

GPS 6: Mon Oct 12

HW 6: Tue Oct 13

Drop without a W: Fri Oct 16

For all  $n \geq 0$   $F_n := \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{o.w.} \end{cases}$

Virahanka  $\sim 800$

Fibonacci 1202

RecFibo(n):

if  $n = 0$

return 0

else if  $n = 1$

return 1

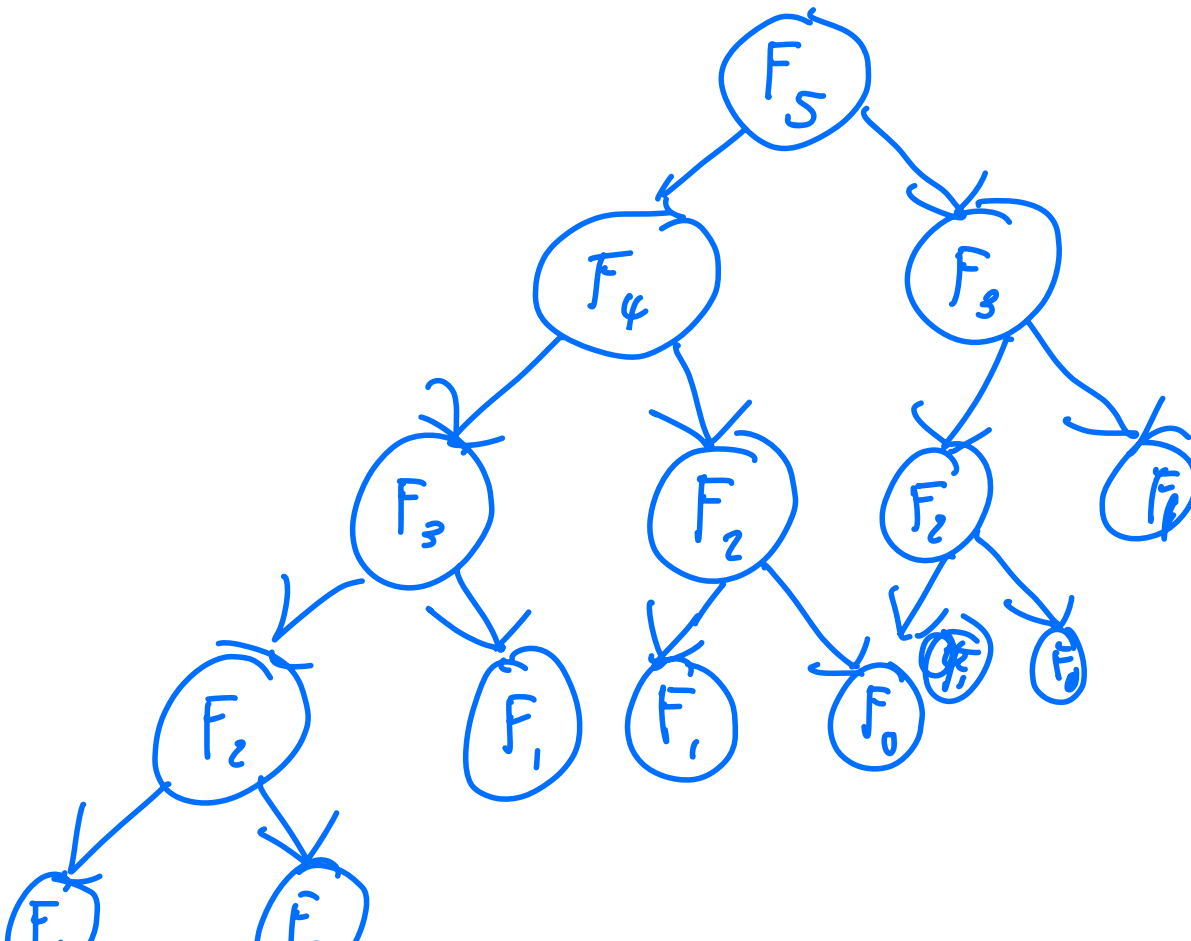
else

```
return RecFibo(n-1)
       + RecFibo(n-2)
```

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

$$T(0) = T(1) = 1$$

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



Donald Michie: memo functions:

