Middlebury

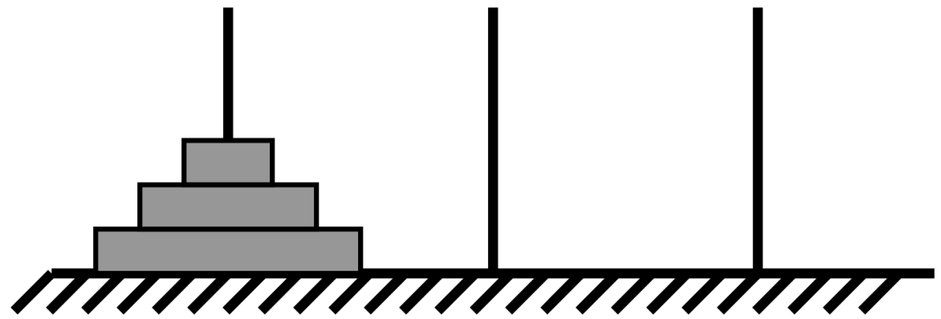
CSCI 201: Data Structures

Spring 2025

Lecture 5M: Recursion

Goals for today:

- Associate a call to a method with a **frame**.
- Use **recursion** to solve problems by building a solution from solutions to smaller problems.
- Ensure we always have a **base case** in our recursive solutions.
- Use **tail recursion** so that there are no pending operations after the recursive call.
- Practice some more with big-O notation.



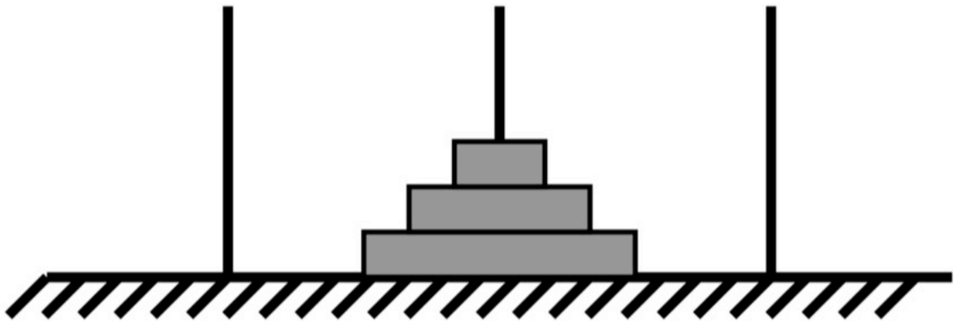
Rules: (also play here! https://www.mathsisfun.com/games/towerofhanoi.html)

1. Move one disk at a time.
2. Every move displaces a disk from the top of one stack to the top of another (or empty rod).
3. No larger disk can ever be placed on top of a smaller one.

Counting the number of "moves" in the Tower of Hanoi problem.

"Moves"

operations
$T(n)$



n	# moves
1	1
2	3
3	7
4	15
5	31

$$T(1) = 1$$

$$T(n) = 2T(n-1) + 1$$

$$= 2(2T(n-2) + 1) + 1$$

$$\cdots$$

$$= 2^n - 1 \qquad O(2^n)$$

$2^n - 1$

Two ingredients for a recursive method:

- **Base case:** represents a problem you know how to solve.
- **Recursive case:** builds the solution to a problem from smaller subproblems.

Arithmetic series:

$$1 + 2 + 3 + \cdots + (n-1) + n = \frac{n(n+1)}{2}$$

```
1 public static int sum(int n) {
2   if (n == 1) { // base case
3     return 1;
4   }
5   // recursive case
6   return n + sum(n - 1);
7 }
```

approach n==1

Factorial:

$$1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n$$

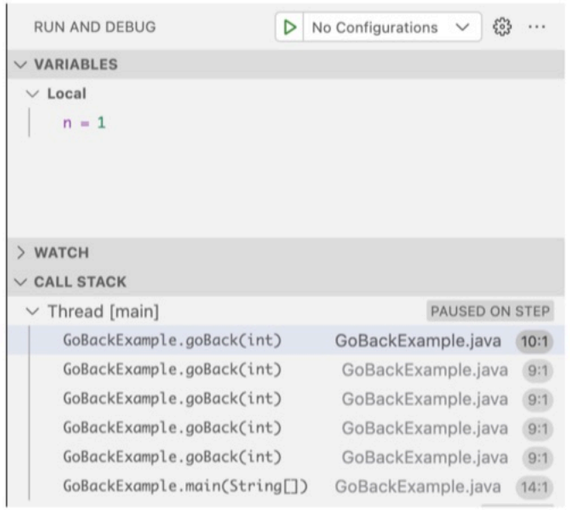
```
1 public static int factorial(int n) {
2   if (n < 2) { // base case
3     return 1;
4   }
5   // recursive case
6   return n * factorial(n - 1);
7 }
```

