Recursion

Introduction

- A repetitive algorithm is simply code that needs to be run over and over again.
- In general, there are 2 approaches to writing repetitive algorithms:
  - Iteration
  - Recursion
- We will define these 2 methods by use of an example.

Factorial

- Let’s consider a function to calculate the factorial of a number.
- For example, let’s determine the factorial of 4:

factorial(4)=4 x 3 x 2 x 1 = 24

Iterative Definition

- In general, we can define the factorial function in the following way:

$$\text{Factorial}(n) = \begin{bmatrix} 1 & \text{if } n = 0 \\ n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1 & \text{if } n > 0 \end{bmatrix}$$

Iterative Definition

- This is an ***iterative*** definition of the factorial function.
- <u>It is iterative because the definition only contains the algorithm parameters and not the algorithm itself.</u>
- This will be easier to see after defining the recursive implementation.

Recursive Definition

- We can also define the factorial function in the following way:

$$\text{Factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times (\text{Factorial}(n-1)) & \text{if } n > 0 \end{cases}$$

Iterative vs. Recursive

- **Iterative**

Function does NOT calls itself

factorial(n) =

$$\begin{cases} 1 & \text{if } n=0 \\ n \times (n-1) \times (n-2) \times \ldots \times 2 \times 1 & \text{if } n>0 \end{cases}$$

