Algorithms

Arthur Hoskey, Ph.D.
Farmingdale State College
Computer Systems Department

Today’s Lecture

- Recursive Method Analysis

- A recursive method to show all number from n to 1.

```
void show(int n)
{
  System.out.println(n);

  if (n== 1) {
    return;
  }

  show(n-1);
}
```

What is the time complexity of this method?

Show All Numbers from n to 1

Show All Numbers from n to 1

- A recursive method to show all number from n to 1.

```
void show(int n)
{
   System.out.println(n);

   if (n== 1) {
      return;
   }

   show(n-1);
}
```

What is the time complexity of this method?

Answer: O(n)

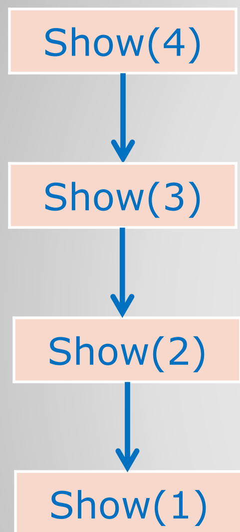
n is reduced by 1 for each recursive call until it reaches the base case (n==1)

show Recursion Trees

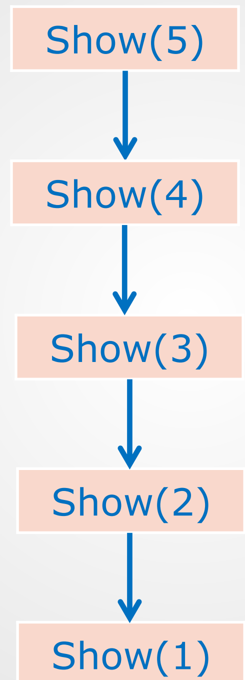
- Recursion trees for calls to show.

Base case is 1

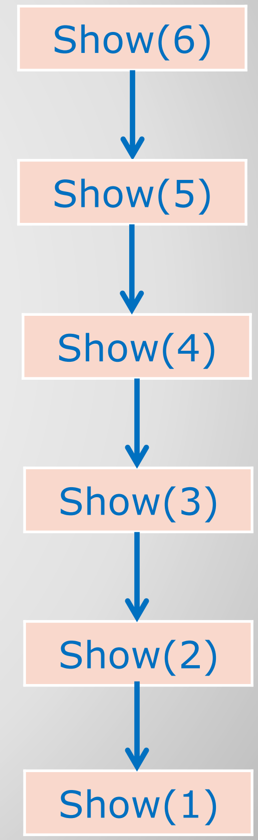
Call show(4)
Total Method Calls: 4



Call show(5)
Total Method Calls: 5



Call show(6)
Total Method Calls: 6



Time complexity O(n)

- A recursive method to show all number from n to 1.

```
void show(int n)
{
  System.out.println(n);

  if (n== 1) {
    return;
  }

  show(n-1);
  show(n-1);
}
```

What is the time complexity of this method?

There are 2 recursive calls

Show All Numbers from n to 1

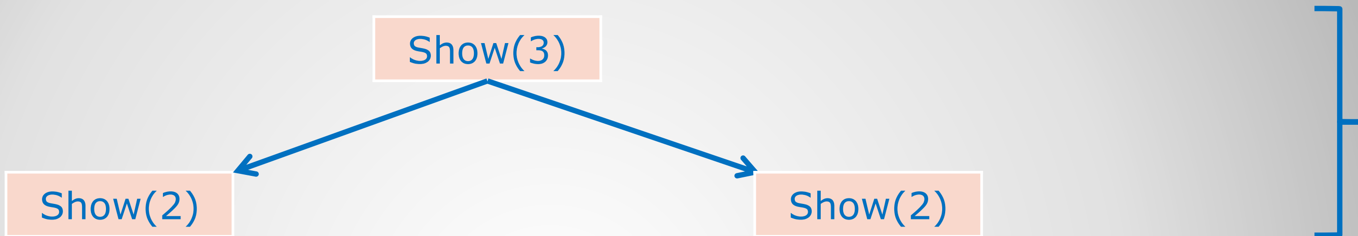
show Recursion Trees

- Recursion trees for calls to show.

Base case is 1

Call show(2)
Total Method Calls: 3

2 levels



Time complexity ???

