Dynamic Programming

Lecture13

Fibonacci

- Fibonacci Numbers (circa 13 th century)

- $F_n =$

0 if n=0
1 if n=1
$F_{n-1}+F_{n-2}$ o/w

Given n, how long does it take to compute F_n?

Fibonacci

- Translates line by line to code:

$$
\begin{array}{|l|}
\hline
\underline{\textsc{RecFibo}(n)\text{:}} \\
\quad \text{if } (n < 2) \\
\qquad\qquad \text{return } n \\
\quad \text{else} \\
\qquad\qquad \text{return } \textsc{RecFibo}(n-1) + \textsc{RecFibo}(n-2) \\
\hline
\end{array}
$$

We will move from mathematical function format to recursive program a lot!

Fibonacci

- Translates line by line to code:

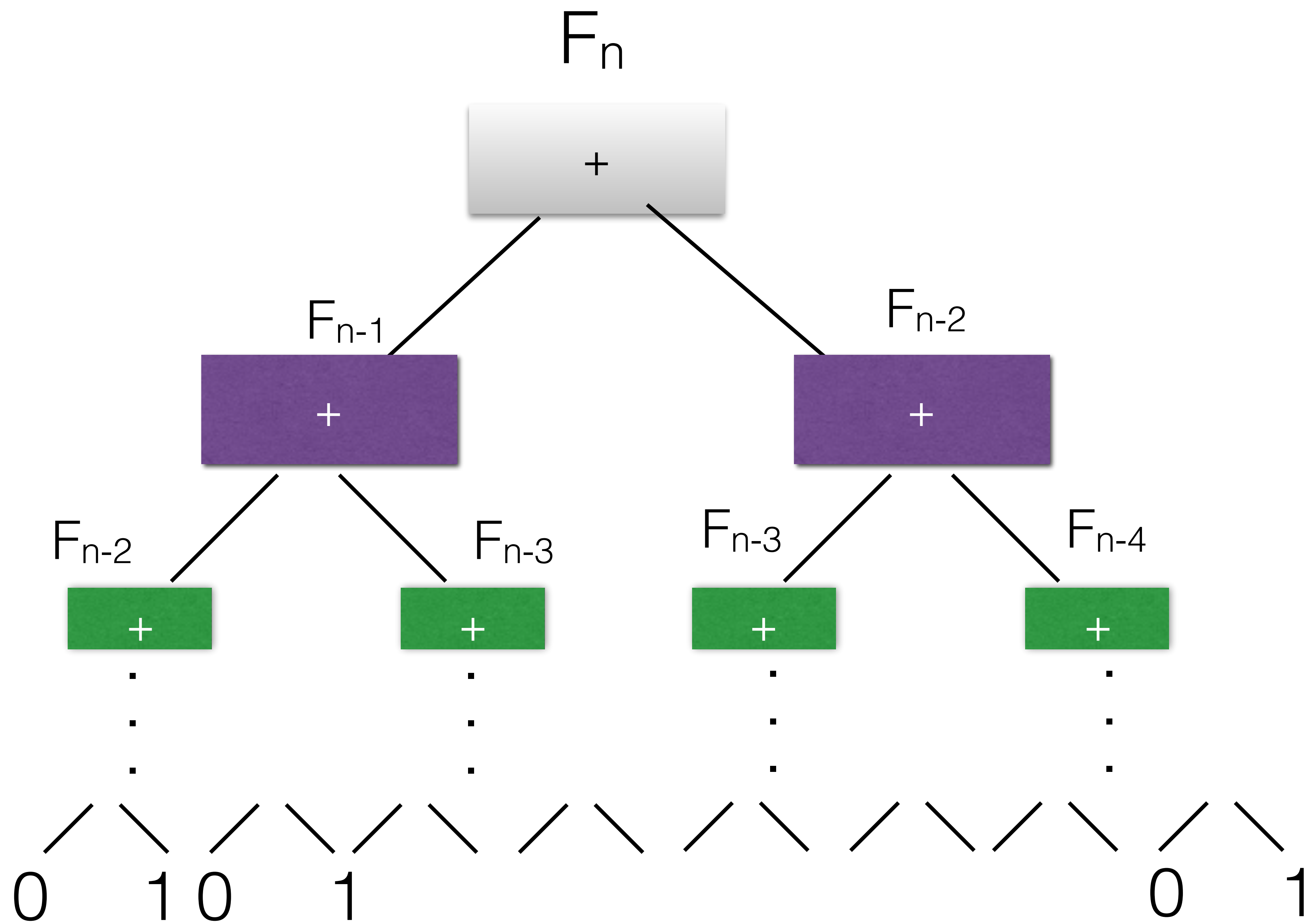
```
RecFibo(n):
    if (n < 2)
        return n
    else
        return RecFibo(n − 1) + RecFibo(n − 2)
```

Running time? (backtracking recurrence)

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

$=\Theta(F_n) = \Theta(1.618^n) = \Theta(((\sqrt{5}+1)/2)^n)$

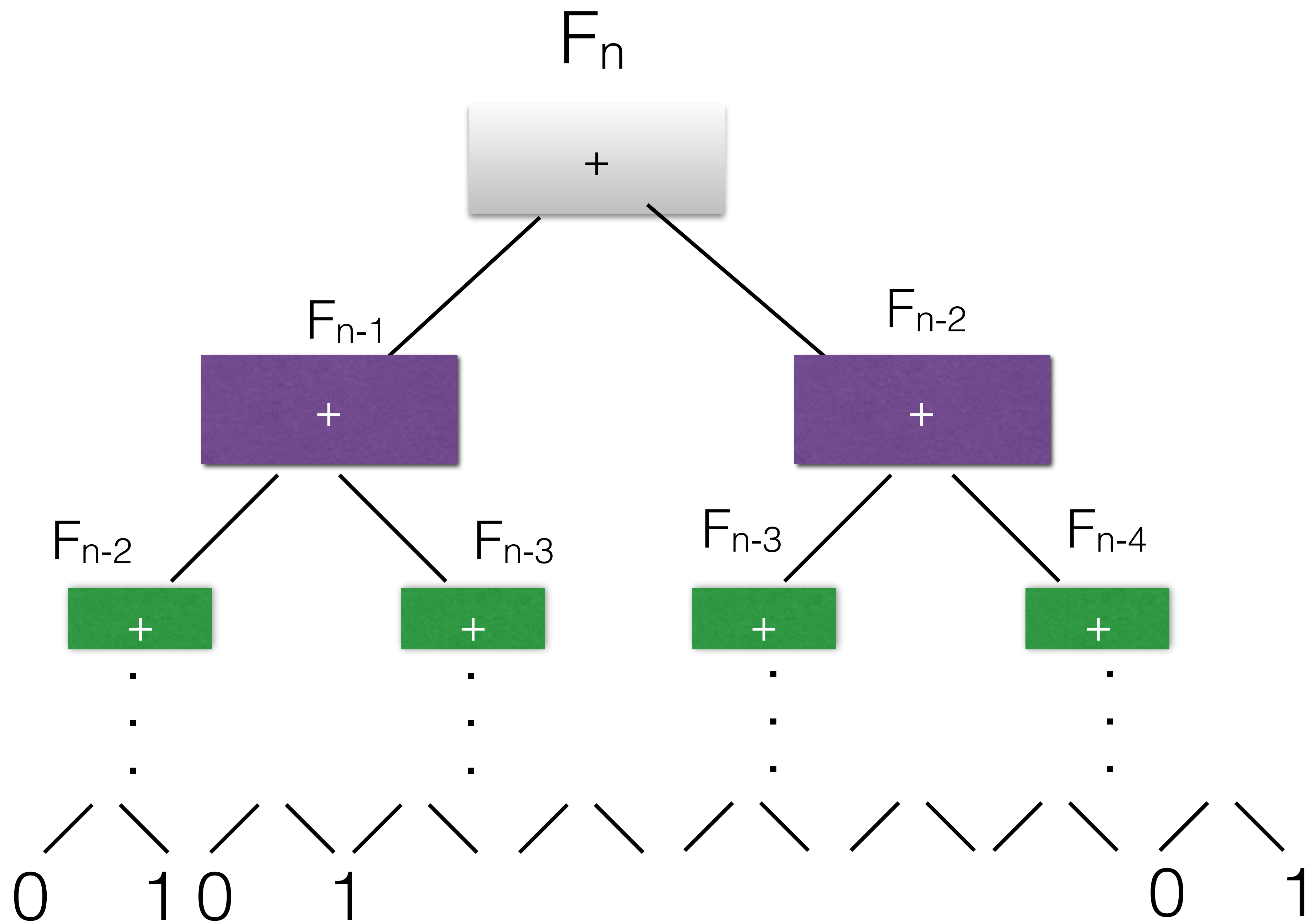
Running time via Rec Tree



Leaves are always 0 or 1.
How many 1's? How many 0s?

