CSCI 136: Data Structures and Advanced Programming

Lecture 10

Recursion, part 2

Instructor: Dan Barowy

Williams

Topics

- Quiz 2—best case
- Recursion costs
- Mathematical Induction

Your to-dos

1. Lab 3, **due Tuesday 3/1 by 10pm**
2. Read **before Wed**: Bailey, Ch 9.4–9.5.

Announcements

- Lab 1: feedback today
- Lab 1: if feedback has mistakes…

What is the “best case”?

```
public static <E> void rev(Vector<E> orig, Vector<E> flip) {
  for (int i = orig.size() - 1; i >= 0; i--) {
    flip.add(orig.get(i));
  }
}
```

2. What is the *best case* runtime for `rev` in Big-O notation? You can assume that n is the length of `orig`.
   Note: write an upper bound using O as in “$O(g(n))$”

   Your answer: _____

Does the best case depend on *n*?

(where **n** is the size of the input)

```
public static <E> void rev(Vector<E> orig, Vector<E> flip) {
  for (int i = orig.size() - 1; i >= 0; i--) {
    flip.add(orig.get(i));
  }
}
```

Yes. Don’t be fooled by the fact that ***n* can be 0**.

```
public static int findFirstSpace(String s) {
    for(int i = 0; i < s.length(); i++) {
        if(s.charAt(i) == ' ') {
            System.out.println("Found space at i = " + i);
            return i;
        }
    }
    return -1;
}
```

No.

The algorithm can finish early **even if *n* is large**.

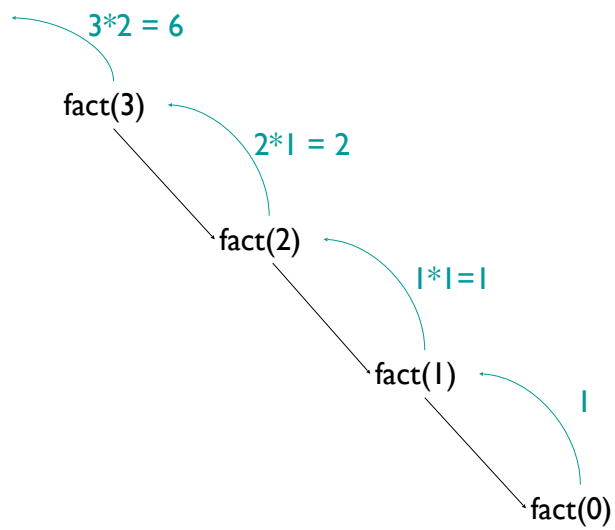
Practice Quiz

Recall: Factorial

- `n! = n × (n-1) × (n-2) × … × 1`
- Work with a partner and see if you can come up with a recursive solution.

How much does a **recusive** solution **cost**?

Graphically…

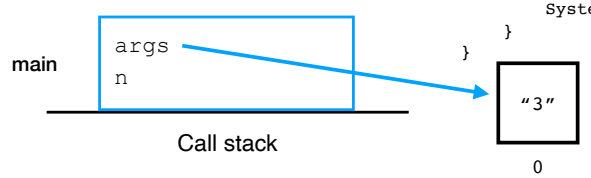



```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

Call stack

Call program with input "3".

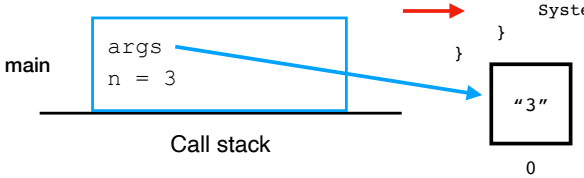
```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```



