

Dynamic Programming: Longest Increasing Subsequence¹

1 Longest Increasing Subsequence

Given a string $s[1 : n]$, a *subsequence* is a subset of the entries of the string in the same order. Formally, a length k subsequence is a string $\sigma = (s[i_1] \circ s[i_2] \circ \dots \circ s[i_k])$ where $1 \leq i_1 < i_2 < \dots < i_k \leq n$. For example, if the string is `algorithms`, of length 10, then `lot` is a subsequence with $i_1 = 2, i_2 = 4$, and $i_3 = 7$. Similarly, `grim` is a subsequence. But, `list` is not a subsequence.

Remark: Note that the i_1, i_2, \dots, i_k need not be contiguous; if they are indeed contiguous, then the subsequence is called a **substring**. The number of substrings is $O(n^2)$, the number of subsequences is 2^n (do you see this?).

In the Longest Increasing Subsequence, or simply LIS, problem the input is an array $A[1 : n]$ of real numbers. A subsequence $(A[i_1], A[i_2], \dots, A[i_k])$ is increasing (actually non-decreasing would be more apt) if $A[i_1] \leq A[i_2] \leq \dots \leq A[i_k]$. As before, we have $i_1 < i_2 < \dots < i_k$. Our goal is to find a *longest* increasing subsequence.

For example, suppose the array $A = [13, 27, 15, 8, 19, 32, 20]$. Then the subsequence $(13, 32)$ is an increasing subsequence, so is $(15, 20)$. The longest one seems to be $(13, 15, 19, 20)$. There could be many such LISs of length 4. For example, $(13, 15, 19, 32)$ is another one.

A First try that *doesn't* work. As we have done before, let us imagine a longest increasing subsequence of A . Let it be $\sigma = (A[i_1], A[i_2], \dots, A[i_k])$. We try to break this solution into solutionettes of smaller LIS problems. As hopefully by now is usual, we ask of $i_k = n$, that if the last element of the array participates in the solution. If not, then this σ should be an LIS of $A[1 : n - 1]$, a smaller instance. And if, $i_k = n$, then $\sigma' := (A[i_1], \dots, A[i_{k-1}])$ should be an LIS of $(A[1 : i_{k-1}])$. We don't know what i_{k-1} is, but we **do** know $A[i_{k-1}] \leq A[n]$ since σ was an LIS. So, perhaps we look at **all** $j < n$ such that $A[j] \leq A[n]$, and assert $|\sigma'|$ is at least as long as LIS of $A[1 : j]$ for all such j .

If you have agreed with the above intuition, you may come up with the definition $L(i) :=$ length of an LIS of $A[1 : i]$ and we are interested in $L(n)$, the base case of $L(0) = 0$, and the recurrence

$$L(i) = \max \left(L(i-1), 1 + \max_{j < i: A[j] < A[i]} L(j) \right) \quad (\text{A Wrong Recurrence!})$$

And as the name suggests, this recurrence is **wrong!** To show this, here is a simple example. Take the array $A = [20, 40, 10, 30]$. First, convince yourself by inspection that $L(4) = 2$, that is, the longest increasing subsequence of this array is of length 2. Now inspect what the recurrence gives. Working bottom up, it sets $L(0) = 0$, then sets $L(1) = 1$. It then sets $L(2) = 1 + L(1)$ since $A[1] < A[2]$, and so $L[2] = 2$. Continuing, it sets $L[3] = \max(L[2], 1) = L[2] = 2$. Indeed, the LIS of $[20, 40, 10]$ is of length 2. And then it makes the mistake of setting $L[4] = 1 + L[3]$ (since $A[3] < A[4]$), which evaluates to 3. Oh no!

¹Lecture notes by Deeparnab Chakrabarty. Last modified : 19th Mar, 2022
These have not gone through scrutiny and may contain errors. If you find any, or have any other comments, please email me at deeparnab@dartmouth.edu. Highly appreciated!

Remark: Many (including me) have made this mistake above. And indeed, if we argue only in English, it is possible we can fool ourselves. Here is an exercise I want **all** of you to do. Imagine, you didn't know that the (A Wrong Recurrence!) was indeed wrong, and you write the proof as you have been doing. You should see that one inequality (the \leq one), will actually go through. You will catch the bug when you try to argue the \geq . This will make you appreciate why one needs to argue both sides.