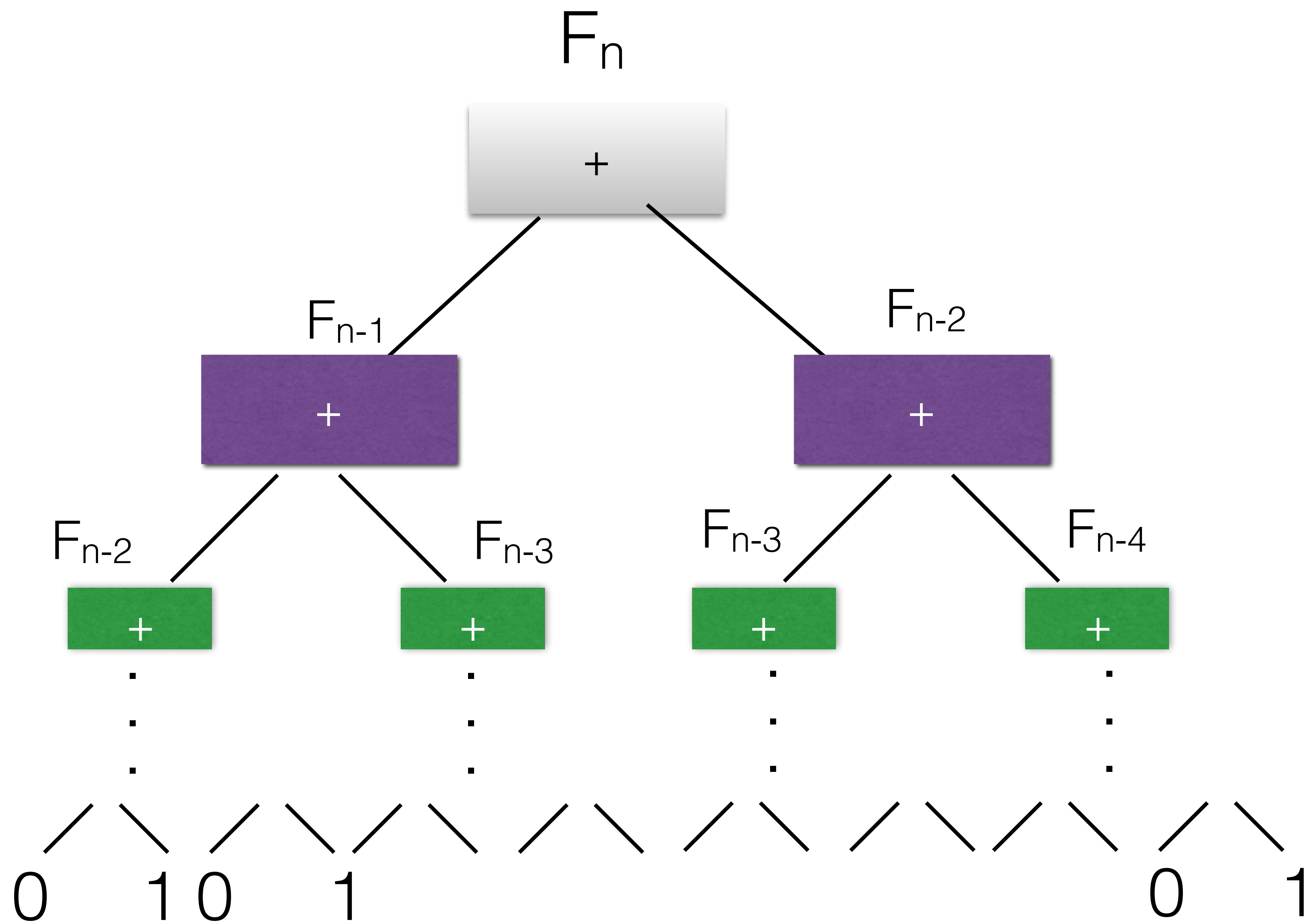
There are F_n 1s and F_{n-1} 0s
F_{n+1} leaves total!

Running time via Rec Tree



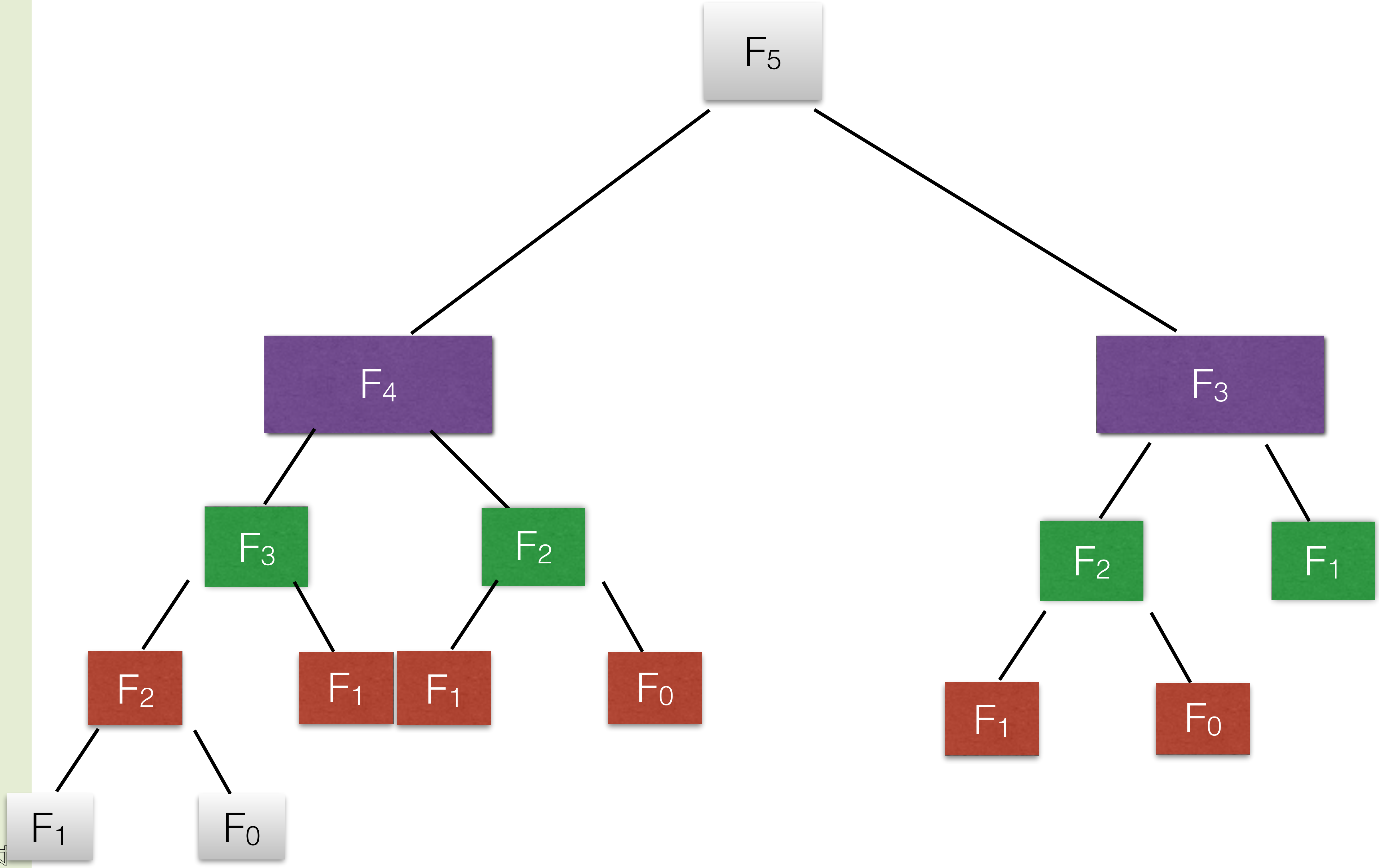
How many intermediate nodes does a full binary tree with m leaves have?

