Specification: Given two indices i and j, where $i < j$, find the longest increasing subseq. of $A[j \ldots n]$ in which every elt is larger than $A[i]$.

Sol'n:

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

Alternatively, if you prefer pseudocode:

```
LISBIGGER(i, j):
1:  if j > n
2:      return 0
3:  else if A[i] ≥ A[j]
4:      return LISBIGGER(i, j + 1)
5:  else
6:      skip ← LISBIGGER(i, j + 1)
7:      take ← LISBIGGER(j, j + 1) + 1
8:      return max{skip, take}
```

1st call: LIS(A[1..n])
A[0] ← −∞
return LISBIGGER(0,1)

the recurrence relation for the worst-case input

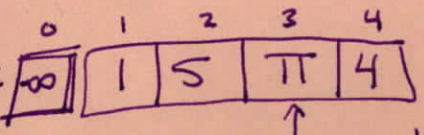
- worst-case is if the input array is in ascending order.
  ∴ our worst-case recurrence:

$$T(n) = 2T(n-1) + \Theta(1)$$

$$\Rightarrow \text{runtime is } \Theta(2^n)$$

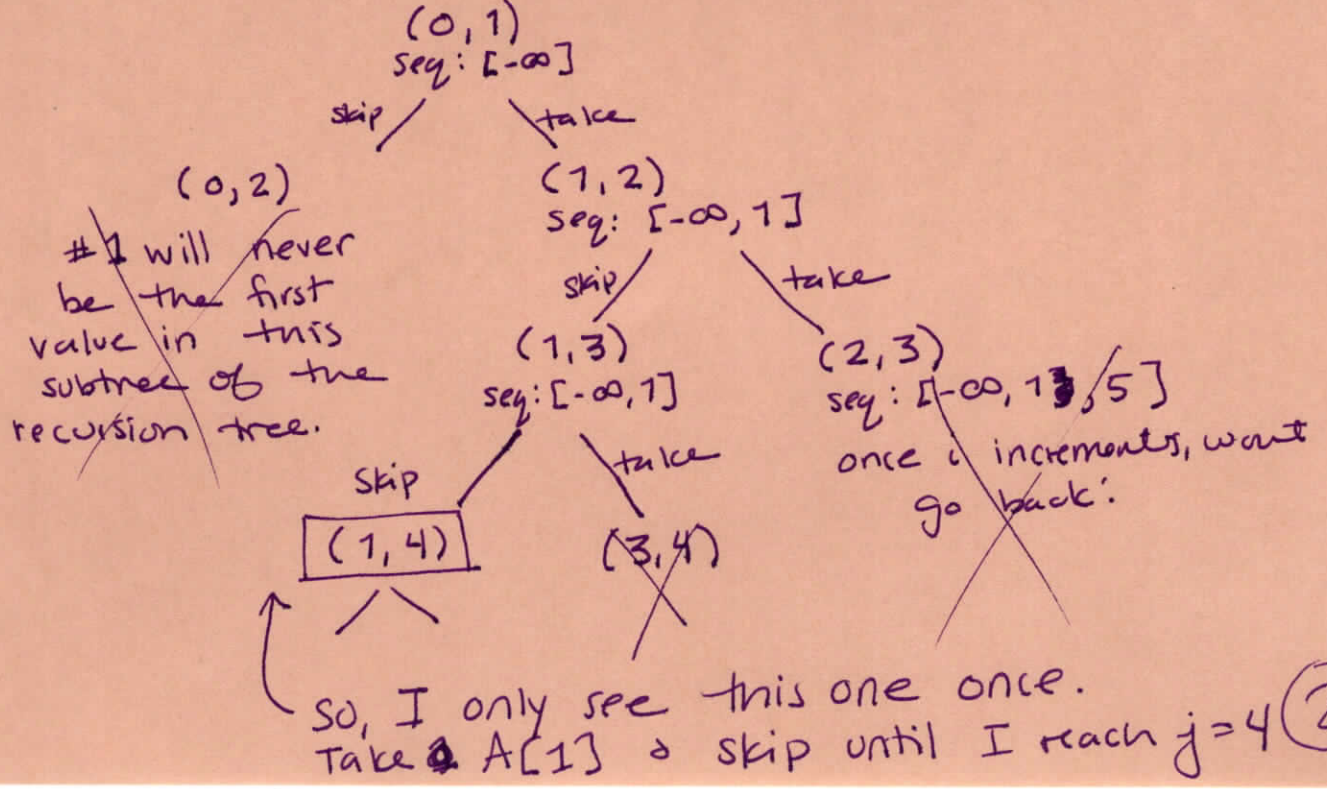
- note: in the textbook, they write $T(n) \leq 2T(n-1) + O(1)$, which is also true. But only gives us $T(n)$ is $O(2^n)$.