Now that we know we made an error, what did we miss? We missed that **an arbitrary** LIS of $A[1 : n-1]$ **cannot** be extended by adding $A[n]$ at the end; it only can be extended if the LIS of $A[1 : n-1]$ ends with a number at most $A[n]$. In the above example, when we were writing $L(4) = 1 + L(3)$ because $A[3] \leq A[4]$ we were inadvertently assuming that the LIS of $A[1 : 3]$ ends at $A[3]$ and thus can be extended. Unfortunately, that assumption is wrong.

But when you do realize what is the error, you can also get out of it. The idea is to make the assumption we made *explicit* by asking for it. So, we will massage our definition such that $L(i)$ will not give us the LIS in the array $A[1 : i]$, but rather give us the length of the longest increasing subsequence of $A[1 : i]$ which ends at $A[i]$. We make this explicit.

- a. Why is this interesting? Because, if we knew this for all $A[1 : m]$, for $1 \leq m \leq n$, then we could scan all these n answers and choose the best one.
- b. Why is this useful? Because now we can assert that if $A[j] < A[n]$, then the LIS of $A[1 : j]$ which ends at $A[j]$ can indeed be extended to an LIS of $A[1 : n]$.

We are now armed to write the full DP solution. With this new twist of LIS, we are ready for our dynamic programming solution, which we provide below.

- a. **Definition:** For any $1 \leq m \leq n$, define $\text{LIS}(m)$ to be the length of the longest increasing subsequence in $A[1 : m]$ which ends with $A[m]$. We are interested in $\max_{1 \leq m \leq n} \text{LIS}(m)$.
- b. **Base Cases:** $\text{LIS}(0) = 0$.
- c. **Recursive Formulation:** For all $m \geq 1$:

$$\text{LIS}(m) = 1 + \max_{1 \leq j < m: A[j] \leq A[m]} \text{LIS}(j)$$

- d. **Formal Proof:** To formally prove the above, it helps to introduce, as we have done for optimization problems, the notation of $\text{Cand}(m)$ to be the set of all *increasing subsequences* of $A[1 : m]$ which end with $A[m]$. Therefore,

$$\text{LIS}(m) = \max_{\sigma \in \text{Cand}(m)} |\sigma|$$

- (\geq): Fix any $1 \leq j < m$ with $A[j] \leq A[m]$, and let σ' be the LIS of $A[1 : j]$ ending with $A[j]$ of length $\text{LIS}(j)$. Well $\sigma = \sigma' \circ A[m]$ is in $\text{Cand}(m)$ and is of length $1 + \text{LIS}(j)$. This means, $\text{LIS}(m) \geq 1 + \text{LIS}(j)$.
- (\leq): Fix the sequence $\sigma \in \text{Cand}(m)$ of length $\text{LIS}(m)$. If $|\sigma| = 1$, then the inequality is vacuous. Otherwise, let $A[j]$ be the second-last entry of σ . Note that $\sigma' = \sigma - A[m]$ is in $\text{Cand}(j)$ and has length $\text{LIS}(m) - 1$.

- e. **Pseudocode for computing $\text{LIS}(m)$ and recovery pseudocode:**

```

1: procedure LIS( $A[1 : n]$ ):
2:   ▷ Returns the Longest Increasing Subsequence
3:   Allocate space  $L[0 : n]$  ▷  $L[m]$  will contain LIS( $m$ ).
4:   Maintain  $\text{parent}[1 : n]$  which is useful for recovery. Initialized to 0
5:    $L[0] = 0$ . ▷ Base Case.
6:   for  $1 \leq m \leq n$  do:
7:      $L[m] = 1 + \max_{1 \leq j < m: A[j] \leq A[m]} L[j]$  ▷ Naively, takes  $O(n)$  time.
8:     Set  $\text{parent}[m]$  to be the  $j$  which maximizes. If no such  $j$ ,  $\text{parent}$  remains 0.
9:      $m = \arg \max L[1 : n]$  ▷  $L[m]$  contains the length of the optimal LIS
10:    Define  $\sigma$  to be the reverse of  $(A[m], A[\text{parent}[m]], A[\text{parent}[\text{parent}[m]]], \dots)$  till  $\text{parent}$  becomes 0.
11:    return  $\sigma$ .

```

f. **Running time and space** As written above, the space is $O(n)$, and the running time is $O(n^2)$ since [Line 7](#) takes $O(n)$ time when implemented naively. In fact this can be improved to $O(n \log n)$ time as we show next.