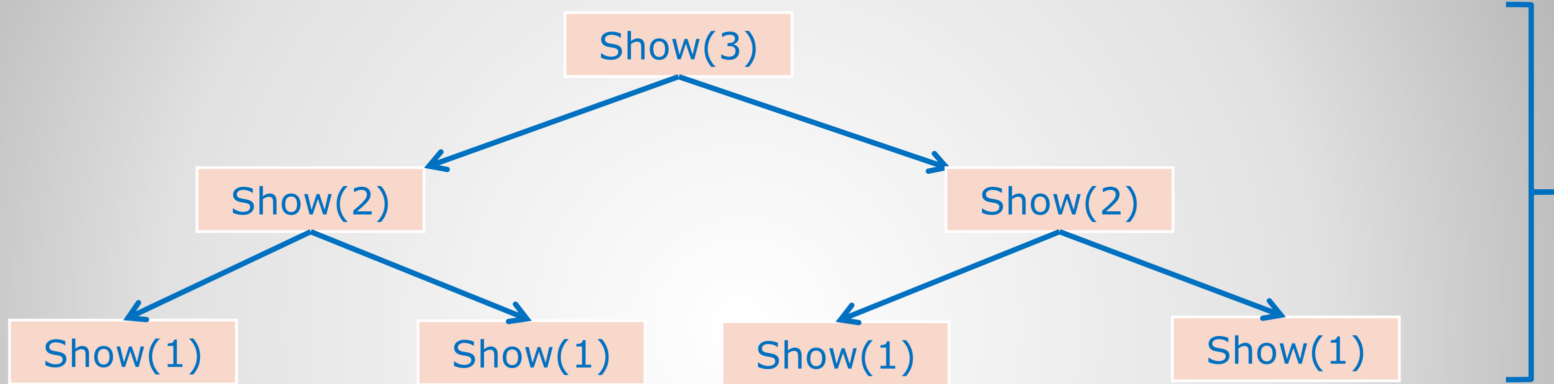
show Recursion Trees

- Recursion trees for calls to show.

Base case is 1

Call show(3)
Total Method Calls: 7

3 levels



Time complexity ???

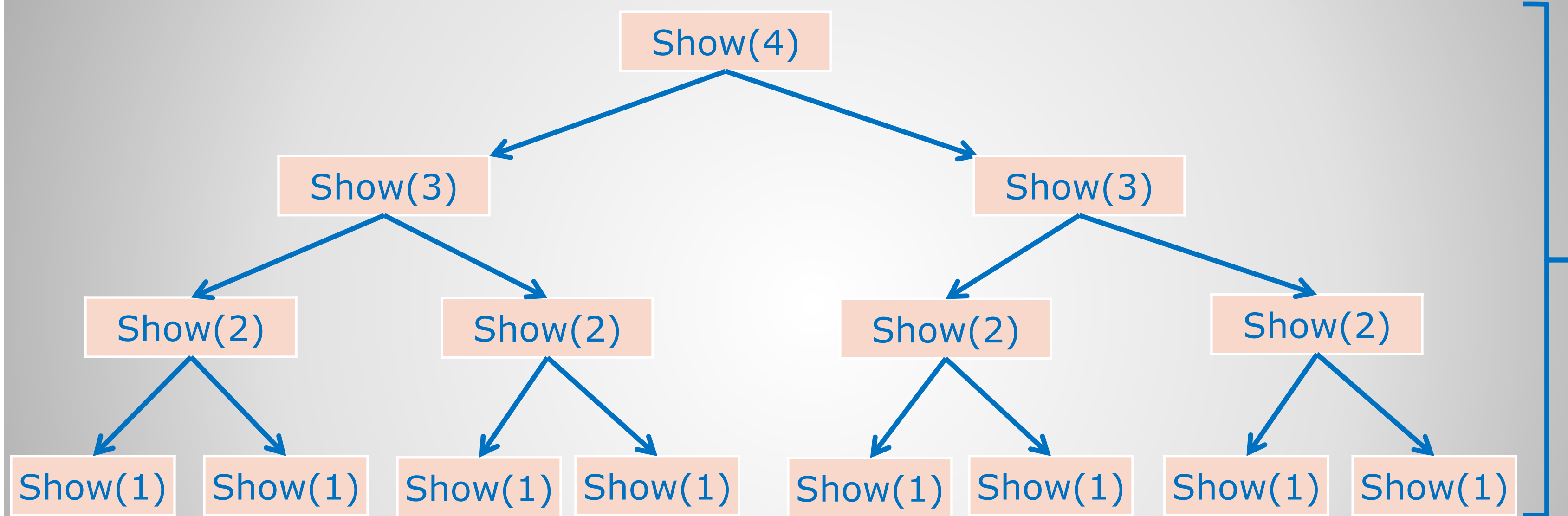
show Recursion Trees

- Recursion trees for calls to show.

