CSE 250
Data Structures

Dr. Eric Mikida
epmikida@buffalo.edu
208 Capen Hall

Lec 12: Recursion

Announcements

- PA1 Implementation due last night, submissions close Tuesday
- WA2 out now, due this Sunday, 2/23 @ 11:59PM

Recursion

Factorial

factorial(n) = n * (n-1) * (n-2) * … * 2 * 1

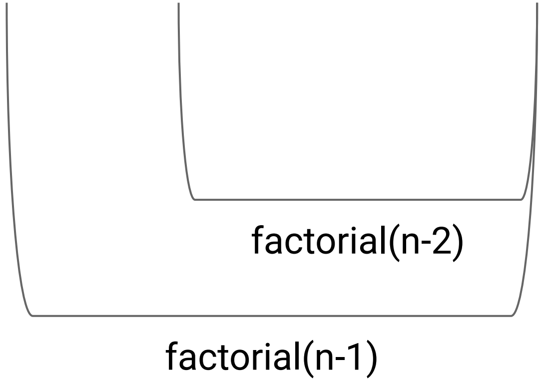
Factorial

factorial(n) = n * (n-1) * (n-2) * … * 2 * 1

factorial(n-1)

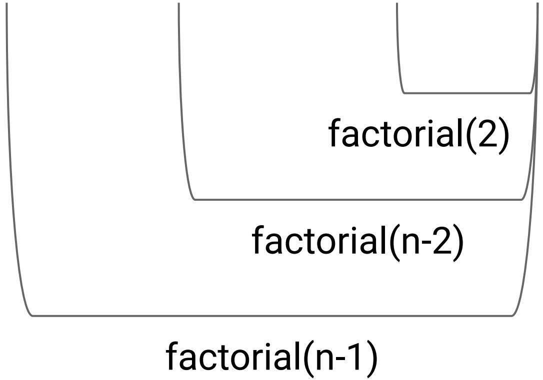
Factorial

factorial(n) = n * (n-1) * (n-2) * … * 2 * 1



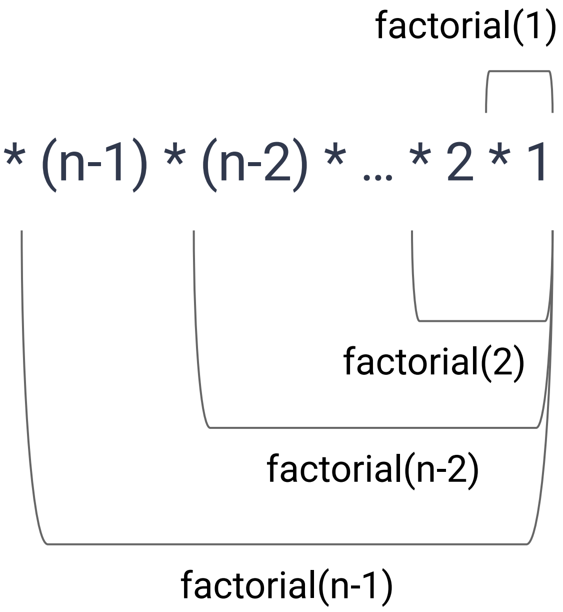
Factorial

factorial(n) = n * (n-1) * (n-2) * … * 2 * 1



Factorial

$$factorial(n) = n * (n-1) * (n-2) * \ldots * 2 * 1$$



Factorial

```
public int factorial(int n) {
    if(n <= 1) { return 1; }
    else { return n * factorial(n - 1); }
}
```

Factorial

```
public int factorial(int n) {
    if(n <= 1) { return 1; }                  ← Base Case
    else { return n * factorial(n - 1); }
}
```

Factorial

```
public int factorial(int n) {
    if(n <= 1) { return 1; }                 ← Base Case
    else { return n * factorial(n - 1); }  ← Recursive Case
}
```