Running time via Rec Tree

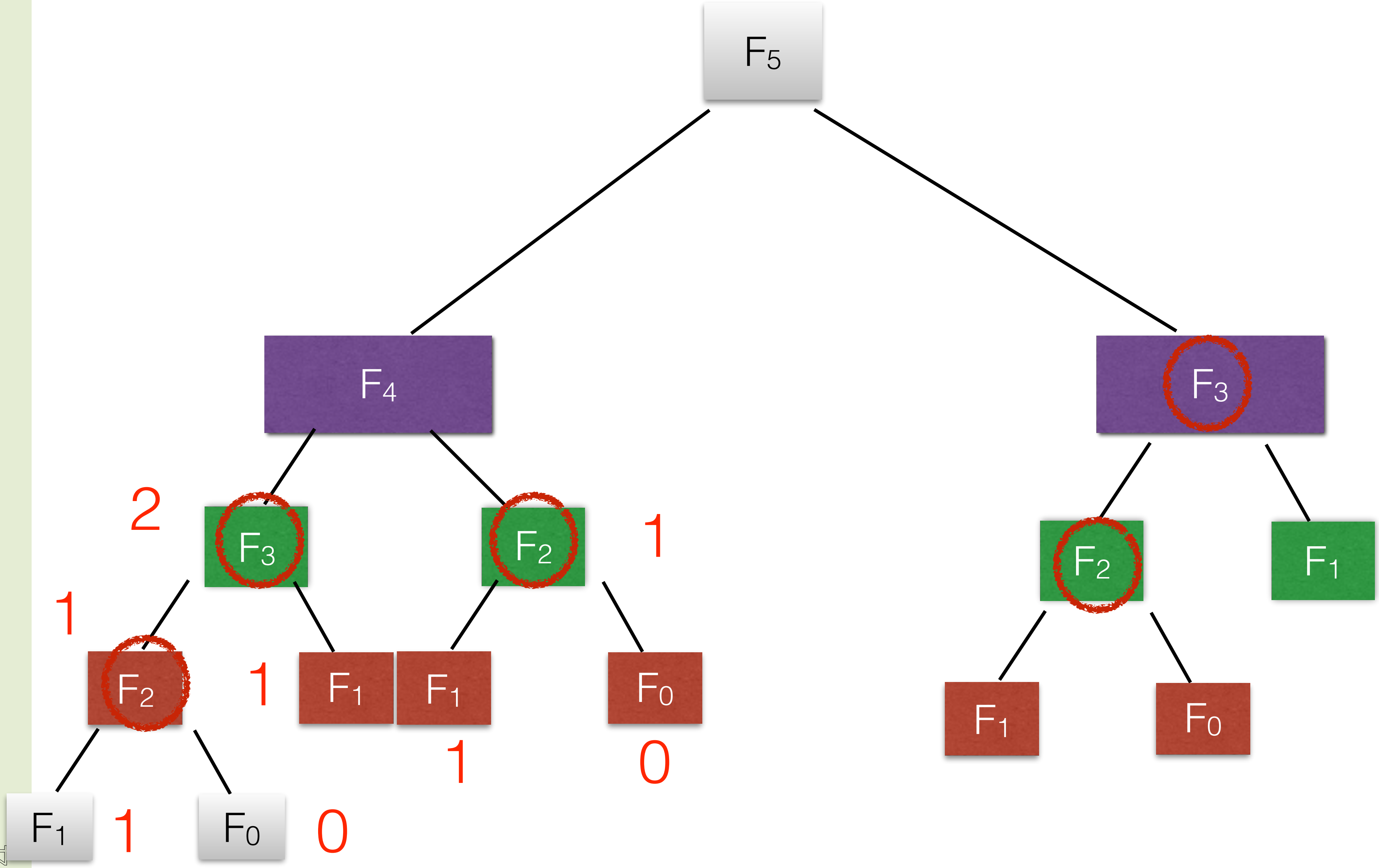


$2F_{n+1}$ -1 nodes (additions)