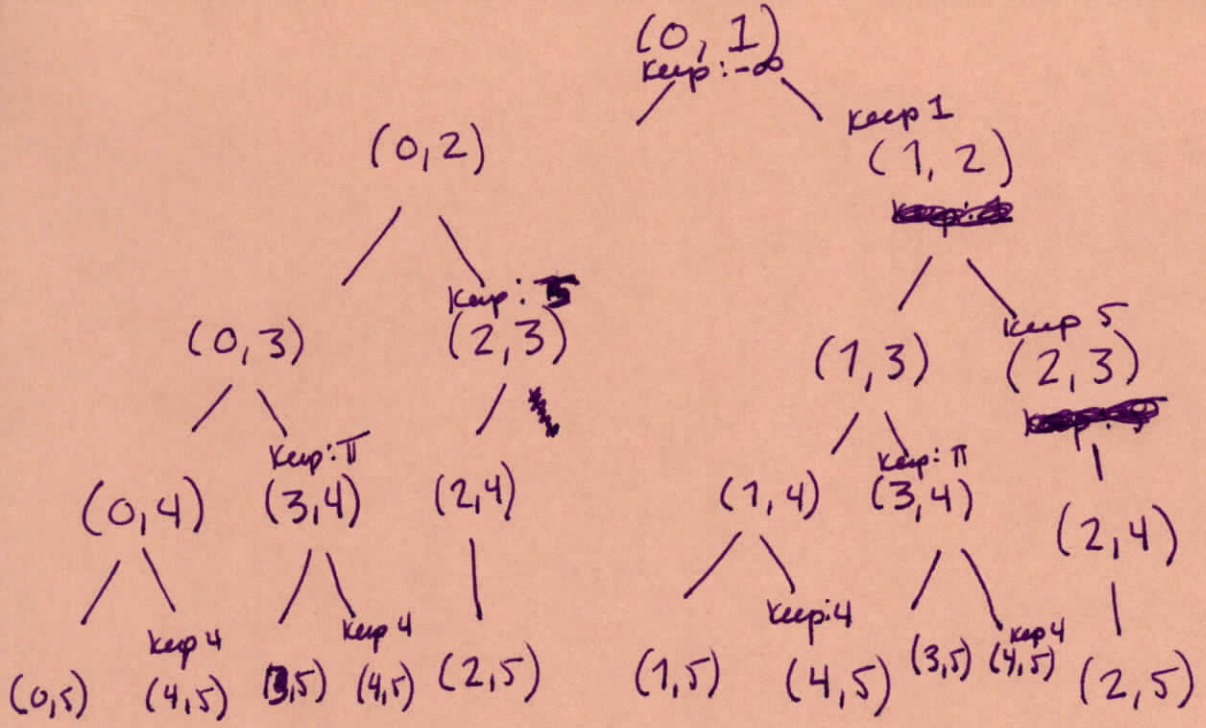
Consider the input



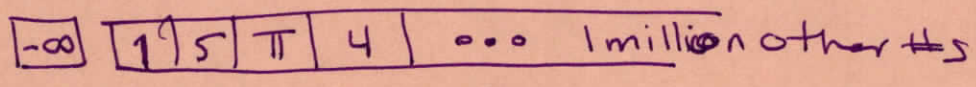
we're using read-ram, so this is ok.

Q1 how many times is LisBigger(1,4) called?
Q2 What do I see multiple times (& how many?)




(0, 1)
keep: -∞
keep 1
(1, 2)
(0, 2)
keep: 5
(2, 3)
(0, 3)
(1, 3)
keep 5
(2, 3)
keep: π
(0, 4)
(3, 4)
(2, 4)
(1, 4)
(3, 4)
(2, 4)
keep 4
keep 4
keep: 4
keep 4
(0, 5)
(4, 5)
(3, 5)
(4, 5)
(2, 5)
(1, 5)
(4, 5)
(3, 5)
(4, 5)
(2, 5)

A2 (3,4) is going to be seen 2x
(2,4) seen 2x
(2,3) seen 2x

What if A = 

Diving into this recursion many times is really bad (2x is twice as bad...)

idea: Let's save the values the first time we encounter them so that the next time we encounter them, we just look them up!

1st: Determine the set of all subproblems.
2nd: Figure out the data structure to store them all!

The subproblems are indexed by i,j.
Specifically

$$\{(i,j) \in \underline{n}^2 \mid i < j\}$$

where $\underline{n} := \{1, 2, \ldots, n\}$

Θ(1) lookup!

We propose to use a 2D array!
→ note: we don't use half the cells, but that's ok

1st call LIS(A[1...n])

Create a global 2D array T, all n×n values are "NaN"

A[0] ← −∞

return LIS BIGGER (0,1)

```
LisBigger(i,j)
    if j>n
    | return 0
    if T[i,j] ≠ NaN
    | return T[i,j]
    else if A[i] ≥ A[j]
    |   v ← LISB(i, j+1)
    |   T[i,j] ← v
    |   return v
    else
    |   skip ← LISB(i, j+1)
    |   take ← LISB(j, j+1) + 1
    |   T[i,j] ← max{skip, take}
    |   return T[i,j]
```

So, now, we're doing them on the fly. But, can we be more direct & know they are computed when we go to use them, thus using the table T and no recursions.

↑ Quest 1

Quest 2 ↙

Why do we know such an order exists?
ie, why is there not a circular dependency?

Quest 2 Define a decrementing fen

$$d : \mathcal{S} \longrightarrow \mathbb{N}$$

$$[i, j, n] \longmapsto 2n - i - j$$

Quest 1