Recursive Function fibonacci

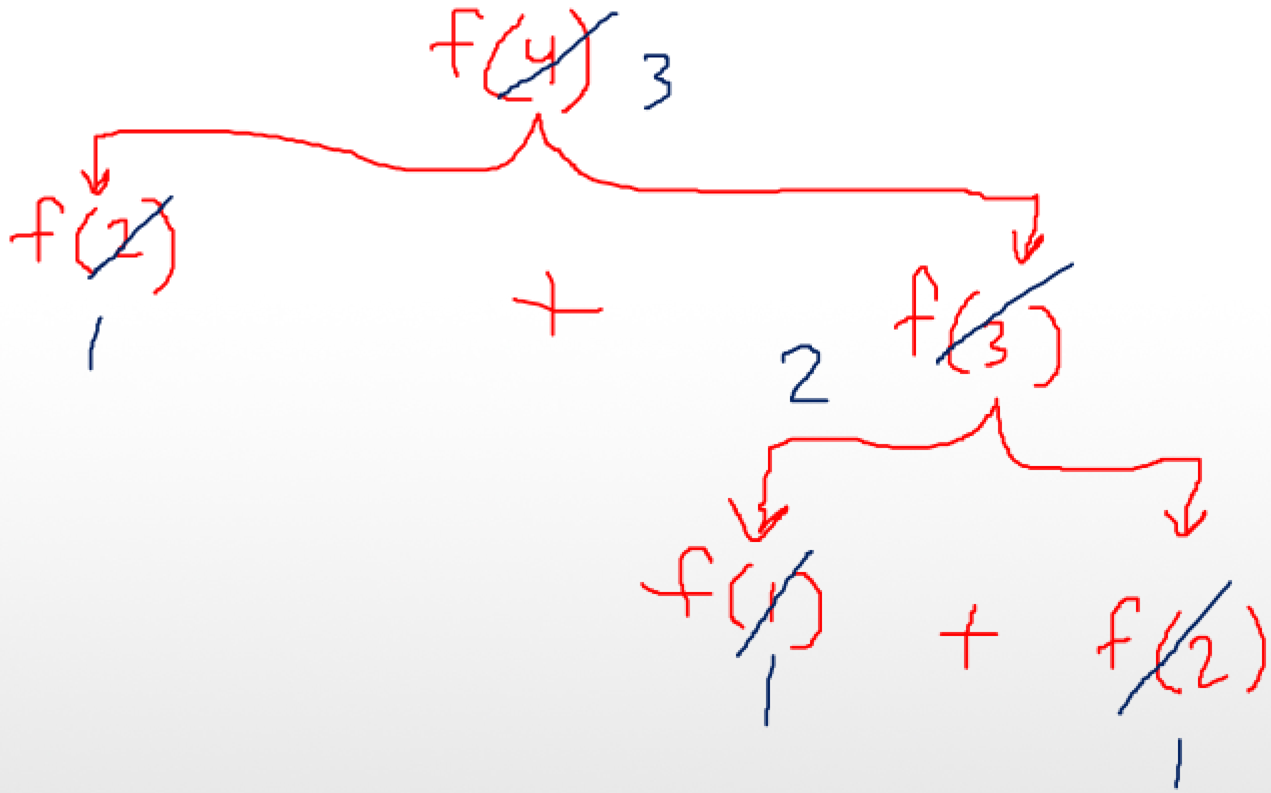
$$
\text{fibonacci(n)} = \begin{cases} 1 & , \ n = 1 \\ 1 & , \ n = 2 \\ \text{fibonacci(n-2)+fibonacci(n-1)} & , \ n > 2 \end{cases}
$$

Recursive Function fibonacci

the Fibonacci sequence 1, 1, 2,3, 5, 8, 13, 21, 34,…………

```
public static long  fibonacci(int n)
{
     if (n==1 || n==2)
        return 1;
     else
        return fibonacci(n-2)+fibonacci(n-1);


}
```

Trace of fibonacci = fibonacci(4)



You have to know !