Call stack

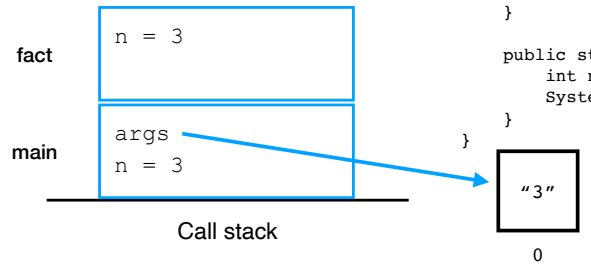
Call program with input "3".

```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

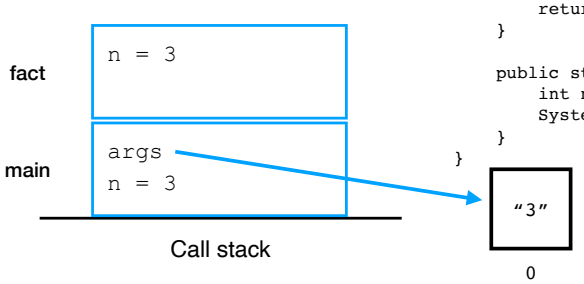


I skipped a subtlety here; did you spot it?

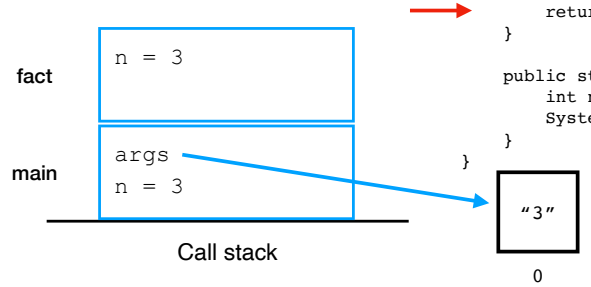
```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

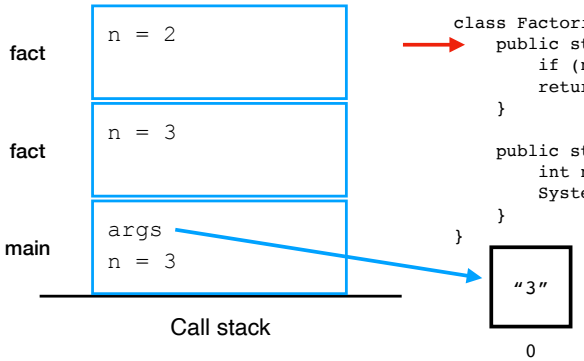


```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

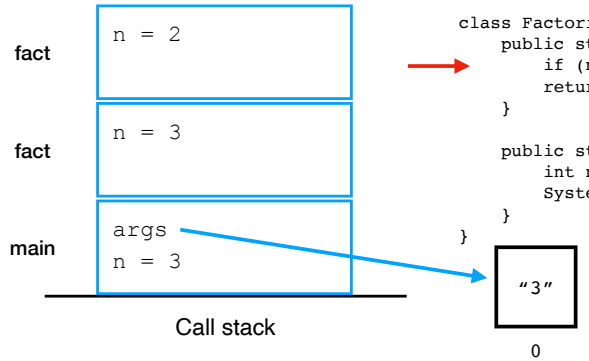