1.1 A Faster Implementation

The time taking step is [Line 7](#) which sets

$$L[m] \leftarrow 1 + \max_{1 \leq j < m: A[j] \leq A[m]} L[j] \quad (\text{Line 7})$$

As noted above, a naive implementation would take $O(m)$ time as we would scan $A[1 : m - 1]$ finding the j 's with $A[j] \leq A[m]$ and then finding the one with maximum $L[j]$. If you feel in your bones that there is something inefficient here, then you are getting a “feel” for good algorithms! Indeed, we can do way better. Let me describe this idea leisurely, and then describe how you would change the pseudocode.

Suppose when you are trying to figure out $L[m]$, we know that the longest LIS seen so far, that is, $\max_{j < m} L[j]$, is some positive integer t . Suppose you *also* knew for every $1 \leq s \leq t$, the locations $A[j]$ where $L[j] = s$. In English, for every s you knew the locations j such that the longest increasing subsequence of $A[1 : j]$ ending in $A[j]$ is of length exactly s . Indeed, let's give this set a name

$$\text{For all } 1 \leq s \leq t, \quad J_s := \{j : L[j] = s\}$$

Using this we can re-cast our value $L[m]$ slightly differently. We can say that $L[m] = 1 + s^*$ where s^* is the *largest* s such that there is some $j \in J_s$ with $A[j] \leq A[m]$. In particular, s^* is the largest s such that $\min_{j \in J_s} A[j] \leq A[m]$. Make sure you understand this. Thus, we can rewrite [Line 7](#) as

$$L[m] \leftarrow 1 + \max_{1 \leq s \leq t: \min_{j \in J_s} A[j] \leq A[m]} s$$

It doesn't seem we made any progress. Instead of searching over all $A[1 : m]$ we are not searching from $1 \leq s \leq t$, but then for each s , we are trying to find the smallest $A[j]$ in J_s . What did we achieve?

The key is to *maintain*, as we go along, the quantity $\min_{j \in J_s} A[j]$ for all $1 \leq s \leq t$. More precisely, we initialize a list D that, at every point of time, will have t entries $D[1 : t]$ defined. Recall t is the maximum LIS length seen so far. And, $D[s]$ will contain the quantity $\min_{j \in J_s} A[j]$. Again, in English, the smallest

array element $A[j]$ such that the LIS of $A[1 : j]$ ending in $A[j]$ is of length exactly s . In parallel, we also maintain $E[s]$ which just contains the index j such that $A[E[s]] = D[s]$ (this can be stored in $D[s]$ itself). So, we can re-case the setting of $L[m]$ yet again as

$$L[m] \leftarrow 1 + \max_{1 \leq s \leq t : D[s] \leq A[m]} s \quad (1)$$

Let us look at an example. Suppose $A = [13, 27, 15, 8, 19, 32, 20]$ with $n = 7$. Suppose we have figured out $L[1 : 6] = [1, 2, 2, 1, 3, 4]$ and we want to figure out what $L[7]$ should be. The value of t , at this point, is 4. And the lists $D[1 : 4]$ and $E[1 : 4]$ are $D = [8, 15, 19, 32]$ and $E = [4, 3, 5, 6]$ indicating the positions. Indeed, $D[2] = 15$ means among all indices j with $L[j] = 2$, which are $\{2, 3\}$, we are storing the smallest value, $D[2] = \min(A[2], A[3]) = 15$.

Remark: Do you see $E[s]$, the position where $D[s]$ is attained, is the largest index in J_s ? This is not an accident. Can you argue this?

The main reason why we make progress is the following key claim. You may have already noticed it in the above example; the claim shows this is no accident.

Claim 1. The list D is sorted increasing (non-decreasing).

Proof. To prove this, let us take s and t such that $s < t$. Let us consider the positions $j := D[s]$ and $k := D[t]$. That is, $j = E[s]$ and $k = E[t]$. Recall, by the definition, $\text{LIS}(j) = s$ and $\text{LIS}(k) = t$. We need to prove $A[j] \leq A[k]$.