Fibonacci

fib(n) = 1, 1

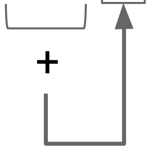
Fibonacci

fib(n) = 1, 1, 2

+

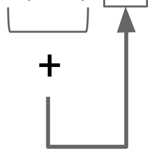
Fibonacci

fib(n) = 1, 1, 2, 3



Fibonacci

fib(n) = 1, 1, 2, 3, 5



Fibonacci

fib(n) = 1, 1, 2, 3, 5, 8, 13, 21, 34, …

Fibonacci

fib(n) = 1, 1, 2, 3, 5, 8, 13, 21, 34, …

fib(n) = fib(n-1) + fib(n-2)

Fibonacci

```
public int fib(int n) {
    if(n < 2) { return 1; }
    else { return fib(n-1) + fib(n - 2); }
}
```

Fibonacci

```
public int fib(int n) {
    if(n < 2) { return 1; }                    ← Base Case
    else { return fib(n-1) + fib(n - 2); }
}
```

Fibonacci

```
public int fib(int n) {
    if(n < 2) { return 1; }                        ← Base Case
    else { return fib(n-1) + fib(n - 2); }  ← Recursive Case
}
```

Towers of Hanoi

Live demo!

But What is the Complexity?

```
public int factorial(int n) {
    if(n <= 1) { return 1; }
    else { return n * factorial(n - 1); }
}
```

But What is the Complexity?

```
public int factorial(int n) {
    if(n <= 1) { return 1; }                ← Θ(1)
    else { return n * factorial(n - 1); }
}
```

But What is the Complexity?

```
public int factorial(int n) {
    if(n <= 1) { return 1; }                     ← Θ(1)
    else { return n * factorial(n - 1); }  ← Θ(1) + Θ(???)
}
```

But What is the Complexity?

```
public int factorial(int n) {
    if(n <= 1) { return 1; }                        ← Θ(1)
    else { return n * factorial(n - 1); }           ← Θ(1) + Θ(???)
}
```

How do we figure out complexity of a function, when part of the runtime of the function is calling itself?

To know the complexity of **`factorial`***, we need to…know the complexity of* **`factorial`***?*

But What is the Complexity?

```
public int factorial(int n) {
    if(n <= 1) { return 1; }                      ← Θ(1)
    else { return n * factorial(n - 1); }  ← Θ(1) + Θ(???)
}
```

What about the growth function for the runtime of **`factorial`***?*

But What is the Complexity?

```
public int factorial(int n) {
    if(n <= 1) { return 1; }                    ← Θ(1)
    else { return n * factorial(n - 1); }  ← Θ(1) + Θ(???)
}
```

What about the growth function for the runtime of **`factorial`***?*

Growth functions can be recursive too!

Complexity of `factorial`

$$T(n) = \begin{cases} \Theta(1) & \textbf{if } n \leq 1 \\ T(n-1) + \Theta(1) & \textbf{otherwise} \end{cases}$$

Solve for *T*(*n*)

Complexity of `factorial`

Solve for $T(n)$

Approach:

1. Generate a hypothesis
2. Prove your hypothesis for the base case
3. Prove the hypothesis for the recursive case *inductively*

Step 1 - Generate a Hypothesis

Let's start by looking at the runtime for increasing values of *n*

Step 1 - Generate a Hypothesis

Let's start by looking at the runtime for increasing values of *n*

$\Theta(1)$

Step 1 - Generate a Hypothesis

Let's start by looking at the runtime for increasing values of *n*

$\Theta(1), 2\Theta(1)$

Step 1 - Generate a Hypothesis

Let's start by looking at the runtime for increasing values of *n*

$\Theta(1)$, $2\Theta(1)$, $3\Theta(1)$

Step 1 - Generate a Hypothesis

Let's start by looking at the runtime for increasing values of *n*

$\Theta(1), 2\Theta(1), 3\Theta(1), 4\Theta(1), 5\Theta(1), 6\Theta(1), 7\Theta(1)$

Step 1 - Generate a Hypothesis

Let's start by looking at the runtime for increasing values of *n*

$\Theta(1), 2\Theta(1), 3\Theta(1), 4\Theta(1), 5\Theta(1), 6\Theta(1), 7\Theta(1)$

What is the pattern?