Running time via Rec Tree

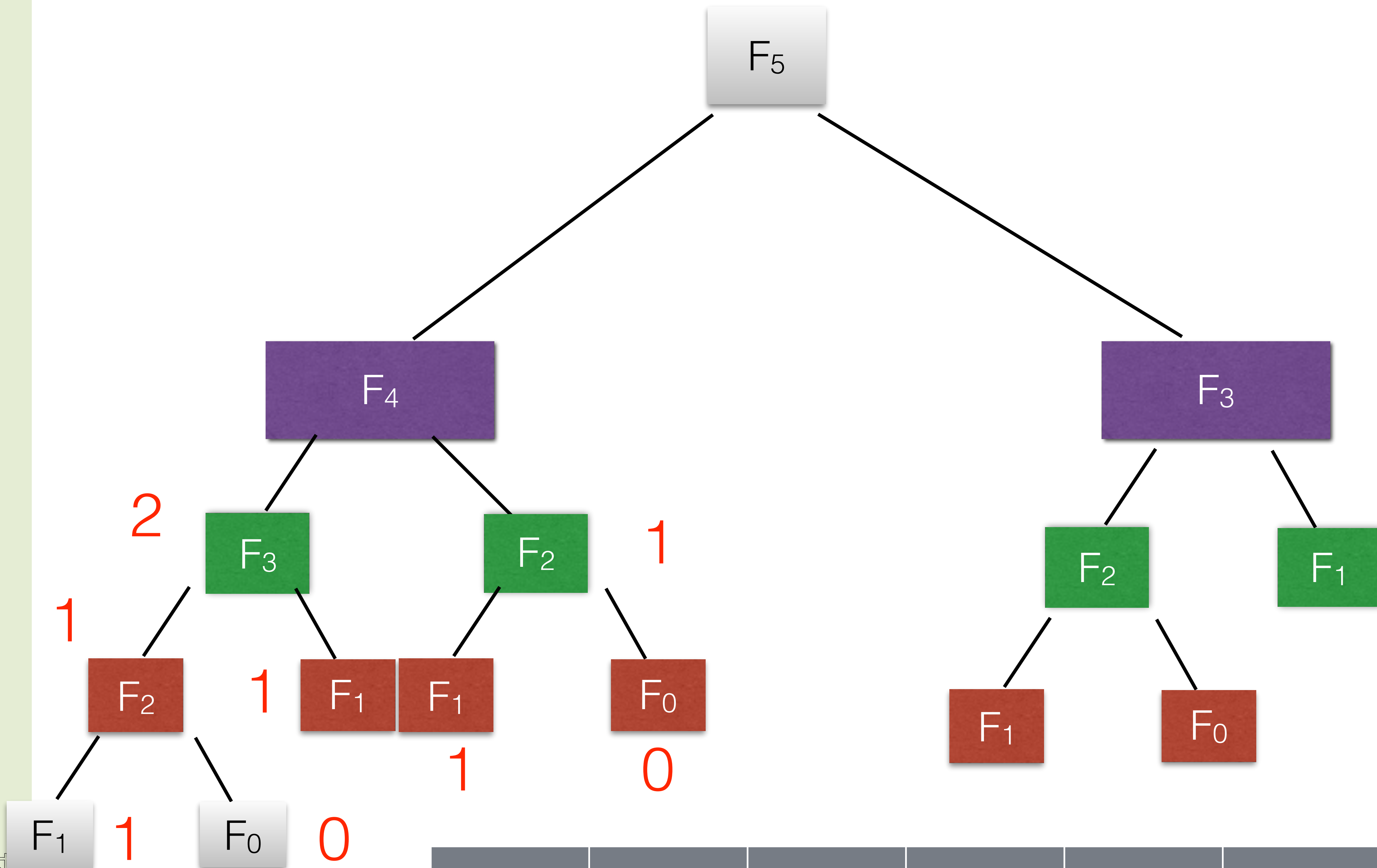


Running time via Rec Tree



Keep an array to remember the previous values!