Base case is 1

Call show(4)
Total Method Calls: 15

4 levels

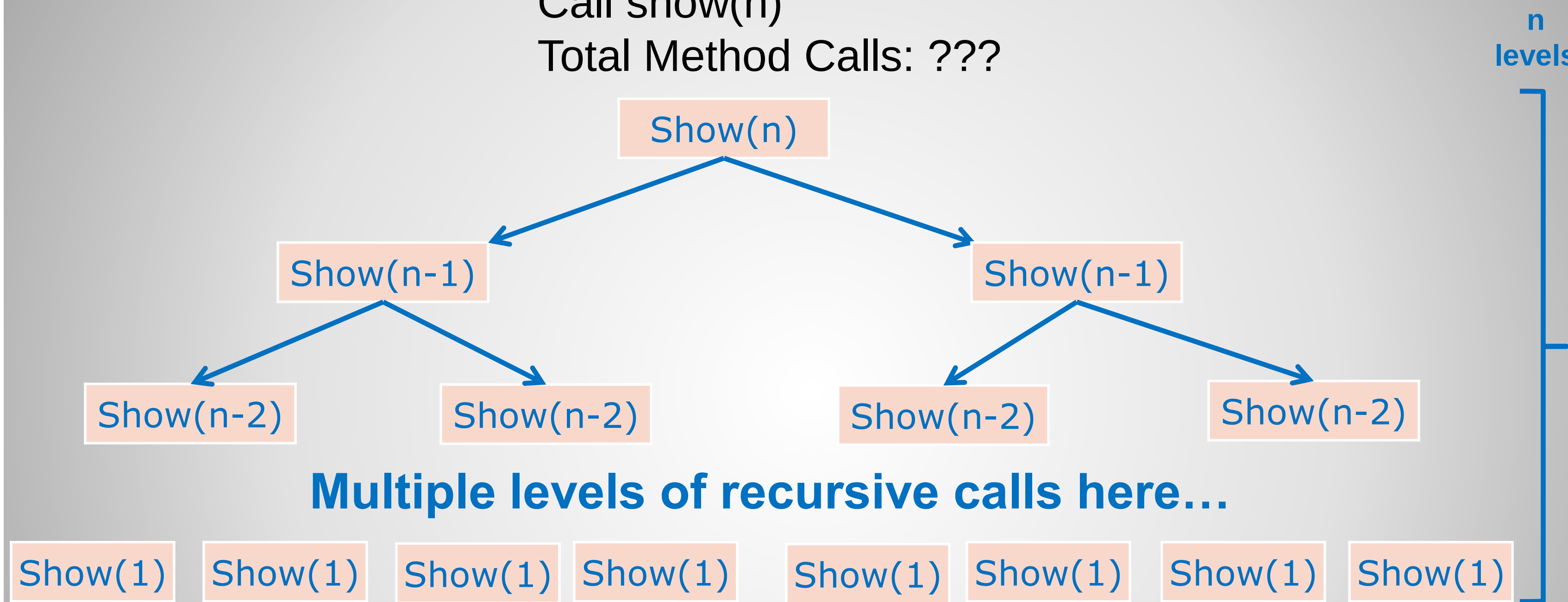


Time complexity ???

- Recursion trees for calls to show.

Base case is 1

Call show(n)
Total Method Calls: ???



Time complexity ???

show Recursion Trees

- Geometric Series

$s = ar^0 + ar^1 + ar^2 + ar^3 + ar^4 + \ldots$

- Geometric Series of the First n Terms

$s_n = \underbrace{ar^0 + ar^1 + ar^2 + ar^3 + ar^4 + \ldots + ar^{n-1}}_{\text{n terms}}$

- a – Coefficient of each term.
- r – Common ratio between terms.

