The Four Steps to Understanding Recursion

Step 1: Write and define the prototype of the function.

Since functions are the basic unit of recursion, it's important to know what the function does. The prototype you use will dictate how the recursion behaves.

Let's look at an example. Here's a function which will sum the first **n** elements of an array.

```
// Sums the first n elements of the array, arr
int sum( int arr[], int n );
```

Step 2: Write out a sample function call

Once you've determined what the function does, then we imagine a function call.

```
int result = sum( arr, n );
```
So, the call is **sum(arr, n)**. This will sum the first **n** elements of **arr**. Pick the most generic function call. For example, you don't want to have a call like:
```
int result = sum( arr, 10 );
```
That's picking a constant. You want to use variables when possible, because that's the most general way to call the function.

Step 3: Think of the smallest version of the problem

The smallest version is called the *base case*. Most people mistakenly pick a base case that's too large. In this case, you will pick a specific value for **n**.

So, what's the smallest version of the problem? Here are three choices:

```
sum( arr, 2 );  // Choice 1
sum( arr, 1 );  // Choice 2
sum( arr, 0 );  // Choice 3
```
Some people pick choice 1, reasoning that if you are to sum elements of an array, then you must have at least two elements to sum. However, that's really not necessary. In math, there is something called a "summation". It's perfectly valid to have a summation of only one element. You just return that one element.

Some people pick choice 2, because it doesn't make sense to sum an array of size 0, whereas an array of size 1 seems to make sense.

However, it turns out choice 3 is the smallest choice possible. You can sum zero elements of an array. What value should it return? It should return 0. As it turns out, 0 is the additive identity. Anything added to 0 is that number. If we wanted to multiply all elements of an array, we would have picked the multiplicative identity, which is 1.

Admittedly, picking such a small value requires a more extensive knowledge of math, but that shouldn't scare you from picking the smallest value possible.

There are several mistakes people make with a base case. The first one we've already mentioned: picking too large a base case. Second, not realizing there may be more than one base case. Finally, thinking that the base case only gets called when the input size is the smallest. In fact, the recursion

ALWAYS makes it to some base case. Thus, the base case is where the recursion eventually stops. Don't think of it as merely called when the input is, say, 0. It gets called for all cases (eventually).

Step 4: Think of smaller versions of the function call.

Here's the function call

```
sum( arr, n )  // sums first n elements of arr
```

It tries to solves a problem of size "n". We want to think of a smaller problem which we will assume can be solved correctly. The next smallest problem is to sum "n - 1" elements.

```
sum( arr, n - 1 )  // sums first n - 1 elements of arr
```

Assume this problem has been solved for you. How would you solve the original, larger problem? If the first n - 1 elements have already been summed then only the n_th element is left to be summed. The n_th element is actually at index n - 1 (because arrays start at index 0).

So, the solution to solving **sum(arr, n)** is to add **sum(arr, n - 1)** to **arr[n - 1]**.

```
int sum( int arr[], int size )
{
   if ( size == 0 )  // base case
      return 0;
   else
     {
        // recursive call
        int smallResult = sum( arr, size - 1 );

        // use solution of recursive call to solve this problem
        return smallResult + arr[ size - 1 ];
     }
}
```

Some people don't like multiple return statements. That can be easily handled:

```
int sum( int arr[], int size )
{
   int result;
   if ( size == 0 )  // base case
      result = 0;
   else
     {
        // recursive call
        int smallResult = sum( arr, size - 1 );

        // use solution of recursive call to solve this problem
        result = smallResult + arr[ size - 1 ];
     }
  return result;
}
```

You may even think there's no reason to declare **smallResult** and prefer to write:

```
int sum( int arr[], int size )
{
   if ( size == 0 )  // base case
      return 0;
   else
     return sum( arr, size - 1 ) + arr[ size - 1 ];
}
```