Running time via Rec Tree

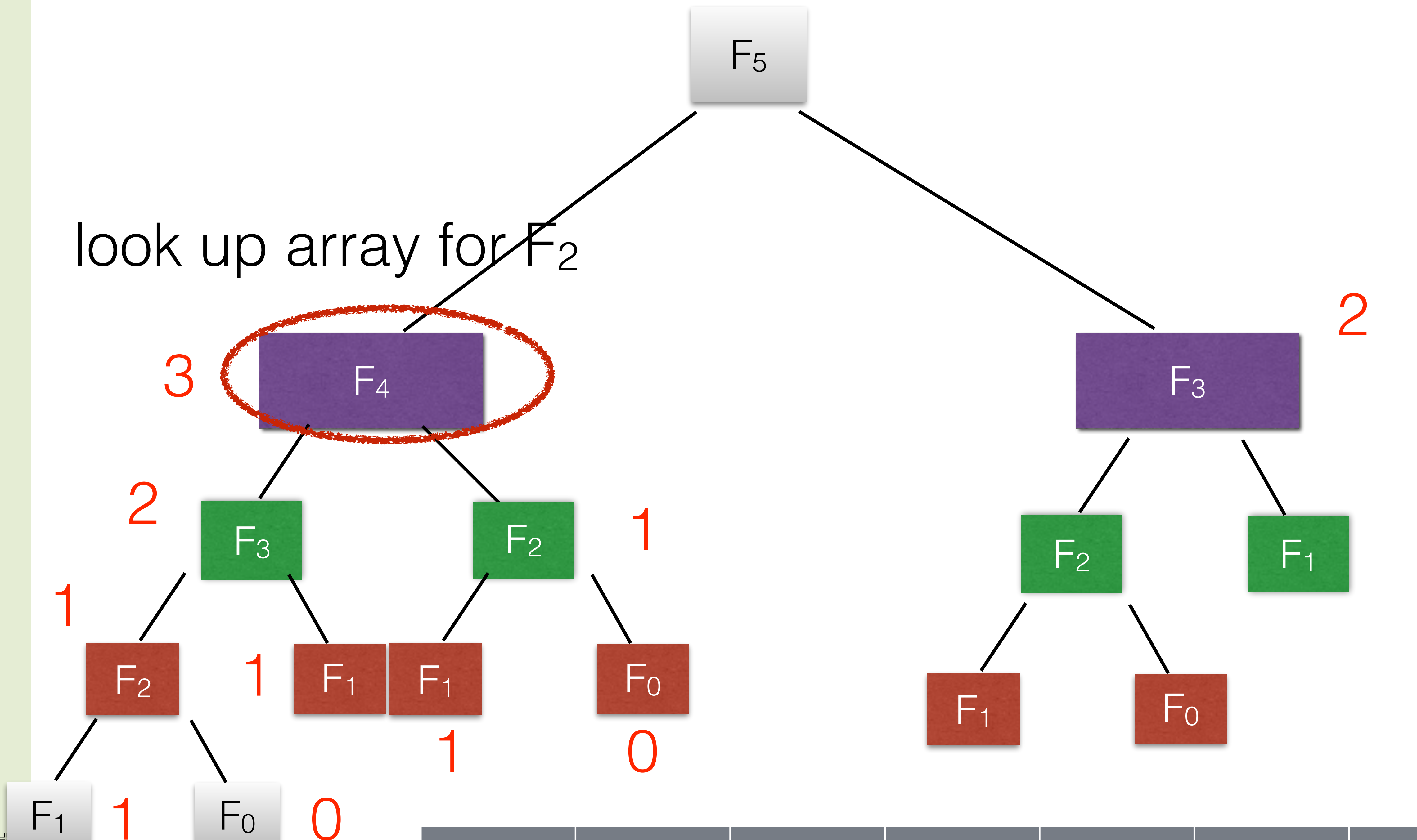


F

0	1	2	3	4	5	…
0	1	1	2			

Running time via Rec Tree

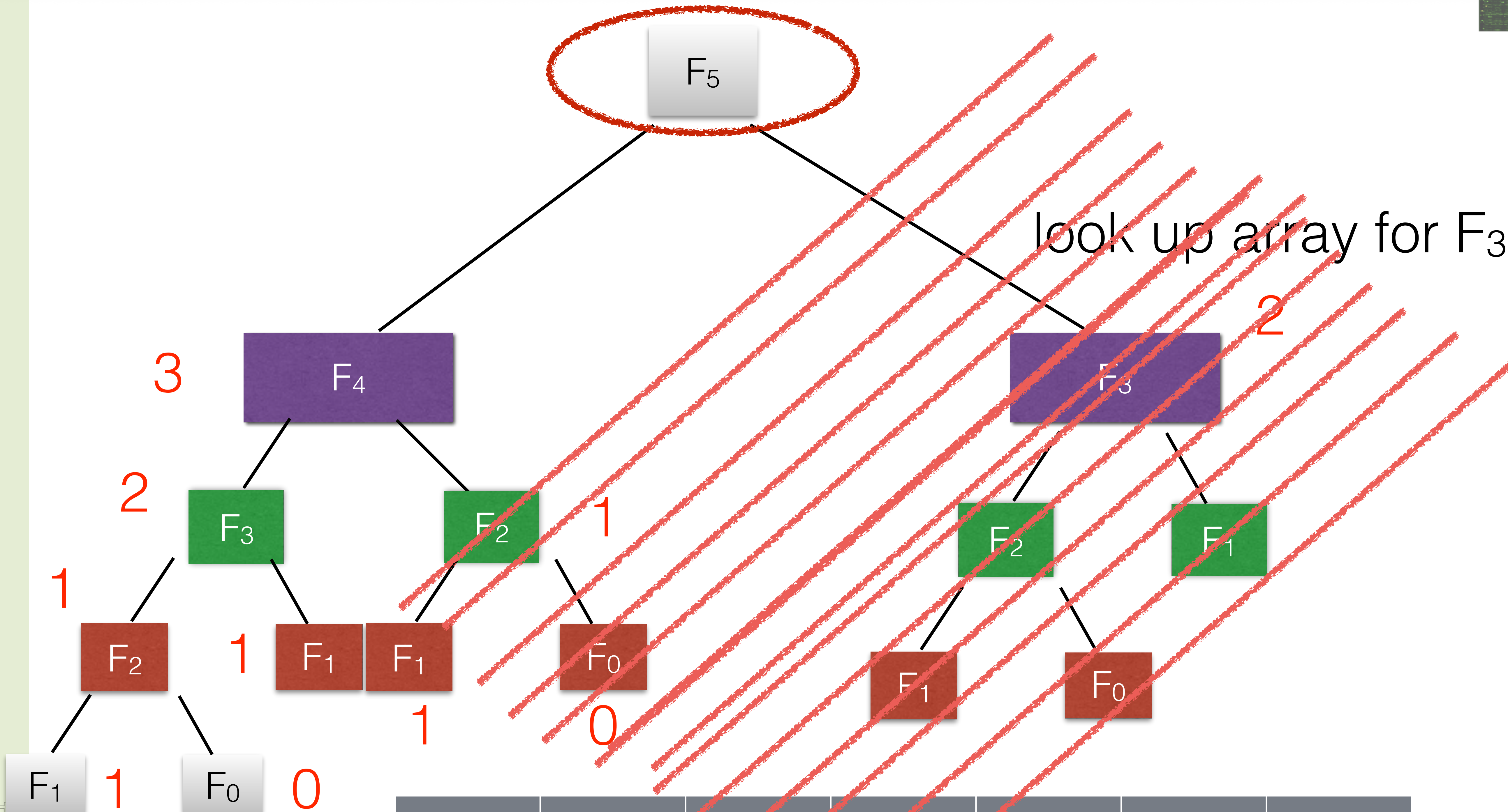
look up array for F_2



F

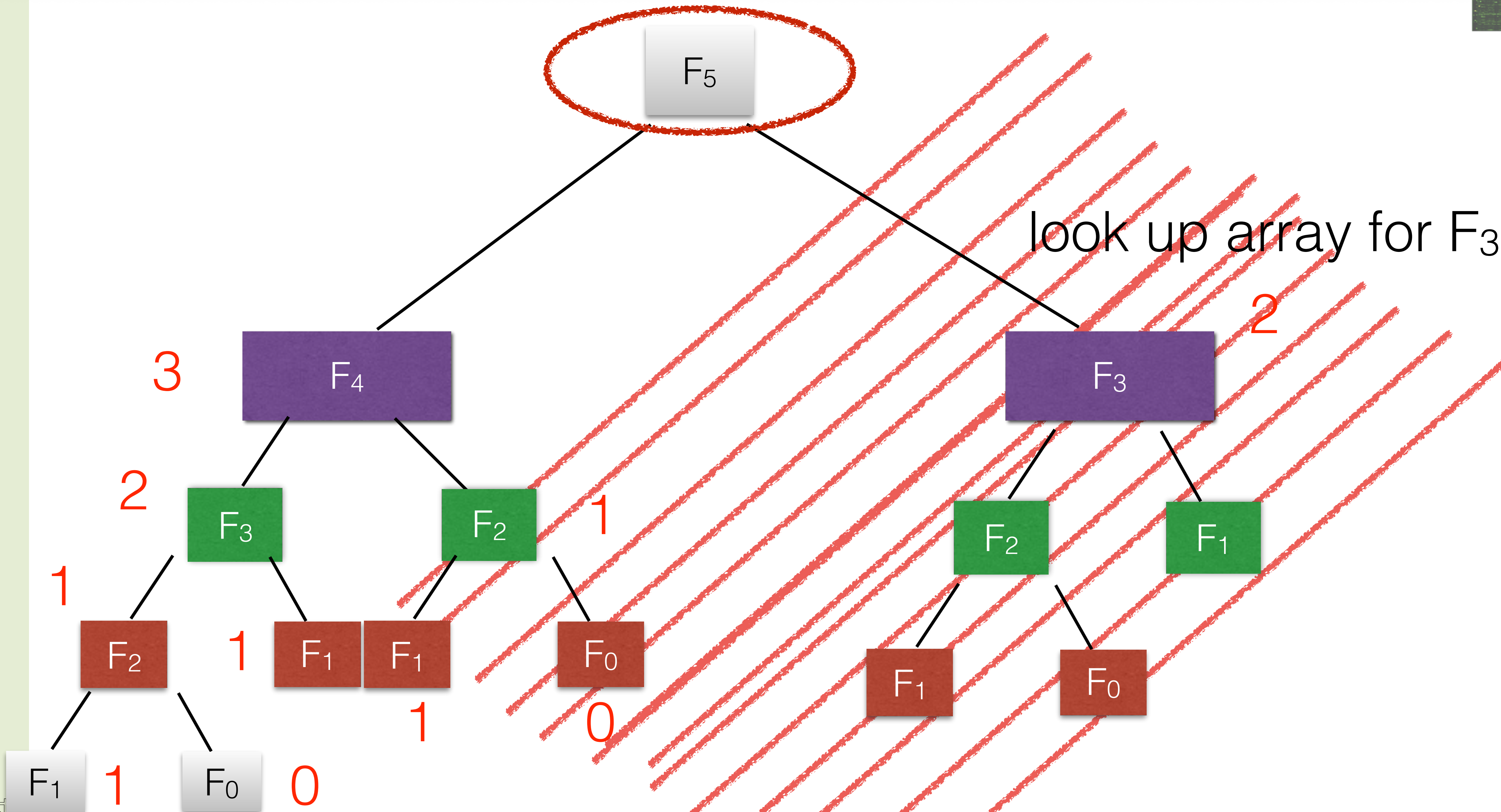
0	1	2	3	4	5	...
0	1	1	2	3		

Running time via Rec Tree



F	0	1	2	3	4	5	...
	0	1	1	2	3	5	

Running time via Rec Tree



Memoization= when I look at the table to see the values I computed before

$\underline{\text{MEMFIBO}(n):}$

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

Given any recursive backtracking algorithm,
you can add memoization and will save time, provided the subproblems repeat

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

How many times did I have to call the recursive function?
exponential!

How many different values did I have to compute?
O(n)!

Memoization decreases running time : performs only O(n) additions, exponential improvement

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

Memoized algorithm fills in the table from left to right.
Why not just do that?

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

Memoized algorithm fills in the table from left to right.
Why not just do that?

We get an iterative algorithm

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

- Clear that the number of additions it does it O(n).