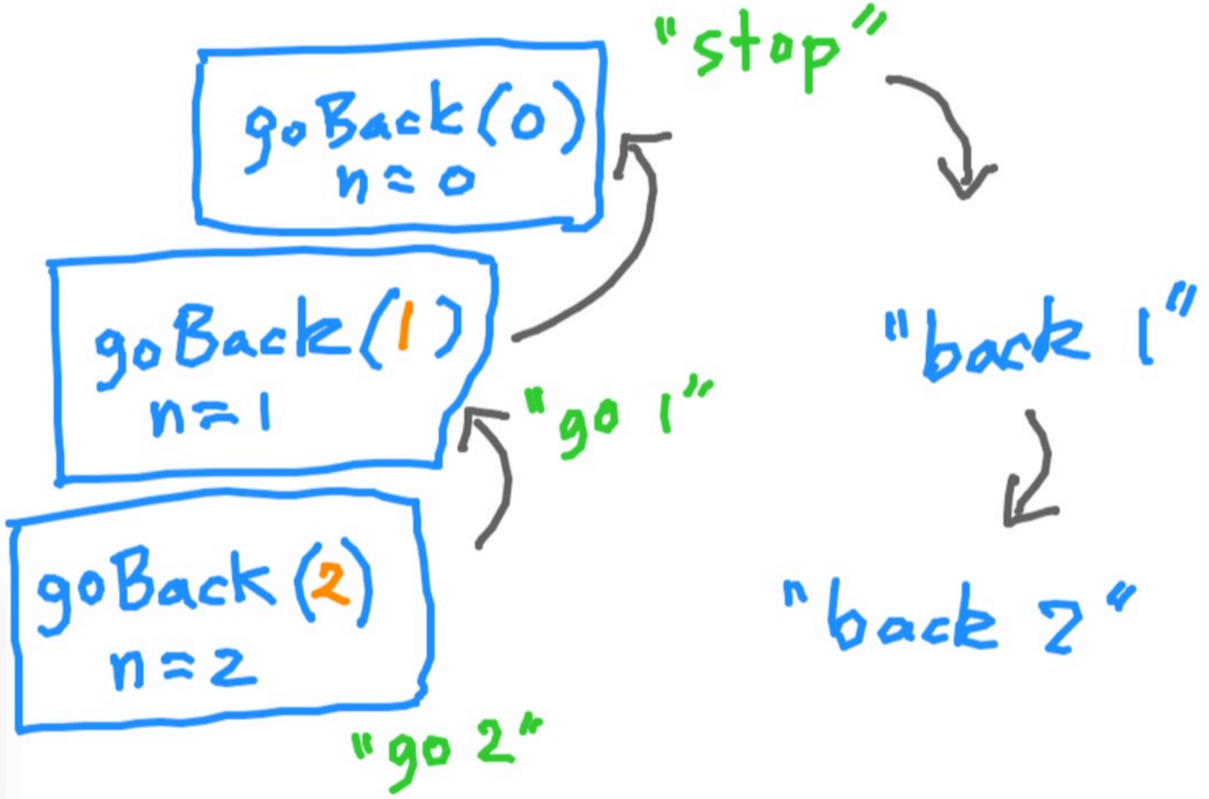
approach n<2

The recursive case needs to make progress towards the base case!

What does the `Call Stack` look like?

Open `GoBackExample.java` and add breakpoint on Line 6.






```java
public static void goBack(int n) {
  if (n == 0) {
    System.out.println("Stop");
  } else {
    System.out.println("Go " + n);
    goBack(n - 1);
    System.out.println("Back " + n);
  }
}
```

recursion (n.)

"return, backward movement," 1610s, from Latin *recursionem* (nominative *recursio*) "a running backward, return," noun of action from past-participle stem of *recurrere* "run back" (see **recur**).

