Dynamic Programming.

LIS

Input: 

$x[1 \ldots n]$

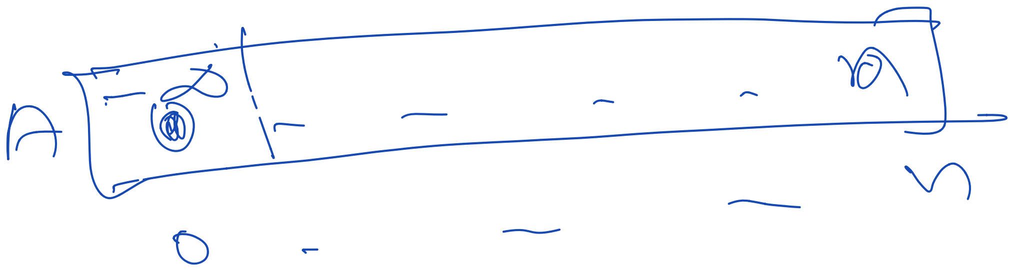
$i < j < z$

Output: The length of a longest ~~inc~~ increasing subseq.

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

$LISbigger(i,j)$ = Length of the LIS in $A[j \ldots n]$ s.t all the elements in the subsequence

is greater than A[i])

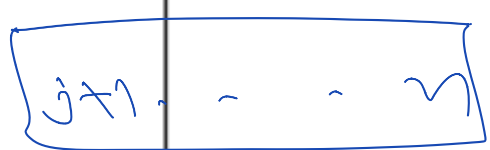


LISbigger(0,1) ← Answer to our question.

(*) One has to prove the above recurrence is correct

[– MI].

```
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}
```



$T(n) \leq 2T(n-1) + 4$

$= O(n^2)$

For all $i < j$,

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

$i < j$

LISbigger(i, j)

i = 0 to n

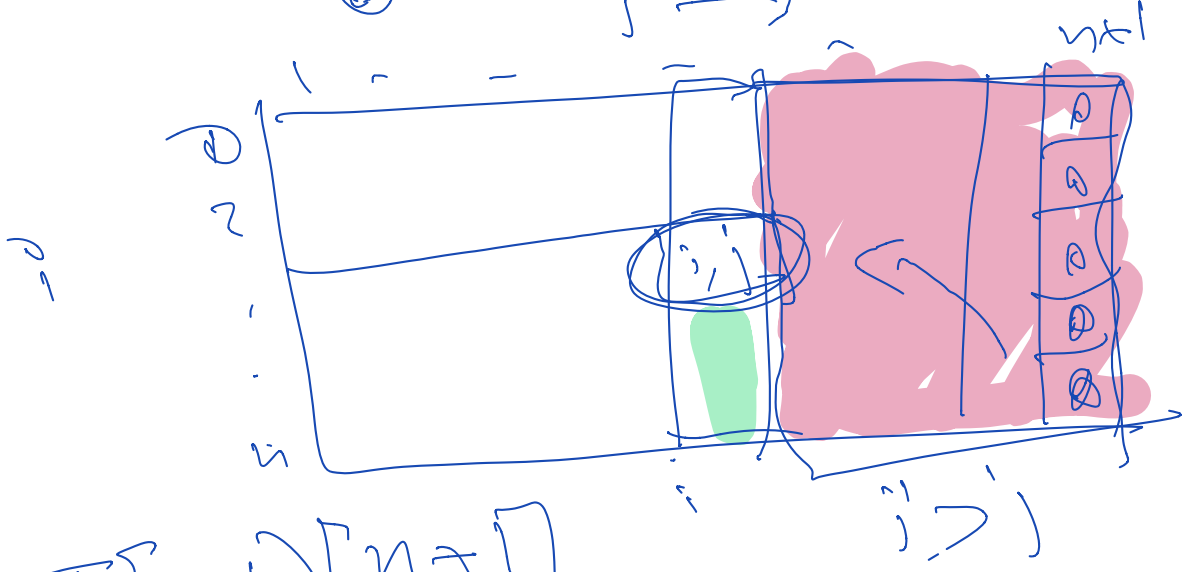
j = i to n+1

j = n+1 → 1

i = 0 to j-1

i → 0

j →



T[n][n+1]