Step 1 - Generate a Hypothesis

Let's start by looking at the runtime for increasing values of *n*

$\Theta(1), 2\Theta(1), 3\Theta(1), 4\Theta(1), 5\Theta(1), 6\Theta(1), 7\Theta(1)$

What is the pattern?

Hypothesis: $T(n) \in O(n)$

Step 1 - Generate a Hypothesis

Let's start by looking at the runtime for increasing values of *n*

$\Theta(1), 2\Theta(1), 3\Theta(1), 4\Theta(1), 5\Theta(1), 6\Theta(1), 7\Theta(1)$

What is the pattern?

Hypothesis: $T(n) \in O(n)$

(there is some $c > 0$ such that $T(n) \leq c \cdot n$)

Prove for the Base Case

First, lets make our constants explicit

$$T(n) = \begin{cases} c_0 & \textbf{if } n \leq 1 \\ T(n-1) + c_1 & \textbf{otherwise} \end{cases}$$

Prove $T(n) \in O(n)$ for the Base Case

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \leq c \cdot n$)

Base Case: n = 1

$$T(1) \leq c \cdot 1$$

Prove $T(n) \in O(n)$ for the Base Case

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \leq c \cdot n$)

Base Case: n = 1

$$T(1) \leq c \cdot 1$$

$$T(1) \leq c$$

Prove $T(n) \in O(n)$ for the Base Case

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \leq c \cdot n$)

Base Case: n = 1

$$T(1) \leq c \cdot 1$$

$$T(1) \leq c$$

$$c_0 \leq c$$

Expand $T(1)$ based on the definition of T

Prove $T(n) \in O(n)$ for the Base Case

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \le c \cdot n$)

Base Case: n = 1

$$T(1) \le c \cdot 1$$

$$T(1) \le c$$

$$c_0 \le c$$

True for any $c \ge c_0$

Prove $T(n) \in O(n)$ for the Base Case + 1

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \leq c \cdot n$)

Base Case + 1: n = 2

$T(2) \leq c \cdot 2$

Prove $T(n) \in O(n)$ for the Base Case + 1

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \leq c \cdot n$)

Base Case + 1: n = 2

Expand $T(2)$ based on the definition of T

$$T(2) \leq c \cdot 2$$

$$T(1) + c_1 \leq 2c$$

Prove $T(n) \in O(n)$ for the Base Case + 1

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \leq c \cdot n$)

Base Case + 1: n = 2

$$T(2) \leq c \cdot 2$$

$$T(1) + c_1 \leq 2c$$

$$c_0 + c_1 \leq 2c$$

Prove $T(n) \in O(n)$ for the Base Case + 1

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \leq c \cdot n$)
Base Case + 1: n = 2

$$T(2) \leq c \cdot 2$$

$$T(1) + c_1 \leq 2c$$

$$c_0 + c_1 \leq 2c$$

We already know there's a $c \geq c_0$, so…

Prove $T(n) \in O(n)$ for the Base Case + 1

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \leq c \cdot n$)

Base Case + 1: n = 2

$$T(2) \leq c \cdot 2$$

$$T(1) + c_1 \leq 2c$$

$$c_0 + c_1 \leq 2c$$

We already know there's a $c \geq c_0$, so…

True for any $c \geq c_1$

Prove $T(n) \in O(n)$ for the Base Case + 2

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \leq c \cdot n$)
Base Case + 2: n = 3

$$T(3) \leq c \cdot 3$$

Prove $T(n) \in O(n)$ for the Base Case + 2

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \le c \cdot n$)
Base Case + 2: n = 3

Expand $T(3)$ based on the definition of T

$$T(3) \le c \cdot 3$$

$$T(2) + c_1 \le 3c$$

Prove $T(n) \in O(n)$ for the Base Case + 2

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \leq c \cdot n$)