- Closed form for sum of first n terms:

$$s_n = \begin{cases} a\left(\frac{1-rn}{1-r}\right) & \text{for r != 1} \\ an & \text{for r = 1} \end{cases}$$

Adapted from the following Wikipedia page:
https://en.wikipedia.org/wiki/Geometric_series

Geometric Series

Geometric Series - Example

- Formula

$s_n = ar^0 + ar^1 + ar^2 + ar^3 + ar^4 + \ldots + ar^{n-1}$

- **a=1, r=2, n=3**

$$s_3 = \underbrace{1*2^0 + 1*2^1 + 1*2^2}_{\text{3 terms (n=3)}}$$

- Closed form formula:

$$s_n = \begin{cases} a\left(\frac{1-rn}{1-r}\right) & \text{for } r!=1 \\ an & \text{for } r=1 \end{cases}$$

$$s_3 = a\left(\frac{1-rn}{1-r}\right) = 1\left(\frac{1-2^3}{1-2}\right) = 1\left(\frac{-7}{-1}\right) = 1(7) = 7$$

- Formula

$s_n = ar^0 + ar^1 + ar^2 + ar^3 + ar^4 + \ldots + ar^{n-1}$

- **a=1, r=2, n=4**

$s_4 = \underbrace{1*2^0 + 1*2^1 + 1*2^2 + 1*2^3}_{\text{4 terms (n=4)}}$

- Closed form formula:

$$s_n = \begin{cases} a\left(\frac{1-rn}{1-r}\right) & \text{for } r!=1 \\ an & \text{for } r=1 \end{cases}$$

$$s_4 = a\left(\frac{1-rn}{1-r}\right) = 1\left(\frac{1-2^4}{1-2}\right) = 1\left(\frac{-15}{-1}\right) = 1(15) = 15$$

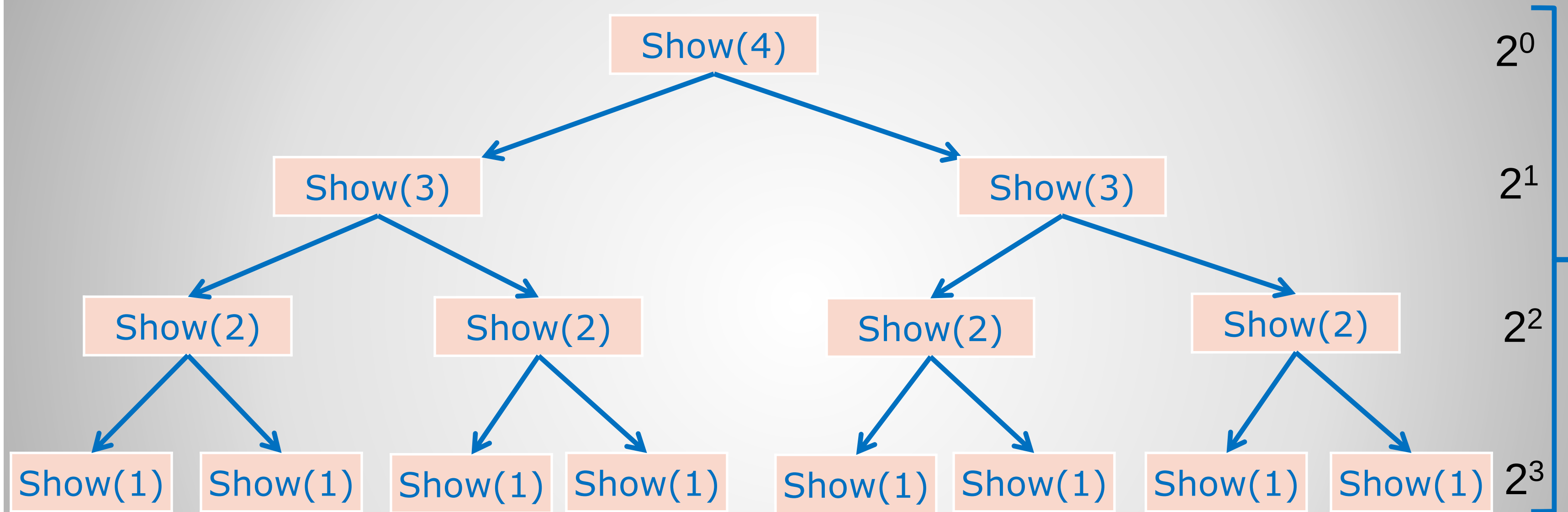
Geometric Series - Example

- Recursion trees for calls to show.