```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

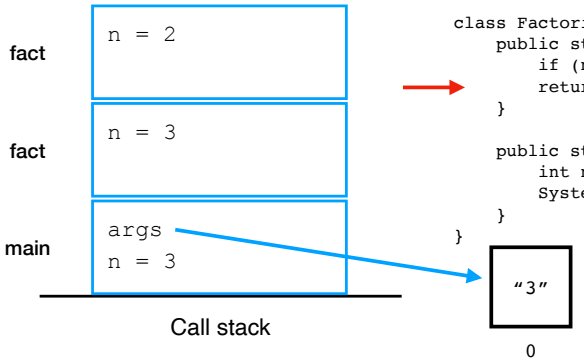




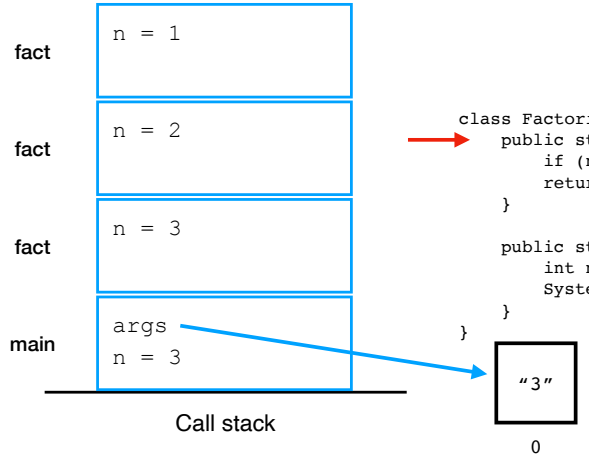
```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```



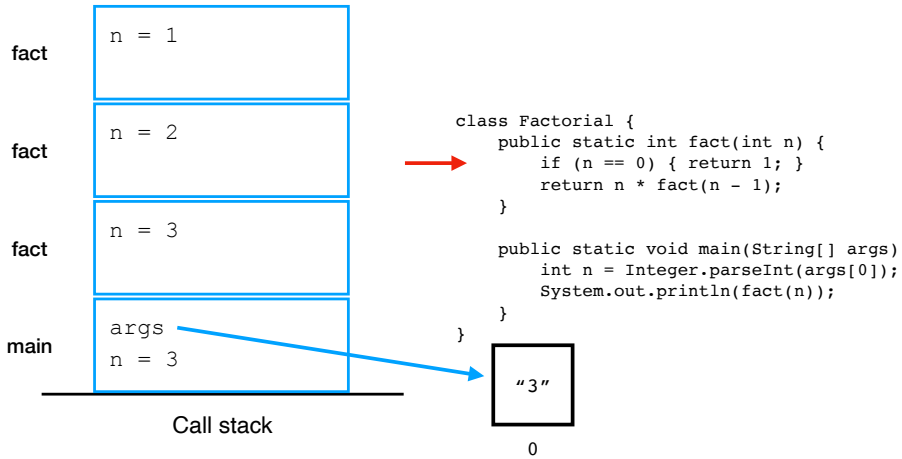
```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```



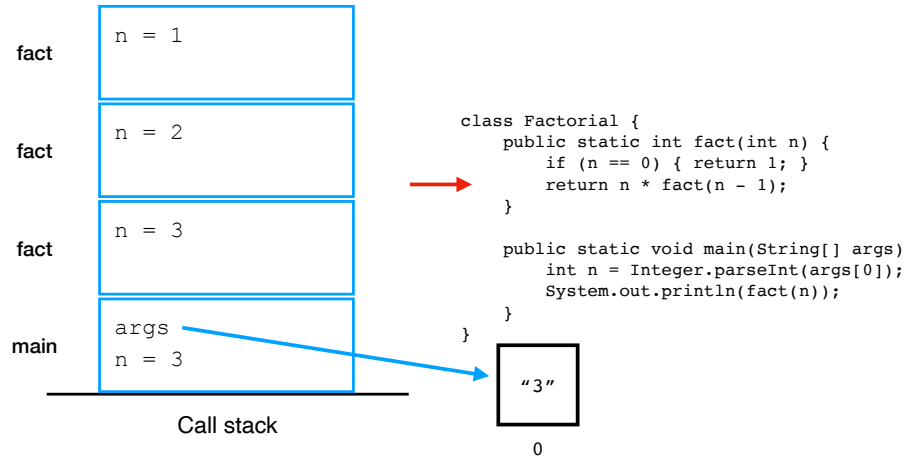
```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```



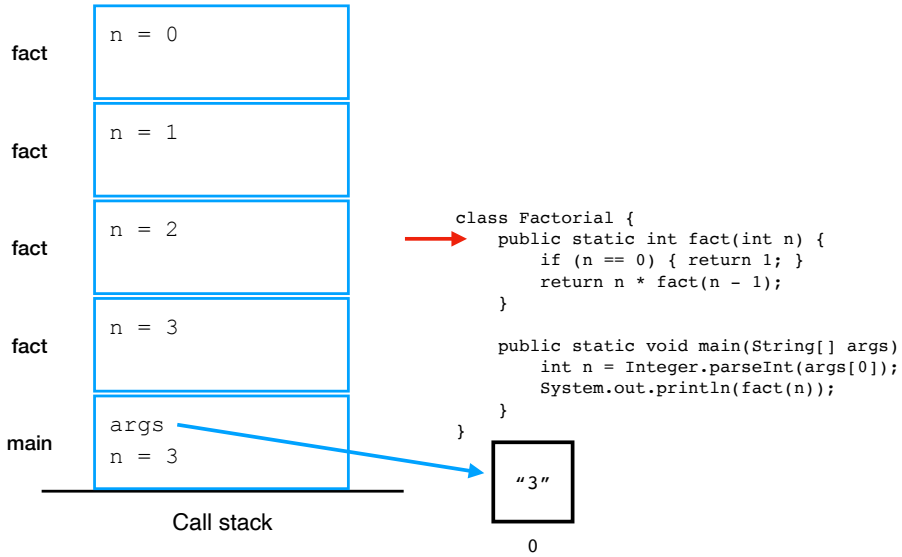
```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```



