CSI - 3105 Design & Analysis of Algorithms
Course 2

Jean-Lou De Carufel

Fall 2019

The Fibonacci Numbers

$$
\begin{aligned}
F_0 &= 0 \\
F_1 &= 1 \\
F_n &= F_{n-1} + F_{n-2} && (n \geq 2)
\end{aligned}
$$

The Fibonacci Numbers

$$
\begin{aligned}
F_0 &= 0 \\
F_1 &= 1 \\
F_n &= F_{n-1} + F_{n-2} && (n \geq 2)
\end{aligned}
$$

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

The Fibonacci Numbers

$$
\begin{aligned}
F_0 &= 0 \\
F_1 &= 1 \\
F_n &= F_{n-1} + F_{n-2} && (n \geq 2)
\end{aligned}
$$

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

Algorithm *fib*(n)

Input: An integer $n \geq 0$.

Output: F_n.

```
if n ≤ 1 then
    return n
else
    return fib(n − 1) + fib(n − 2)
end if
```

The Running Time of *fib*(n)

Algorithm *fib*(n)

Input: An integer $n \geq 0$.

Output: F_n.

```
if n ≤ 1 then
    return n
else
    return fib(n − 1) + fib(n − 2)
end if
```

The Running Time of *fib*(n)

Algorithm *fib*(n)

Input: An integer $n \geq 0$.

Output: F_n.

if $n \leq 1$ **then**
    return n
else
    return $fib(n-1) + fib(n-2)$
end if

Let $T(n)$ be the number of steps when running *fib*(n).

The Running Time of *fib*(n)

Algorithm *fib*(n)

Input: An integer $n \geq 0$.
Output: F_n.

```
if n ≤ 1 then
    return n
else
    return fib(n − 1) + fib(n − 2)
end if
```

Let $T(n)$ be the number of steps when running *fib*(n).

If $n = 0$ or $n = 1$,

The Running Time of $fib(n)$

Algorithm $fib(n)$

Input: An integer $n \geq 0$.

Output: F_n.

> **if** $n \leq 1$ **then**
>   **return** n
> **else**
>   **return** $fib(n-1) + fib(n-2)$
> **end if**

Let $T(n)$ be the number of steps when running $fib(n)$.

If $n = 0$ or $n = 1$, we do

$$\left.\begin{array}{l}\text{comparison } \text{“}n \leq 1\text{”} \\ \text{return value}\end{array}\right\} \quad 2 \text{ steps}$$

The Running Time of $fib(n)$

If $n \geq 2$, we do

comparison "$n \leq 1$"	:	1 step
compute $n - 1$	:	1 step
call $fib(n - 1)$	:	?
compute $n - 2$	:	1 step
call $fib(n - 2)$	:	?
compute sum of two results	:	1 step
return output	:	1 step

The Running Time of $fib(n)$

If $n \geq 2$, we do

comparison "$n \leq 1$"	:	1 step
compute $n-1$	:	1 step
call $fib(n-1)$	:	$T(n-1)$ steps
compute $n-2$	:	1 step
call $fib(n-2)$	:	?
compute sum of two results	:	1 step
return output	:	1 step

The Running Time of $fib(n)$

If $n \geq 2$, we do

comparison "$n \leq 1$"	:	1 step
compute $n - 1$	:	1 step
call $fib(n-1)$	:	$T(n-1)$ steps
compute $n - 2$	:	1 step
call $fib(n-2)$	:	$T(n-2)$ steps
compute sum of two results	:	1 step
return output	:	1 step

The Running Time of *fib*(n)

If $n \geq 2$, we do

comparison "$n \leq 1$"	:	1 step
compute $n - 1$	:	1 step
call *fib*($n-1$)	:	$T(n-1)$ steps
compute $n - 2$	:	1 step
call *fib*($n-2$)	:	$T(n-2)$ steps
compute sum of two results	:	1 step
return output	:	1 step

$$T(n) = \begin{cases} 2 & \text{if } n = 0, \\ 2 & \text{if } n = 1, \\ T(n-1) + T(n-2) + 5 & \text{if } n \geq 2. \end{cases}$$

The Running Time of $fib(n)$

If $n \geq 2$, we do

comparison "$n \leq 1$"	:	1 step
compute $n-1$	:	1 step
call $fib(n-1)$	:	$T(n-1)$ steps
compute $n-2$	:	1 step
call $fib(n-2)$	:	$T(n-2)$ steps
compute sum of two results	:	1 step
return output	:	1 step

$$T(n) = \begin{cases} 2 & \text{if } n = 0, \\ 2 & \text{if } n = 1, \\ T(n-1) + T(n-2) + 5 & \text{if } n \geq 2. \end{cases}$$

What do we do with this?

