Transition: DS+ Algorithms

Dynamic Pro: LIS

Worksheet:

Key: What is greed vs DP? (don't do it!)

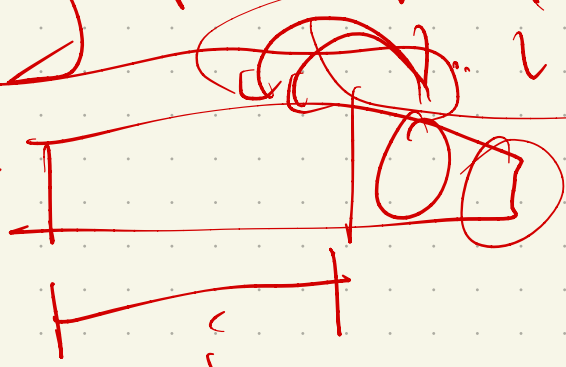
F[i] = money at house i

→ local optimal choice
"pick largest & recurse
often not obvious if greedy = optimal

1b) Let F[i] = max money from houses 1..i

$F[0] = 0$

$F[1] = R[1]$ ← size 1



$F[i] = \max(F[i-1], F[i-2] + R[i])$

return $F[n]$ = max in houses 1..n

$G[i]$ = max amount I can get from houses $i \ldots n$ (ignoring $1 \ldots i-1$)

$G[n+1] = 0$ ← "dummy" at end

$G[n] = R[n]$ ← size 1

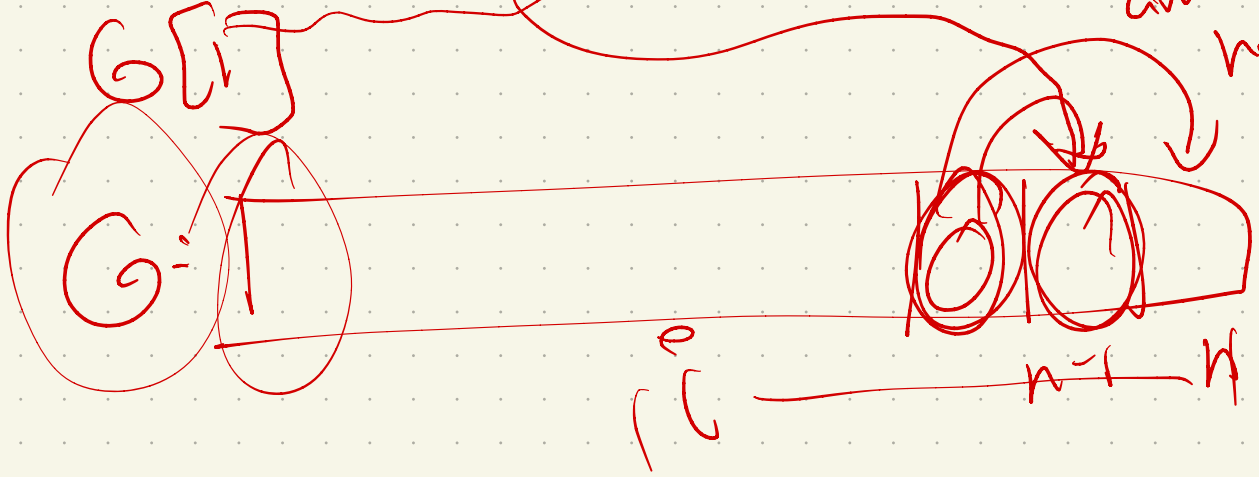
$G[i] = \max\{G[i+1], R[i] + G[i+2]\}$

$\rightarrow G[n-1] = \max\{R[n], R[n-1]\}$

amt at house n

amount at house $n-1$

return $G[1]$



Houses 1 2 ... 10

Fill F[0] → F[1]

for i = 2 to n

fill F[i] using F[i-1] + F[i-2]

return F[n]

by max formula 2 slides back

Let $F[i]$ = max amount I can steal from $1 \dots i$

in loop:

$$F[i] \leftarrow \max\{F[i-1], F[i-2] + Haves[i]\}$$

Splitting text:

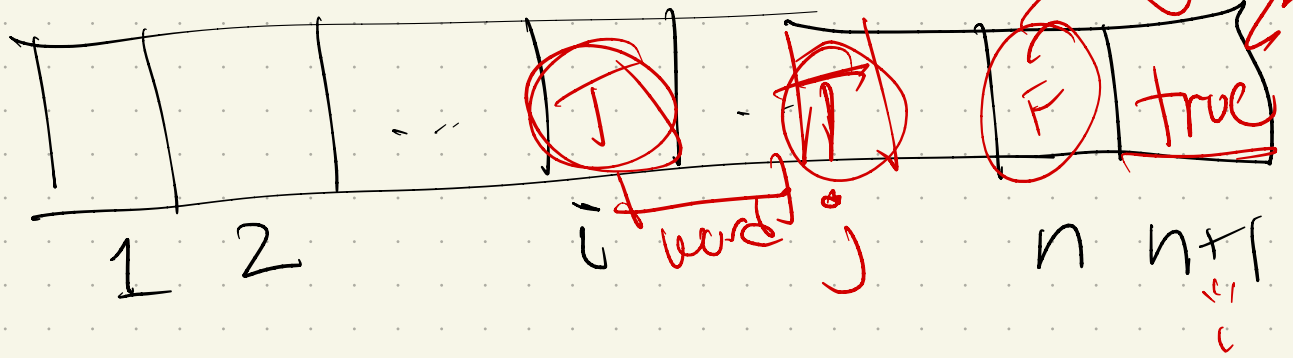
Recursive

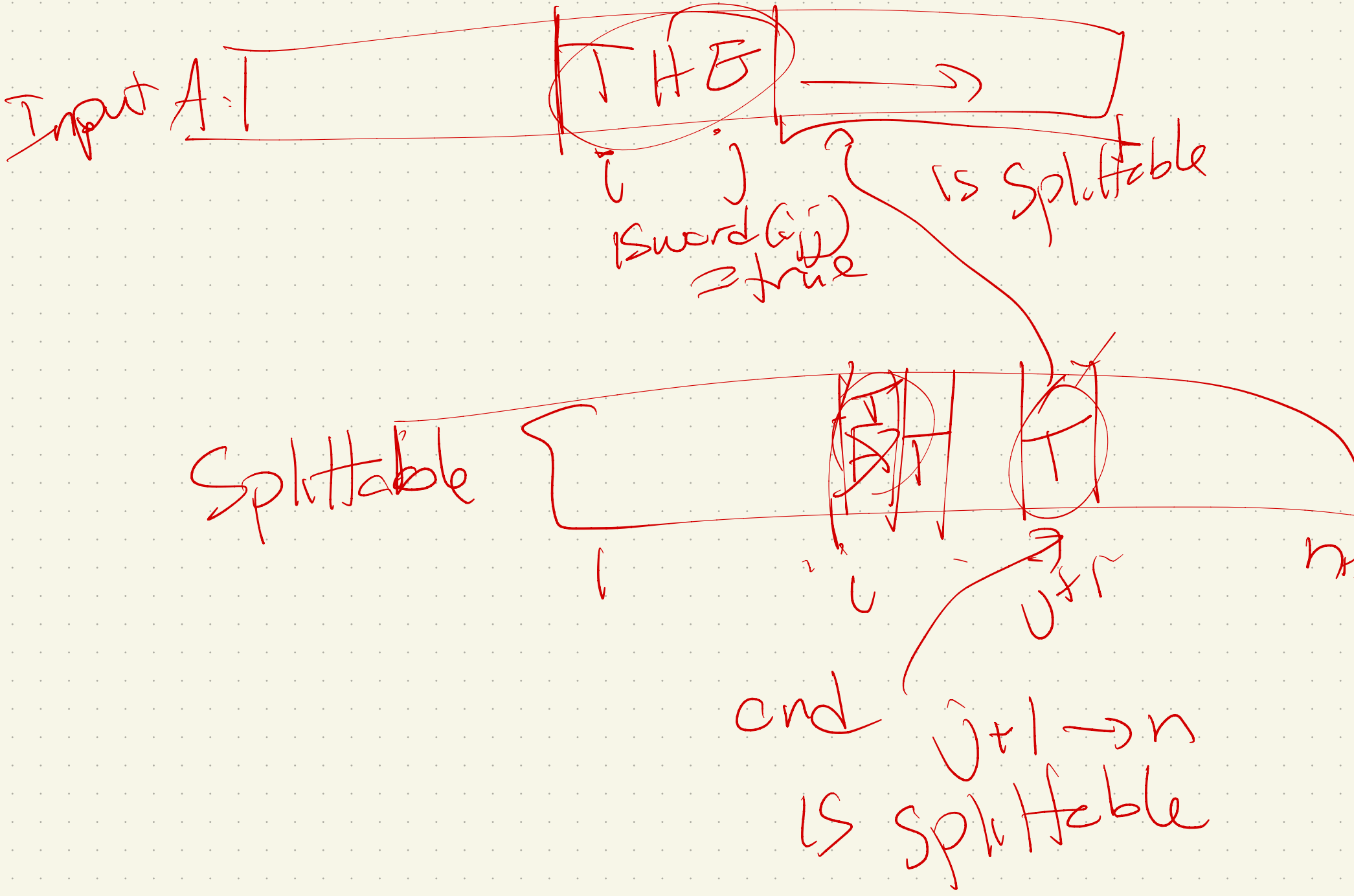
```
⟨⟨Is the suffix A[i..n] Splittable?⟩⟩
SPLITTABLE(i):
  if i > n
    return TRUE
  for j ← i to n
    if ISWORD(i, j)
      if SPLITTABLE(j + 1)
        return TRUE
  return FALSE
```

```
FASTSPLITTABLE(A[1..n]):
  SplitTable[n + 1] ← TRUE
  for i ← n down to 1
    SplitTable[i] ← FALSE
    for j ← i to n
      if ISWORD(i, j) and SplitTable[j + 1]
        SplitTable[i] ← TRUE
  return SplitTable[1]
```

