

← [course home \(/table-of-contents\)](/table-of-contents)

# Bottom-Up Algorithms

Going **bottom-up** is a way to avoid recursion, saving the **memory cost** that recursion incurs when it builds up the **call stack**.

Put simply, a bottom-up algorithm "starts from the beginning," while a recursive algorithm often "starts from the end and works backwards."

For example, if we wanted to multiply all the numbers in the range  $1..n$ , we could use this cute, **top-down**, recursive one-liner:

```
def product_1_to_n(n):  
    # We assume n >= 1  
    return n * product_1_to_n(n - 1) if n > 1 else 1
```

Python 3.6 ▼

This approach has a problem: it builds up a **call stack** of size  $O(n)$ , which makes our total memory cost  $O(n)$ . This makes it vulnerable to a **stack overflow error**, where the call stack gets too big and runs out of space.

To avoid this, we can instead go **bottom-up**:

```
def product_1_to_n(n):  
    # We assume n >= 1  
    result = 1  
    for num in range(1, n + 1):  
        result *= num  
  
    return result
```

Python 3.6 ▼

This approach uses  $O(1)$  space ( $O(n)$  time).

*Some* compilers and interpreters will do what's called **tail call optimization** (TCO), where it can optimize *some* recursive functions to avoid building up a tall call stack. Python and Java decidedly do not use TCO. Some Ruby implementations do, but most don't. Some C implementations do, and the JavaScript spec recently *allowed* TCO. Scheme is one of the few languages that *guarantee* TCO in all implementations. In general, best not to assume your compiler/interpreter will do this work for you.

Going bottom-up is a common strategy for **dynamic programming** problems, which are problems where the solution is composed of solutions to the same problem with smaller inputs (as with multiplying the numbers  $1..n$ , above). The other common strategy for dynamic programming problems is **memoization (/concept/memoization)**.

← [course home \(/table-of-contents\)](/table-of-contents)

Next up: Recursive String Permutations → [\(/question/recursive-string-permutations?course=fc1&section=dynamic-programming-recursion\)](/question/recursive-string-permutations?course=fc1&section=dynamic-programming-recursion)

---

Want more coding interview help?

Check out **interviewcake.com** for more advice, guides, and practice questions.