Dynamic Programming — Recursion without repetition iteratively!

String splitting

string $\longrightarrow$ sequence of "words" oracle IsWord

↳ empty or

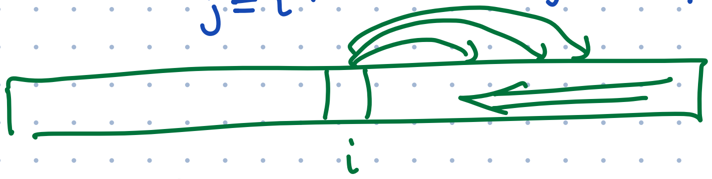
word + seq of words

$\boxed{\text{my job}}$ $\boxed{\text{recursion fairy's job}}$

Splittable(i) = True iff A[i..n] can be split

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

SplitTable



O(n) space

```
For i ← n down to 1
    For j ← i to n
        ⋮
```

$O(n^2)$ time

Longest Increasing Subsequence

incr sequence = empty

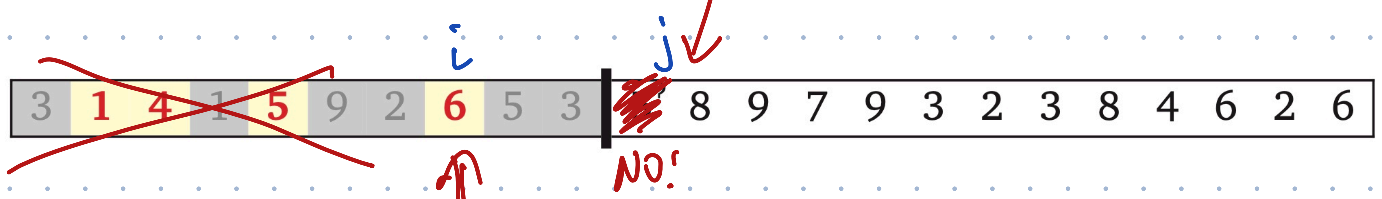
or number + incr seq
 x $\qquad\qquad$ > x

incr seq > x = empty

or $\boxed{\text{number} > x}$ + $\boxed{\text{incr seq} > y}$
 y

$\boxed{\text{my job}}$ $\boxed{\text{rec fairy's job}}$

Is this in LIS?



LIS bigger(i,j) =

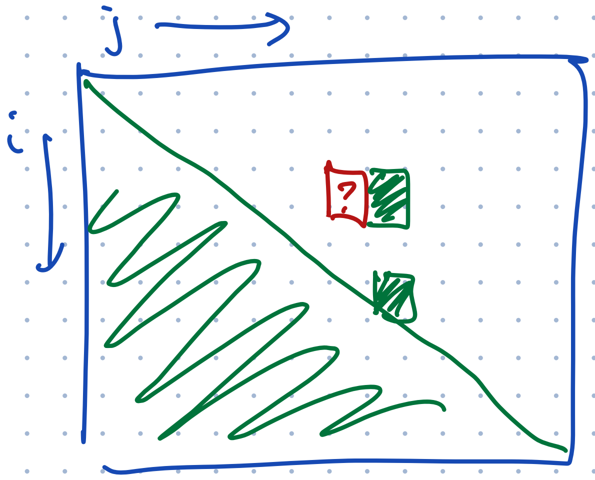
Length of Longest increasing subsequence of A[j..n] all bigger than A[i]

Is A[j] in this subsequence?

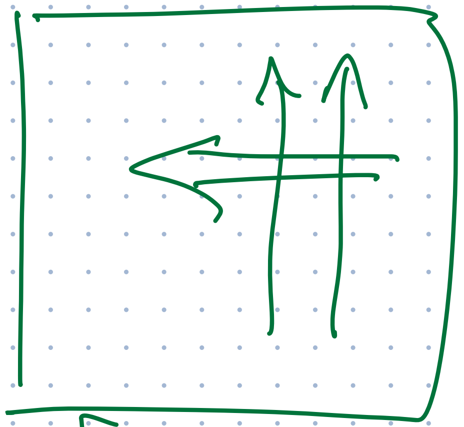
$$
LISbigger(\underline{i,j}) = \begin{cases} 0 & \text{if } j > n \\ LISbigger(\underline{i, j+1}) \quad \text{NO} & \text{if } A[i] \geq A[j] \\ \max \begin{Bmatrix} LISbigger(\underline{i, j+1}) \\ 1 + LISbigger(\underline{j, j+1}) \end{Bmatrix} & \begin{matrix} \text{NO} \\ \text{otherwise} \\ \text{YES} \end{matrix} \end{cases}
$$