The Running Time of *fib*(n)

$$T(n) = \begin{cases} 2 & \text{if } n = 0, \\ 2 & \text{if } n = 1, \\ T(n-1) + T(n-2) + 5 & \text{if } n \geq 2. \end{cases}$$

The Running Time of *fib*(n)

$$T(n) = \begin{cases} 2 & \text{if } n = 0, \\ 2 & \text{if } n = 1, \\ T(n-1) + T(n-2) + 5 & \text{if } n \geq 2. \end{cases}$$

In Exercise #20, you will prove that $T(n) \geq F_n$. So the running time of *fib*(n) is at least F_n.

The Running Time of $fib(n)$

$$T(n) = \begin{cases} 2 & \text{if } n = 0, \\ 2 & \text{if } n = 1, \\ T(n-1) + T(n-2) + 5 & \text{if } n \geq 2. \end{cases}$$

In Exercise #20, you will prove that $T(n) \geq F_n$. So the running time of $fib(n)$ is at least F_n.

The problem is that F_n is very large. Indeed, in Exercise #19, you will prove that $F_n \geq 2^{n/2}$ for all $n \geq 6$.

The Running Time of *fib*(n)

$$T(n) = \begin{cases} 2 & \text{if } n = 0, \\ 2 & \text{if } n = 1, \\ T(n-1) + T(n-2) + 5 & \text{if } n \geq 2. \end{cases}$$

In Exercise #20, you will prove that $T(n) \geq F_n$. So the running time of *fib*(n) is at least F_n.

The problem is that F_n is very large. Indeed, in Exercise #19, you will prove that $F_n \geq 2^{n/2}$ for all $n \geq 6$.

So *fib*(n) takes at least exponential time: *fib*(200) will not terminate during our lifetime!

The Running Time of *fib*(n)

$$T(n) = \begin{cases} 2 & \text{if } n = 0, \\ 2 & \text{if } n = 1, \\ T(n-1) + T(n-2) + 5 & \text{if } n \geq 2. \end{cases}$$

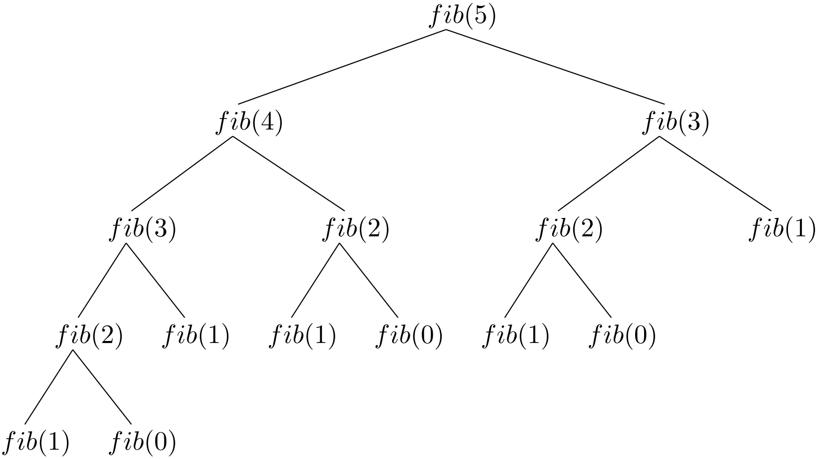
In Exercise #20, you will prove that $T(n) \geq F_n$. So the running time of *fib*(n) is at least F_n.

The problem is that F_n is very large. Indeed, in Exercise #19, you will prove that $F_n \geq 2^{n/2}$ for all $n \geq 6$.