Figure 3.3. Interpunctio verborum velox

Base case + how we fill in:




Input A[1..n]
THEN
i
j
isword(i,j) = true
is Splitable
Splitable
i
j
j+1
and
j+1 → n
is Splitable

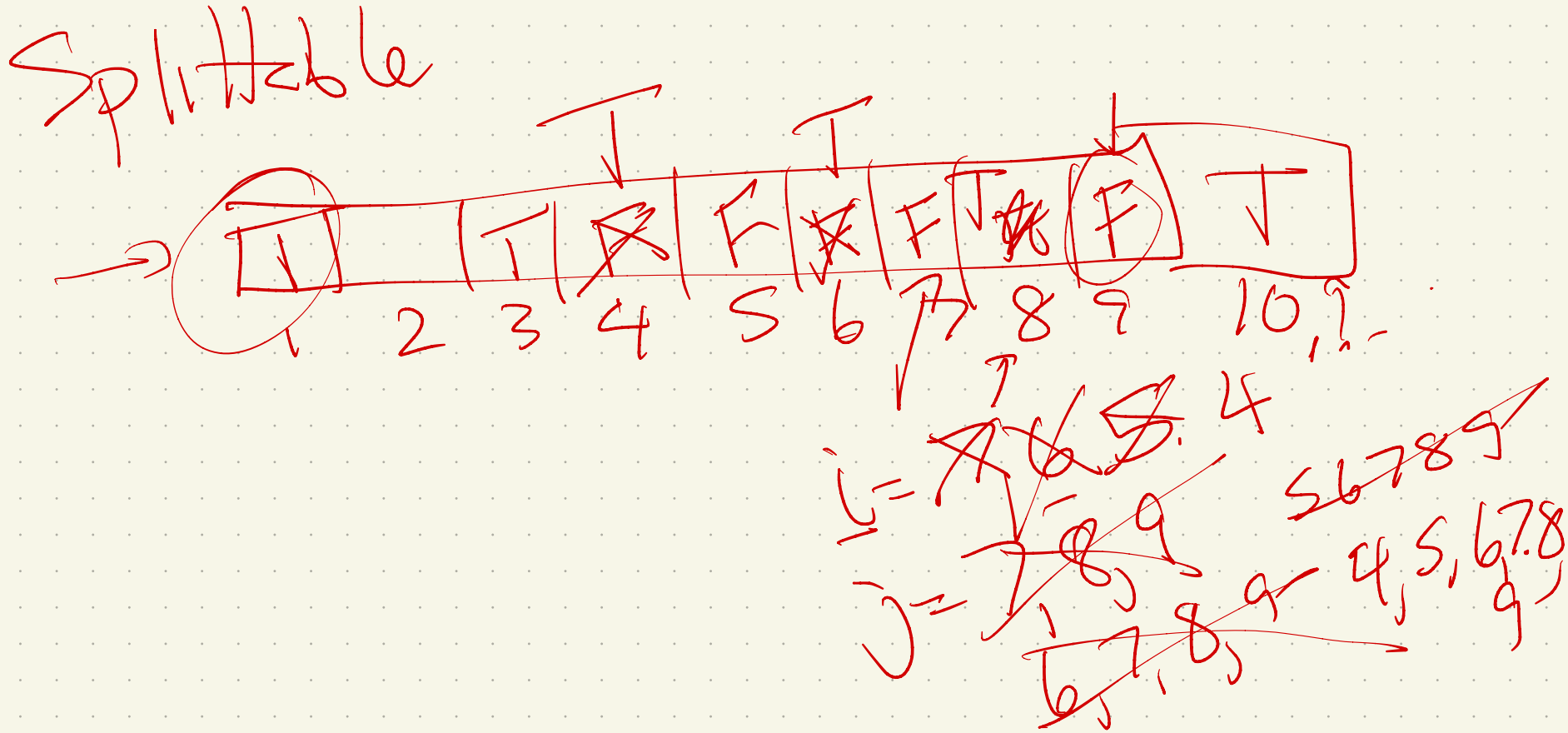
A: HI I AM ER IN

2 2 3 4 5 6 7 8 9

n = 9

```
FastSplittable(A[1..n]):
  SplitTable[n+1] ← True
  for i ← n down to 1
      SplitTable[i] ← False
      for j ← i to n
          if IsWord(i, j) and SplitTable[j+1]
              SplitTable[i] ← True
  return SplitTable[1]
```

Figure 3.3. Interpunctio verborum velox



Runtime & Space

```
FastSplittable(A[1..n]):
  SplitTable[n+1] ← True
  for i ← n down to 1
      SplitTable[i] ← False
      for j ← i to n
          if IsWord(i, j) and SplitTable[j+1]
              SplitTable[i] ← True
  return SplitTable[1]
```

Figure 3.3. Interpunctio verborum velox

array $O(n)$

$$\text{Time} = \sum_{i=1}^{n} \left(1 + \sum_{j=i}^{n} 3 \right)$$

$$= \sum_{i=1}^{n} 1 + \sum_{i=1}^{n} \sum_{j=i}^{n} 3 = n + \sum_{i=1}^{n} 3(n-i)$$

$$= O(n^2)$$

Next: Longest Increasing Subsequence

Recall: Recursive version

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 $A[i]$.

