Algorithms and Data Structures for Beginners

9 / 35

8 - Factorial

Mark Lesson Complete View Code Prev Next

Suggested Problems

Status	Star	Problem	Difficulty	Video Solution	Code
		Reverse Linked List	Easy		C++

Recursion (One Branch)

Recursion can be a perplexing concept to wrap your head around so don't be discouraged if you don't get it straightaway.

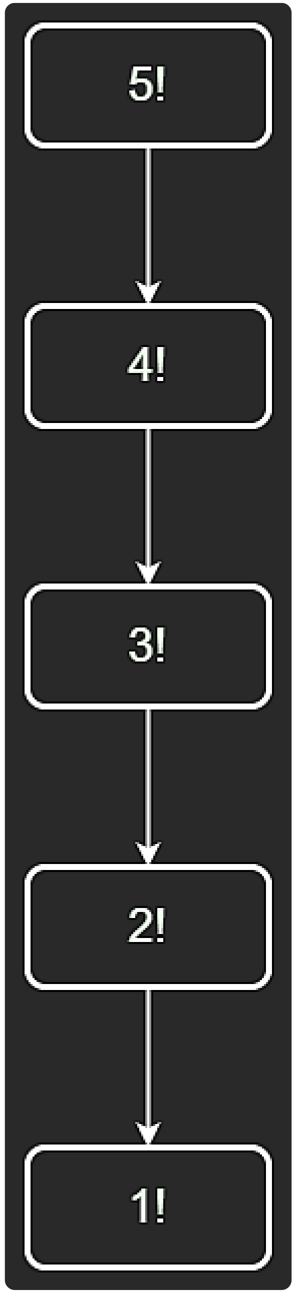
Recursion is when a function calls itself with a smaller output. So while an iterative function will make use of for loop and while loop, a recursive function achieves this by calling itself until a base case is reached.

> *Recursive functions have two parts:*
>
> 1. The base case
> 2. The function calling itself with a different input.

> *There are two types of recursion, one-branch and two-branch. Let's discuss one-branch recursion first.*

About

Recursion is best explained with an example. Let's take n factorial from math, the formula for which is: $n! = n * (n-1) * (n-2) * ...1$. $n!$ is just a short way of representing the cumulative product of all numbers from n to 1. A shorter way of writing this would be as $n! = n * (n-1)!$, i.e. $5!$ = $5 * 4!$. The visual and pseudocode below demonstrate this.


5!
4!
3!
2!
1!

