Recursion

Divide + Conquer $\quad T(n) = f(n) + T(\frac{n}{a}) + T(\frac{n}{b}) + T(\frac{n}{c})$ polynomial

Backtracking $\quad T(n) = f(n) + T(n-a) + T(n-b)$ exponential

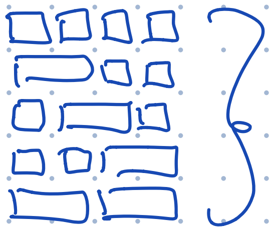
---

Pingala

prosody $\quad$ 200 BCE

short
long

4 beats



5

Virahanka $\quad M(n)$ = # meters last n beats

~ 700 CE

$M(1) = 1$

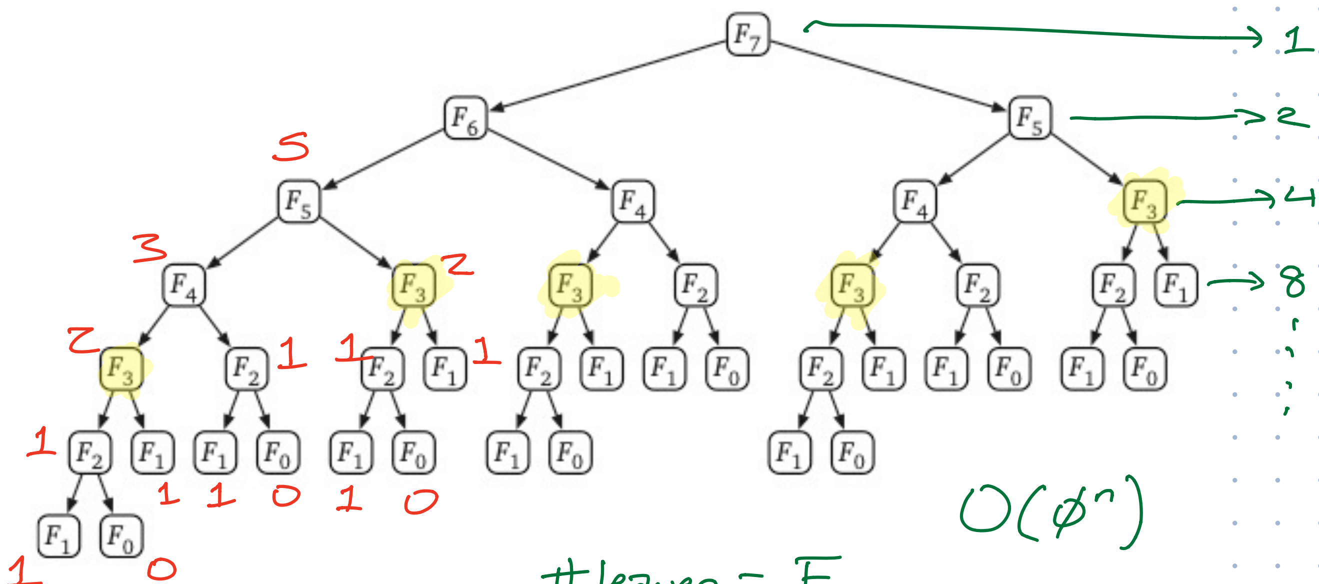
$M(2) = 2$

$M(n) = M(n-1) + M(n-2)$

1 2 3 5 8 13 21 34 55 89 144 . . .

Fibonacci #s $\quad F_n = \begin{cases} 0 & n=0 \\ 1 & n=1 \\ F_{n-1} + F_{n-2} & n>1 \end{cases}$