The code of a recursive method must be structured to handle both **the base case** and **the recursive case**

Each call sets up **a new execution environment, with new parameters and new local variables**

You have to know !

Comparison

1. Some problems are more easily solved recursively.

2.Recursion can be highly inefficient as resources are allocated for each method invocation.

If you can solve it iteratively, you usually should!

Iterative Factorial

```
public long factorial(int n) {
        long product = 1;
        for(int i = 1; i <= n; i++) {
                product *= i;
                System.out.println("Product " + product);
        }
        return product;
}
```

Recursive Factorial

(Write the previous method without using a loop!)

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

Try running this for large values of n!

You have to know !

Recursion

```
public static long  fib(int n)
{
    if (n==1 || n==2)
       return 1;
    else
       return fib(n-2)+fib(n+1);


}
```

Iteration

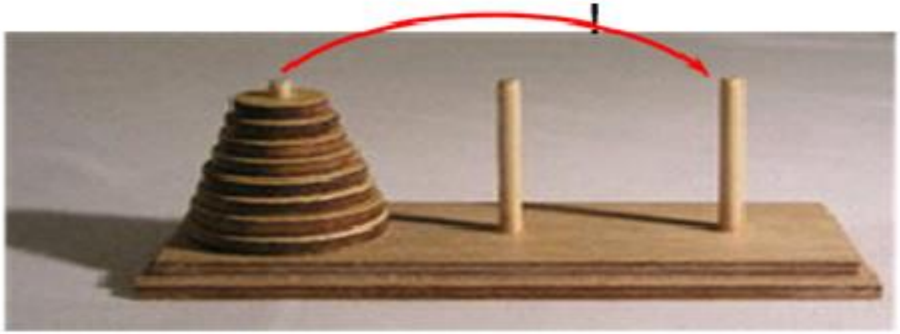
```
public static long fib(int n) {
   int f0 = 0, f1 = 1, currentFib;

   if (n == 0) return 0;
   if (n == 1) return 1;

   for (int i = 2; i <= n; i++) {
     currentFib = f0 + f1;
     f0 = f1;
     f1 = currentFib;
   }

   return f1;
 }
```

Recursion: Advantages/Disadvantages

Advantage of recursion: Simple to write and understand.

Disadvantage: Amount of memory required is proportional to the depth of recursion

Case Study: Towers of Hanoi



The "Towers of Hanoi" puzzle was invented by the french mathematician Édouard Lucas

- Tower of Hanoi is a very famous game. In this game there **are 3 pegs and N number of disks** placed one over the other in decreasing size.

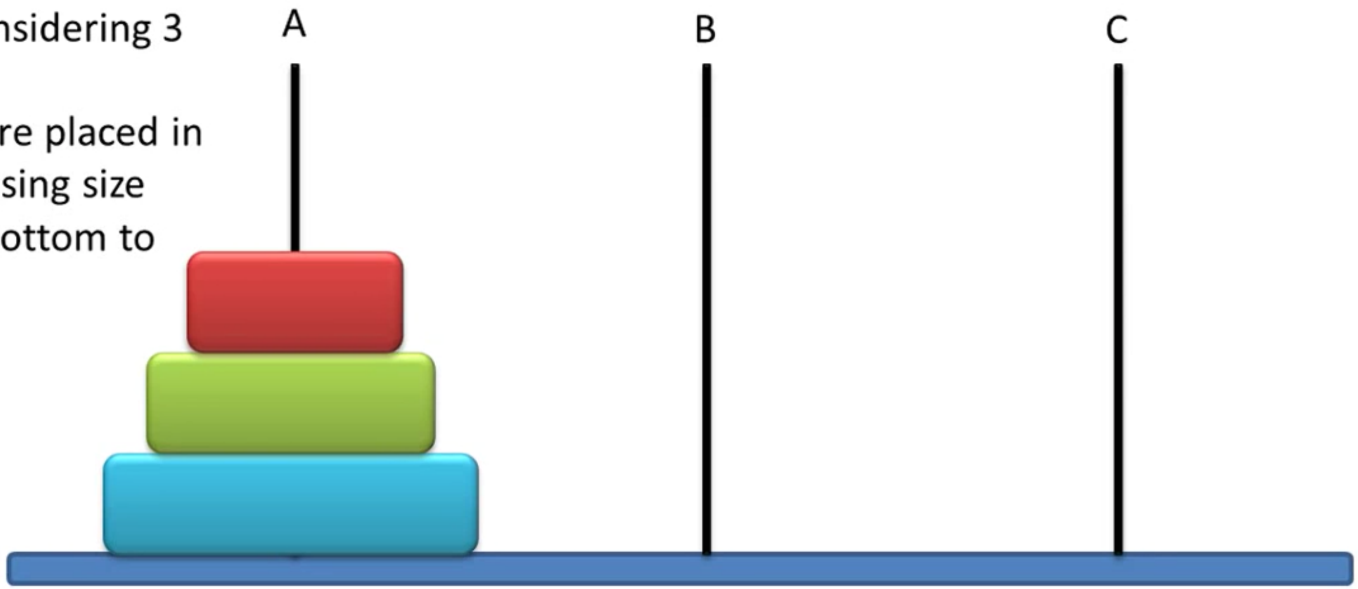
- The objective of this game is to move the disks one by one from the first peg to the last peg (Only the top disk can be moved each time).