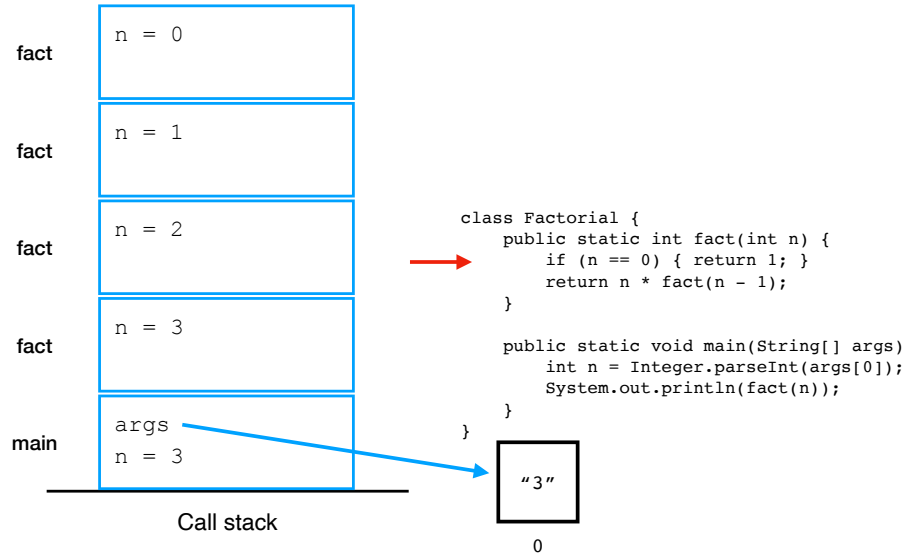

```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```




