Recursion vs. Iteration

With recursion, each recursive call requires allocation of memory on the stack.

So, recursive algorithms can be expensive in terms of memory use.

Except that "requires" is not always true.

Tail recursion

Consider a recursive definition in a language such as F# or Haskell.

$$f(x_1, \ldots, x_n) = \ldots \quad f(y_1, \ldots, y_n) \ldots$$

For a concrete example:

$$fact(0) = 1$$
$$fact(n+1) = (n+1) * fact(n)$$

We can envision the call stack by "unwinding" the recursion:

$$fact(5) =$$
$$5 * fact(4) =$$
$$4 * fact(3) =$$
$$3 * fact(2) =$$
$$2 * fact(1) =$$
$$1 * fact(0) =$$
$$1$$

Each layer involves allocation on the stack.

Instead, use tail recursion; recursive calls have the form

$$f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$$

For factorial, this involves a helper function:

$$fact\_iter(0, r) = r$$
$$fact\_iter(n+1, r) = fact\_iter(n, (n+1) * r)$$

then

$$fact(n) = fact\_iter(n, 1)$$

Unfold this:

$fact(5) =$
$fact\_iter(5,1) =$
$fact\_iter(4, 5) =$
$fact\_iter(3, 20) =$
$fact\_iter(2, 60) =$
$fact\_iter(1, 120) =$
$fact\_iter(0, 120) =$
120

No need to return to each level once done.

Just return directly to the caller of fact.

Reuse the memory on the call-stack each time.

In Oz, a procedure/function is tail-recursive if the last expression/statement in the body is the (only) recursive call.