Base case is 1

Call show(4)
Total Method Calls: 15

4 levels



Time complexity
2^n

$s_4 = 1*2^0 + 1*2^1 + 1*2^2 + 1*2^3$

$s_4 = a\left(\frac{1-rn}{1-r}\right) = 1\left(\frac{1-2^4}{1-2}\right) = 1\left(\frac{-15}{-1}\right) = 1(15) = 15$

$2^4 = 16$ which is approximately 15

show Recursion Trees

Mathematical Definition for Calculating the n^{th} Fibonacci Number

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

Calculate n^{th} Fibonacci Number

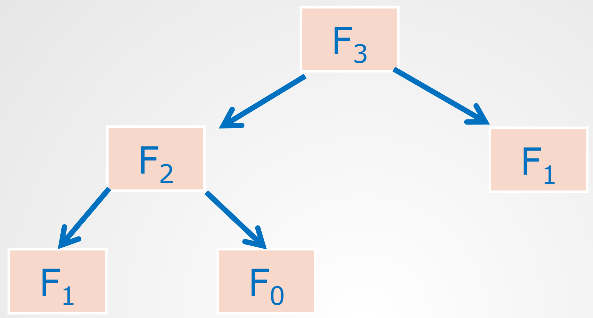
- Method to calculate the nth Fibonacci number.
- There is a small amount of code, but it is not very efficient.

```
int fibonacci(int n)
{
   if (n == 0 || n == 1)
         return n;
   else
         return fibonacci(n-2) + fibonacci(n-1);
}
```

Fibonacci Number Implementation

- Calculate F_3 (so n=3)

F_0 and F_1 are base cases

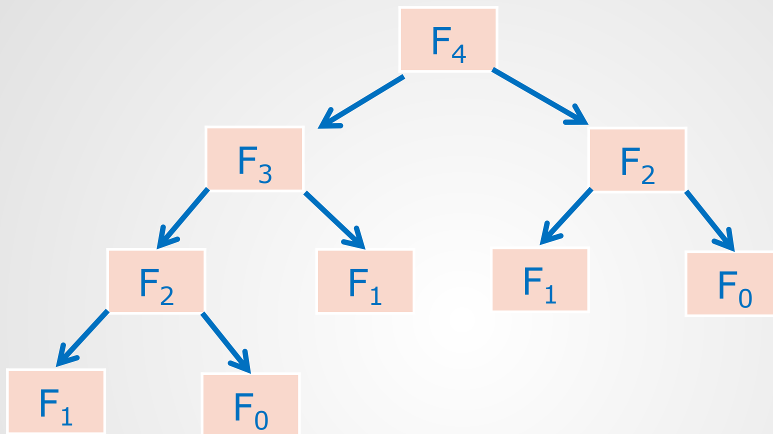


F_3 Recursion Tree

F_4 Recursion Tree

- Calculate F_4 (so n=4)