- In practice this is faster than memoized algo, cause we don’t use stack/ look up the table etc.

ITERFIBO(n):

$F[0] \leftarrow 0$

$F[1] \leftarrow 1$

for $i \leftarrow 2$ to n order

    $F[i] \leftarrow F[i-1] + F[i-2]$

return $F[n]$

- Structure mirrors the recurrence
- Only subtle thing is that we want to fill in the array in increasing order.

$$
\begin{array}{l}
\underline{\text{ITERFIBO}(n):} \\
\quad F[0] \leftarrow 0 \\
\quad F[1] \leftarrow 1 \\
\quad \text{for } i \leftarrow 2 \text{ to } n \\
\quad\quad F[i] \leftarrow F[i-1] + F[i-2] \\
\quad \text{return } F[n]
\end{array}
$$

order

- This is Dynamic Programing Algorithm!
- Dynamic Programming= pretend to do Memoization but do it on purpose

- Memoization: accidentally use something efficient
- Backwards induction =Dynamic Programming

Dynamic Programming

- Dynamic programming is about smart recursion.
- Not about filling out tables!
- How do I solve the problem, how do I not repeat work, then how to fill up my data structure.

Dynamic Programming

- How can I speed up my algorithm?

$\underline{\text{ITERFIBO}(n)\text{:}}$

$F[0] \leftarrow 0$
$F[1] \leftarrow 1$
for $i \leftarrow 2$ to n
    $F[i] \leftarrow F[i-1] + F[i-2]$
return $F[n]$

- I only need to keep my last two elements of the array.
- Even more efficient algorithm

Dynamic Programming

- How can I speed up my algorithm?