```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

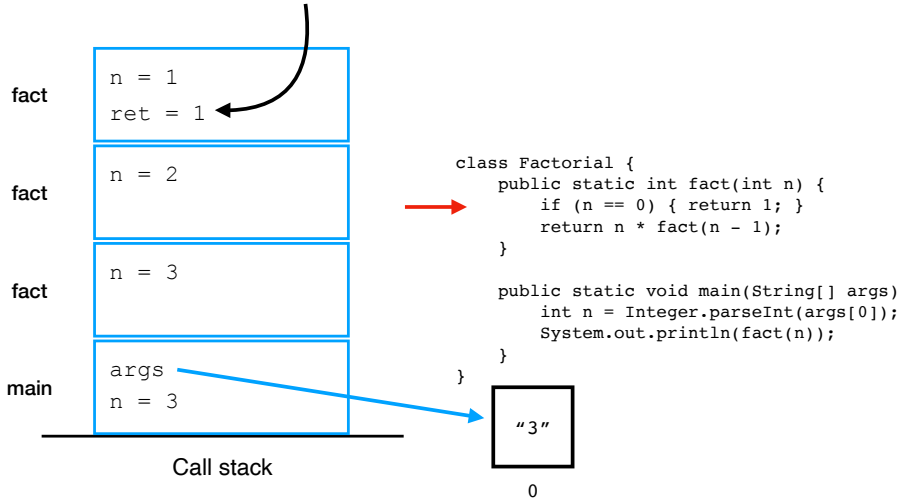



```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

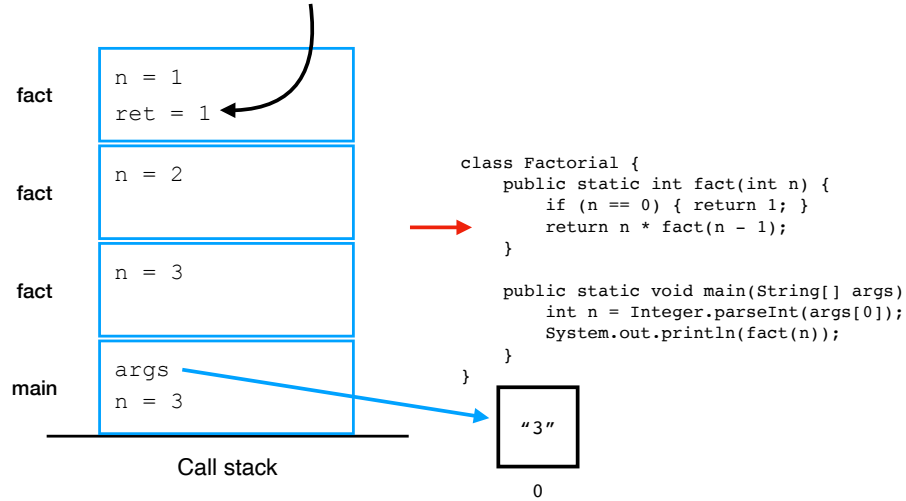



```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

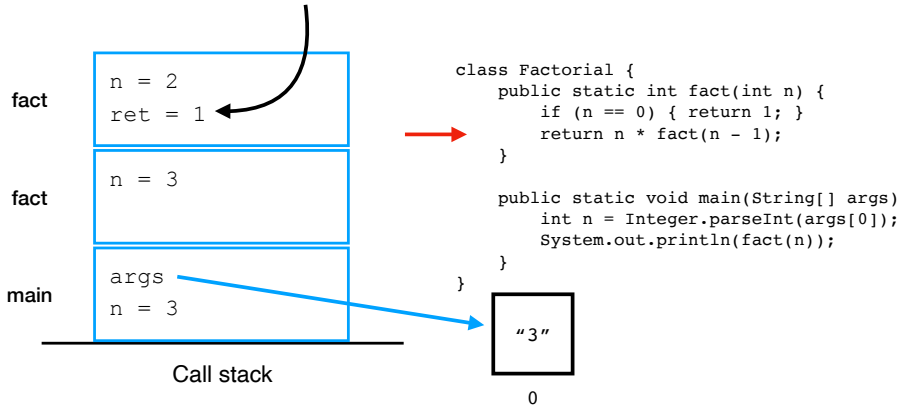
Base case: recursion terminates.



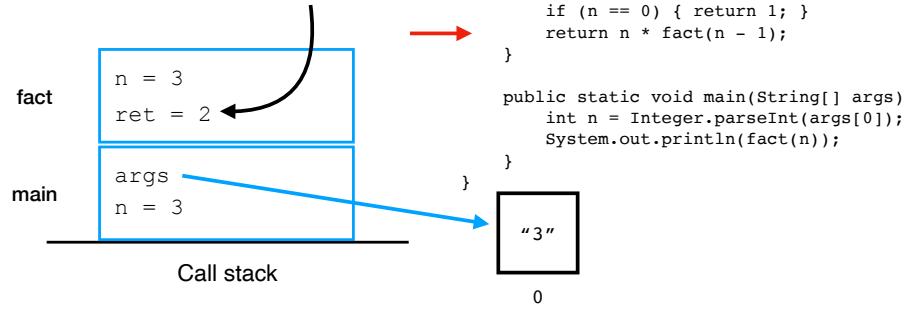

```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```