T[i, j] = LISbigger(i, j)

```
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]
```

Worst-case running time
$= O(n^2)$

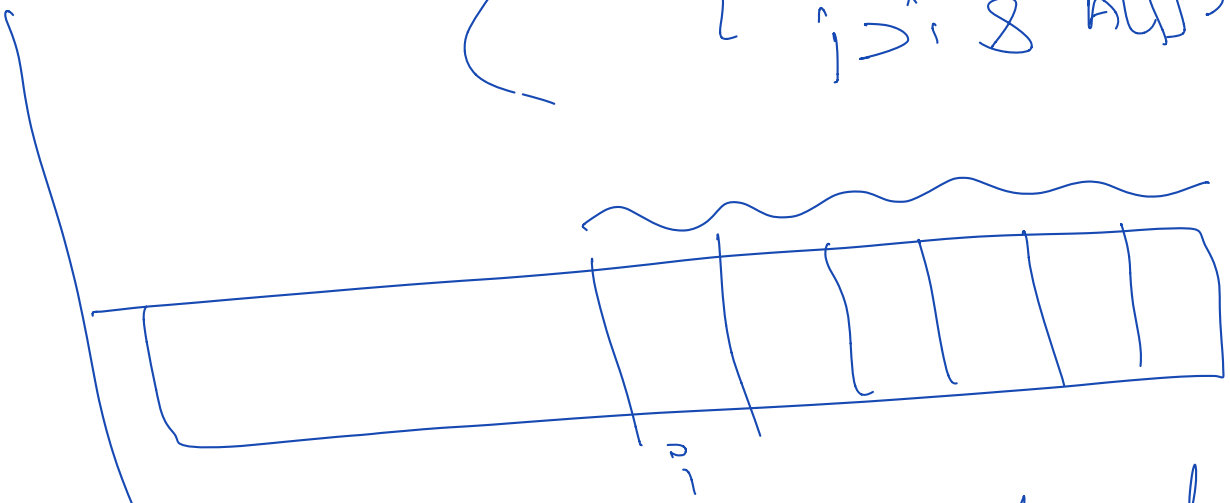
Space complexity: $O(n^2)$ → $O(n^2)$?

- Can we improve space complexity.

$LISFirst(i)$ = the length of the longest increasing subsequence in $A[i \ldots n]$ starting at $A[i]$.

for $i = 1$ to n:

$$LISFirst(i) = \begin{cases} 0 & i > n \\ \max\{1 + LISFirst(j) \mid j > i \text{ and } A[j] > A[i]\} & \text{(2)} \end{cases}$$



Exc

Prove that the recursive formula (2) is correct.

LISF(1), LISFirst(2), ..., LISFirst(n)

Computation order: i = n → 1

LISFirst[1 ... n]

→ O(n)

```
FastLIS2(A[1..n]):
  A[0] = -∞                                  ⟨⟨Add a sentinel⟩⟩
  for i ← n downto 0
      LISfirst[i] ← 1
      for j ← i + 1 to n                     ⟨⟨...or whatever⟩⟩
          if A[j] > A[i] and 1 + LISfirst[j] > LISfirst[i]
              LISfirst[i] ← 1 + LISfirst[j]
  return LISfirst[0] − 1                     ⟨⟨Don't count the sentinel⟩⟩
```

LISFirst[i].

- Space complexity = O(n)
- Time complexity = O(n²)

O(n²) and Ω(n²)

Dynamic Programming (DP)

1. First come up with a recursive formula for your problem (a generalization of your problem (prove it)
2. Find out the correct evaluation order
3. Find suitable data structure.
4. Populate the D.S in the order you found in step 2

have memoized!

Next Class

- Text segmentation } Backtracking
- subset sum

↳ DP

- more problems