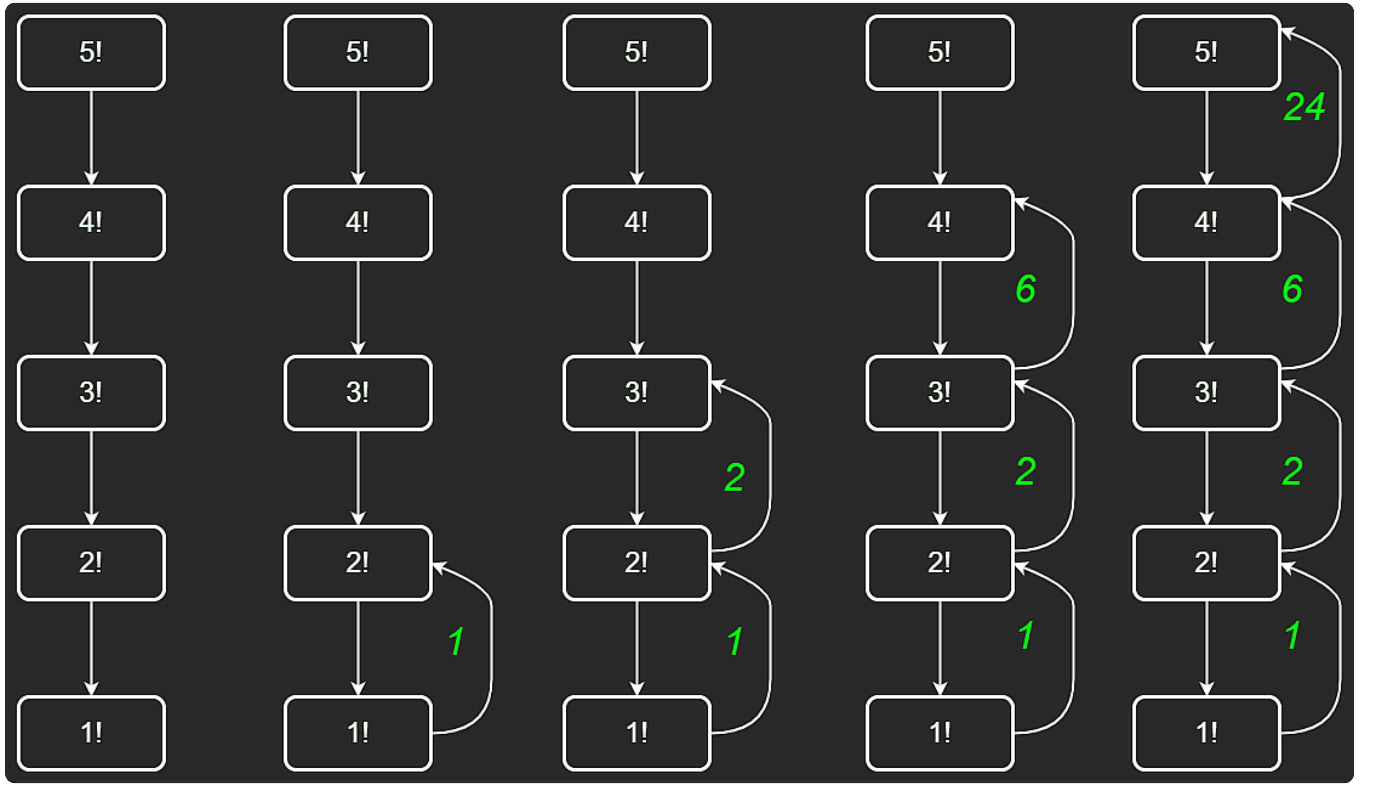
```
int factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n-1)
```

In the last line return statement, we notice that the function is calling itself with a different input. Let's analyze this. We have our two parts: **base case** and the **function calling itself**.

When the code reaches the last line with the initial input of 5, we get: `5 * factorial(4)` , which starts executing the function again from line 1, only now with input 4, so we get `4 * factorial(3)` and then `3 * factorial(2)` and lastly `2 * factorial(1)` after which the base case is reached.

But what happens when the base case is reached? When the function is called with 1 as the input, 1 is returned, and now it can be multiplied by 2, which will result in 2, which is the answer to `2!` . We have only solved the first sub-problem so far. Now, we compute `3 * factorial(2)` , which results in 6, then `4 * factorial(3)` , which is 24, and finally `5 * factorial(4)` , which is 120 - the ultimate answer to $5!$ **The most important part** is that when we trigger the base case, we move back "up" the recursion tree because now we have to "piece" together the answers to our sub-problems to get to the final solution.

This process is visualized below.



> *As observed, we took the original problem,* `factorial(5)` *and broke it down into smaller sub-problems, and by combining the answer to those sub-problems, we were able to solve the original problem. It is important to node that if there is no base case in recursion, the last line would execute forever resulting in a stack overflow!*

Time Space Complexity Analysis

In total, n calls are being made to the `factorial` function, making the time complexity $O(n)$. Furthermore, the space complexity will also be in $O(n)$.

Recursion operates off of a stack, and because there are n recursive calls, there will be n stacks, which results in $O(n)$ space.

Iteration and Recursion

Any recursive algorithm can be written iteratively, and the other way around. The iterative implementation of this is the following:

```
n = 5
res = 1
while n > 1:
    res = res * n
    n--
```

In the iterative case, we store our answer in a variable named `res` and decrement `n` until `n` becomes 1.

Closing Notes

Recursion will become very useful once we get to trees as it can be easily used to perform depth-first search.