Divide + conquer $T(n) = T(\frac{n}{a}) + T(\frac{n}{b}) + T(\frac{n}{c}) + O(n^d)$

$\Rightarrow$ polynomial time

Backtracking $T(n) = T(n-a) + T(n-b) + T(n-c) + O(n^d)$

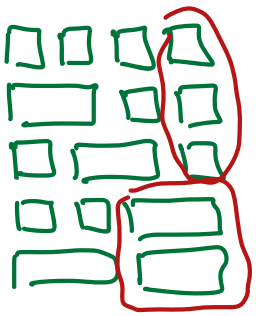
$\Rightarrow$ exponential time

↓ Recursive brute force

Dynamic Programming polynomial time

Pingala ~700 BCE

mātrāvṛtta



Virahanka 600 CE

$M(n)$ = # meters lasting n beats

$M(0) = 1$

$M(1) = 1$

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

1
1
2
3
5
8
13
21
34
55
89
144
⋮

Fibonacci Liber Abaci 1202

$F_0 = 0$

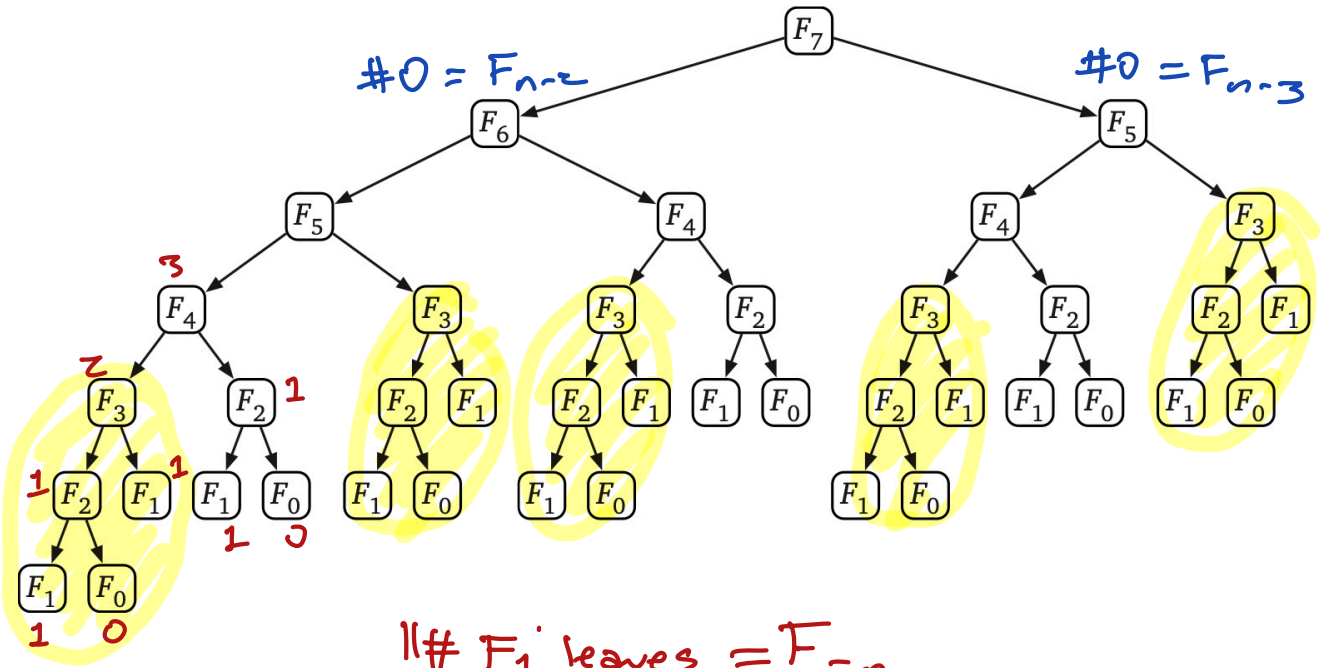
$F_1 = 1$

$F_n = F_{n-1} + F_{n-2}$

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

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

$= O(F_n) = O(\phi^n)$ $\quad \phi = \frac{\sqrt{5}+1}{2} \approx 1.61$



F_1 leaves $= F_{n}$

F_0 leaves $= F_{n-1}$

leaves $= F_{n+1}$

int nodes $= F_{n+1} - 1$

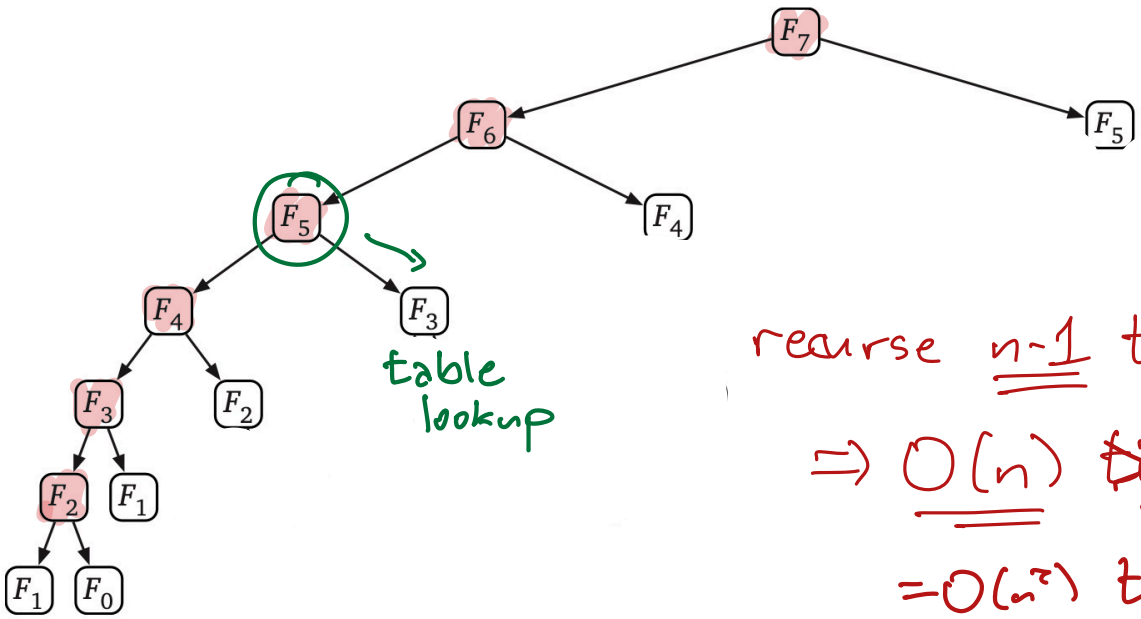
We're repeating a LOT of work a LOT

Donald Mitchie 1967 memorization

Write stuff down

Choose a data structure ~~Hashtable~~ array!

```
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]
```



recurse n-1 times

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