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

too small must skip

skip

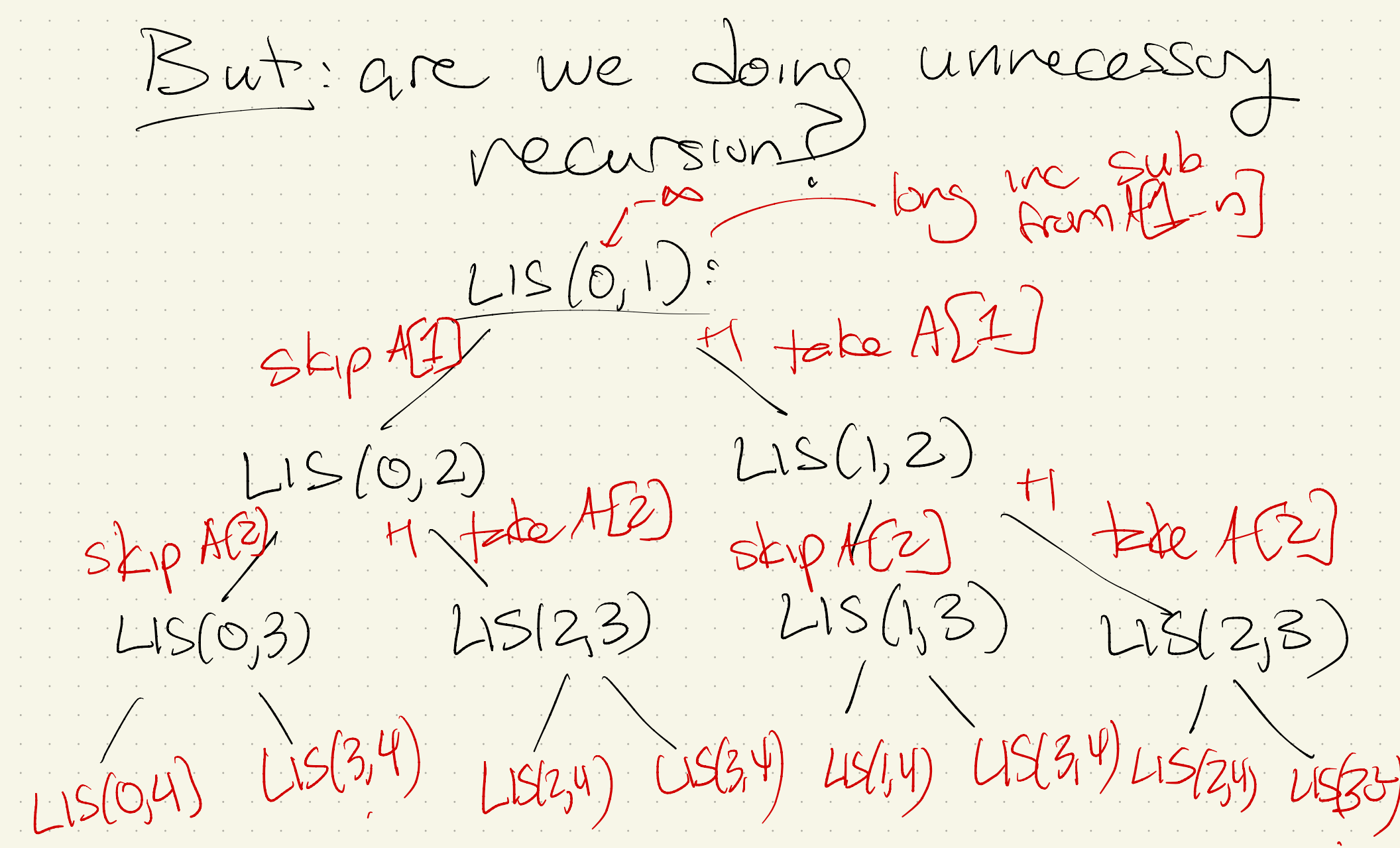
take A[j]

Long. inc sub from j..n where A[i] was last included element

8, 2, 9

i

j must skip


But: are we doing unnecessary recursion?
-∞
long inc sub from A[1-n]
LIS(0,1):
skip A[1]
+1 take A[1]
LIS(0,2)
LIS(1,2)
skip A[2]
+1 take A[2]
skip A[2]
+1 take A[2]
LIS(0,3)
LIS(2,3)
LIS(1,3)
LIS(2,3)
LIS(0,4)
LIS(3,4)
LIS(2,4)
LIS(3,4)
LIS(1,4)
LIS(3,4)
LIS(2,4)
LIS(3,4)

Next: memoize?

What sort of calls are we making often? → LIS(i, j)

Can we save them, & avoid recomputing over and over?

Recursive code:

```
LISBIGGER(i, j):
  if j > n
      return 0
  else if A[i] ≥ A[j]
      return LISBIGGER(i, j + 1)
  else
      skip ← LISBIGGER(i, j + 1)
      take ← LISBIGGER(j, j + 1) + 1
      return max{skip, take}
```

For $j > n$, $LIS(i, j) = 0$

For each (i, j), store result for later lookup.

So, data structure!

