MECHTRON 2MD3

Data Structures and Algorithms for Mechatronics

Winter 2022

13 Recursion

Department of Computing and Software

Instructor:

Omid Isfahanialamdari

February 10, 2022

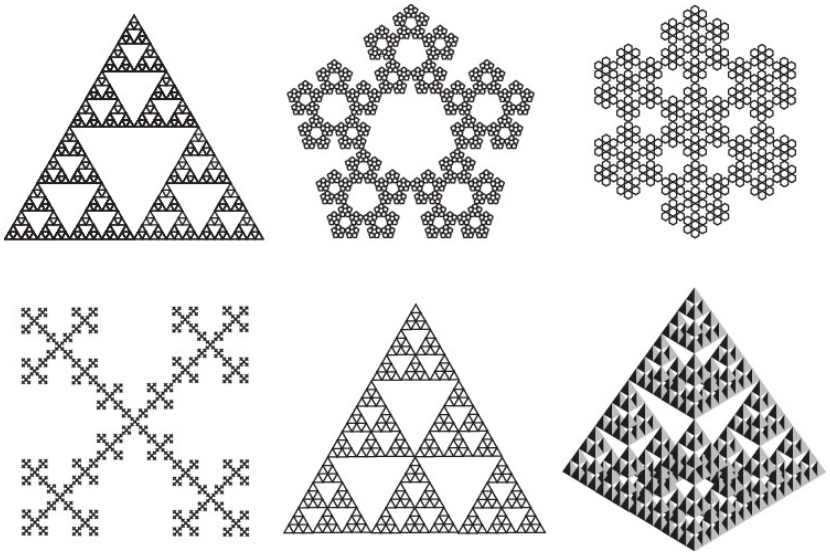
Administration

- Today's lecture is shorter
  - I will stay until 2:30 to answer questions of students who can not attend my office hour.

What is Recursion

- **Recursion** is the concept of defining a function that makes a call to itself.
- We call a function **Recursive** if it calls itself.
- When a function calls itself, we refer to this as a **Recursive Call**.
  - if function M calls another function that ultimately leads to a call back to M, this is also a recursive call and function M is recursive.

- A lot of use-cases in real-life:
  - Nature : fractals
  - Mathematics: recursive functions



Recursive Algorithms

- The idea is to avoid loops!
- A recursive implementation can be significantly simpler and easier to understand than an iterative implementation.
- Types of Recursion:
  - Linear Recursion
  - Binary Recursion
  - Tail Recursion
  - Multiple Recursion

Recursive Algorithms - Example

- Factorial of a whole number n is defined as:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1 & \text{if } n \geq 1. \end{cases}$$

  - factorial(4) = 4*(3*2*1) = 4*factorial(3)
  - factorial(4) can be defined in terms of factorial(3)
- Recursive definition of the factorial function is:

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot \text{factorial}(n-1) & \text{if } n \geq 1. \end{cases}$$

  - base case
    - are non-recursive
  - recursive case
    - no circularity (function terminates)

Recursive Algorithms - Example

- Iterative Factorial

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1 & \text{if } n \geq 1. \end{cases}$$

```
int factorial_iter(int n){
    int factorial = 1;
    for (int i = 2; i <= n; i++){
        factorial = factorial * i;
    }
    return factorial;
}
```

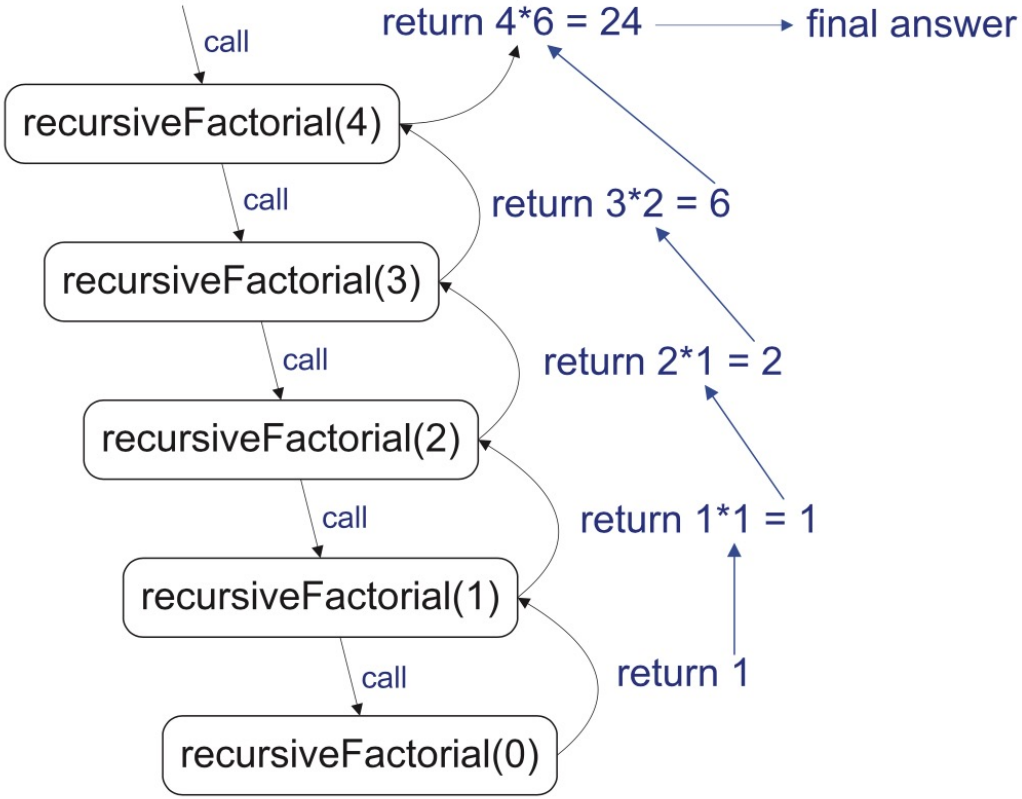
- Recursive Factorial:

