CPSC 320: Intermediate Algorithm Design 2019/20 Term 2

Tutorial 9

Prof. Nick Harvey *University of British Columbia*

Let A be an array of n distinct integers. In this problem, we are interested in finding the *longest increasing subsequence* of A. That is, we want to find indices $i_1, i_2, ..., i_t$ such that

$$\begin{array}{rl} \text{Indices:} & i_1 < i_2 < ... < i_t \\ \text{Values:} & A[i_1] < A[i_2] < ... < A[i_t], \end{array}$$

and t is as big as possible.

For instance, consider the array

$$A = (1, 9, 17, 5, 8, 6, 4, 7, 12, 3).$$

In this example, one increasing subsequence is:

$$\begin{array}{rl} \text{Indices:} & 1 < 2 < 3 \\ \text{Values:} & A[1] = 1 < A[2] = 9 < A[3] = 17. \end{array}$$

It has length 3, but there is no way to extend it to get a longer increasing subsequence.

Another increasing subsequence is:

$$\begin{array}{rl} \text{Indices:} & 1 < 4 < 6 < 8 < 9 \\ \text{Values:} & A[1] = 1 < A[4] = 5 < A[6] = 6 < A[8] = 7 < A[9] = 12 \end{array}$$

It has length 5.

We will design an algorithm that uses dynamic programming to find the longest increasing subsequence.

1. We must find a recurrence that relates the solution of bigger subproblems to the solution for smaller subproblems.

   We will define $L[j]$ to be the length of the longest increasing subsequence of $A[1..j]$ and which *definitely includes index* j as part of the increasing subsequence.

   Give a recurrence relation for $L[j]$. (It may depend on $L[1], ..., L[j-1]$.)

2. Write pseudo-code for a memoization algorithm that finds the **length** of the longest increasing subsequence of $A[1..n]$. What is the runtime of your algorithm?

3. Write pseudo-code for an bottom-up dynamic programming algorithm that finds the longest increasing subsequence of $A[1..n]$. What is the runtime of your algorithm?

4. **Keener Kwestion.** For this problem it is possible to design an algorithm with improved runtime $O(n \log n)$. The algorithm is very short, but clever, and not so easy to discover.

   Hint: Moving from right-to-left in the array, maintain an array B for which $B[i]$ is the "best" location at which to start an increasing subsequence of length i.