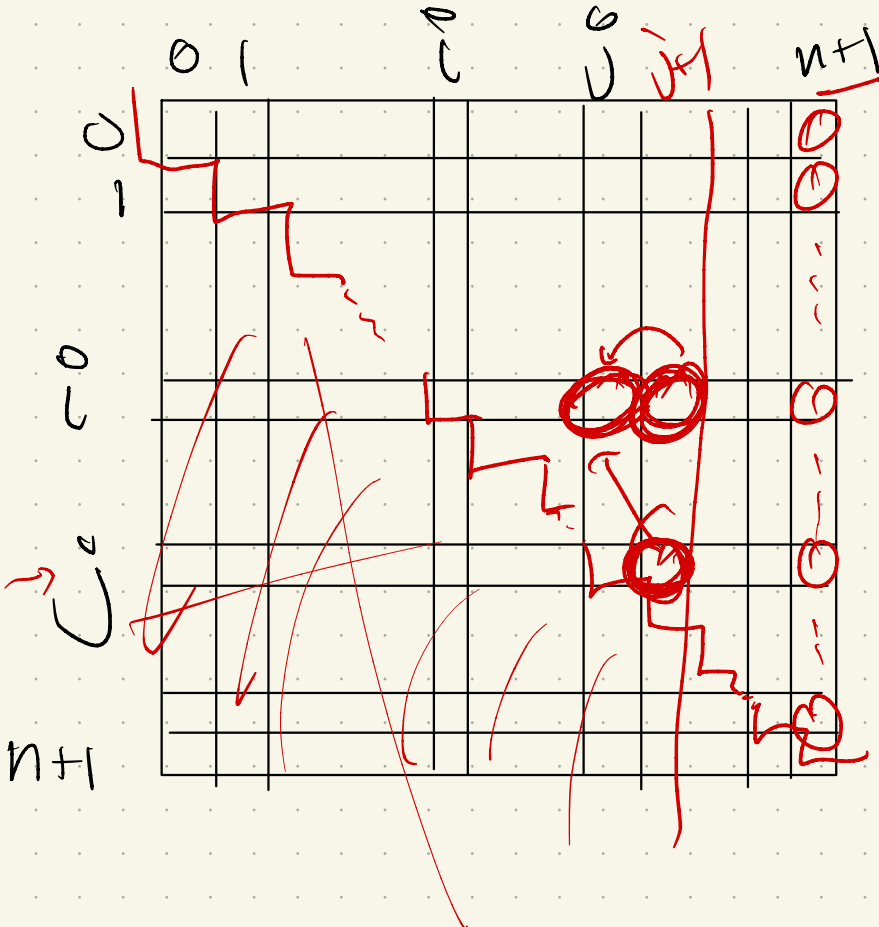
Need $LIS[i,j]$ for all $i < j$.

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