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

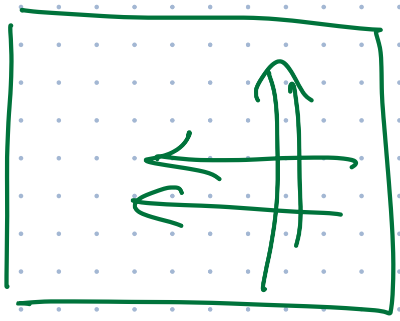
```
LIS($A[1..n]$):
    $A[0] \leftarrow -\infty$
    return LISBIGGER($0, 1$)
```

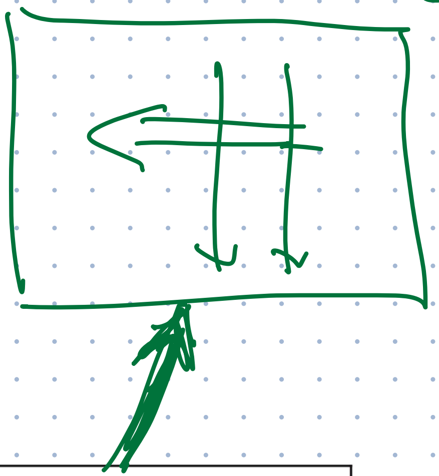


LIS bigger



for j ← n to 1
for i ← n to j





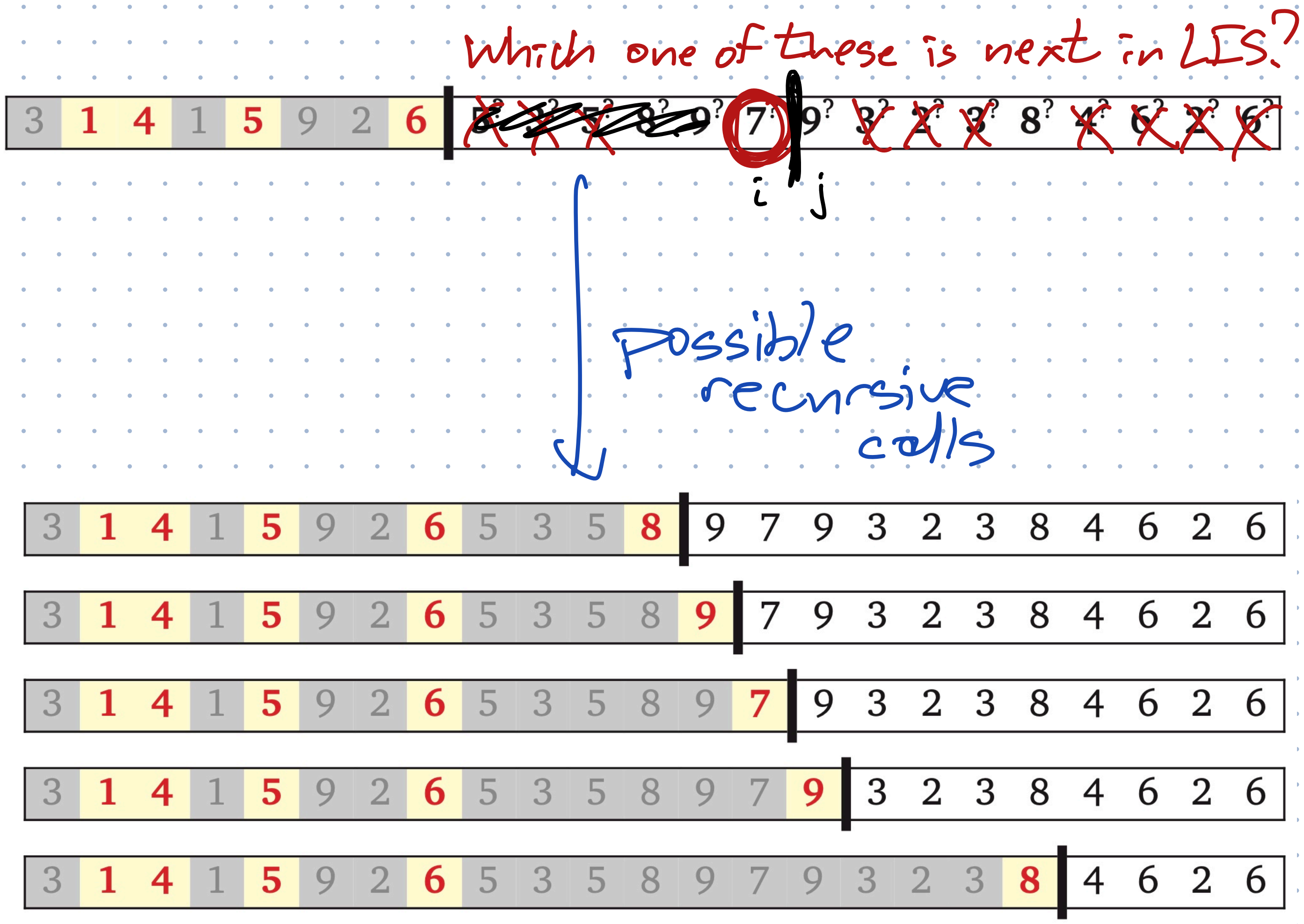
$\boxed{O(n^2)\text{ time}}$

```
FastLIS($A[1..n]$):
    $A[0] \leftarrow -\infty$                      ⟨⟨Add a sentinel⟩⟩
    for $i \leftarrow 0$ to $n$                    ⟨⟨Base cases⟩⟩
        $LISbigger[i, n+1] \leftarrow 0$
    for $j \leftarrow n$ down to 1
        for $i \leftarrow 0$ to $j-1$              ⟨⟨...or whatever⟩⟩
            $keep \leftarrow 1 + LISbigger[j, j+1]$
            $skip \leftarrow LISbigger[i, j+1]$
            if $A[i] \geq A[j]$
                $LISbigger[i, j] \leftarrow skip$
            else
                $LISbigger[i, j] \leftarrow \max\{keep, skip\}$
    return $LISbigger[0, 1]$
```