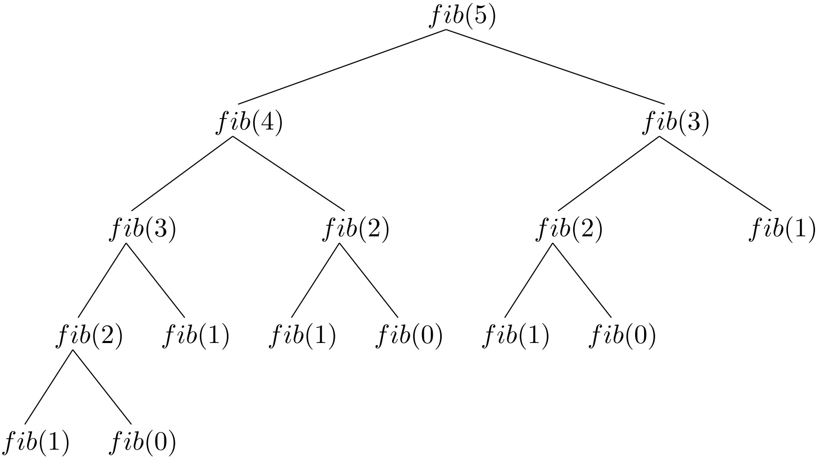
So *fib*(n) takes at least exponential time: *fib*(200) will not terminate during our lifetime!

Why is *fib*(n) so slow?

The Running Time of *fib*(n)



The Running Time of *fib*(n)



Too many things are called multiple times.

A Better Algorithm

Algorithm *fib2*(n)

Input: An integer $n \geq 0$.
Output: F_n.

if $n \leq 1$ **then**
  return n
else
  initialize array $f[0..n]$
  $f[0] = 0$
  $f[1] = 1$
  for $i = 2$ to n **do**
    $f[i] = f[i-1] + f[i-2]$
  end for
  return $f[n]$
end if

The running time of *fib*2(n) is *linear in n*, i.e. it is $O(n)$. In this case, we can even say that it is $\Theta(n)$.

The running time of *fib*2(n) is *linear in n*, i.e. it is $O(n)$. In this case, we can even say that it is $\Theta(n)$.

Running time of *fib*(n):	exponential
Running time of *fib*2(n):	linear

The running time of $\mathit{fib2}(n)$ is *linear in n*, i.e. it is $O(n)$. In this case, we can even say that it is $\Theta(n)$.

Running time of $\mathit{fib}(n)$:	exponential
Running time of $\mathit{fib2}(n)$:	linear

But!

The running time of *fib*2(n) is *linear in n*, i.e. it is $O(n)$. In this case, we can even say that it is $\Theta(n)$.

Running time of *fib*(n):	exponential
Running time of *fib*2(n):	linear

But! Is it realistic to say that *fib*2(n) takes a linear number of steps?

The running time of $fib2(n)$ is *linear in n*, i.e. it is $O(n)$. In this case, we can even say that it is $\Theta(n)$.

Running time of $fib(n)$:	exponential
Running time of $fib2(n)$:	linear

But! Is it realistic to say that $fib2(n)$ takes a linear number of steps?

In our analysis, one step corresponds to

$$\left.\begin{array}{l}\text{comparison}\\ \text{addition}\\ \text{subtraction}\end{array}\right\}\quad \text{involving very large numbers}$$

In kindergarden, we learned that

Two n-bit numbers can be added in a linear number of bit-operations.

In kindergarden, we learned that

Two n-bit numbers can be added in a linear number of bit-operations.

When running *fib2*(n), we do roughly n additions of numbers, each of these numbers is at most F_n, each of these numbers has roughly n bits.

In kindergarden, we learned that

Two n-bit numbers can be added in a linear number of bit-operations.

When running *fib*2(n), we do roughly n additions of numbers, each of these numbers is at most F_n, each of these numbers has roughly n bits.

Therefore, *fib*2(n) makes a *quadratic number* of bit-operations, i.e. $O(n^2)$ bit-operations.

In kindergarden, we learned that

Two n-bit numbers can be added in a linear number of bit-operations.

When running *fib*2(n), we do roughly n additions of numbers, each of these numbers is at most F_n, each of these numbers has roughly n bits.

Therefore, *fib*2(n) makes a *quadratic number* of bit-operations, i.e. $O(n^2)$ bit-operations.

In Exercise #23, you will prove that the number of bit-operations done by *fib*(n) is $O(n \cdot F_n)$.

In kindergarden, we learned that

Two n-bit numbers can be added in a linear number of bit-operations.

When running $fib2(n)$, we do roughly n additions of numbers, each of these numbers is at most F_n, each of these numbers has roughly n bits.