2nd index "j"

one column over!

Picture:



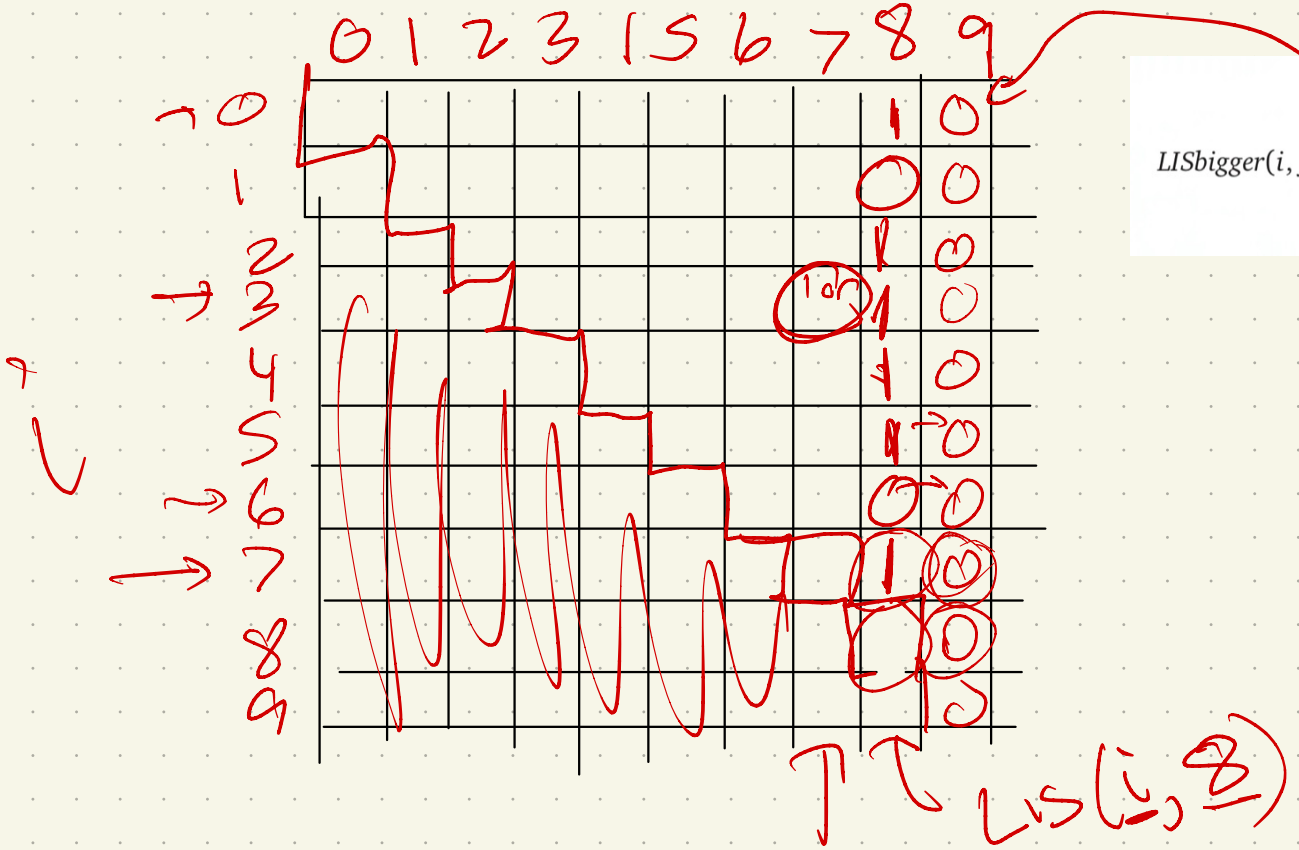
1st index "i"