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

- I only need to keep my last two elements of the array.
- Even more efficient algorithm
- Where is the recursion?

Dynamic Programming

- How can I speed up my algorithm?

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

- I only need to keep my last two elements of the array.
- Even more efficient algorithm
- Where is the recursion?
- Saves space, sometimes important

Dynamic Programming

- How can I speed up my algorithm?

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

- Is this the fastest Algorithm for Fibonacci?

Dynamic Programming

- How can I speed up my algorithm?

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

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} y \\ x + y \end{bmatrix}$$

This matrix vector multiplication does exactly the same thing as one iteration of the loop!

What to do to compute the nth Fibonacci number?

Dynamic Programming

- How can I speed up my algorithm?

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

Compute the nth power of the matrix.

- With repeated squaring, O(logn) multiplications
- Compute F_n in O(logn) arithmetic operations
- Double exponential speedup!

Dynamic Programming

- How can I speed up my algorithm?

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

Compute the nth power of the matrix.

- But how many bits is the nth Fibonacci number?
- O(n)!
- Can’t perform arbitrary precision arithmetic in constant time

Longest Increasing Subsequence (LIS)

- 3 1 4 1 5 9 2 6 5 3 8 2 7 9 4 6 1 0 4 8

Longest Increasing Subsequence (LIS)

- 3 1 4 1 5 9 2 6 5 3 8 2 7 9 4 6 1 0 4 8
- LIS(A[1…n],p) = length of LIS of A[1…n] where everything is bigger than p

Longest Increasing Subsequence (LIS)

- 3 1 4 1 5 9 2 6 5 3 8 2 7 9 4 6 1 0 4 8

- LIS(A[1…n],p)=

$$\begin{cases} 0 \text{ if } n=0 \\ \text{LIS(A[2…n],p) if } A[1] \le p \\ \text{MAX \{ LIS(A[2…n],p)} \\ \quad 1+\text{LIS(A[2…n],A[1])\}} \end{cases}$$

Longest Increasing Subsequence (LIS)

- $\text{LIS}(A[1\ldots n],p) = \begin{cases} 0 \text{ if } n=0 \\ \text{LIS}(A[2\ldots n],p) \text{ if } A[1]\leq p \\ \text{MAX}\{\, \text{LIS}(A[2\ldots n],p) \\ \qquad 1+\text{LIS}(A[2\ldots n],A[1])\} \end{cases}$

- The argument p is always either $-\infty$ or and element of the array A
- Add $A[0]=-\infty$
- We can identify any recursive subproblem with two array indices.
- LIS(i,j) = length or LIS of A[j…n] with all elements larger than A[i]

Longest Increasing Subsequence (LIS)

For i<j

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

- LIS(i,j) = length or LIS of A[j…n] with all elements larger tha A[i]
- We want to compute LIS(0,1)
- Memoize? what data structure to use?
- Two dimensional Array **LIS[0…n,1…n+1]**

For i<j

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

j

i

	1	2	3	4				n+1
0								
1								
2								
3								
n								

For $i<j$

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

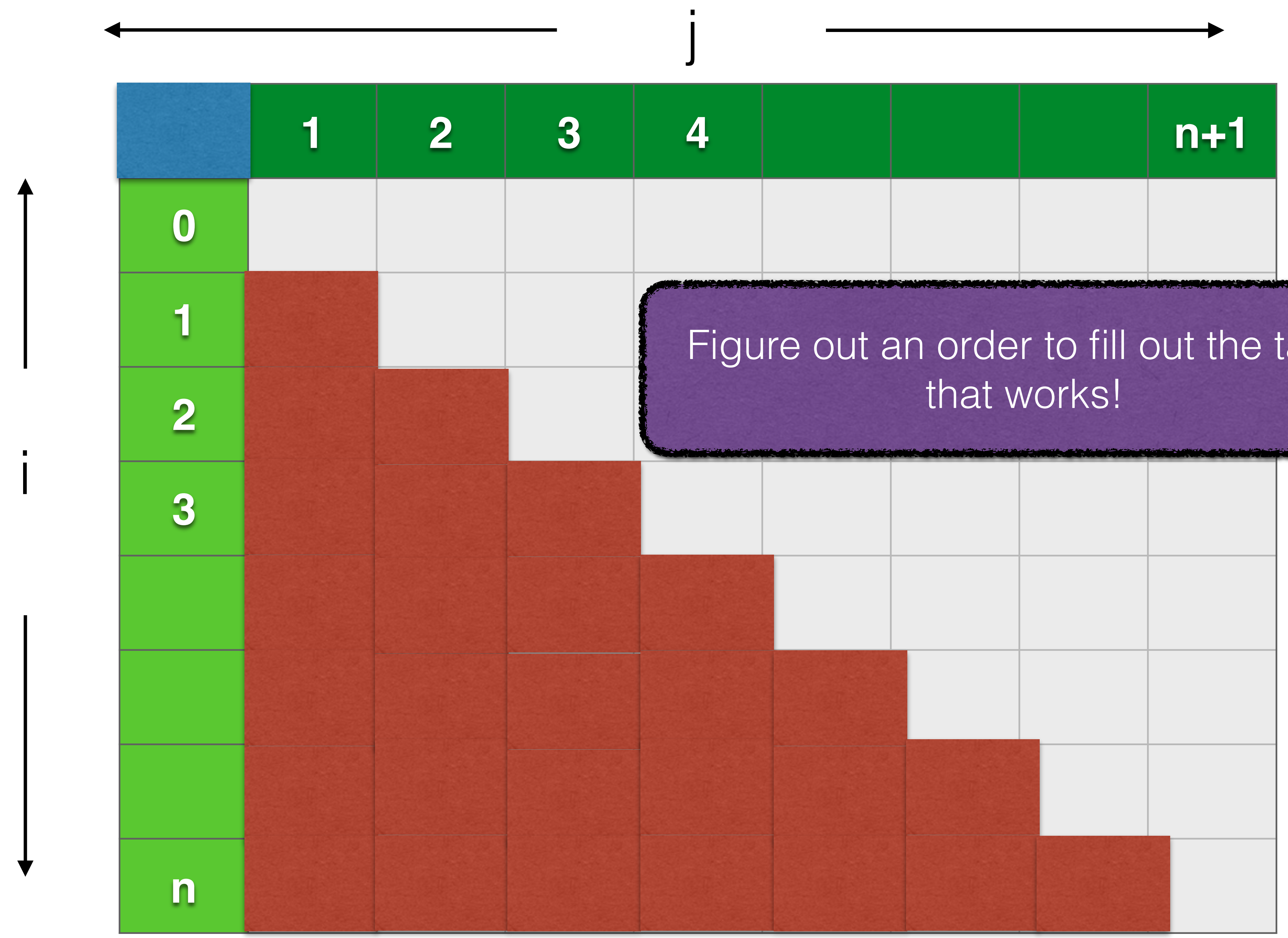
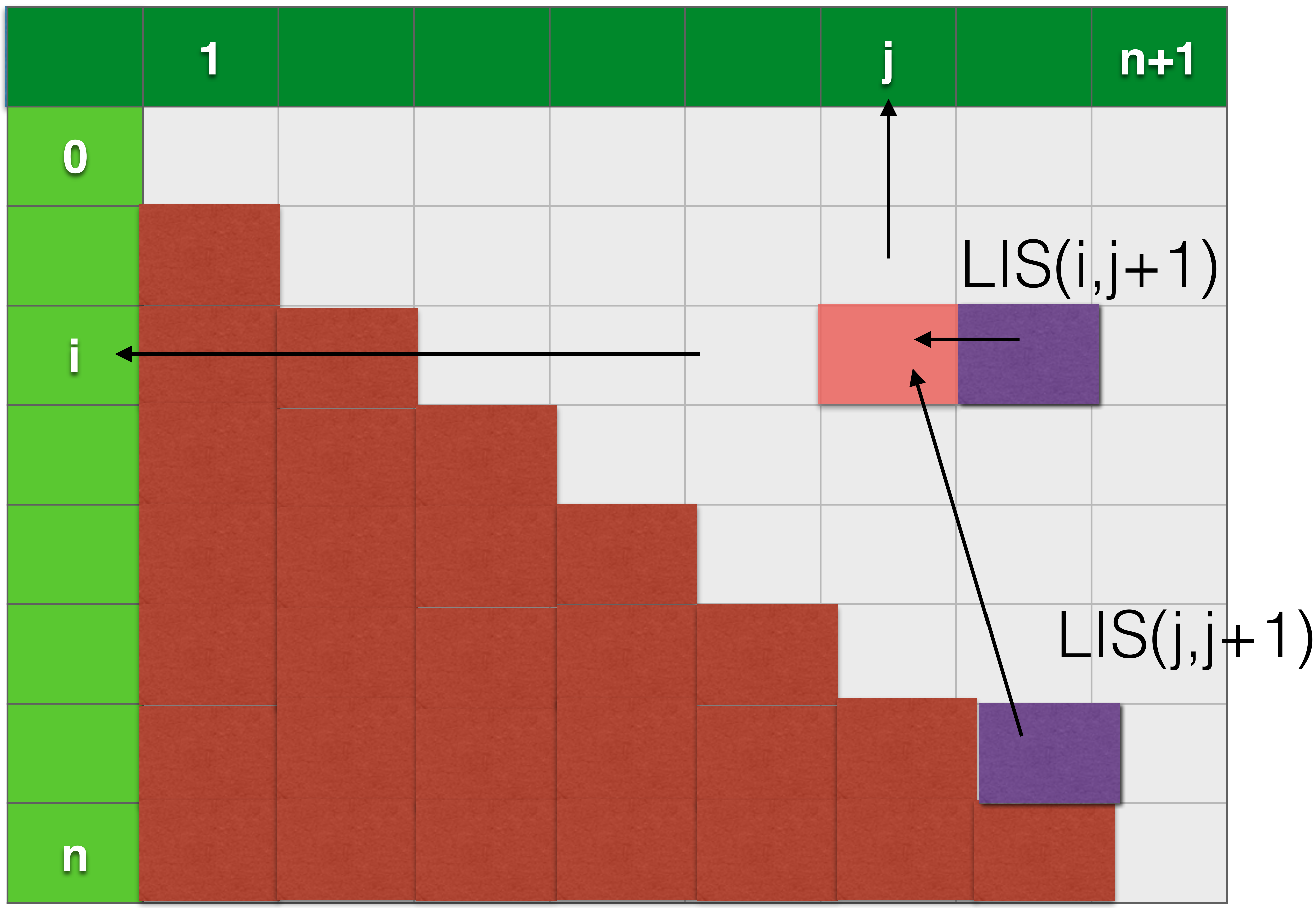


