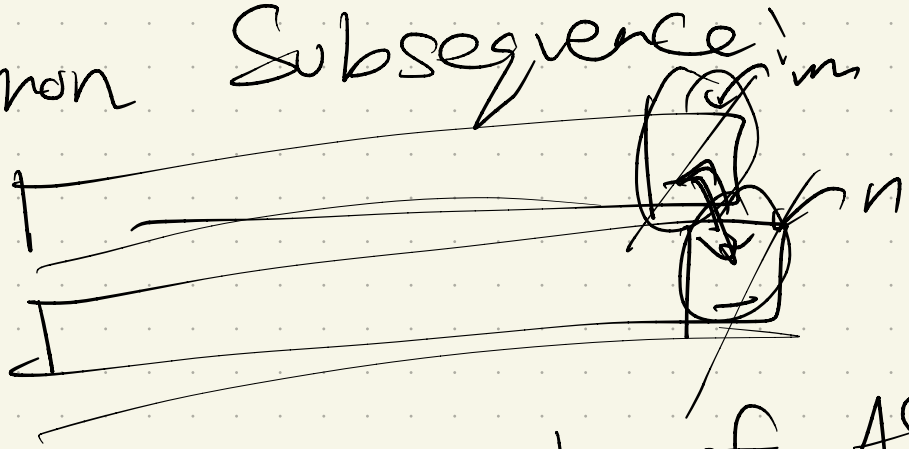
Transition: DS + Algorithms

Intro to DP

Worksheet :

Longest Common Subsequence :

A[1..m]
B[1..n]

