**How To Approach Recursive Problems**

* Recursive solutions, by definition, are built off of **solutions to subproblems**.
* Many times, this will mean simply to compute f(n) by
  + adding something,
  + removing something,
  + or
  + otherwise changing the solution for f(n-1)
* In other cases, you might solve the problem for the first half of the data set, then the second half, and then merge those results.
* There are many ways you might divide a problem into subproblems.
* Three of the most common approaches to develop an algorithm are bottom-up, top-down, and half-and-half.

1. **Bottom-Up Approach**

The bottom-up approach is often the most intuitive.

We start with knowing how to solve the problem for a simple base case.

Then we figure out how to solve the problem for two elements, then for three elements, and so on.

The key here is to think about how you can build the solution for one case off of the previous case (or multiple previous cases).

1. **Top-Down Approach**

The top-down approach can be more complex since it's less concrete. But sometimes, it's the best way to think about the problem.

In these problems, we think about how we can divide the problem for case N into subproblems.

Be careful of overlap between the cases.

1. **Half-and-Half Approach**

In addition to top-down and bottom-up approaches, it's often effective to divide the data set in half.

For example, binary search works with a "half-and-half" approach. When we look for an element in a sorted array, we first figure out which half of the array contains the value. Then we recurse and search for it in that half.

Merge sort is also a "half-and-half" approach. We sort each half of the array and then merge together the sorted halves.