- **Recursive**

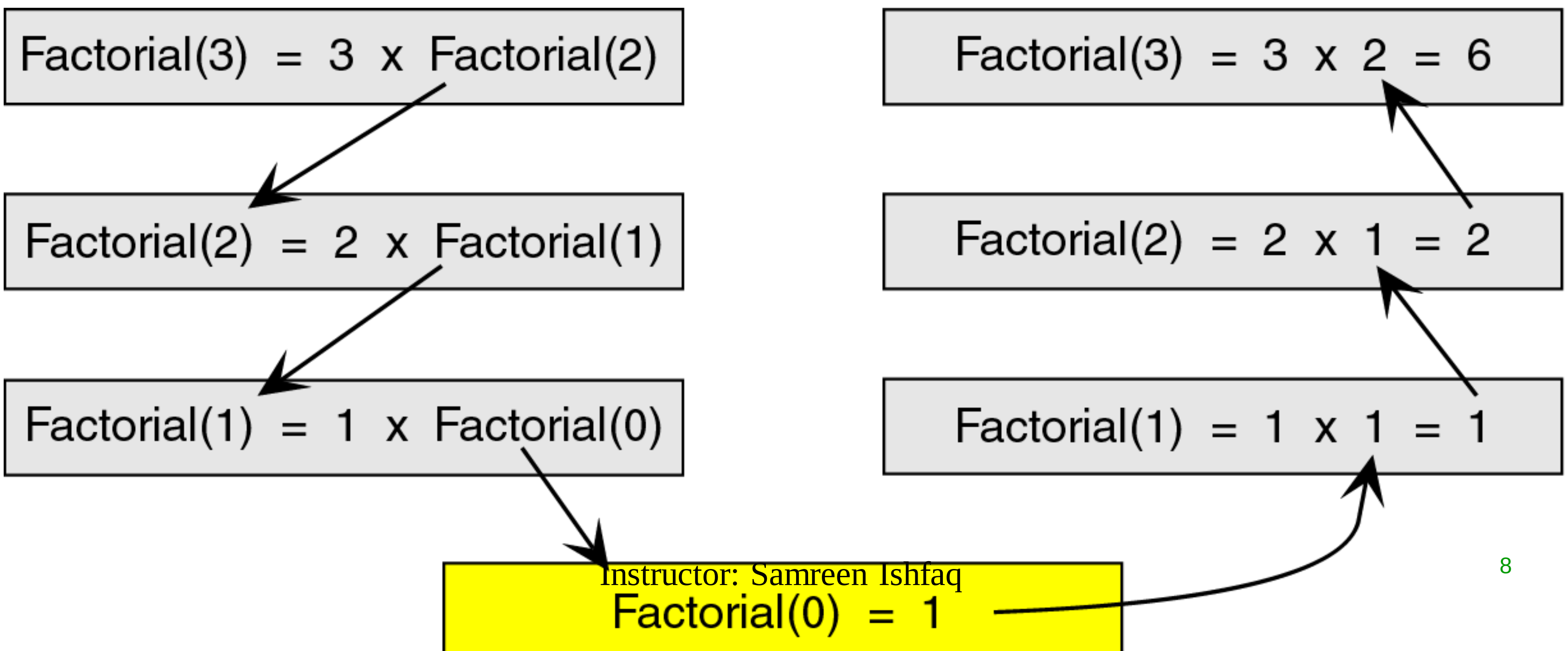
Function calls itself

factorial(n) =

$$\begin{cases} 1 & \text{if } n=0 \\ n \times \text{factorial}(n-1) & \text{if } n>0 \end{cases}$$

Recursion

- To see how the recursion works, let's break down the factorial function to solve factorial(3)



Breakdown

- This known solution is called the ***base case***.
- Every recursive algorithm must have a base case to simplify to.
- Otherwise, the algorithm would run forever (or until the computer ran out of memory).

Breakdown

- The other parts of the algorithm, excluding the base case, are known as the general case.
- For example:

    3 x factorial(2) ➔ general case
    2 x factorial(1) ➔ general case
    etc …

Breakdown

- After looking at both iterative and recursive methods, it appears that the recursive method is much longer and more difficult.
- If that's the case, then why would we ever use recursion?
- It turns out that recursive techniques, although more complicated to solve by hand, are very simple and elegant to implement in a computer.

Iterative Algorithm

```
factorial(n) {
    i = 1
    factN = 1
    loop (i <= n)
        factN = factN * i
        i = i + 1
    end loop
    return factN
}
```

Recursive Algorithm

```
factorial(n) {
   if (n == 0)
        return 1;
   else
        return n*factorial(n-1);
   end if
}
```

How Recursion Works

- When a program calls a subroutine (function) the current function must suspend its processing.
- The called function then takes over control of the program.
- When the function is finished, it needs to return to the function that called it.
- The calling function then ‘wakes up’ and continues processing.

How Recursion Works

- To do recursion we use a stack.
- Before a function is called, all relevant data is stored in a ***stackframe***.
- This stackframe is then pushed onto the system stack.
- After the called function is finished, it simply pops the system stack to return to the original state.

How Recursion Works

- By using a stack, we can have functions call other functions, which can call other functions, etc.
- Because the stack is a first-in, last-out data structure, as the stackframes are popped, the data comes out in the correct order.

Designing Recursive Algorithms

- We first note that all recursive algorithms have 2 parts:
  - General Case
  - Base Case
- Every call to a recursive algorithm must either solve a part of the problem (base case) or reduce it in size (general case).

Designing Recursive Algorithms

- Design rules:
  1) Determine the base case.
  2) Determine the general case.
  3) Combine the base case and the general case into an algorithm.
- When combining the 2 parts of the algorithm, we must make sure that we either reduce the size of the problem (i.e., move it closer to the base case) or reach the base case.

Advantages of Recursion

- Recursion is a powerful problem-solving technique that often produces very clean solutions to even the most complex problems.
- Recursive solutions can be easier to understand and to describe than iterative solutions.

Limitations of Recursion

- Recursion works the best when the algorithm and/or data structure that is used naturally supports recursion.
- One such data structure is the tree (more to come).
- One such algorithm is the binary search algorithm.

Limitations of Recursion

- Recursive solutions may involve extensive overhead because they use calls.
- When a call is made, it takes time to build a stackframe and push it onto the system stack.
- Conversely, when a return is executed, the stackframe must be popped from the stack and the local variables reset to their previous values – this also takes time.