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



$O(\phi^n)$

$\#$ leaves $= F_{n+1}$

$\#$ additions $= F_{n+1} - 1$

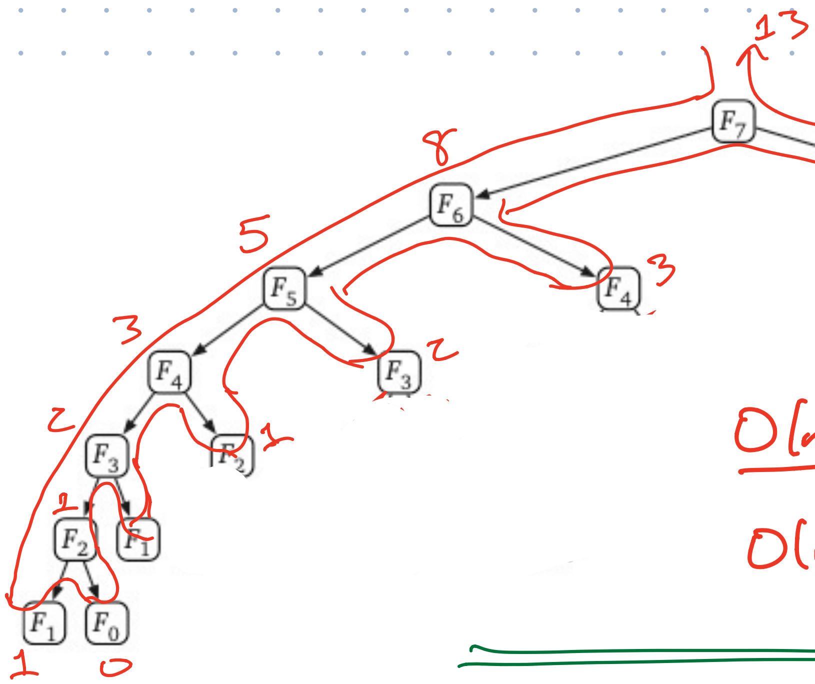
$\Theta(n)$ time just to write F_n

~~hash table / dictionary~~

array

Memoization — Remember your past work

```
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]
```



$O(n)$ additions

$O(n^2)$ time

F	0	1	1	2	3	5	8	13	...
	0	1	2	3	4	5	6	7	

Dynamic Programming

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

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

$O(n)$ additions

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

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

```
⟨⟨Compute the pair F_{n−1}, F_n⟩⟩
FASTRECFIBO(n) :
  if n = 1
      return 0, 1
  m ← ⌊n/2⌋
  hprv, hcur ← FASTRECFIBO(m)      ⟨⟨F_{m−1}, F_m⟩⟩
  prev ← hprv^2 + hcur^2           ⟨⟨F_{2m−1}⟩⟩
  curr ← hcur · (2 · hprv + hcur)  ⟨⟨F_{2m}⟩⟩
  next ← prev + curr               ⟨⟨F_{2m+1}⟩⟩
  if n is even
      return prev, curr
  else
      return curr, next
```

$T(n) = T(\frac{n}{2}) + O(1)$
arith. ops.

$O(n \log n)$ time

```
SPLITTABLE(A[1..n]):
    if n = 0
        return TRUE
    for i ← 1 to n
        if ISWORD(A[1..i])
            if SPLITTABLE(A[i+1..n])
                return TRUE
    return FALSE
```

```
⟨⟨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
```

only n ways to call this function

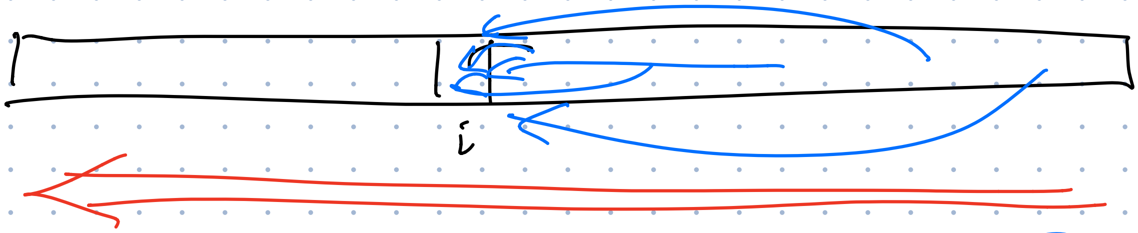
Is $A[i..n]$ splittable into words

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

$\boxed{\text{What is the first word?}}$

Memoize into an array

SplitTable[1..n+1]



1. What are the subproblems? — suffixes of A
2. Data structure — 1d array
3. Evaluation order — ←
4. Time

```
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]
```

$O(n^2)$ calls to IsWord