- There is only ONE condition, we can not place a bigger disk on top of a smaller disk.

Towers of Hanoi: Example

The 3 pegs are labeled A, B and C.

In this example we are considering 3 disks.
They are placed in decreasing size from bottom to top.



Our objective is to move the disks from peg A to peg C in such a way that they are in the same order:
RED then GREEN then BLUE
from top to bottom as they are in peg A.

Towers of Hanoi

Before solving the example, let's learn how to solve this problem with lesser amount of disks, say → 1 or 2

Towers of Hanoi for 1 disk

- ❑ If we have only one disk, then it can easily be moved from source to destination peg

Towers of Hanoi for 2 disks

- ❑First, we move the smaller (top) disk to aux peg.
- ❑Then, we move the larger (bottom) disk to destination peg.
- ❑And finally, we move the smaller disk from aux to destination peg

Towers of Hanoi

How to solve Tower Of Hanoi?

To solve this game we will follow 3 simple steps recursively.

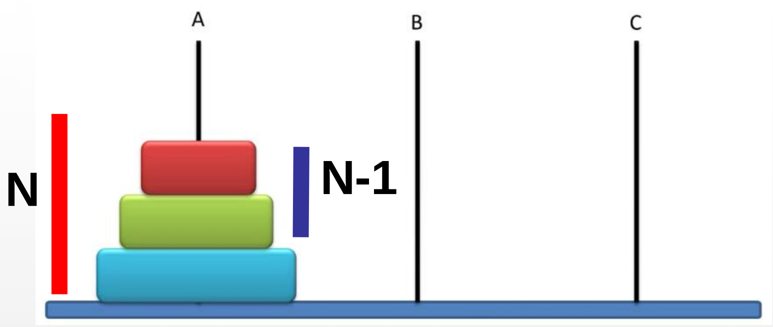
We will use a general notation:
T(N, Beg, Aux, End)

***T** denotes our procedure*
***N** denotes the number of disks*
***Beg** is the initial peg*
***Aux** is the auxiliary peg* (only to help moving the disks).
***End** is the final peg*

Towers of Hanoi

Steps

1. T (N-1, Beg, End, Aux)
2. Move (1, Beg, Aux, End)
3. T (N-1, Aux, Beg, End)



Step 1 − Move top (N-1) disks from **Beg** to **Aux** peg
Step 2 − Move 1 disk from **Beg** to **End** peg
Step 3 − Move top (N-1) disks from **Aux** to **End** peg