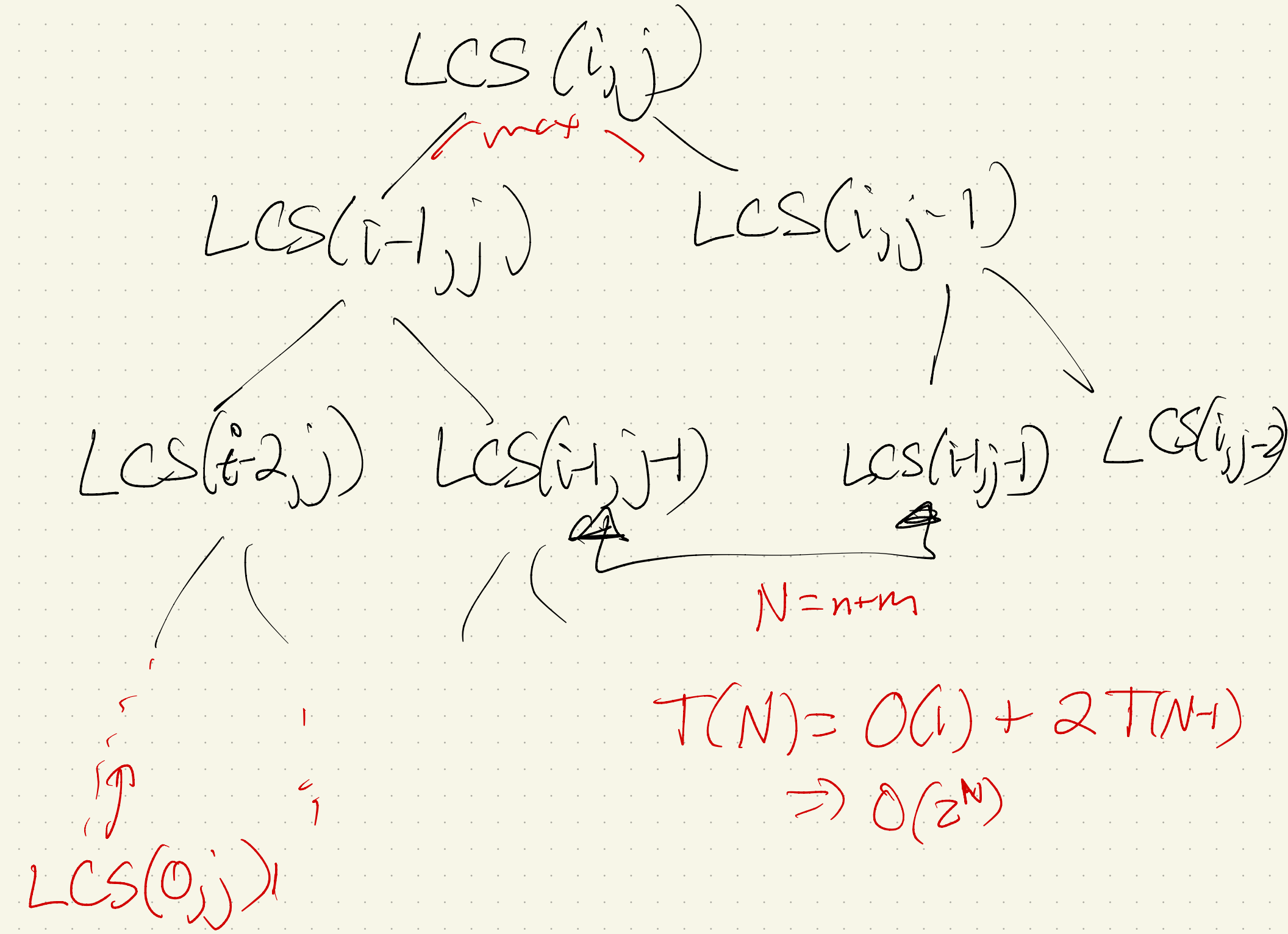


LCS(i,j) = longes com sub of A[1..i] & B[1..j]

$$LCS(m,n) = \begin{cases} \text{if } A[m] = B[n], \\ \quad LCS(m,n) = 1 + LCS(m-1,n-1) \\ \text{if } A[m] \neq B[n], \\ \quad = \max \begin{cases} LCS(m-1,n) \\ LCS(m,n-1) \\ LCS(m-1,n-1) \end{cases} \end{cases}$$

Base case

LCS(0,0) = 0 ←
LCS(i,0) = 0
LCS(0,j) = 0


LCS(i,j)
max
LCS(i-1,j)
LCS(i,j-1)
LCS(i-2,j)
LCS(i-1,j-1)
LCS(i-1,j-1)
LCS(i,j-2)
LCS(0,j)
N=n+m
T(N) = O(1) + 2T(N-1)
⇒ O(2^N)

Fibonacci Computations

Original runtime? → rec. formula

```
ComputeFib(n)
  IF n ≤ 1          n=0
                    n=1
    return n
  else
    return ComputeFib(n-1) +
           ComputeFib(n-2)
```

$$T(n) = O(1) + T(n-1) + T(n-2)$$

linear inhomogeneous rec.

$a_n = a_{n-1} + a_{n-2} + c$ ← # of ops

$a=1 \quad b=-1 \quad c=-1$

$r^2 - r - 1 = 0$

$b^2 - \sqrt{4ac}$

$\hookrightarrow r = \frac{1 \pm \sqrt{5}}{2}$

let $\phi = \frac{1+\sqrt{5}}{2} > 1$, $\hat{\phi} = \frac{1-\sqrt{5}}{2}$

$\Rightarrow T(n) = \Theta(\phi^n)$ bad

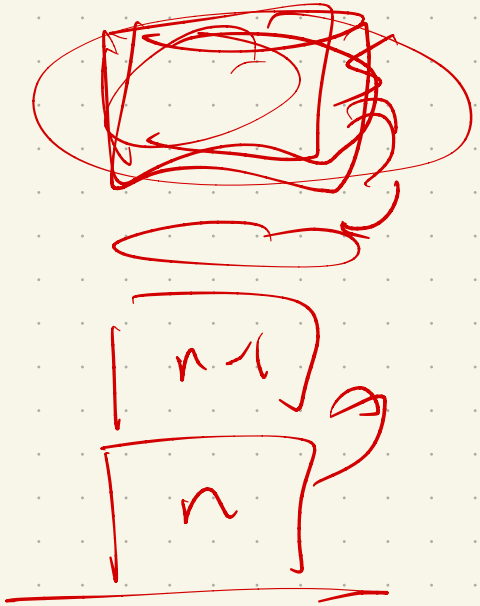
both $\begin{cases} 0 \\ \Omega \end{cases}$

