$$F_n := \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$

```
RecFibo(n):
  if n = 0
      return 0
  else if n = 1
      return 1
  else
      return RecFibo(n − 1) + RecFibo(n − 2)
```

$T(0) = 1$

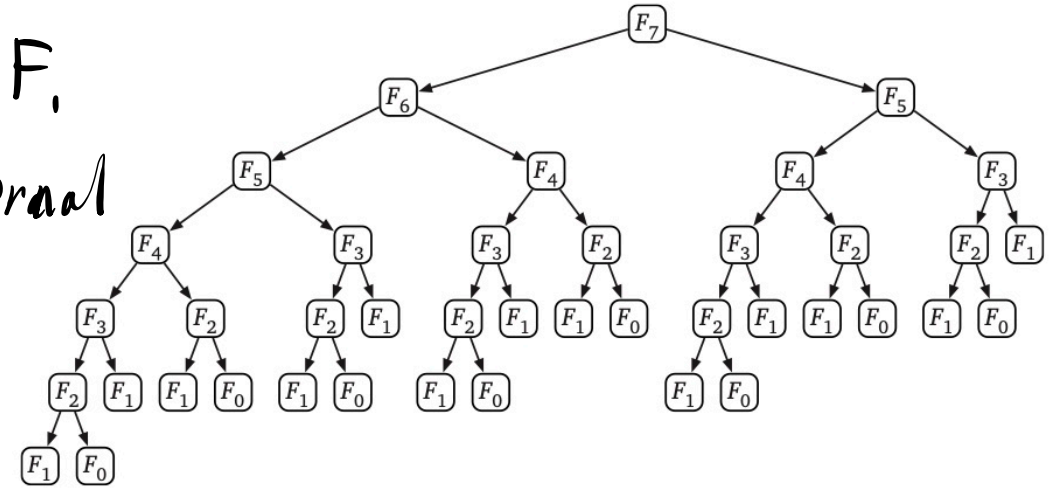
$T(1) = 1$

$T(n) = T(n-1) + T(n-2) + 1 \quad \text{for } n \geq 2$

$T(n) = 2F_{n+1} - 1 = \Theta(\phi^n) = O(1.62^n)$

F_n calls for F_1

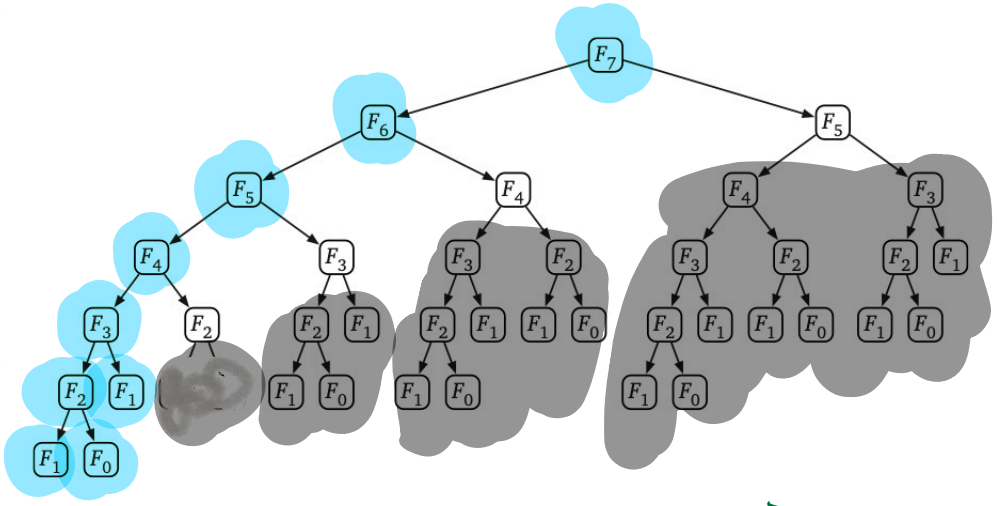
$\Rightarrow \geq F_n - 1$ internal nodes



Donald Michie: Memoization: Store results in a "global" data structure.

~~hash table~~? array! F[2...]

```
MemFibo(n):
  if n = 0
      return 0
  else if n = 1
      return 1
  else
      if F[n] is undefined
          F[n] ← MemFibo(n − 1) + MemFibo(n − 2)
      return F[n]
```



$O(n)$ ~~time~~ additions

F | | 2 | 3 | 5 | 8 | ...