Therefore, $fib2(n)$ makes a *quadratic number* of bit-operations, i.e. $O(n^2)$ bit-operations.

In Exercise #23, you will prove that the number of bit-operations done by $fib(n)$ is $O(n \cdot F_n)$.

Therefore, the running time, in terms of bit-operations:

Running time of $fib(n)$:	exponential
Running time of $fib2(n)$:	quadratic

Chapter 2: Divide-and-Conquer Algorithms

To solve a problem of size n:

- **Divide** the problem into subproblems, each of size $< n$.
- **Conquer**: Solve each subproblem recursively (and independantly of the other subproblems).
- **Combine/Merge** the solutions to the subproblems into a solution to the original problem.

Chapter 2: Divide-and-Conquer Algorithms

To solve a problem of size n:

- **Divide** the problem into subproblems, each of size $< n$.
- **Conquer**: Solve each subproblem recursively (and independantly of the other subproblems).
- **Combine/Merge** the solutions to the subproblems into a solution to the original problem.

For a given problem,

$\rightarrow$ How do we divide the problem, into how many subproblems?

$\rightarrow$ How to combine/merge?

Example: Merge Sort

To sort n numbers:

If $n \leq 1$: do nothing.

If $n \geq 2$: divide the n numbers arbitrarily into two sequences, both of size $n/2$, run Merge sort twice, once for each sequence.

Then merge the two sorted sequences into one sorted sequence.

Example: Merge Sort

To sort n numbers:

If $n \leq 1$: do nothing.

If $n \geq 2$: divide the n numbers arbitrarily into two sequences, both of size $n/2$, run Merge sort twice, once for each sequence.

Then merge the two sorted sequences into one sorted sequence.

What is the running time?

Let $T(n)$ be the running time of Merge Sort on an input of n numbers.

Let $T(n)$ be the running time of Merge Sort on an input of n numbers.

From CSI-2110, we know that the merge step takes $O(n)$ time.

Let $T(n)$ be the running time of Merge Sort on an input of n numbers.

From CSI-2110, we know that the merge step takes $O(n)$ time.

Hence, there is a constant $c > 0$ such that

$$T(n) \leq \begin{cases} c & \text{if } n = 1, \\ 2\,T(n/2) + c \cdot n & \text{if } n \geq 2, \end{cases}$$

Let $T(n)$ be the running time of Merge Sort on an input of n numbers.

From CSI-2110, we know that the merge step takes $O(n)$ time.

Hence, there is a constant $c > 0$ such that

$$T(n) \leq \begin{cases} c & \text{if } n = 1, \\ 2\,T(n/2) + c \cdot n & \text{if } n \geq 2, \end{cases}$$

What do we do with this?

Let $T(n)$ be the running time of Merge Sort on an input of n numbers.

From CSI-2110, we know that the merge step takes $O(n)$ time.

Hence, there is a constant $c > 0$ such that

$$T(n) \leq \begin{cases} c & \text{if } n = 1, \\ 2\,T(n/2) + c \cdot n & \text{if } n \geq 2, \end{cases}$$

What do we do with this?

We solve by *unfolding*!

In general, if we do not assume anything about the value of c, we find $T(n) \leq 2c\, n \log_2(n)$.

In general, if we do not assume anything about the value of c, we find $T(n) \leq 2c\, n \log_2(n)$.

So the running time of Merge Sort is $T(n) = O(n \log_2(n))$ if n is a power of two.

In general, if we do not assume anything about the value of c, we find $T(n) \leq 2c\, n \log_2(n)$.

So the running time of Merge Sort is $T(n) = O(n \log_2(n))$ if n is a power of two.

For a general n, we have

$$T(n) \leq \begin{cases} c & \text{if } n = 1, \\ T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + c \cdot n & \text{if } n \geq 2, \end{cases}$$

In general, if we do not assume anything about the value of c, we find $T(n) \leq 2c\, n \log_2(n)$.

So the running time of Merge Sort is $T(n) = O(n \log_2(n))$ if n is a power of two.

For a general n, we have

$$T(n) \leq \begin{cases} c & \text{if } n = 1, \\ T\left(\lfloor n/2 \rfloor\right) + T\left(\lceil n/2 \rceil\right) + c \cdot n & \text{if } n \geq 2, \end{cases}$$

By induction, we can prove that $T(n) = O(n \log_2(n))$.