Store the result for  
fast retrieval if you  
call it again with  
same parameters

memoization: make functions into memo  
functions

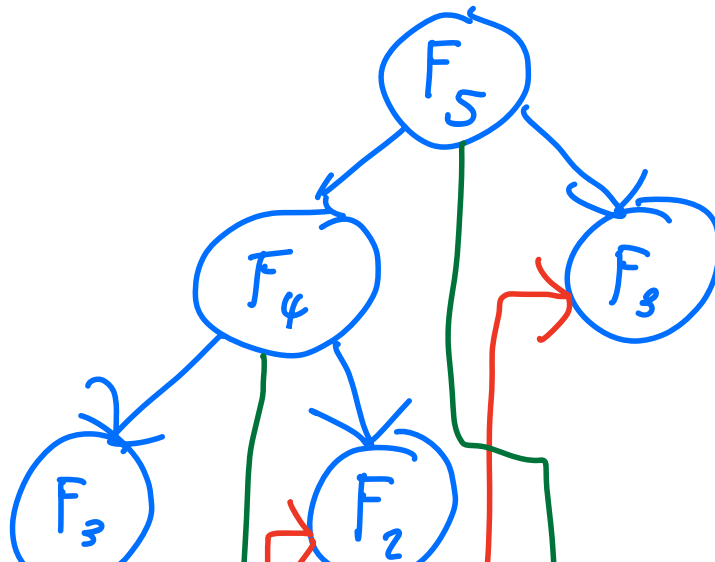
use a global... array (not a hash  
table!)

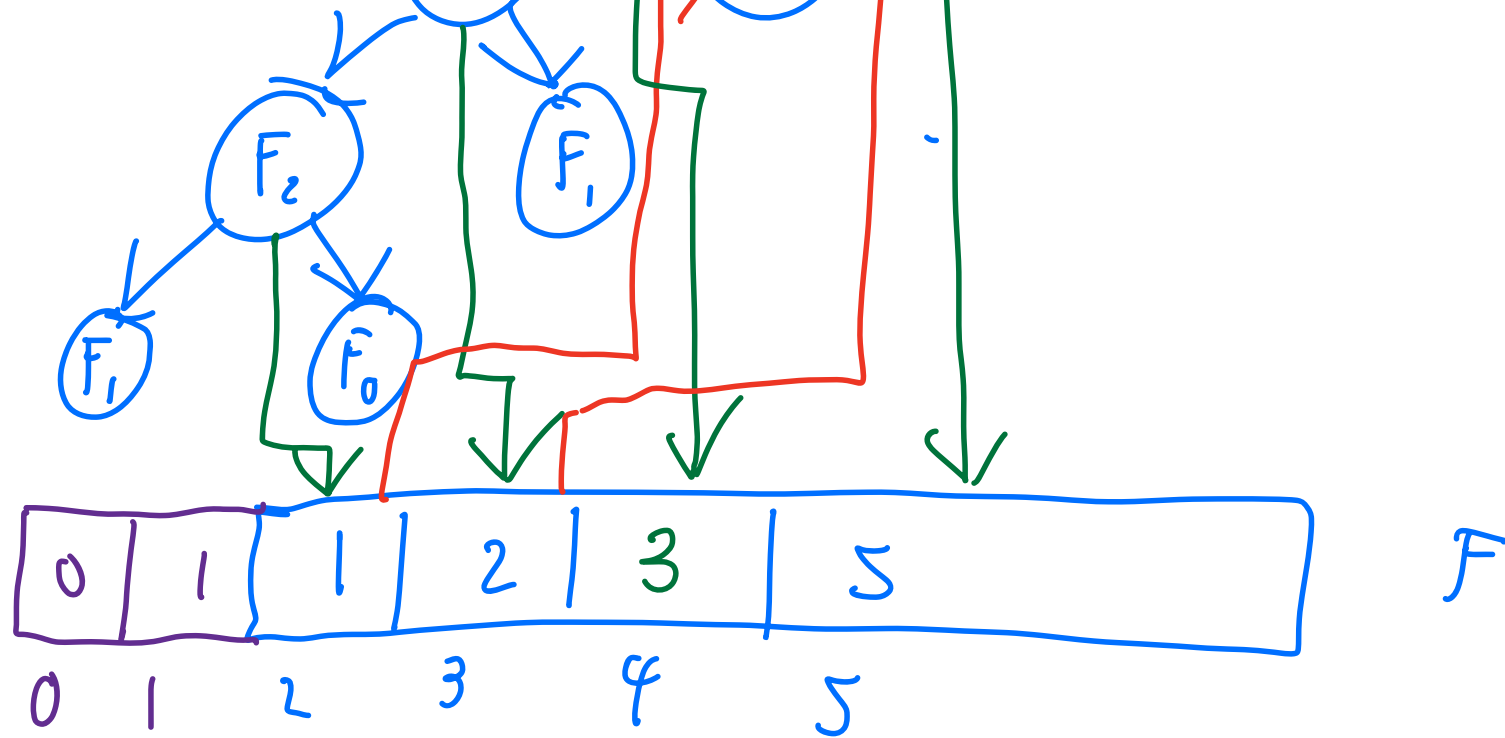
$F[1..n]$

MemFibo(i):

if  $i = 0$

```
return 0
else if  $i = 1$ 
    return 1
else
    if  $F[i]$  is undefined
         $F[i] \leftarrow \text{Mem } F:60(i-1) + \text{Mem } F:60(i-2)$ 
    return  $F[i]$ 
```





Numbers were computed from small to big /  
left to right.