0 1 2 3 4 5

F_n

n

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

$O(n)$ additions

(dynamic programming)

```
ITERFIBO2(n):
    prev ← 1   (F_{-1} = 1)
    curr ← 0
    for i ← 1 to n
        next ← curr + prev
        prev ← curr
        curr ← next
    return curr
```

storing $O(1)$ integers

$$\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} prev \\ curr \end{bmatrix} = \begin{bmatrix} curr \\ prev + curr \end{bmatrix} = \begin{bmatrix} curr \\ next \end{bmatrix}$$

$$\Rightarrow \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} F_{n-1} \\ F_n \end{bmatrix}$$

can compute with $O(\log n)$ 2×2 matrix mults.

$\Rightarrow F_n$ with $O(\log n)$ arithmetic operations

Take $O(n)$ time to compute (and write!)

$F_{n-2} + F_{n-1}$.

$\Rightarrow$ IterFibo takes $O(n^2)$ time.

For multiplying?: $T(n) = T(n/2) + M(n)$

↑ time to multiply n-digit numbers

Karatsuba: $O(n^{\log_2 3})$

or $O(n \log n)$ with fastest mults

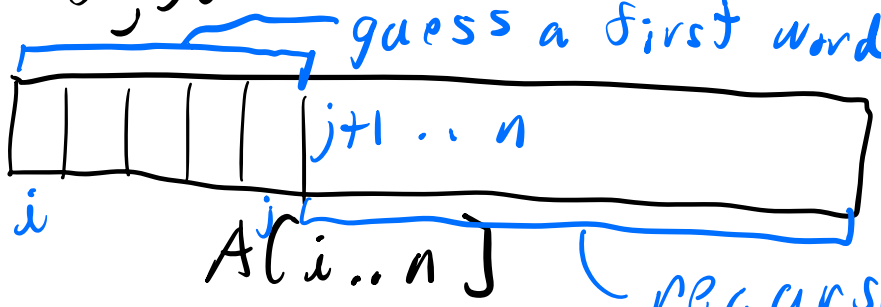
Text segmentation: Given A[1..n] of letters + a function IsWord(),

Is A the concatenation of words?

need Splittable (1)

Splittable(i) = True iff A[i..n] is the concatenation of words.

IsWord(i, j): True iff A[i..j] is a word.

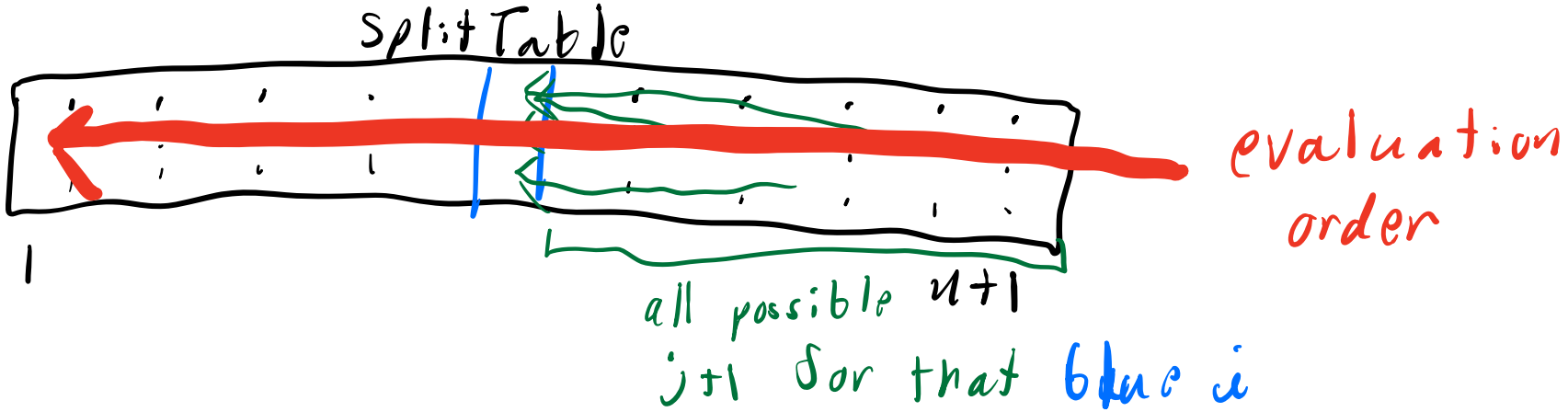


$$Splittable(i) = \begin{cases} \text{TRUE} & \text{if } i > n \\ \bigvee_{j=i}^{n} \big(IsWord(i,j) \wedge Splittable(j+1)\big) & \text{otherwise} \end{cases}$$

```
⟨⟨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
```

Need to know SplittableBlo(i) for $i \in \{1, \ldots, n+1\}$.

Memoize in SplitTable[1..n+1].




```
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]
```

How many calls to IsWord?

$O(n^2)$.

$O(n)$ subproblems.

$O(n)$ per

$= O(n^2)$ total