LISbigger(i, j) = LIS of A[j..n] all > A[i]

But j is always i+1!

LISFirst(i) = LIS A[i..n] starting with A[i]



$\Rightarrow$ LIS First(i) = LIS A[i..n] starting with A[i]

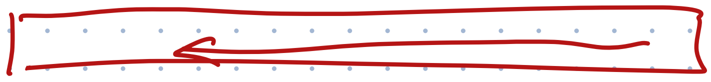
$\rightarrow$ $\max \emptyset = 0$

$$\Rightarrow \quad \mathit{LISfirst}(i) = 1 + \underline{\max \left\{ \mathit{LISfirst}(j) \;\middle|\; j > i \text{ and } A[j] > A[i] \right\}}$$

```
LISFIRST(i):
    best ← 0
    for j ← i + 1 to n
        if A[j] > A[i]
            best ← max{best, LISFIRST(j)}
    return 1 + best
```

$\rightarrow$

```
LIS(A[1 .. n]):
    A[0] ← −∞
    return LISFIRST(0) − 1
```

1d array  $O(n^2)$ time

```
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⟩⟩
```

Patience Sorting

input = sequence of cards
we're going to arrange in piles

~~3~~ ~~1~~ ~~4~~ ~~5~~ ~~9~~ ~~2~~ ~~6~~ ~~3~~ ~~8~~
~~7~~

$$\begin{tikzcd}
\infty & 3 \arrow[l] & 4 \arrow[l] & 5 \arrow[l] & 9 \arrow[l] & 7 \arrow[dl] & 8 \arrow[l] \\
 & 1 \arrow[ul] & 2 \arrow[l] & & 6 \arrow[ul] & &
\end{tikzcd}$$

$\boxed{O(n \log n)}$

place each card $A[i]$ on
leftmost pile whose
top card is $> A[i]$

#piles = LIS