Base case?

↓

column $n+1$

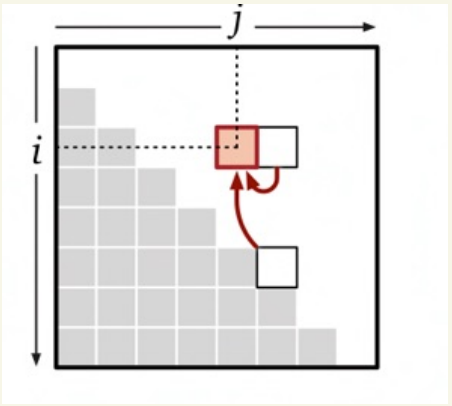
Ex: A = [10 2 4 1 6 11 7 9]



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

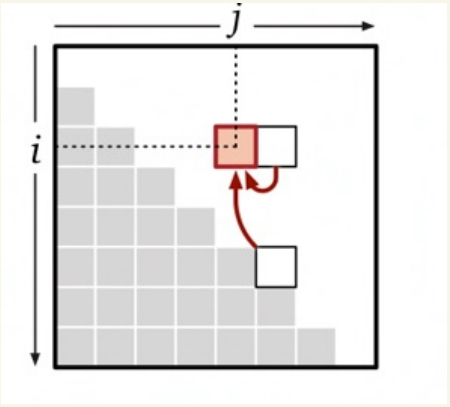
LIS(i, 8)

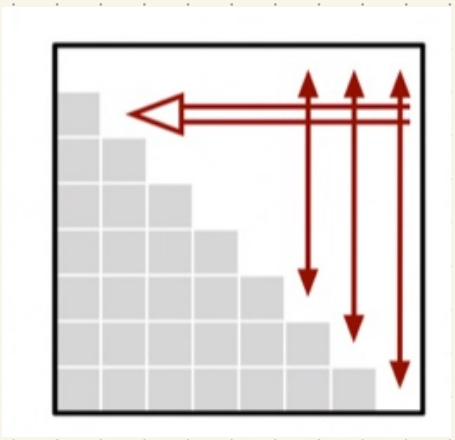
LIS(i, 7)



So: code to compute!

```
FastLIS(A[1..n]):
  A[0] ← −∞                          ⟨⟨Add a sentinel⟩⟩
  for i ← 0 to n                     ⟨⟨Base cases⟩⟩
      LISbigger[i, n+1] ← 0
  for j ← n down to 1
      for i ← 0 to j − 1             ⟨⟨... or whatever⟩⟩
          keep ← 1 + LISbigger[j, j+1]
          skip ← LISbigger[i, j+1]
          if A[i] ≥ A[j]
              LISbigger[i, j] ← skip
          else
              LISbigger[i, j] ← max{keep, skip}
  return LISbigger[0, 1]
```





Runtime & space?

Edit distance:

HUGE in bioinformatics!

One of the basic tools in sequence alignment.

(I have a book with an entire chapter on how to optimize.)