Exercise: write a recursive function to compute Fibonacci numbers.

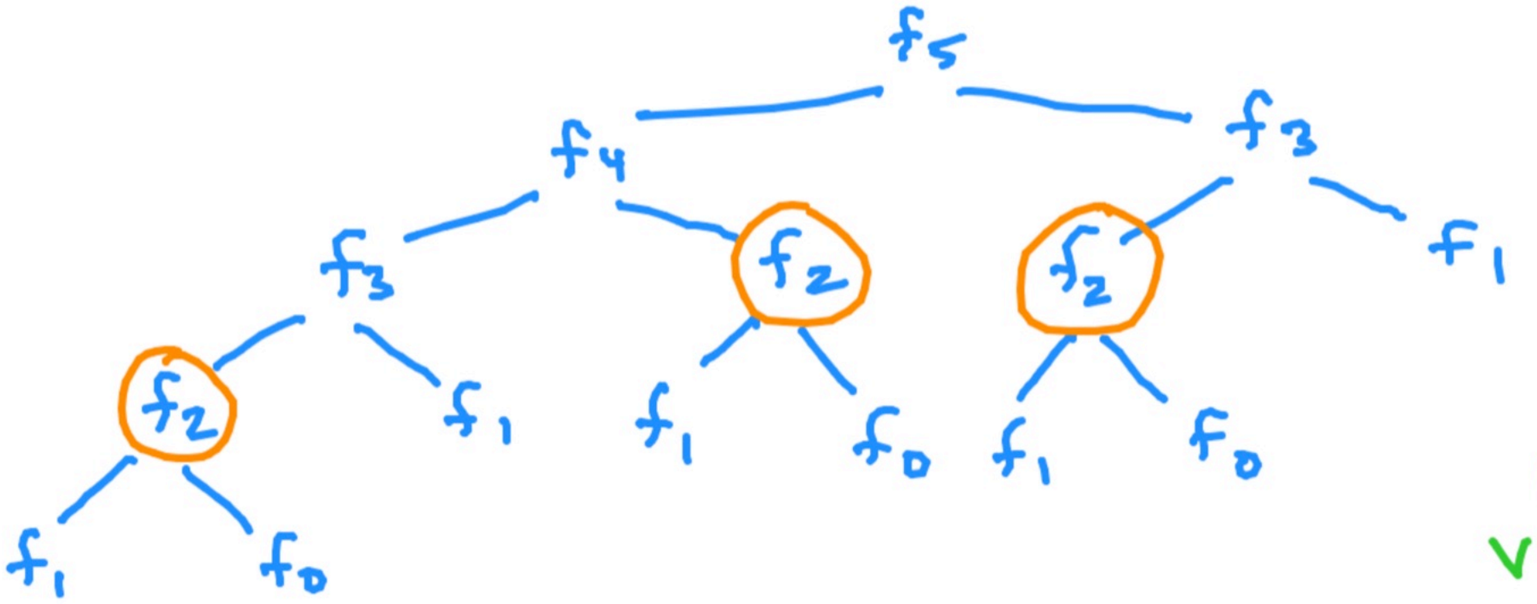
$$f(n) = f(n-1) + f(n-2), \quad \text{with } f(0) = 0,\ f(1) = 1.$$

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

levels = n

branches = 2

Try $n = 40, n = 45, n = 48$. What do you notice?



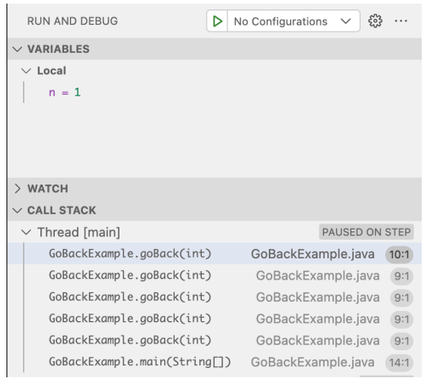
$\approx 2^n$ calls to fib

$O(2^n)$ exponential very slow!

The downsides of recursion.

- Performance might not be great.
- Recursion depth is limited by the stack size.

Go back to `GoBackExample.java` (remove breakpoints) and try $n = 100000$.