Base Case + 2: n = 3

$$T(3) \leq c \cdot 3$$

$$T(2) + c_1 \leq 3c$$

We know there's a c s.t. $T(2) \leq 2c$...,

Prove $T(n) \in O(n)$ for the Base Case + 2

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \leq c \cdot n$)
Base Case + 2: n = 3

$$T(3) \leq c \cdot 3$$

$$T(2) + c_1 \leq 3c$$

We know there's a c s.t. $T(2) \leq 2c$…therefore $T(2) + c_1 \leq 2c + c_1$,

Prove $T(n) \in O(n)$ for the Base Case + 2

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \leq c \cdot n$)
Base Case + 2: n = 3

$$T(3) \leq c \cdot 3$$

$$T(2) + c_1 \leq 3c$$

We know there's a c s.t. $T(2) \leq 2c$…therefore $T(2) + c_1 \leq 2c + c_1$,

So if we show that $2c + c_1 \leq 3c$, then $T(2) + c_1 \leq 2c + c_1 \leq 3c$

Prove $T(n) \in O(n)$ for the Base Case + 2

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \leq c \cdot n$)
Base Case + 2: n = 3

$T(3) \leq c \cdot 3$

$T(2) + c_1 \leq 3c$

We know there's a c s.t. $T(2) \leq 2c$…therefore $T(2) + c_1 \leq 2c + c_1$,

So if we show that $2c + c_1 \leq 3c$, then $T(2) + c_1 \leq 2c + c_1 \leq 3c$

True for any $c \geq c_1$

Prove $T(n) \in O(n)$ for the Base Case + 3

Prove: $T(n) \in O(n)$ (ie: there exists a constant, c, such that $T(n) \leq c \cdot n$)

Base Case + 2: n = 4

$$T(4) \leq c \cdot 4$$

$$T(3) + c_1 \leq 4c$$

We know there's a c s.t. $T(3) \leq 3c$…therefore $T(3) + c_1 \leq 3c + c_1$,

So if we show that $3c + c_1 \leq 4c$, then $T(3) + c_1 \leq 3c + c_1 \leq 4c$

True for any $c \geq c_1$

Proving the Hypothesis Inductively

We're starting to see a pattern…

Proving the Hypothesis Inductively

We can prove our hypothesis for specific values of n…

n=1 n=2 n=3 n=4 n=5 n=6 n=7 n=8 …

Proving the Hypothesis Inductively

We can prove our hypothesis for specific values of n…

n=1 n=2 n=3 n=4 n=5 n=6 n=7 n=8 …

Proving the Hypothesis Inductively

We can prove our hypothesis for specific values of n…

n=1	n=2	n=3	n=4	n=5	n=6	n=7	n=8	…

Proving the Hypothesis Inductively

We can prove our hypothesis for specific values of n…

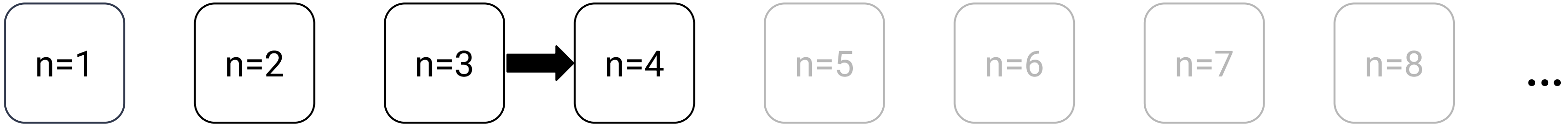
…but there are infinitely many possible values of n

n=1 n=2 n=3 n=4 n=5 n=6 n=7 n=8 …

Proving the Hypothesis Inductively

We can prove our hypothesis for specific values of n…

…but there are infinitely many possible values of n



Instead, let's prove that we can derive an unproven case from a proven one!

Proving the Hypothesis Inductively

Approach: Assume our hypothesis is true for any $n' < n$;
Now prove it must also hold true for n.

