

Recursion

Recursion

- Idea: Some problems can be broken down into smaller versions of the same problem
- Example: $n!$
 - $1*2*3*\dots*(n-1)*n$
 - $n*\text{factorial of } (n-1)$

Base Case

- $5! = 5*4!$
- $4! = 4*3!$
- $3! = 3*2!$
- $2! = 1*1!$
- $1! = 1$ ← Base case – you always need a terminating condition to end

Function: factorial

```
int factorial(int n) {  
    if(n == 1)  
        return 1;  
    else  
        return (n*factorial(n-1));  
}
```

Iterative Factorial

```
int factorial(int n) {  
    int i;  
    int product = 1;  
    for(i = n; i > 1; i--) {  
        product = product * i;  
    }  
    return product;  
}
```

Comparison

- Why use iteration over recursion or vice versa?

Linear Recursion

- At most 1 recursive call at each iteration
 - Example: Factorial
- General algorithm
 - Test for base cases
 - Recurse
 - Make 1 recursive call
 - Should make progress toward the base case
- Tail recursion
 - Recursive call is last operation
 - Can be easily converted to iterative

Higher-Order Recursion

- Algorithm makes more than one recursive call
- Binary recursion
 - Halve the problem and make two recursive calls
 - Example?
- Multiple recursion
 - Algorithm makes many recursive calls
 - Example?

Rules of Recursion

1. *Base cases.* You must always have some bases cases, which can be solved without recursion.
2. *Making progress.* For the cases that are to be solved recursively, the recursive call must always be to a case that makes progress toward a base case.
3. *Design rule.* Assume that all the recursive calls work.
4. *Compound interest rule.* Never duplicate work by solving the same instance of a problem in separate recursive calls.

Towers of Hanoi

- Three pegs and a set of disks
- Goal: move all disks from peg 1 to peg 3
- Rules:
 - move 1 disk at a time
 - a larger disk cannot be placed on top of a smaller disk
 - all disks must be on some peg except the disk in-transit