$O(1)$  time for each computation

over  $O(n)$  entries in the array,

So  $O(n)$  additions total.

IterFibo(n):

$F[0] \leftarrow 0$

$F[1] \leftarrow 1$

for  $i \leftarrow 2$  to  $n$

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

return  $F[n]$

$O(n)$  additions

# Dynamic Programming

hard part,  
but needed  
for correctness

1) Find a recursive backtracking algorithm for your problem

2) Build subproblem solutions from "bottom up" usually iteratively.

↑  
fast but  
easier

Read the steps in

— Erickson Algorithms 3

T

Homework: Lab  
Rubrics

Text segmentation: Given an array

$A[1..n]$  of characters & a function  $\text{IsWord}$  where  $\text{IsWord}(w)$  returns  $\text{is } w \text{ is a word.}$

Can we partition  $A$  into substrings that are all words?

# Dynamic Programming:

1) Formulate the problem recursively.

(a) Specify the recursive subproblems.

Closed form definition of what that subproblem returns.

\*  $\text{Splittable}(i)$ : Can  $A[i..n]$  be segmented?

(b) Give a recursive solution (usually a recurrence)

$(\text{IsWord}(i, j))$ : is  $A[i..j]$  a word?

$$\text{Splittable}(i) = \begin{cases} \text{True} & \text{if } i \geq n \end{cases}$$

\*

$$\left[ \bigvee_{j=i}^n (\text{IsWord}(i, j) \wedge \text{Splittable}(j+1)) \right] \text{ o.w.}$$

(c) Say what to do with function for the original problem.

\* Want to know  $\text{Splittable}(1)$ .

2) Build solutions from the bottom up.

(a) Identify the subproblems (parameters)

\* All  $i \in \{1, \dots, n+1\}$ .

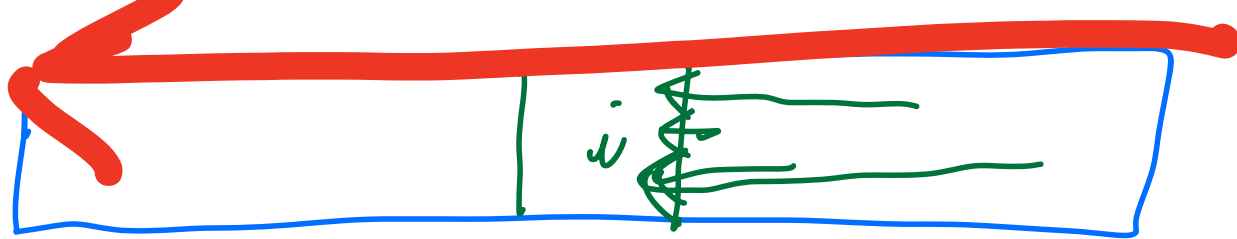
(b) Choose a memoization structure.

Usually a multi-dimensional array.

\* Split Table[1..n+1].

Do not use DP as the table  
or function name!

(c) Figure out dependencies.



Split Table

1 —  $i$  —  $n$

What needs to be known first to compute subproblems?

\* Split Table( $i$ ) need Split Table( $j+1$ )  
for all  $j \geq i$ , (bigger indices)

(d) Find an evaluation order.

\* Solve from  $i \leftarrow n+1$  down to 1

Or right to left.

Or see the giant arrow.

(e) Write the algorithm code

Wrap your recurrence in some  
for loops.

FastSplittable( $A[1..n]$ ):

for  $i \leftarrow n+1$  down to 1

if  $i = n+1$

SplitTable[ $i$ ]  $\leftarrow$  True

else

SplitTable[ $i$ ]  $\leftarrow$  False

for  $j \leftarrow i$  to  $n$

if  $IsWord(i, j)$  and  $SplitTable[j+1]$   
 $SplitTable[i] \leftarrow True$

return  $SplitTable[1]$

(f) Run time: usually (time per  
subproblem)

\* (number of  
subproblems)

\*  $O(n) \cdot O(n) = \underline{O(n^2)}$  calls to  
 $IsWord$

↑  
time per

↑  
A