Proving the Hypothesis Inductively

Assume: There is a $c > 0$ s.t. $T(n - 1) \leq c \cdot (n - 1)$

Prove: There is a $c > 0$ s.t. $T(n) \leq c \cdot n$

$$T(n) \leq c \cdot n$$

Proving the Hypothesis Inductively

Assume: There is a $c > 0$ s.t. $T(n - 1) \leq c \cdot (n - 1)$

Prove: There is a $c > 0$ s.t. $T(n) \leq c \cdot n$

$$T(n) \leq c \cdot n$$

$$T(n - 1) + c_1 \leq c \cdot n$$

Expand $T(n)$ based on the definition of T

Proving the Hypothesis Inductively

Assume: There is a $c > 0$ s.t. $T(n - 1) \leq c \cdot (n - 1)$

Prove: There is a $c > 0$ s.t. $T(n) \leq c \cdot n$

$$T(n) \leq c \cdot n$$

$$T(n - 1) + c_1 \leq c \cdot n$$

By the inductive assumption, there is a c s.t. $T(n - 1) \leq (n - 1)c$

Proving the Hypothesis Inductively

Assume: There is a $c > 0$ s.t. $T(n - 1) \leq c \cdot (n - 1)$

Prove: There is a $c > 0$ s.t. $T(n) \leq c \cdot n$

$$T(n) \leq c \cdot n$$

$$T(n - 1) + c_1 \leq c \cdot n$$

By the inductive assumption, there is a c s.t. $T(n - 1) \leq (n - 1)c$

So if we show that $(n - 1)c + c_1 \leq nc$, then…

Proving the Hypothesis Inductively

Assume: There is a $c > 0$ s.t. $T(n - 1) \leq c \cdot (n - 1)$

Prove: There is a $c > 0$ s.t. $T(n) \leq c \cdot n$

$$T(n) \leq c \cdot n$$

$$T(n - 1) + c_1 \leq c \cdot n$$

By the inductive assumption, there is a c s.t. $T(n - 1) \leq (n - 1)c$

So if we show that $(n - 1)c + c_1 \leq nc$, then…

$$T(n - 1) + c_1 \leq (n - 1)c + c_1 \leq nc$$

Proving the Hypothesis Inductively

Assume: There is a $c > 0$ s.t. $T(n - 1) \leq c \cdot (n - 1)$

Prove: There is a $c > 0$ s.t. $T(n) \leq c \cdot n$

$$T(n) \leq c \cdot n$$

$$T(n - 1) + c_1 \leq c \cdot n$$

By the inductive assumption, there is a c s.t. $T(n - 1) \leq (n - 1)c$

So if we show that $(n - 1)c + c_1 \leq nc$, then…

$$T(n - 1) + c_1 \leq (n - 1)c + c_1 \leq nc$$

True for any $c \geq c_1$

Proving the Hypothesis Inductively

Assume: There is a $c > 0$ s.t. $T(n - 1) \leq c \cdot (n - 1)$

Prove: There is a $c > 0$ s.t. $T(n) \leq c \cdot n$

$$T(n) \leq c \cdot n$$

$$T(n - 1) + c_1 \leq c \cdot n$$

By the inductive assumption, there is a c s.t. $T(n - 1) \leq (n - 1)c$

So if we show that $(n - 1)c + c_1 \leq nc$, then…

$$T(n - 1) + c_1 \leq (n - 1)c + c_1 \leq nc$$

True for any $c \geq c_1$

Therefore, we've proven our hypothesis for the Base Case, and inductively for the Recursive Case.
Therefore, the complexity of factorial is $\Theta(n)$

But wait...

How did that prove $T(n) \in O(n)$, it was based on an assumption!?

But wait...

How did that prove $T(n) \in O(n)$, it was based on an assumption!?

If we are trying to prove some proposition, $P(n)$, the inductive step of the proof **doesn't** prove $P(n - 1)$ is true! It **doesn't** prove $P(n)$ is true!

All it proves is $P(n - 1) \rightarrow P(n)$

Inductive Proofs

This is why we have to prove a base case!

By itself, all the inductive step tells us is $\boldsymbol{P(n - 1) \rightarrow P(n)}$

On its own, that does nothing for us…

…but when combined with a base case, for example proving $\boldsymbol{P(1)}$, we now have everything we need to state that $\boldsymbol{P(n)}$ is true for all $\boldsymbol{n}$!