Towers of Hanoi: Algorithm

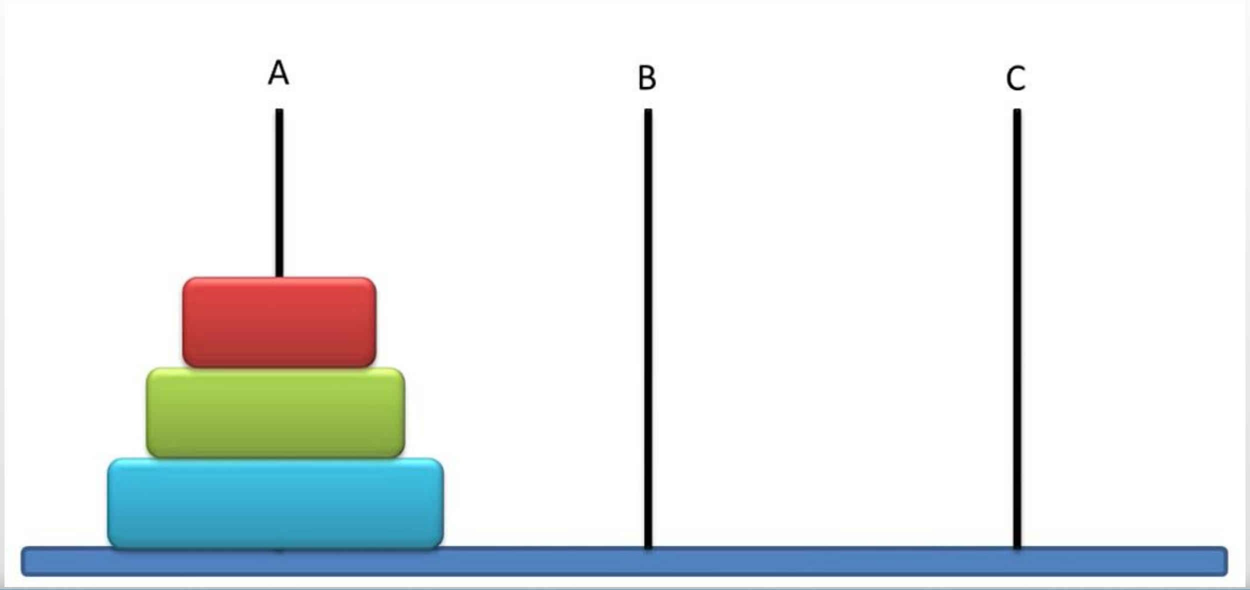
Let's solve this game together. ☺

Towers of Hanoi: Algorithm

We have 3 disks Red, Green and Blue all placed in peg A.

So, N = 3 (Number of disks)

Therefore, we will start with T(3, A, B, C)



Towers of Hanoi: Algorithm

N = 3
Beg = A
Aux = B
End = C

We will follow the 3 steps recursively to find the moves.

Step 1: T (N-1, Beg, End, Aux)
Step 2: Move (1, Beg, Aux, End)
Step 3: T (N-1, Aux, Beg, End)

T(3, A, B, C) ← We will apply the 3 steps on this

Towers of Hanoi: Algorithm

$$T(n, A, B, C) = \begin{cases} T(n-1, A, C, B) \\ A \to C \\ T(n-1, B, A, C) \end{cases}, \quad n > 0$$

$$T(n, Beg, Anx, end) = \begin{cases} T(n-1, Beg, end, Anx) \\ Beg \to end \\ T(n-1, Anx, beg, end) \end{cases}$$

Beg Anx end

$T(3, A, B, C)$

# of disks

- $T(2, A, C, B) \longrightarrow$
  - $T(1, A, B, C) \longrightarrow$
    - $T(0, A, C, B)$
    - $A \to C$ (1)
    - T(0,B,A,C)
  - $A \to B$ (2)
  - $T(1, C, A, B) \longrightarrow$
    - $T(0, C, B, A)$
    - $C \to B$ (3)
    - $T(0, A, C, B)$
- $A \to C$ (4)
- $T(2, B, A, C) \longrightarrow$
  - $T(1, B, C, A) \longrightarrow$
    - $T(0, B, A, C)$
    - $B \to A$ (5)
    - $T(0, C, B, A)$
  - $B \to C$ (6)
  - $T(1, A, B, C) \longrightarrow$
    - $T(0, A, C, B)$
    - $A \to C$ (7)
    - $T(0, B, A, C)$