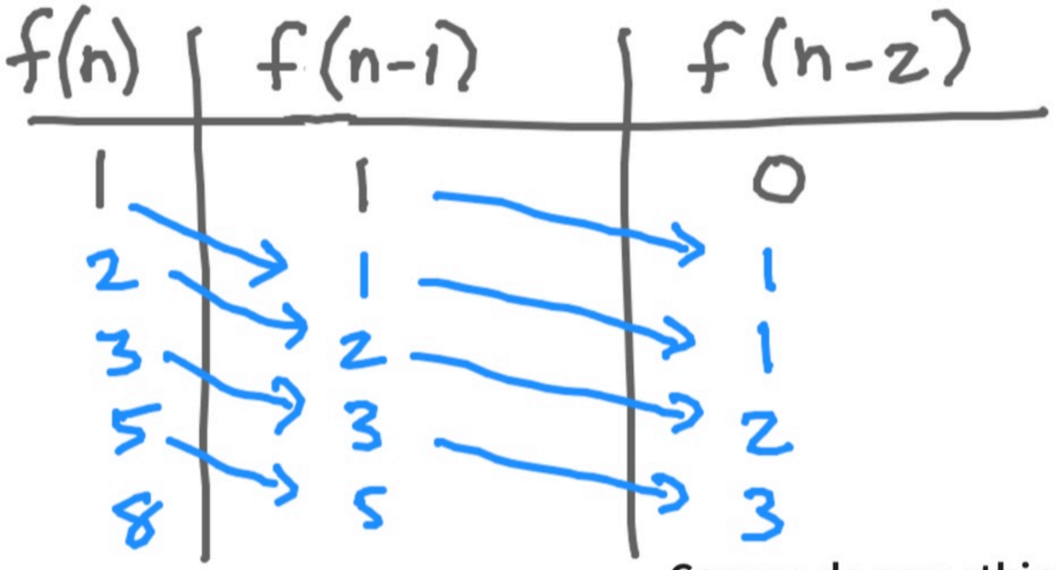



```java
public static void goBack(int n) {
  if (n == 0) {
    System.out.println("Stop");
  } else {
    System.out.println("Go " + n);
    goBack(n - 1);
    System.out.println("Back " + n);
  }
}
```

```
Go
Go
Go
Exception in thread "main" java.lang.StackOverflowError
        at java.base/java.io.FileOutputStream.write(FileOutputStream.java:349)
        at java.base/java.io.BufferedOutputStream.flushBuffer(BufferedOutputStream.java:81)
        at java.base/java.io.BufferedOutputStream.flush(BufferedOutputStream.java:142)
        at java.base/java.io.PrintStream.write(PrintStream.java:570)
        at java.base/sun.nio.cs.StreamEncoder.writeBytes(StreamEncoder.java:234)
        at java.base/sun.nio.cs.StreamEncoder.implFlushBuffer(StreamEncoder.java:313)
        at java.base/sun.nio.cs.StreamEncoder.flushBuffer(StreamEncoder.java:111)
        at java.base/java.io.OutputStreamWriter.flushBuffer(OutputStreamWriter.java:178)
        at java.base/java.io.PrintStream.writeln(PrintStream.java:723)
        at java.base/java.io.PrintStream.println(PrintStream.java:1028)
        at GoBackExample.goBack(GoBackExample.java:8)
        at GoBackExample.goBack(GoBackExample.java:9)
        at GoBackExample.goBack(GoBackExample.java:9)
```

Can we make our Fibonacci number calculation more efficient?

Let's try to write it as a `while`-loop instead.




```java
public static int fibWhile(int n) {
  int fnMinus2 = 0;
  int fnMinus1 = 1;
  int i = n;
  while (i > 1) {
    int fn = fnMinus1 + fnMinus2;
    fnMinus2 = fnMinus1;
    fnMinus1 = fn;
    i--;
  }
  return fnMinus1;
}
```

Can we do something similar using recursion?

```java
public static int fib(int n) {
  if (n < 2) return n;
  return fibHelper(n, 1, 0);
}

private static int fibHelper(int n, int fnMinus1, int fnMinus2) {
  if (n == 1) return fnMinus1;
  return fibHelper(n - 1,          ,          );
}
```

