Recursion

Coding Self-Similar Algorithms

Have you ever seen...

Things with parts that look like the whole?

... And Algorithms

- How would you describe how to draw those pictures?
- You might say “Draw a little version here, then here, then draw a circle around them…”

Recursion

- Defining something in terms of itself
- Have part of the problems that looks like the whole.
- Example:
  \[ 5! = 4! \times 5 \]
  \[ n! = (n-1)! \times n \]
Writing Recursive Algorithms

1. Find part of the problem that can be solved using the algorithm you want to write
   \( n! = (n-1)! \times n \)

Writing Recursive Algorithms

2. The recursive part of the problem MUST be a smaller version!
   \( n! = n! \times 1 \) Not allowed!

Writing Recursive Algorithms

3. Assume that the algorithm works for the smaller problem.
   \( n! = (n-1)! \times n \) (n-1)! Will be computed correctly

Writing Recursive Algorithms

4. Define the base case: the smallest version of the program. This part cannot be written recursively!
   \( 1! = 1 \)
Example

```c
int factorial(int n) {
    if (n == 1)
        return 1;
    else return factorial(n-1) * n;
}
```

But how does it work?

• Each recursive subroutine call is done separately
• Similar to “calling a friend” – you can’t ask them to do all the work!
• Eventually the base case is computed and the answer is “built up” from there