$A \to C$
$A \to B$
$C \to B$
$A \to C$
$B \to A$
$B \to C$
$A \to C$

# of moves $= 2^n - 1 = 2^3 - 1 = 8 - 1 = \underline{\underline{7}}$

# of calls $= 2^{n+1} - 1 = 2^4 - 1 = 16 - 1 = \underline{\underline{15}}$

Towers of Hanoi: Algorithm

Moves

A → C

A → B

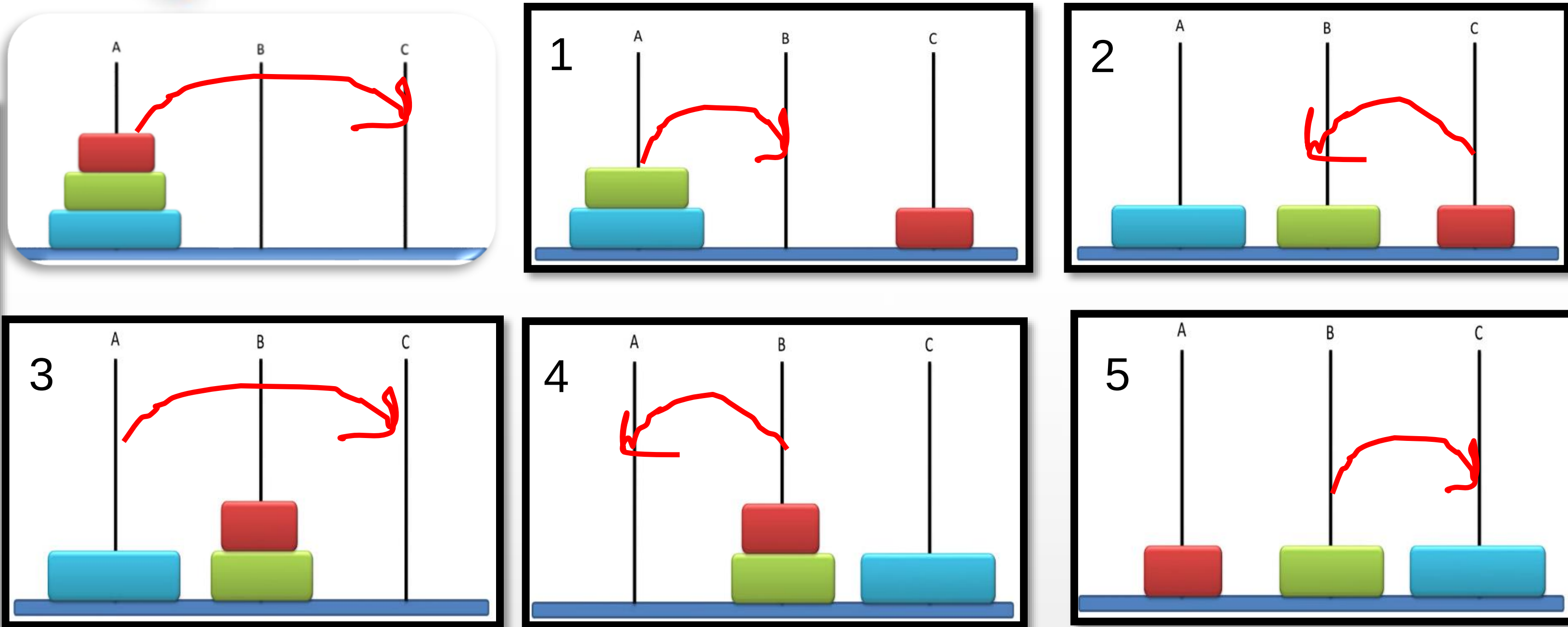
C → B

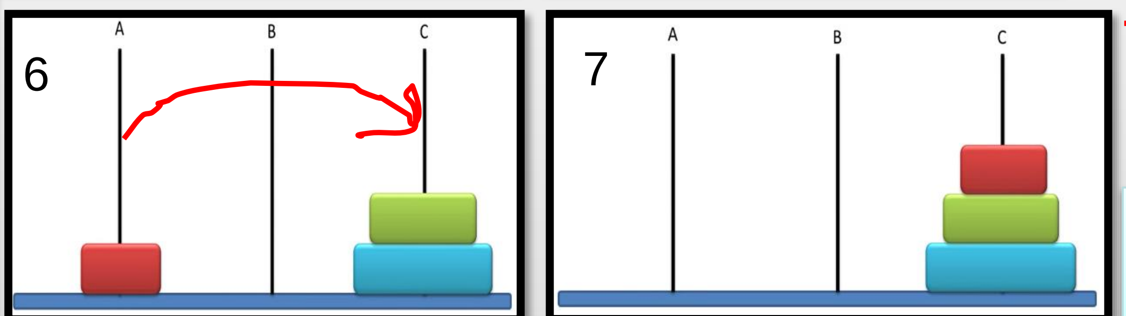
A → C

B → A

B → C

A → C





Tower of Hanoi puzzle with n disks can be solved in minimum 2^n-1 steps

puzzle with 3 disks has taken $2^3 - 1 = 7$ steps

Towers of Hanoi: Algorithm

```
public static void T(int n,char A, char B, char C){

   if (n>0){
        T(n-1,A,C,B);
        System.out.println(A+" ---> "+C);
        T(n-1,B,A,C);
   }

}
```