Make it faster:

```
MEMFIBO(n):
  if (n < 2)
      return n
  else
      if F[n] is undefined
          F[n] ← MEMFIBO(n − 1) + MEMFIBO(n − 2)
      return F[n]
```

called

function



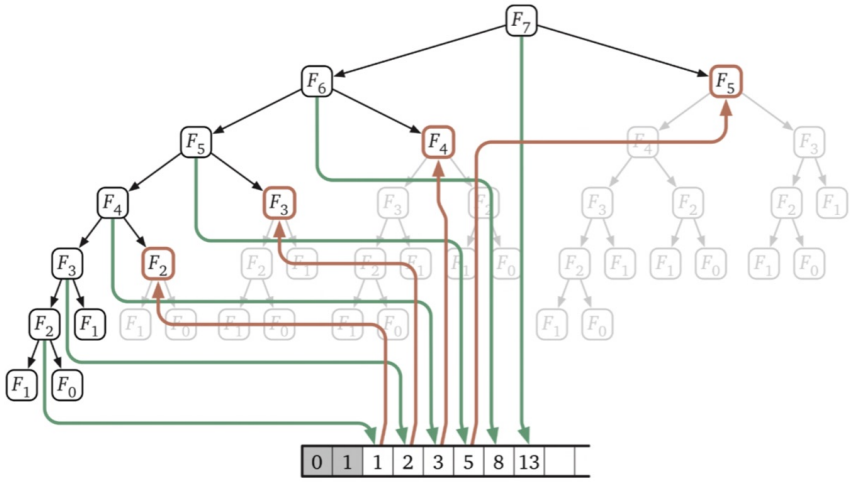


Figure 3.2. The recursion tree for F_7 trimmed by memoization. Downward green arrows indicate writing into the memoization array; upward red arrows indicate reading from the memoization array.

Fine! runtime: 2 lookups in table per value plus one addition & save

$\Rightarrow O(n)$

Reset our view:

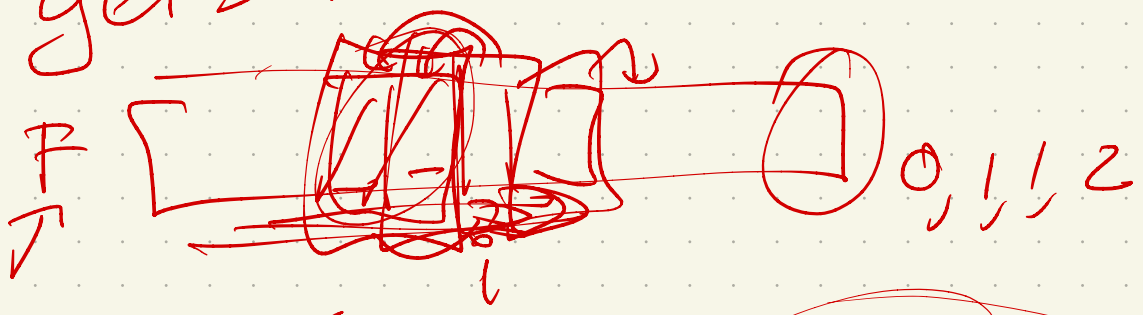
What do we need to save?

rewrite as a loop!

gets rid of recursion

```
ITERFIBO(n):
    F[0] ← 0
    F[1] ← 1
    for i ← 2 to n
        F[i] ← F[i−1] + F[i−2]
    return F[n]
```

table lookup in array




```
ITERFIBO2(n):
    prev ← 1 0
    curr ← 0 1
    for i ← 1 to n
        next ← curr + prev
        prev ← curr
        curr ← next
    return curr
```

O(1) space

O(n) time

O(n) time

space? O(n)

needed? NO!

prev ← 0 1 1 2

curr ← 1 1 2 3

next ← 2 3

This ♡ section: Can actually do better!

Fancy math tricks

$$\begin{bmatrix}0&1\\1&1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix}=\begin{bmatrix}0\\1\end{bmatrix}\begin{matrix}F_0\\F_1\end{matrix}$$

$$\begin{bmatrix}0&1\\1&1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix}=\begin{bmatrix}1\\1\end{bmatrix}\begin{matrix}F_1\\F_2\end{matrix}$$

$$\begin{bmatrix}0&1\\1&1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}=\begin{bmatrix}1\\2\end{bmatrix}\begin{matrix}F_2\\F_3\end{matrix}$$

$$\begin{bmatrix}0&1\\1&1\end{bmatrix}\begin{bmatrix}1\\2\end{bmatrix}=\begin{bmatrix}2\\3\end{bmatrix}\begin{matrix}F_3\\F_4\end{matrix}$$

$$\begin{bmatrix}0&1\\1&1\end{bmatrix}\begin{bmatrix}2\\3\end{bmatrix}=\begin{bmatrix}3\\5\end{bmatrix}\begin{matrix}F_4\\F_5\end{matrix}$$

$$\begin{bmatrix}0&1\\1&1\end{bmatrix}^5\begin{bmatrix}1\\0\end{bmatrix}$$

$$\begin{bmatrix}0&1\\1&1\end{bmatrix}^n\begin{bmatrix}0\\1\end{bmatrix}=\begin{bmatrix}F_{n-1}\\F_n\end{bmatrix}$$

