CSCI 270 Lecture 6: Longest Increasing Subsequence

Given a sequence of numbers $s_1, s_2, ..., s_n$, delete the fewest numbers possible so that what is left is in increasing order.

Example input: $3, 4, 1, 2, 8, 6, 7, 5, 9$

We'll try the same bite-size questions we have in the previous problems.
Q1: Do we keep s_1 or not?

- If I **do not** keep s_1, which number should I try next?
- If I **do** keep s_1, which number should I try next?
- What information must be passed down to the next level of recursion?
- Is this information **concise**?

We lost some critical information: the largest number we've included so far. This information needs to appear in the parameters, or else we will have to pass a ton of information down the recursion chain.

Attempt 2 bite-size questions:
Qi: If I include s_i, what number should I include next?

LIS[z] will store the length of the longest increasing subsequence which starts with s_z.

LIS[$n+1$] $= 0$

- I don't know which number to include after s_z, so I try all of them. Well, not really all of them, there are some constraints. What constraints do I place on which number I can include next?
- What does the recursive formula look like for LIS[z]?
- What order do I fill in the array?
- Where is the answer stored in the completed array?
- How do I reconstruct the actual sequence?
- What is the runtime of the algorithm?

Polynomial Runtimes

Primality(int X)
1: For $i = 2$ to $X - 1$
2: If $X\%i = 0$ Then Return False
3: Return True

- What is the runtime of the above algorithm?
- Is this a polynomial-running time algorithm?
- When I ask if an algorithm is polynomial, it must be in relation to something. In relation to what, specifically?