prune \( s_2 \) without deleting all optimal solutions from the tree.

Lastly, if two sequences on the same depth end in the same number, we can prune the one with less length: let \( s_1 \) and \( s_2 \) be these sequences, where \( s_2 \) is of less length. Consider any feasible solution in the subtree of \( s_2 \), namely a sequence of the form \( s_2 \cdot s_3 \). Then since \( s_1 \) ends on the same value as \( s_2 \), if \( s_2 \cdot s_3 \) is a feasible solution then so is \( s_1 \cdot s_3 \). Moreover, the length does not decrease by performing this exchange, so the solution is just as good. Thus we are not removing all optimal solutions by pruning \( s_2 \) from the recursion tree.

2. In class we tried to use i) and iii) to develop an algorithm. Technically the one we came up with in class ended up being \( O(n^3) \) to compute. Let \( \text{Opt}(j,k) \) = the longest increasing subsequence of the first \( j \) numbers that ends on \( x_k \). Show that we can compute an array \( A \) of size \( n \times n \) such that \( A[j,k] = \text{Opt}(j,k) \) in \( O(n^2) \) time. 

**Solution:** Consider the following code:

Initialize \( A \) as an \( n \times k \) array with every entry set to 0
for \( j = 0\ldots n-1 \):
    for \( k = 1\ldots j \):
        \[ A[j+1, k] = \max(A[j+1, k], A[j,k]) \]
        if \( x_{j+1} > x_k \):
            \[ A[j+1, j+1] = \max(A[j+1, j+1], 1+A[j,k]) \]

This code is explicitly tracing through what the tree creates - on line 5, it prunes according to the first rule, and on line 6 it prunes according to the 3rd rule. Note that we are doing a constant amount of work for each pair \((j,k)\). Thus the overall runtime of this algorithm is \( O(n^2) \).

3. Let’s try to use i) and ii) to develop an algorithm. Let \( \text{Opt}(j,k) \) = the minimal last number of length \( k \) subsequences from the first \( j \) numbers. Show that we can compute an array \( A \) of size \( n \times n \) such that \( A[j,k] = \text{Opt}(j,k) \) in \( O(n^2) \) time.

**Solution:**

Initialize \( A \) as an \( n \times k \) array with every entry set to \( \infty \)
Set \( A[0, 0] = -\infty \)
for \( j = 0\ldots n-1 \):
    for \( k = 1\ldots j \):
        \[ A[j+1, k] = \min(A[j+1, k], A[j, k]) \]
        if \( x_{j+1} > A[j,k] \):
            \[ A[j+1, k+1] = \min(A[j+1, k+1], x_{-\{j+1\}}) \]

We set \( A[0, 0] = -\infty \) because on a 0 length sequence anything can be appended, ie the last index of 0 length string is so small that every \( x_i \) is bigger than it.