Also: spell checkers, word prediction, etc.

operations: insert, delete, edit

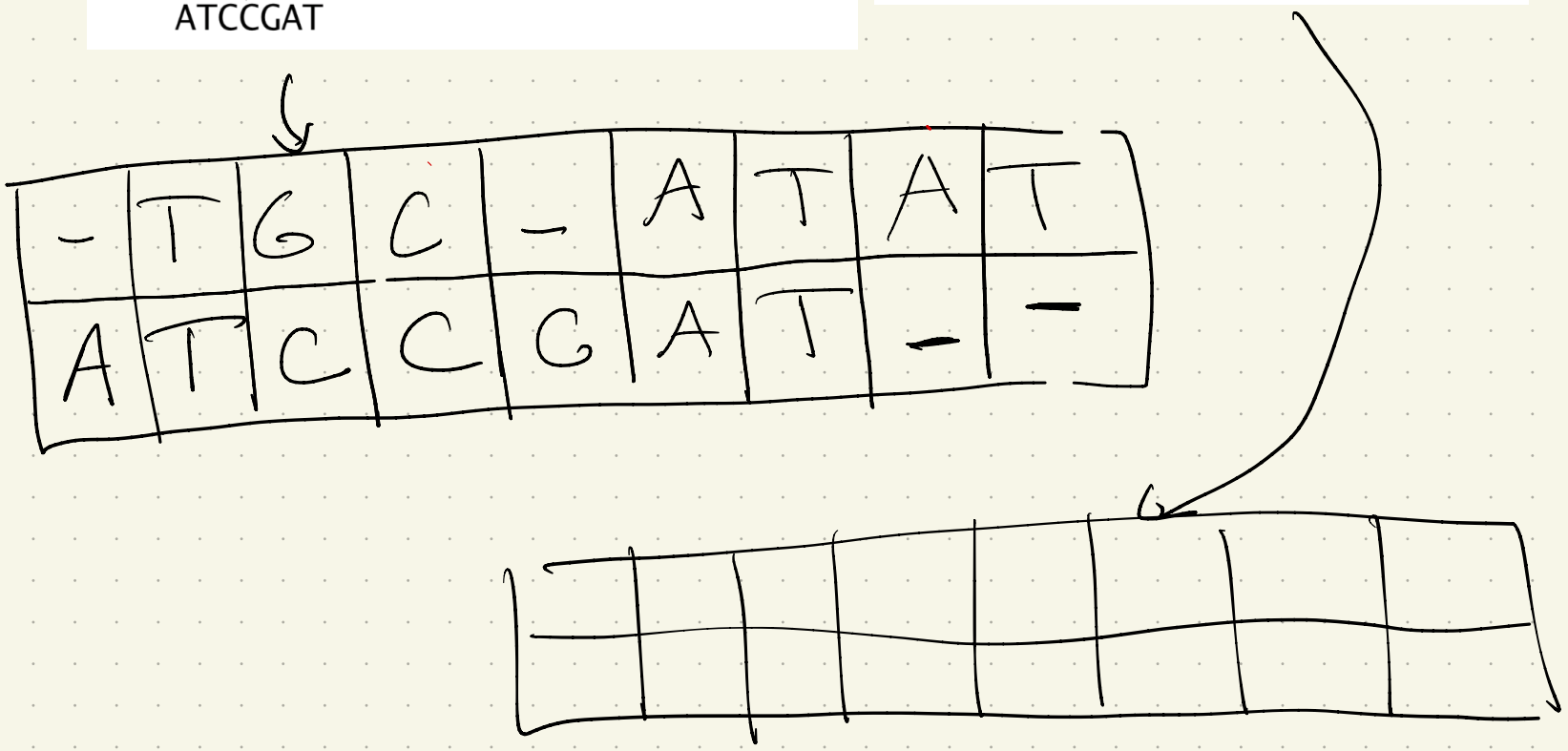
ALGORITM

→ ALTRUISTIC

Example: TGCATAT to ATCCGAT

TGCATAT
↓ delete last T
TGCATA
↓ delete last A
TGCAT
↓ insert A at the front
ATGCAT
↓ substitute C for G in the third position
ATCCAT
↓ insert a G before the last A
ATCCGAT

TGCATAT
↓ insert A at the front
ATGCATAT
↓ delete T in the sixth position
ATGCAAT
↓ substitute G for A in the fifth position
ATGCGAT
↓ substitute C for G in the third position
ATCCGAT



How to start? Recursively.

Let's try:

A: ALGORITM

B: ALTRUISTIC

Look at last letter: 3 options!