```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

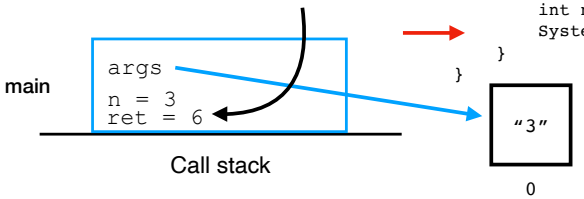



```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

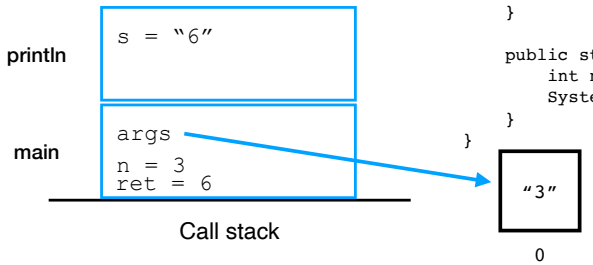



```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```

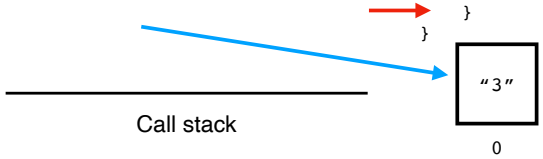



```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```



I skipped another subtlety here; did you spot it?

```
class Factorial {
    public static int fact(int n) {
        if (n == 0) { return 1; }
        return n * fact(n - 1);
    }