$= O(n^2)$ time

F:

0	1	2	3	4	5	6	7
~~1~~	~~1~~	~~2~~	~~3~~	~~5~~	~~8~~	13	

$$F_n = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}$$

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]
```

O(n) time

```
ITERFIBO2(n):
  prev ← 1        = F_1
  curr ← 0        = F_0
  for i ← 1 to n
      next ← curr + prev
      prev ← curr
      curr ← next
  return curr
```

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

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

```
⟨⟨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
```

O(log n) +'s and x's

O(n²) time

↓ karatsuba

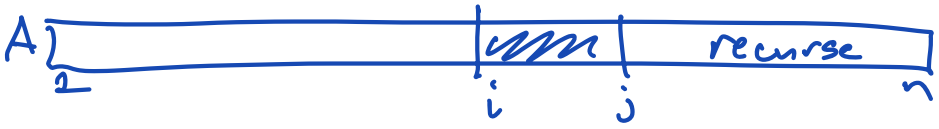
O(n^lg 3) time

↓

O(n log n) time

PRIMUS SCIENTIA...

Given string $A[1..n]$
IsWord(i,j) = True iff $A[i..j]$ is a "word"



True iff $A[i..n]$ is splittable into words

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

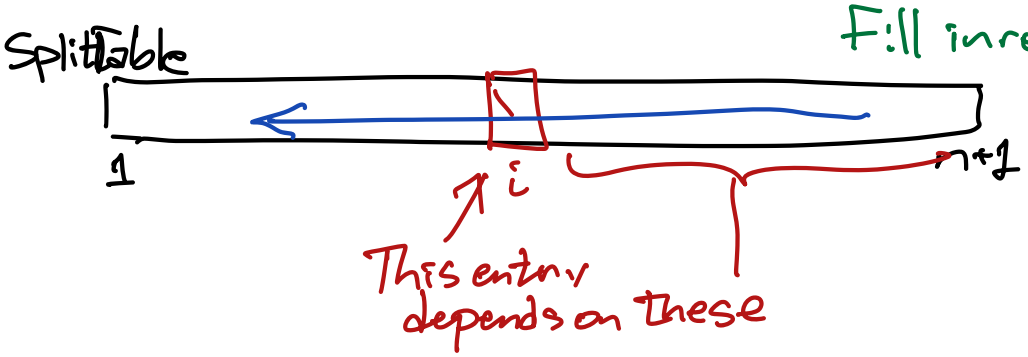
rec arg is bigger

```
⟨⟨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
```

$O(2^n)$ time

Memoize i-to array SplitTable[1..n]

Fill in reverse order




```
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]
```

calls to IsWord = $O(n^2)$

Figure 3.3. Interpunctio verborum velox