“The main purpose of deliberate practice is to develop effective mental representations, and, as we will discuss shortly, mental representations in turn play a key role in deliberate practice. The key change that occurs in our adaptable brains in response to deliberate practice is the development of better mental representations, which in turn open up new possibilities for improved performance.”

-- Peak: Secrets from the New Science of Expertise by Ericsson and Pool

CSCI 136:
Data Structures
and
Advanced Programming
Lecture 10
Recursion, part 2

Instructor: Kelly Shaw
Williams

Topics

• Recursion costs
• Mathematical Induction

Your to-dos

1. Lab 3, due Tuesday 10/4 by 10pm
2. Read before Mon: Bailey, Ch 9.4–9.5.
3. Weekly quiz, due Saturday 10/1 at 12pm
Recall: Factorial

- \( n! = n \times (n-1) \times (n-2) \times \ldots \times 1 \)

How much does a recursive 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 program with input “3”.

Call stack
Call program with input “3”.

I skipped a subtlety here; did you spot it?
```java
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));
    }
}
```
```java
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:
```
main
  args
    n = 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:
```
main
  args
    n = 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:
```
main
  args
    n = 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:
```
main
  args
    n = 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:
```
main
  args
    n = 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:
```
main
  args
    n = 3
```
```java
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.
```java
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));
    }
}
```

```java
Call stack
main
args
n = 3
"3"
0
ret = 2
fact
n = 3
ret = 6
```

```java
println
x = 6
```

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

```java
Call stack
main
args
n = 3
"3"
0
ret = 6
```
```java
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 than a loop
  - Code is usually clearer
  - 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 $k \geq a$, if $P(k)$ is true then $P(k + 1)$ is true.

then the statement

for all integers $n \geq a$, $P(n)$ 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 $n \geq a$, $P(n)$ is true

is also true.

To be clear:

If you want to prove that $P(n)$ is true for all integers $n \geq a$,

1. You must first prove that $P(a)$ is true.
2. Then you must prove that:

   For all integers $k \geq a$, 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 \( n \geq a \),

1. **Base case**: \( P(a) \) is true.

2. **Inductive step**: For all integers \( k \geq a \), 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

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

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 \( P(k) \Rightarrow P(k+1) \)

Therefore,

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

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 \( P(k) \Rightarrow P(k+1) \)

Assume the following is true:

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

Prove that \( P(k) \) implies:

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

Let’s handle the left side first.

\[ 1 + 2 + 3 + \ldots + (k + 1) \]

Looks familiar. Isn’t it the same as:

\[ (1 + 2 + 3 + \ldots + k) + (k + 1) \]

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 \( P(k) \Rightarrow P(k+1) \)

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

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 $P(k) \Rightarrow P(k+1)$

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

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 $P(k) \Rightarrow P(k+1)$

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

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 $P(k) \Rightarrow P(k+1)$

Therefore,

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

For all $n \geq 1$.

Is true.

Recap & Next Class

Today:

• Recursion costs
• Mathematical induction

Next class:

• Vector doubling
• ADTs
• Lists
Expanding vectors: why double?

Why is the **array doubling** strategy for Vector **better** than expanding the array **one element at a time**?

One-at-a-time expansion

Initial array.

Insert element.

New array; copy previous; insert element.

New array; copy previous; insert element.

New array; copy previous; insert element.

Insertion into an array

How much does **array insertion** cost?

It costs \( O(1) \).

In fact, lookup and insertion both cost \( O(1) \).

Tradeoff: arrays are fixed size.

Copying an array

How much does an **array copy** cost?

It costs \( O(1) \times m \), where \( m \) is the size of the original array.

\( \approx O(m) \)
One-at-a-time expansion costs?

(in the worst case, each time)

Initial array.

Insert element.

New array; copy previous; insert element.

$O(m) + O(1) \approx O(m)$, where $m$ is the size of the original array.

Cost is dominated by the size of the array being copied.

How many copies?

# of copies for one-at-a-time expansion:

\[
\begin{align*}
1 \quad & \quad 2 \quad & \quad 3 \quad & \quad \ldots \quad & \quad + (n-1) \\
\text{add( )} \quad & \quad \text{2nd} \quad & \quad \text{3rd} \quad & \quad \text{4th} \quad & \quad \ldots \quad & \quad \text{nth} \\
\text{elem.} \quad & \quad \text{elem.} \quad & \quad \text{elem.} \quad & \quad \ldots \quad & \quad \text{elem.}
\end{align*}
\]

Recall theorem: \( 1 + 2 + 3 + \ldots + k = \frac{k(k+1)}{2} \)

Sub $n-1$ for $k$: \( (n-1) \left( \frac{(n-1)+1}{2} \right) = \frac{n(n-1)}{2} \)

\( = \frac{n^2 - n}{2} \)

One-at-a-time expansion costs \( \approx O(n^2) \)