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot \text{factorial}(n-1) & \text{if } n \geq 1. \end{cases}$$

```
int factorial_rec(int n){
    if (n==0){
        return 1;
    }
    else{
        return n * factorial_rec(n - 1);
    }
}
```

Recursive Algorithms - Example

- Recursive Factorial
- **Recursion Trace**:




```
int factorial_rec(int n){
    if (n==0){
        return 1;
    }
    else{
        return n * factorial_rec(n - 1);
    }
}
```

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot \text{factorial}(n-1) & \text{if } n \geq 1. \end{cases}$$

Linear Recursion

- The simplest form of recursion
- function makes at most one recursive call each time it is invoked
- When we have a sequence, we can view some problems in terms of:
  - a first or last element
  - a remaining sequence that has the same structure as the original sequence

Linear Recursion

- The simplest form of recursion
- function makes at most one recursive call each time it is invoked
- When we have a sequence, we can view some problems in terms of:
  - a first or last element
  - a remaining sequence that has the same structure as the original sequence

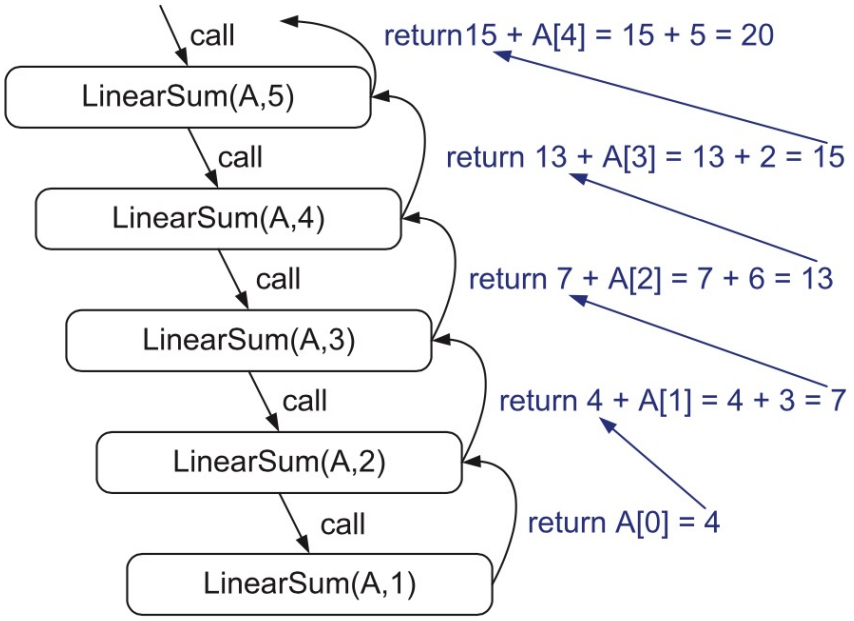
Algorithm LinearSum(A, n):
Input: A integer array A and an integer $n \geq 1$, such that A has at least n elements
Output: The sum of the first n integers in A

```
if n = 1 then
    return A[0]
else
    return LinearSum(A, n − 1) + A[n − 1]
```

Linear Recursion

Algorithm LinearSum(A, n):
Input: A integer array A and an integer $n \geq 1$, such that A has at least n elements
Output: The sum of the first n integers in A

if $n = 1$ **then**
  return $A[0]$
else
  return LinearSum$(A, n-1) + A[n-1]$




```java
int linearSum_iter(int data[], int n){
    int sum = 0;
    for (int i =0; i < n; i++){
        sum += data[i];
    }
    return sum;
}
```

```java
int linearSum(int data[], int n){
    if (n <= 0)
        return 0;
    return (linearSum(data, n - 1) + data[n - 1]);
}
```

Recursion in Computer Science

- Recursion is heavily used in Functional Programming
  - In Pure Functional Languages it is the only way to iterate!
    - Like Haskell
  - Non-pure programming languages allow iteration
    - Like Scala
- Recursion has overheads in Imperative Programming Languages
  - Like in C++ and Java
- Pure Functional Language use tail recursion conversions to reduce significant impact on memory consumption of heavy recursive calls
- It has many applications in Big Data Tools
  - Laziness!

Questions?

Those who can not attend the office hours can stay and ask questions for mid-term

If you don't have questions, feel free to leave ☺