Since $\text{LIS}(k) = t$, there is a length t subsequence σ ending at $A[k]$. Look at the s th entry in this sequence: suppose it is k' . By definition of LIS, $A[k'] \leq A[k]$. Now note that $\text{LIS}(k') = s$ as well; there definitely is a length s subsequence ending at $A[k']$ (the first s entries of σ), and if there was anything strictly longer, than we can append the $(t - s)$ entries of σ coming after $A[k']$ to get a longer than t LIS ending at $A[k]$. Which is a contradiction to $\text{LIS}(k) = t$. So, $\text{LIS}(k') = s$, that is, $k' \in J_s$. Which means $D[s] \leq A[k']$, and so $D[s] \leq A[k] = D[t]$. \square

Why is having D sorted non-decreasing allow us to run (1) faster than $O(m)$ or $O(t)$ time? The answer is “binary search”. We need to find $\max_{1 \leq s \leq t : D[s] \leq A[m]} s$. That is, in the *sorted* array $D[1 : t]$ we are trying to find the largest entry $\leq A[m]$. This is precisely what Binary Search does². To give more details, we start with $L \leftarrow 0$ and $U \leftarrow t + 1$, and we start with $s = \lceil \frac{L+U}{2} \rceil$. If $D[s] \leq A[m]$, then our $s^* \geq s$, and so we set $L \leftarrow s$. Otherwise, if $D[s] > A[m]$, then $s^* < s$, and so we set $U \leftarrow s$. Thus, the invariant $L \leq s^* < U$ is always maintained. We stop when $U - L \leq 1$, and we return the value of L . This takes $O(\log t) = O(\log n)$ time. Note that the answer could be 0, which would mean $A[m]$ is smaller than $D[1]$ as well. In the pseudocode below, this bit is just called $\text{BINSEARCH}(D, A[m])$.

Are we done explaining the whole algorithm? Well, no. We need to tell *how* this list D is maintained. We maintain the value t and D together. Initially $t = 0$ and $D = []$. After setting $L[m]$ for a certain m , we first check if the current $t < L[m]$. If so, then it must be $L[m] = t + 1$ (do you see why?) and so we set $t \leftarrow t + 1$ and $D[t] = A[m]$. Otherwise, $L[m] = s \leq t$. Then, we check if $A[m] < D[s]$. If so, we modify $D[s] \leftarrow A[m]$ thus maintaining $D[s] = \min_{j \in J_s} A[j]$. Otherwise, we do nothing. This completes the description of the algorithm.


²Ok, so binary search only answers YES or NO. But in the NO case, we can always make it return the largest entry of the array \leq the element you are searching for

```

1: procedure LISFASTER( $A[1 : n]$ ):
2:   ▷ Returns the Longest Increasing Subsequence
3:   Allocate space  $L[0 : n]$  ▷  $L[m]$  will contain LIS( $m$ ).
4:   Maintain  $\text{parent}[1 : n]$  which is useful for recovery. Initialized to 0.
5:   Maintain  $t \leftarrow 0$  and  $D, E \leftarrow []$ .
6:    $L[0] = 0$ . ▷ Base Case.
7:   for  $1 \leq m \leq n$  do:
8:      $s^* \leftarrow \text{BINSEARCH}(D[1 : t], A[m])$ .
9:      $j^* \leftarrow E[s^*]$  if  $s^* \neq 0$ ; otherwise,  $j^* \leftarrow 0$ .
10:    Set  $L[m] \leftarrow 1 + s^*$  and  $\text{parent}[m] \leftarrow j^*$ 
11:    if  $L[m] > t$  then: ▷ Then  $L[m] = t + 1$ 
12:       $t \leftarrow t + 1$ 
13:       $D[t] \leftarrow A[m]$ 
14:       $E[t] \leftarrow m$ 
15:    else if  $A[m] < D[L[m]]$  then: ▷  $m \in J_s$  and  $A[m]$  is now the minimum, and  $D[s]$  should be set, where  $s = L[m]$ 
16:       $D[L[m]] \leftarrow A[m]$ 
17:       $E[L[m]] \leftarrow m$ .
18:     $m = \arg \max L[1 : n]$  ▷  $L[m]$  contains the length of the optimal LIS
19:    return  $\sigma$ .

```

Theorem 1. The Longest Increasing Subsequence can be solved in $O(n \log n)$ time.

To see if you have understood the LIS problem above, try doing this weighted version. First get an $O(n^2)$ time algorithm. Then get an $O(n \log n)$ time algorithm. 

Exercise: Imagine a weighted version of LIS. In this, every index $1 \leq j \leq n$ has a weight $w(j)$. The goal is to find an LIS $(j_1 < j_2 < \dots < j_k)$ which maximizes the total weight $w(j_1) + w(j_2) + \dots + w(j_k)$. Solve this weighted LIS problem, hopefully in $O(n \log n)$ time.