Proof: induction. **BC**

IH: assume $\begin{bmatrix}0&1\\1&1\end{bmatrix}^n\begin{bmatrix}0\\1\end{bmatrix}=\begin{bmatrix}F_{n-1}\\F_n\end{bmatrix}$

IS:

$$\begin{bmatrix}0&1\\1&1\end{bmatrix}^{n+1}\begin{bmatrix}0\\1\end{bmatrix}=\begin{bmatrix}0&1\\1&1\end{bmatrix}\cdot\begin{bmatrix}0&1\\1&1\end{bmatrix}^n\begin{bmatrix}0\\1\end{bmatrix}\overset{IH}{=}\begin{bmatrix}0&1\\1&1\end{bmatrix}\cdot\begin{bmatrix}F_{n-1}\\F_n\end{bmatrix}=\begin{bmatrix}F_n\\F_{n+1}\end{bmatrix}$$

(can read rest of section on your own! :))

Fibonacci Recap: good/bad

- "Simple" yet interesting example
- Illustrates how powerful this concept can be

Downside:

- Not always so obvious how to convert the recursion into an iterative structure!

Text segmentation : from Ch 2!

Given an index i, find a segmentation of the suffix $A[i..n]$.

$$Splittable(i) = \begin{cases} \text{TRUE} & \text{if } i > n \\ \bigvee_{j=i}^{n} \left(\text{IsWord}(i, j) \wedge Splittable(j+1) \right) & \text{otherwise} \end{cases}$$

(i) check if it's a word from i to j

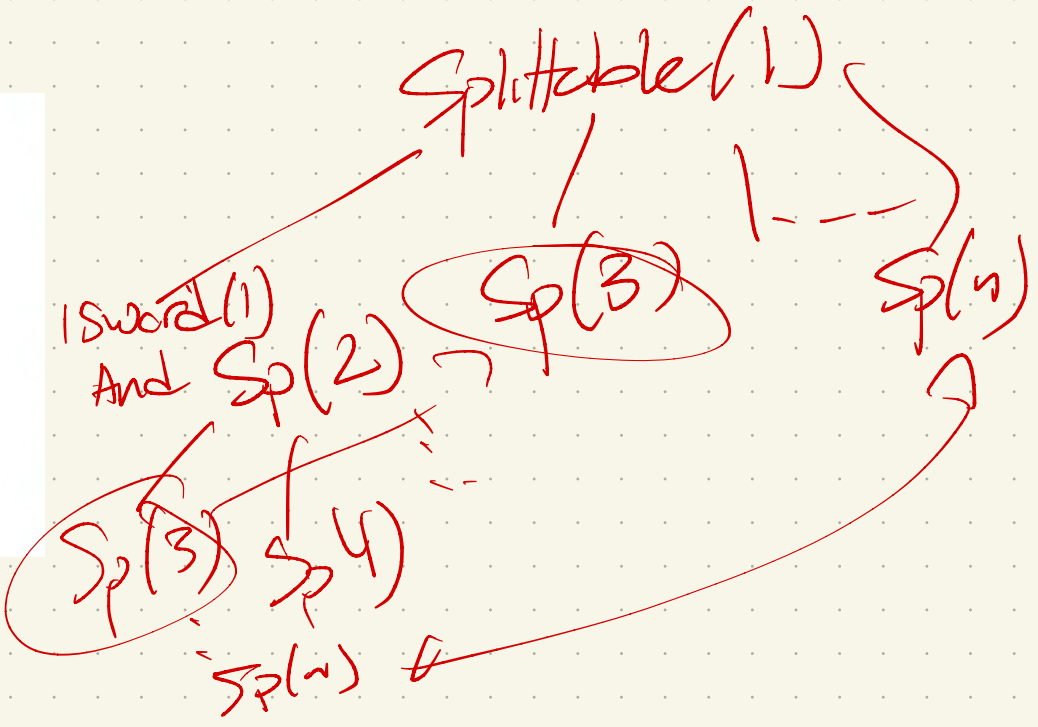
or

and

check if j+1 to end is Splittable

if any (IsWord(i,j) and Splittable(j+1)) is true, I'll return true

```
⟨⟨Is the suffix A[i..n] Splittable?⟩⟩
SPLITTABLE(i):
  if i > n
      return TRUE
  for j ← i to n
      if ISWORD(i, j)
          if SPLITTABLE(j + 1)
              return TRUE
  return FALSE
```



