Dynamic Programming
Fibonacci Example

Example

- Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, …
  Recurrence:

  $F(n)=F(n-1)+F(n-2)$ for $n>1$

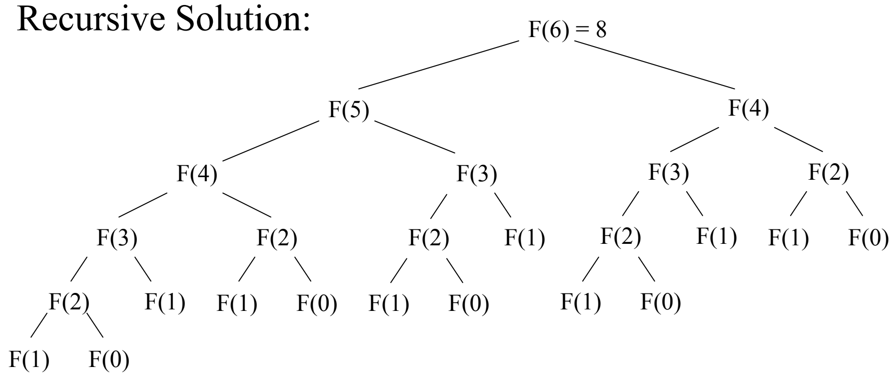
  $F(0)=0, F(1)=1$

```
ALGORITHM   F(n)
if n≤1 return n
else return F(n-1)+F(n-2)
```

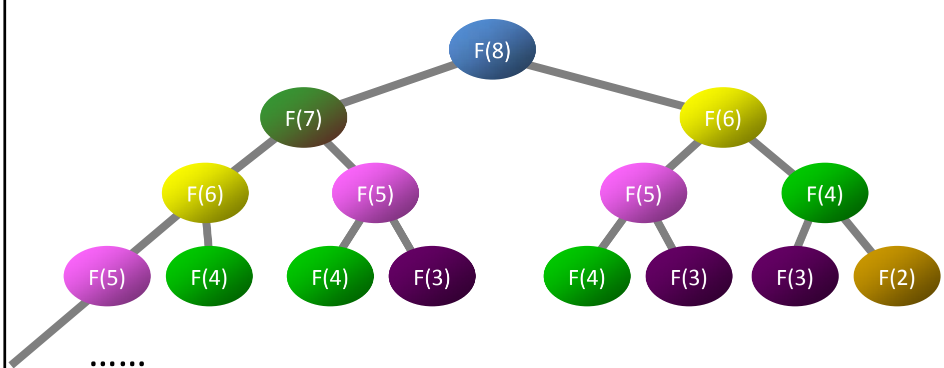
Recursive Computation

$F(n) = F(n\text{-}1) + F(n\text{-}2)$; $F(0) = 0$, $F(1) = 1$

Recursive Solution:



Inefficient!



Exponential time complexity

Bottom-up computation

We can calculate F(n) in linear time by storing small values.

```
F[0] = 0
F[1] = 1
for i = 2 to n
    F[i] = F[i-1] + F[i-2]
return F[n]
```

Moral*: We can sometimes trade space for time.*

Efficiency Example: Fibonacci numbers

- F(n) = F(n-1) + F(n-2)
  - F(0) = 0
  - F(1) = 1
- Top-down recursive computation is very inefficient
  - Many F(i) values are computed multiple times
- Bottom-up computation is much more efficient
  - Compute F(2), then F(3), then F(4), etc. using stored values for smaller F(i) values to compute next value
  - Each F(i) value is computed just once

- Footprints in the sand show where one has been

- Use ***additional memory*** *to save computation time*

The End