F_0 and F_1 are base cases

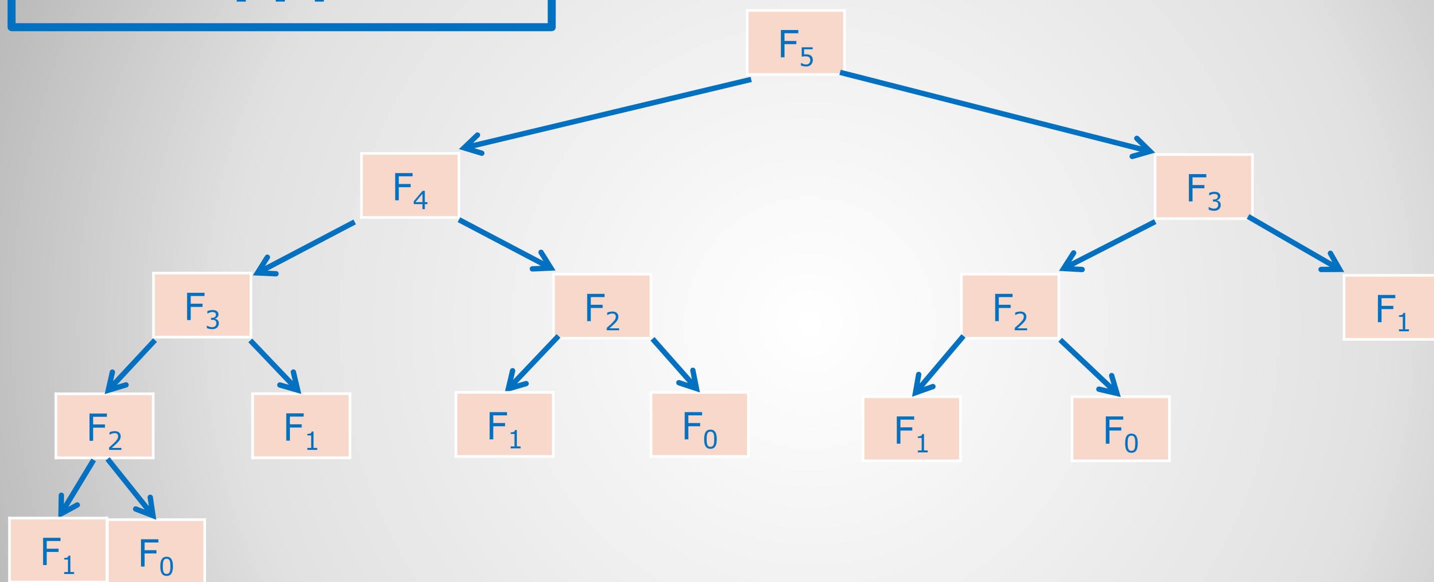


F_5 Recursion Tree

- Calculate F_5 (so n=5)

Time complexity ???

F_0 and F_1 are base cases

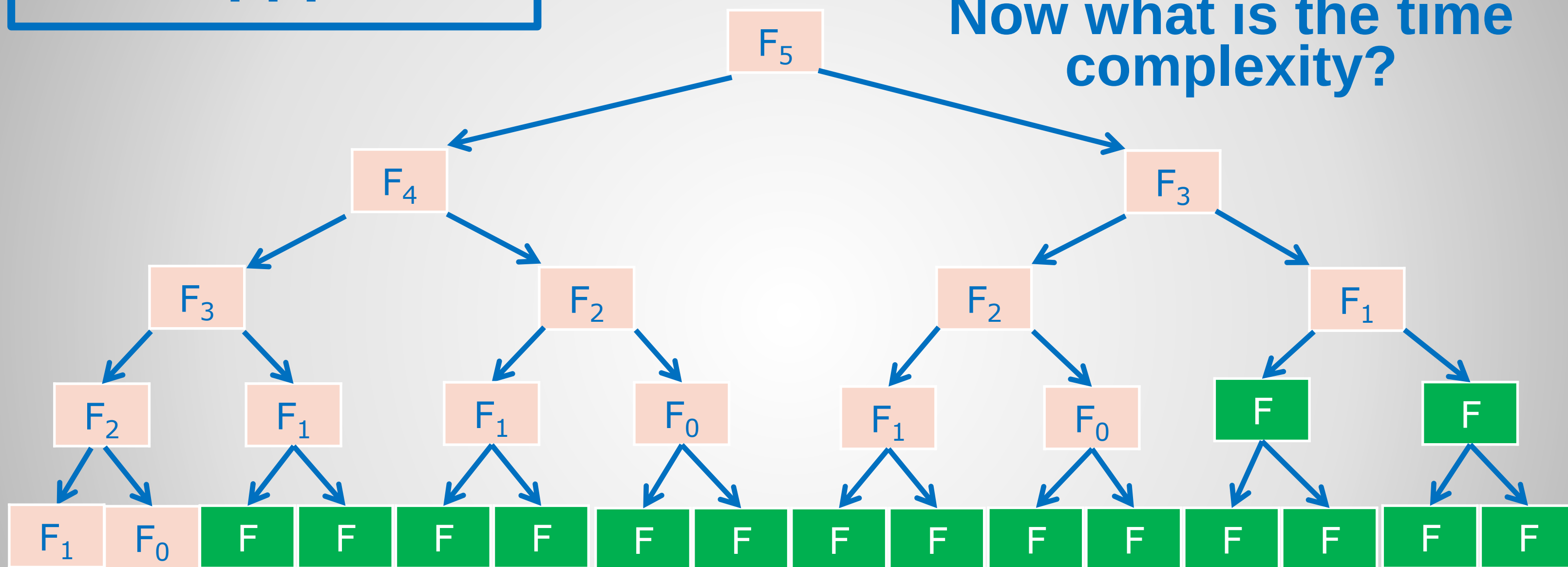


- Calculate F_5 (so n=5)

Time complexity ???