Can we make our *recursive* Fibonacci number calculation more efficient?

```java
public static int fibWhile(int n) {
  int fnMinus2 = 0;
  int fnMinus1 = 1;
  int i = n;
  while (i > 1) {
    int fn = fnMinus1 + fnMinus2;
    fnMinus2 = fnMinus1;
    fnMinus1 = fn;
    i--;
  }
  return fnMinus1;
}
```

```java
1 public static int fib(int n) {
2   if (n < 2) return n;
3   return fibHelper(n, 1, 0);
4 }
5
6 private static int fibHelper(int n, int fnMinus1, int fnMinus2) {
7   if (n == 1) return fnMinus1;
8   return fibHelper(n - 1,        ,        );
9 }
```

fnMinus1

fnMinus1 + fnMinus2

Fill in the blanks! See slido.com # 4017686

CS 201 Lecture 9

What should the recursive call to fibHelper look like on Line 8? 17

- fibHelper(n - 1, fnMinus1, fnMinus1 + fnMinus2);
- fibHelper(n - 1, fnMinus2, fnMinus1 + fnMinus2);
- fibHelper(n - 1, fnMinus1 + fnMinus2, fnMinus1);

Send

Voting as Anonymous

Two things going on here:

- Computing any given Fibonacci number *once*.
- **Tail-Recursive:** the recursive call is the **last** thing done in the method.
  - **Strategy to develop:** think about how to write it as a `while`-loop first.

```
1 public static int fib(int n) {
2   if (n < 2) return n;
3   return fibHelper(n, 1, 0);
4 }
5
6 private static int fibHelper(int n, int fnMinus1, int fnMinus2) {
7   if (n == 1) return fnMinus1;
8   return fibHelper(n - 1, fnMinus1 + fnMinus2, fnMinus1);
9 }
```

helper

How many times is `fibHelper` called?

Same as while loop

O(n)

why tail recursion?

→ some compilers detect and will optimize creation of stack frames

Exercise: rewrite `factorial` to be tail-recursive.

```
int result=1;
int i=n;
while(i>1){
  result*=i;
  i--;
}
```

```java
public static int factorial(int n) {
  if (n < 2) { // base case
    return n;
  }
  // recursive case
  return n * factorial(n - 1);
}
```

```java
public static int factorialTail(int n) {
  return factorialTailHelper(n, 1);
}

private static int factorialTailHelper(int n, int result) {
  if (n == 1) return result;
  return factorialHelper(n - 1, n * result);
}
```

In this case, we still have $\mathcal{O}(n)$ multiplications ($*$), but a compiler might perform *tail call optimization* (TCO), eliminating the need for a new stack frame (just not in Java).

Back to `GoBackExample.java`.

Try to pinpoint *exactly* the value of n that gives a stack overflow.

Design your own algorithm!

This is the algorithm *binary search*.

```java
public static int binarySearch(int value, int[] sortedArray) {
  return binarySearch(value, sortedArray, 0, sortedArray.length - 1);
}

private static int binarySearch(int value, int[] sortedArray, int left, int right) {
 if (right < left) {
   return -1;
 }

 int mid = left + (right - left) / 2;
 if (sortedArray[mid] == value) {
    return mid;
 } else if (sortedArray[mid] < value) {
    return binarySearch(value, sortedArray, mid + 1, right); // search right half
 } else {
    return binarySearch(value, sortedArray, left, mid - 1); // search left half
  }
}
```

More practice

Convert the following (recursive) string reversal method to use tail recursion.

```java
public static String reverse(String message) {
  if (message.length() < 2) return message;
  return message.charAt(message.length() - 1) + reverse(message.substring(0, message.length() - 1));
}
```

```java
public static String reverseTail(String message) {
  return reverseHelper(message, "");
}

private static String reverseHelper(String message, String result) {
  if (message.length() == 0) {
    return result;
  }
  int end = message.length() - 1;
  return reverseHelper(message.substring(0, end), result + message.charAt(end));
}
```

See you on Wednesday!

- Lab 4 (Bucket Sort) due tonight at 11:59pm.
- Work on Homework 4!
    1. Implement Radix Sort.
    2. Derive number of `=` for `DIYList add` method with a different growth technique.
- Reminder that Noah (go/noah) and Smith (go/smith) have office hours throughout the week and the 201 Course Assistants have drop-in hours in the late afternoons/evenings (go/cshelp).