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:

JavaScript

```javascript
function product1ToN(n) {
    // we assume n >= 1
    return (n > 1) ? (n * product1ToN(n-1)) : 1;
}
```

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**:

JavaScript ▼

```
function product1ToN(n) {
    // we assume n >= 1

    var result = 1;
    for (var num = 1; num <= n; num++) {
        result *= num;
    }

    return result;
}
```

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 the fibonacci problem, above). The other common strategy for dynamic programming problems is **memoization (/concept/memoization)**.

See also:

- Overlapping Subproblems (/concept/overlapping-subproblems)

- Memoization (/concept/memoization)

Bottom-Up/Dynamic Programming Coding Interview Questions

5 **Making Change »**

Write a function that will replace your role as a cashier and make everyone rich or something. keep reading »

(/question/coin)

15✓ **Compute nth Fibonacci Number »**

Computer the nth fibonacci number. Careful--the recursion can quickly spin out of control! keep reading »

(/question/nth-fibonacci)

16 **The Cake Thief »**

You've hit the motherload: the cake vault of the Queen of England. Figure out how much of each cake to carry out to maximize profit. keep reading »

(/question/cake-thief)

All Questions ➡ (/all-questions)