CSCI 261
Computer Science II
Outline

- Recursion
  - Definition and Setup
  - Tracing
  - Pitfalls
  - Running Time
- Examples
- Recursive Data Structures
- Conclusion
Thinking Recursively

- Recursion is the lazy approach to looping.
  - *(Demo: Adding up a list of ints)*

- When given a problem, ask two questions:
  - For what problem size does the solution become obvious?
  - What's the *least* we can do to *reduce* the problem size?
    - Do that tiny unit work to drive problem size toward the obvious case above.
    - Call itself on smaller problem size to recursively solve remaining problem.
Recursion

- Recursive algorithms are characterized by:
  
  • **Base case**: The problem is trivial! Solution is usually obvious.
    - Usually (doesn't always) returns a constant value to the caller, or does nothing.
    - *Does not* call itself (terminates recursion!)
  
  • **Recursive case**: Problem decomposition
    - Do a tiny unit of work to reduce problem into a sub-problem of the same nature.
      - Make sure the sub-problem approaches the base case!
    - Call *itself* to solve the smaller sub-problem
Example: MultiPrint

- Suppose I want to print a string \( n \) times:

```plaintext
multiPrint("Hello World", 4);
> Hello World
> Hello World
> Hello World
> Hello World

multiPrint("Hello World", 1);
> Hello World

multiPrint(0);
```
Example: MultiPrint (Version 1)

- Rewritten to remove empty if-statement

```java
public void multiPrint(String str, int n) {
    if (n != 0) {
        System.out.println(str);
        multiPrint(str, n-1);
    }
}
```

- Recursive case
  - Print the String once (the minimal work to reduce problem size)
  - Recursively `multiPrint()` the remaining `n-1` strings

- Base case
  - When `n == 0`: we don't want to do anything (That terminates recursion!)
Stack Overflow (Caused by Infinite Recursion)

- Current implementation:

```java
public void multiPrint(String str, int n) {
    if (n != 0) {
        System.out.println(str);
        multiPrint(str, n-1);
    }
}
```

- What if we called `multiPrint("Hello", -1);`?

```
> Hello
> Hello
> Hello
> Hello
> Hello
> Hello
...
java.lang.StackOverflowError
```
Example: MultiPrint (Final)

- Re-written to avoid stack overflow

```java
public void multiPrint(String str, int n) {
    if (n > 0) {
        System.out.println(str);
        multiPrint(str, n-1);
    }
}
```
Outline

- Recursion
  - Definition and Setup
  - Tracing
  - Pitfalls
  - Running Time
- Examples
- Recursive Data Structures
- Conclusion
Another Example: Factorials

- The factorial \( n! \) is the product of \( n, n-1, n-2, \ldots, 1 \)

- For example:
  - \( 7! = 7 \times 6 \times 5 \times 4 \times 3 \times 2 \times 1 = 5040 \)
  - \( 3! = 3 \times 2 \times 1 = 6 \)
  - \( 1! = 1 \)
  - \( 0! = 1 \)

- Side note
  - Factorial-time algorithms \( O(n!) \) are among the worst you can do
Factorial Expressed Recursively

/** iterative (non-recursive) */
public static long factorial(long n) {
    long fac = 1;
    for (int i = n; i > 0; i--)
        fac *= i;
    return fac;
}

/** @pre n >= 0 */
public static long factorial(long n) {
    if (n == 0) {
        return 1;
    }
    else if (n == 1) {
        return 1;
    }
    return n * factorial(n-1);
}
Let's see a trace for \texttt{factorial}(4)

- Remember that callers of recursive methods must \textit{wait} until callee returns
- Remember that each call to factorial produces its own set of:
  - Parameters (when applicable)
  - Local variables (when applicable)

Never forget to check \textit{edge cases} for recursive methods!

- Trace these edge cases:
  - \texttt{factorial}(1)
  - \texttt{factorial}(0)
Another Tracing Example

- Track contents of all variables, inputs, outputs, and return values.
  - Keep track of the call stack(!) of recursive calls that are waiting

- Another Example: What does mysteryTwo do?
  - Try a trace on mysteryTwo("AB", 2)

```java
public static void mysteryTwo(String str, int n) {
    if (n == 0) {
        return "";
    }
    return mysteryTwo(str, n-1) + str;
}
```

(Trace demo on board)
Tracing a Recursive Method Call (Important)

- Another Example: What does this mystery method do?
  - Try a trace with n = 4

```java
public static void mysteryTwo(int n) {
    if (n != 0 && n != 1) {
        mysteryTwo(n-2);
        System.out.println(n);
        mysteryTwo(n-1);
    }
}
```

- When a method makes multiple recursive calls, sometimes it's easier to visualize using a trace with a recursion tree.