Fill out the rest of the tree with pretend nodes to make it complete.

Now what is the time complexity?

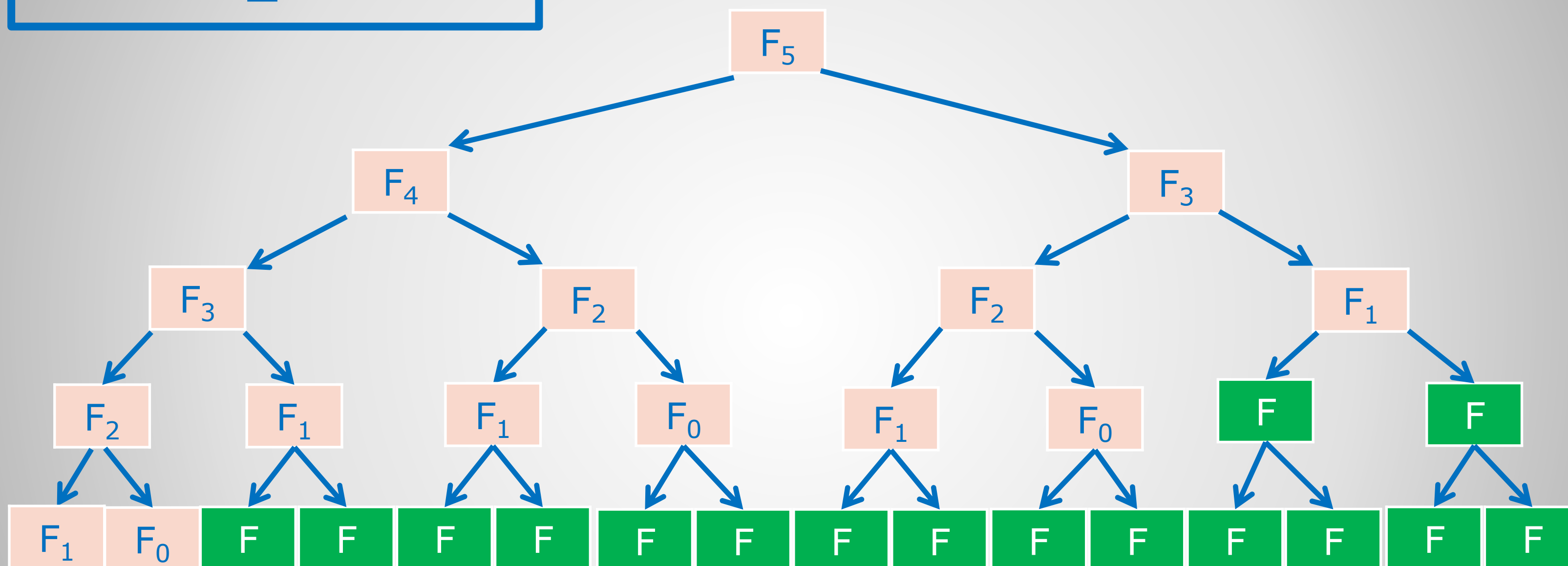


F_5 Recursion Tree

- Calculate F_5 (so $n=5$)

Time complexity
2^n

Fill out the rest of the tree with pretend nodes to make it complete



$s_5 = 1*2^0 + 1*2^1 + 1*2^2 + 1*2^3 + 1*2^4$

$s_5 = a\left(\frac{1-rn}{1-r}\right) = 1\left(\frac{1-2^5}{1-2}\right) = 1\left(\frac{-31}{-1}\right) = 1(31) = 31$

$2^5 = 32$ which is approximately 31

F_5 Recursion Tree

Recursive Fibonacci Time Complexity

- The recursive Fibonacci time complexity is generally thought of as $O(2^n)$.
- A more precise time complexity is actually $O(1.6^n)$.
- For this course on exams and quizzes we will use the generally accepted $O(2^n)$ as the time complexity.

Recursive Fibonacci Time Complexity

- **End of Slides**

End of Slides