    public static void main(String[] args)
        int n = Integer.parseInt(args[0]);
        System.out.println(fact(n));
    }
}
```



Recursion tradeoffs

- Advantages
  - Often **easier** to construct recursive solution
  - Code is usually **cleaner**
  - Some problems do not have **obvious** non-recursive solutions
- Disadvantages
  - **Time cost** of recursive calls
  - **Memory cost** (need to store state for each recursive call until base case is reached)

Mathematical Induction

A note about “formal methods”

If the problem “fits” the mold, there is
a procedure for determining truth.

Mathematical Induction

- The **mathematical cousin** of **recursion** is **induction**
- Induction is a **proof technique**
- Purpose: to **simultaneously prove** an **infinite number** of theorems!

Principle of Mathematical Induction

Let **P(n)** be a **predicate** that is defined for **integers n**, and let **a** be a **fixed integer**.

If the following two statements are **true**:

1. **P(a)** is **true**.
2. For all integers **k ≥ a**, **if P(k)** is **true then P(k + 1)** is **true**.

then the statement

for all integers **n ≥ a**, **P(n)** is **true**

is **also true**.

Principle of Mathematical Induction (variant)

Let **P(n)** be a **predicate** that is defined for **integers n**, and let **a** be a **fixed integer**.

If the following two statements are **true**:

1. **P(a)** is **true**.
2. For all integers **k > a**, **if P(k-1)** is **true then P(k)** is **true**.

then the statement

for all integers **n ≥ a**, **P(n)** is **true**

is **also true**.

To be clear:

If you want to prove that **P(n)** is **true** for all integers **n ≥ a,**

1. You must first prove that **P(a)** is **true**.

2. Then you must prove that:

For all integers **k ≥ a**, **if P(k)** is **true then P(k+1)** is **true**.

Critically, when proving #2, **assume** that **P(k)** is **true** and **show** that **P(k+1) must also be true**.

Names for things and "form"

Hypothesis: **P(n)** is **true** for all integers **n ≥ a,**

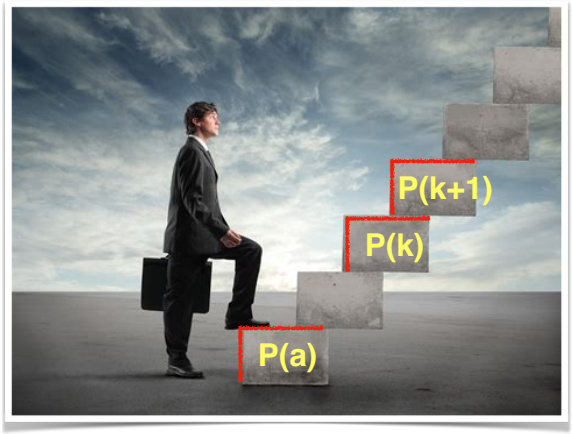
1. <u>Base case</u>: **P(a)** is **true**.

2. <u>Inductive step</u>:

For all integers **k ≥ a**, **if P(k)** is **true then P(k+1)** is **true**.

Like recursion, there is an analogy

Like recursion, there is an analogy



Example

Prove that the sum of the first n integers is:

$$\frac{n(n+1)}{2}$$

Example

Put another way, prove

$$\mathbf{P(n)} : 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$$

for all $n \geq 1$.

We have an unbounded number of hypotheses ("for all $n \geq 1$").

Use **mathematical induction**.

Remember the template!

Step 1: Prove **P(a)**

Step 2: Prove **P(k) ⇒ P(k+1)**

Therefore,

$$\mathbf{P(n)} : 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$$

For all $n \geq 1$.

Is **true**.

Example

Step 1: Prove **P(a)**

What would a good **a** be?

$$P(n) : 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$$

The "simplest" instance is **a = 1**. Let's start there.

Example

Step 1: Prove **P(a)**

$$P(a) : 1 = \frac{1(1+1)}{2}$$

Is this statement true? **Yes.**

Proof: $\frac{1(1+1)}{2} = \frac{2}{2} = 1$

Example

Step 2: Prove **P(k) ⇒ P(k+1)**

Assume the following is true:

$$\mathbf{P(k)} : 1 + 2 + 3 + \ldots + k = \frac{k(k+1)}{2}$$

Prove:

$$\mathbf{P(k+1)} : 1 + 2 + 3 + \ldots + (k + 1) = \frac{(k+1)((k+1)+1)}{2}$$

Example

Step 2: Prove **P(k) ⇒ P(k+1)**

$$\mathbf{P(k+1)} : 1 + 2 + 3 + \ldots + (k + 1) = \frac{(k+1)((k+1)+1)}{2}$$

Let's handle the left side first.

$$1 + 2 + 3 + \ldots + (k + 1)$$

Looks familiar. Isn't it the same as:

$$(1 + 2 + 3 + \ldots + k) + (k + 1)$$

Example

Step 2: Prove **P(k) ⇒ P(k+1)**

$$(1 + 2 + 3 + \ldots + k) + (k + 1)$$

According to P(k), which is true,
it must be equal to:

$$(1 + 2 + 3 + \ldots + k) + (k + 1) = \frac{k(k+1)}{2} + (k + 1)$$

Example

Step 2: Prove **P(k) ⇒ P(k+1)**

Simplify

$$= \frac{k(k+1)}{2} + (k + 1)$$

$$= \frac{k(k+1)}{2} + \frac{2(k+1)}{2}$$

$$= \frac{k(k+1) + 2(k+1)}{2}$$

Let's stop here.
The left side is

$$= \frac{(k+1)(k+2)}{2}$$

Example

Step 2: Prove **P(k) ⇒ P(k+1)**

P(k+1) : $1 + 2 + 3 + \ldots + (k + 1) = \frac{(k+1)((k+1)+1)}{2}$

Let's handle the right side now.

$$\frac{(k+1)((k+1)+1)}{2}$$

Simplify

$$\frac{(k+1)(k+2)}{2}$$

Let's stop here.

Example

Step 2: Prove **P(k) ⇒ P(k+1)**

P(k+1) : $1 + 2 + 3 + \ldots + (k + 1) = \frac{(k+1)((k+1)+1)}{2}$

We just showed that the left side

$$\frac{(k+1)(k+2)}{2}$$

equals the right side

$$\frac{(k+1)(k+2)}{2}$$

Example

Step 1: Prove **P(a)** ✓

Step 2: Prove **P(k) ⇒ P(k+1)** ✓

Therefore,

$$\mathbf{P(n)} : 1 + 2 + 3 + \ldots + n = \frac{n(n+1)}{2}$$

For all $n \geq 1$.

Is **true**. ✓

Recap & Next Class

Today:

- Recursion costs
- Mathematical induction

Next class:

- ADTs
- Lists