Figure out an order to fill out the table that works!

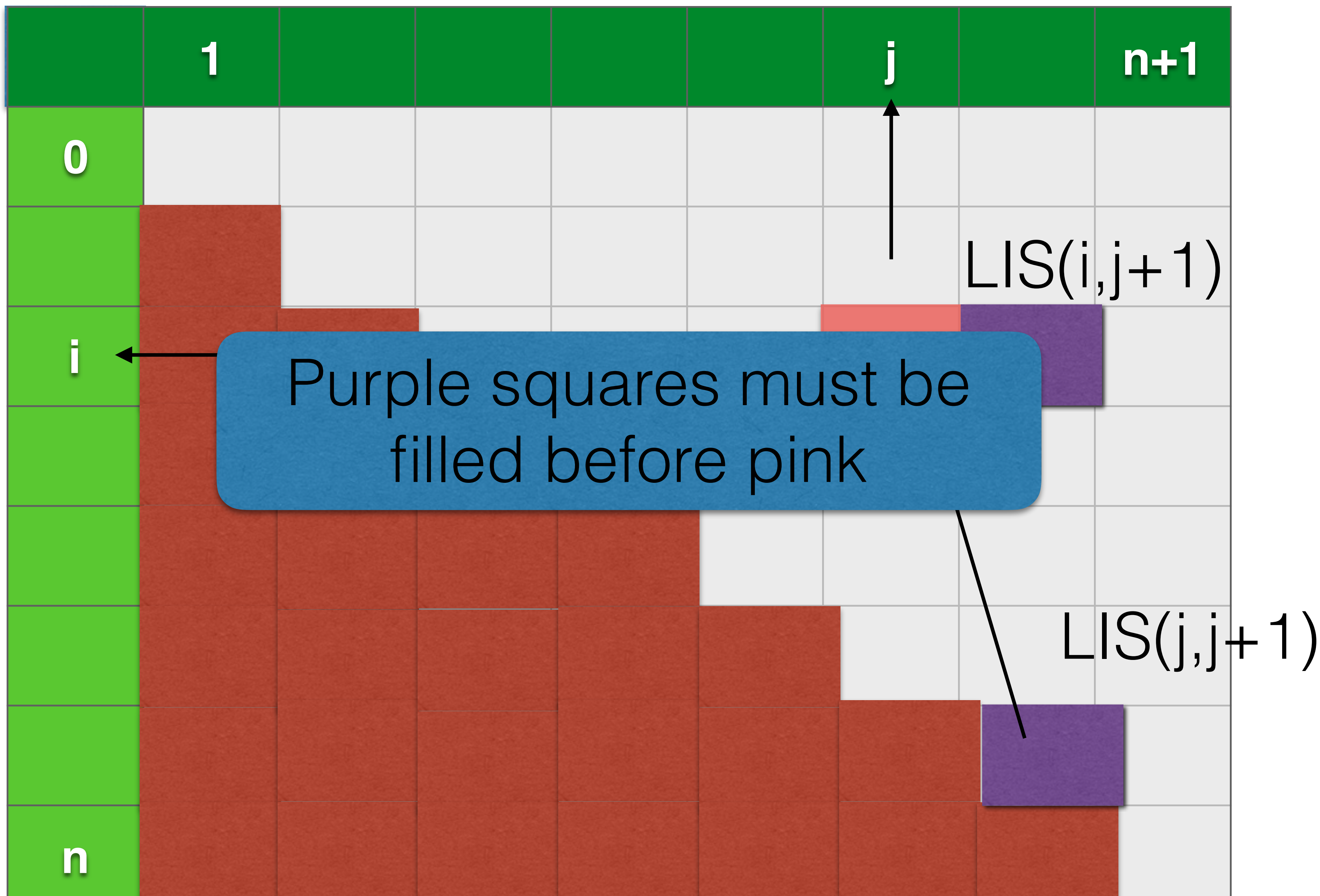
For i<j

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



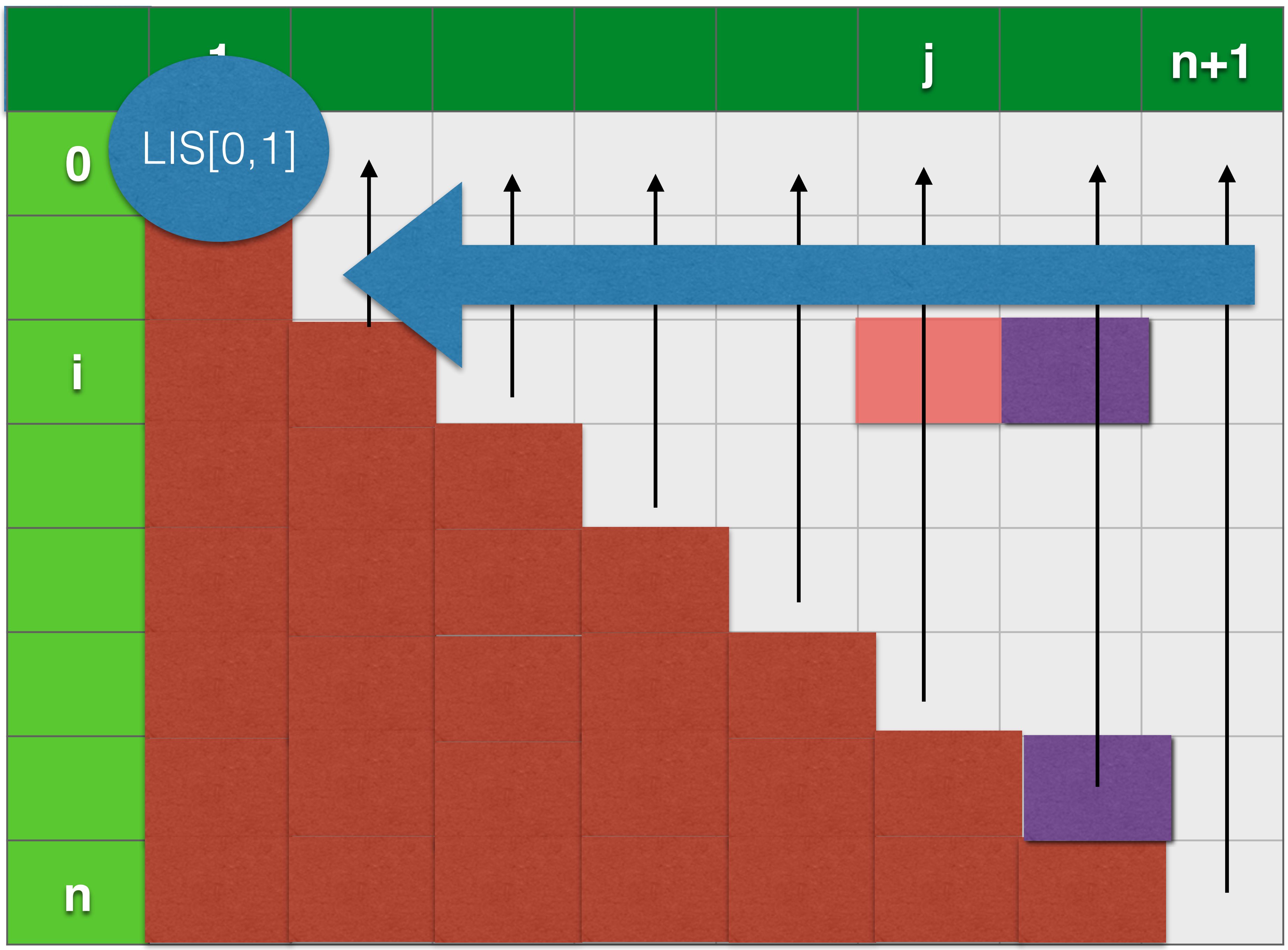
For i<j

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

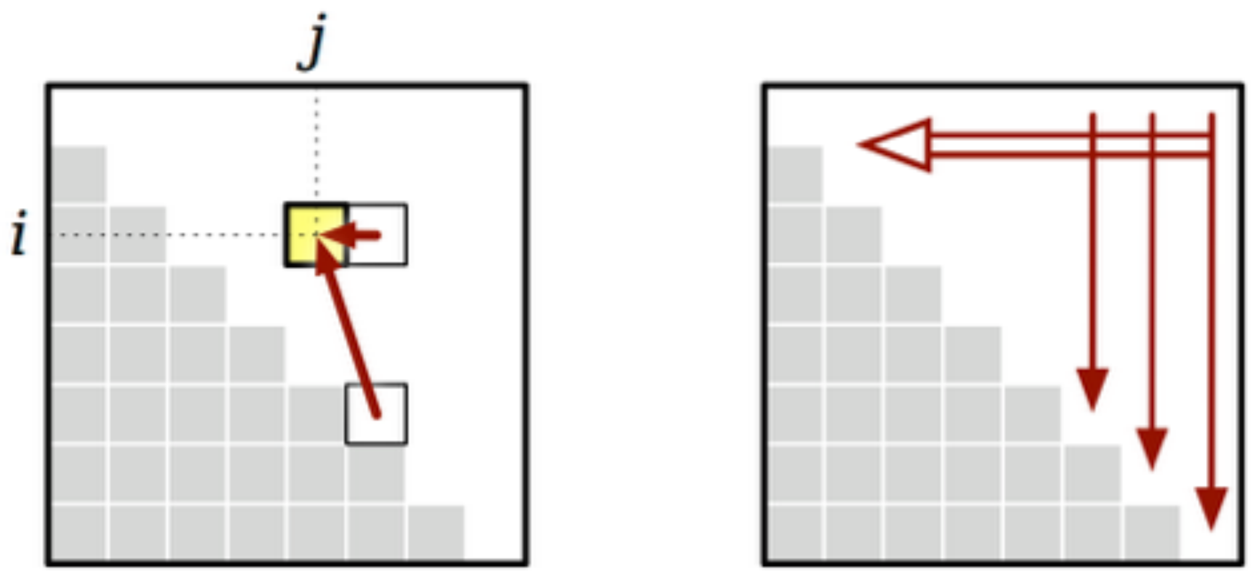


For i<j

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



Longest Increasing Subsequence (LIS)



doesn't matter what order I fill the columns in

```
LIS(A[1..n]):
  A[0] ← −∞                      ⟨⟨Add a sentinel⟩⟩
  for i ← 0 to n                 ⟨⟨Base cases⟩⟩
      LIS[i, n+1] ← 0

  for j ← n downto 1
      for i ← 0 to j−1
          if A[i] ≥ A[j]
              LIS[i, j] ← LIS[i, j+1]
          else
              LIS[i, j] ← max{LIS[i, j+1], 1 + LIS[j, j+1]}
  return LIS[0, 1]
```

Longest Increasing Subsequence (LIS)

- Running time?
- $O(n^2)$
- Two nested for loops
- How man values are there in the recurrence?

<u>LIS($A[1..n]$):</u>

$A[0] \leftarrow -\infty$ ⟨⟨*Add a sentinel*⟩⟩

for $i \leftarrow 0$ to n ⟨⟨*Base cases*⟩⟩

  $LIS[i, n+1] \leftarrow 0$

for $j \leftarrow n$ downto 1

  for $i \leftarrow 0$ to $j-1$

    if $A[i] \geq A[j]$

      $LIS[i, j] \leftarrow LIS[i, j+1]$

    else

      $LIS[i, j] \leftarrow \max\{LIS[i, j+1],\ 1 + LIS[j, j+1]\}$

return $LIS[0, 1]$

Longest Increasing Subsequence (LIS)

For i<j

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

- As general rule of thumb:
- # variables on the left =space $O(n^2)$ array for i,j taking n values each
- # variables on the right =time $O(n^2)$

Dynamic Programming

General Recipe for DP

- **Step 1**: Find Backtracking Recursive algorithm (e.g. for LIS we leveraged the recursive def. Either empty or there is something that comes first) (6 pts)

- **Step 2**: Identify the subproblems (e.g. indices i,j for LIS), need english description

- **Step 3**: Analyze time and space

- **Step 4**: Choose a memoization data structure (e.g. two dim array)

- **Step 5**: Find evaluation order (draw picture!!!)

Dynamic Programming

General Recipe for DP

- **Step 3**: Analyze time and space
- **Step 6**: write iterative pseudocode