Memoization

Think about our recursion:
calling splittable(i) quite a bit.

After first time it's computed, store the answer.

Then, later calls just look it up!

How to store?

```
⟨⟨Is the suffix A[i..n] Splittable?⟩⟩
SPLITTABLE(i):
  if i > n
    return TRUE
  for j ← i to n
    if ISWORD(i, j)
      if SPLITTABLE(j + 1)
        return TRUE
  return FALSE
```

check if exists & look up answer if already done

S[1..n]: S[i] store true if A[i..n] is splittable

So code:

```
⟨⟨Is the suffix A[i..n] Splittable?⟩⟩
SPLITTABLE(i):
  if i > n
      return TRUE
  for j ← i to n
      if ISWORD(i, j)
          if SPLITTABLE(j + 1)
              return TRUE
  return FALSE
```

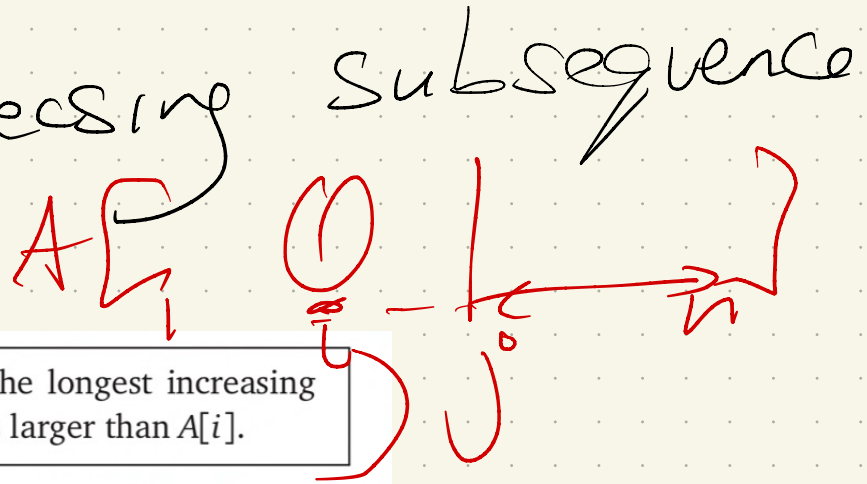


How to implement

Run time & space?

Next reading:

Longest increasing subsequence

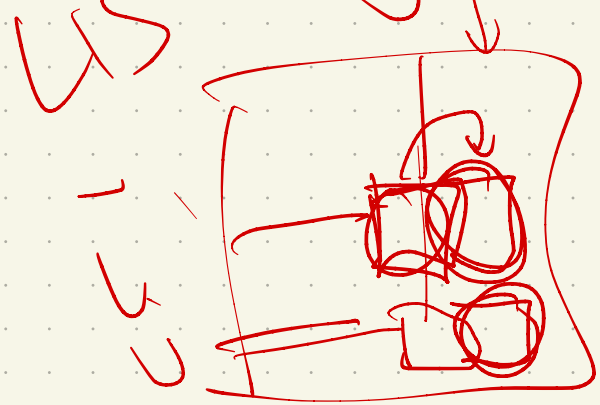


Given two indices i and j, where $i < j$, find the longest increasing subsequence of $A[j..n]$ in which every element is larger than $A[i]$.

$$LISbigger(i,j) = \begin{cases} 0 & \text{if } j > n \\ LISbigger(i, j+1) & \text{if } A[i] \geq A[j] \\ \max \begin{Bmatrix} LISbigger(i, j+1) \\ 1 + LISbigger(j, j+1) \end{Bmatrix} & \text{otherwise} \end{cases}$$

Why?

lookup $LISbigger(i,j)$ quickly



How to store for DP?

How to code?