Longest Increasing Subsequence

Input: Sequence S of n numbers

Output: Length of the longest increasing subsequence of S.

If $1 \leq i_1 < i_2 < \ldots < i_k \leq n$, then $S[i_1], S[i_2], \ldots, S[i_k]$ is a subsequence. This subsequence is increasing if $S[i_1] < S[i_2] < \ldots < S[i_k]$.

Example: $S = [5, 2, 8, 6, 3, 6, 9, 7]$

- [5, 2, 3, 6] is a subsequence, but is not increasing, since $5 > 2$.
- [2, 8, 9] is an increasing subsequence.
- [2, 3, 6, 7] is a longest increasing subsequence.

There are 2^n subsequences, so trying all of them would take too much time.

Divide and conquer? Maybe... but it is not clear how to merge

- What does the optimal solution look like?

$S = [5, 2, 8, 6, 3, 6, 9, 7]$

$L = [\underbrace{2, \quad 3, 6,}_{\text{longest increasing subsequence ending with 6.}} \quad 7]$

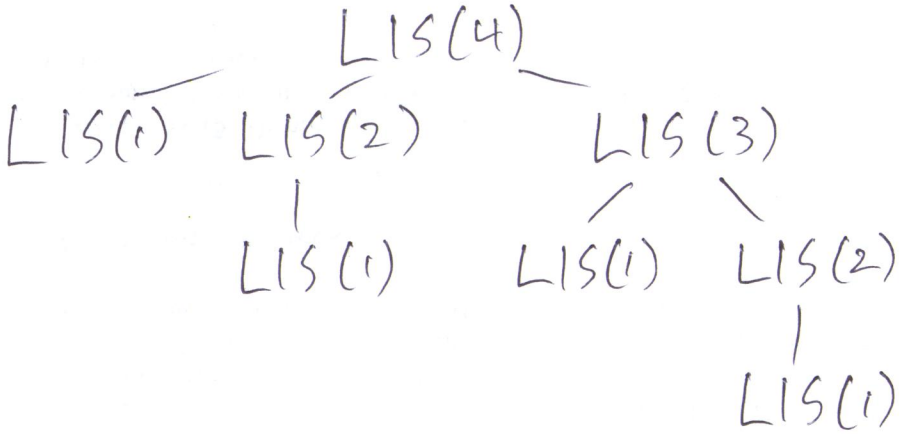
In general: $L = [S[i_1], S[i_2], \ldots, S[i_{k-1}], S[i_k]]$ with $S[i_{k-1}] < S[i_k]$ and $[S[i_1], S[i_2], \ldots, S[i_{k-1}]]$ is a longest increasing sequence ending with $S[i_{k-1}]$.

- This gives a recurrence for the LIS ending at position j:

$$LIS(j) = \begin{cases} \max_i \{1 + L(i) : S[i] < S[j] \text{ and } i < j\}, \\ 1, \text{ if there are no such } i \end{cases}$$

- The actual longest increasing subsequence of S has to end somewhere, so it is the maximum over all $LIS(j)$.
- We can ~~compu~~turn this recurrence into a recursive algorithm; but it isn't very efficient!

Suppose $S = [1, 2, 3, 4]$. Then the recursion tree for LIS(4) is:



There are many repeated subproblems.

- Instead, compute LIS(1), then LIS(2), etc: If we store these answers in an array, we can simply look them up when computing the next entry:

Algorithm LIS(S)
  Initialize array L[1..n]
  maxL ← 0
  for j ← 1 to n do
    L[j] ← 1
    for i ← 1 to j-1 do
      if S[i] < S[j] then
        L[j] ← max(L[j], 1 + L[i])
    maxL ← max(maxL, L[j])
  return maxL

- Correct? Y
- Terminates? Y.
- Efficient? $O(n^2)$

Can we do better?

Yes - $O(n \log n)$ by maintaining M[i] = smallest element ending an inc. s.s. of length i.

Example: $S = [4, 2, 9, 4, 6, 3, 8]$

$L = [1, 1, 2, 2, 3, 2, 4]$

$\Rightarrow$ maxL = 4, sequence 2, 4, 6, 8.

Dynamic Programming

① Study the structure of the optimal solution.

Show that there is optimal substructure: the optimal solution contains the optimal solution for smaller subproblems.

② Set up a recurrence for the opt. solution, using the optimal substructure from ①.

③ Solve the recurrence bottom-up.

First solve the smallest subproblems (usually base cases of the recurrence). Then solve the second smallest, etc.

Subproblems in	D&C	DP
	- few	- many
	- disjoint	- overlapping
	- much smaller	- only slightly smaller