# of disks= n

Beg = A

Aux = B

End = C

Recursion or Iteration?

- Overhead associated with executing a (recursive) function in terms of:
  - Memory space
  - Computer time
- A recursive function executes more slowly than its iterative counterpart
- Today's computers are fast
  - Overhead of a recursion function is not noticeable

Extra Exercises

***Problem 1* :**

Write a recursive method `isPalindrome` that takes a string and returns whether the string is the same forwards as backwards.

Problem 2 :

Given integers a and b where a >= b, find their greatest common divisor ("GCD"), which is the largest number that is a factor of both a and b.

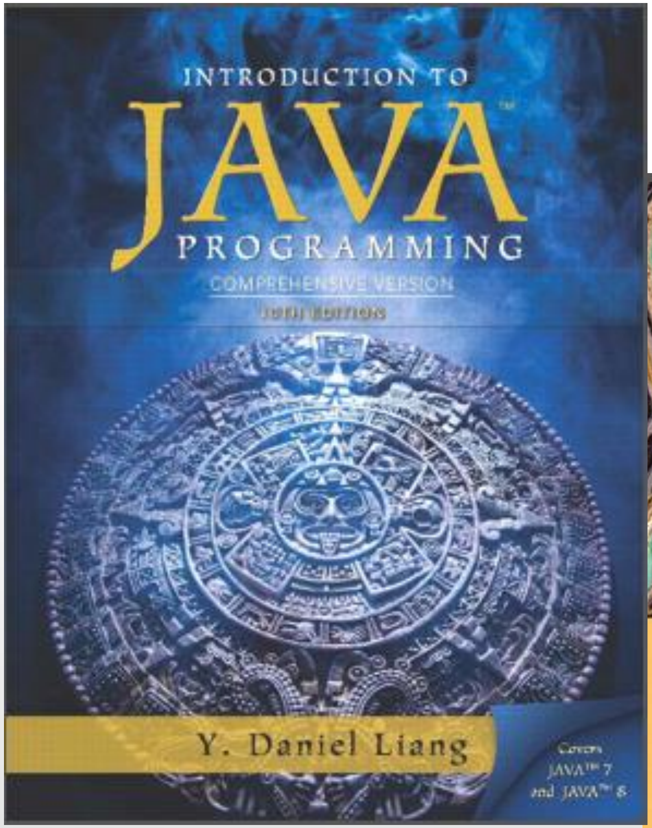
GCD(a, b) = GCD(b, a MOD b)

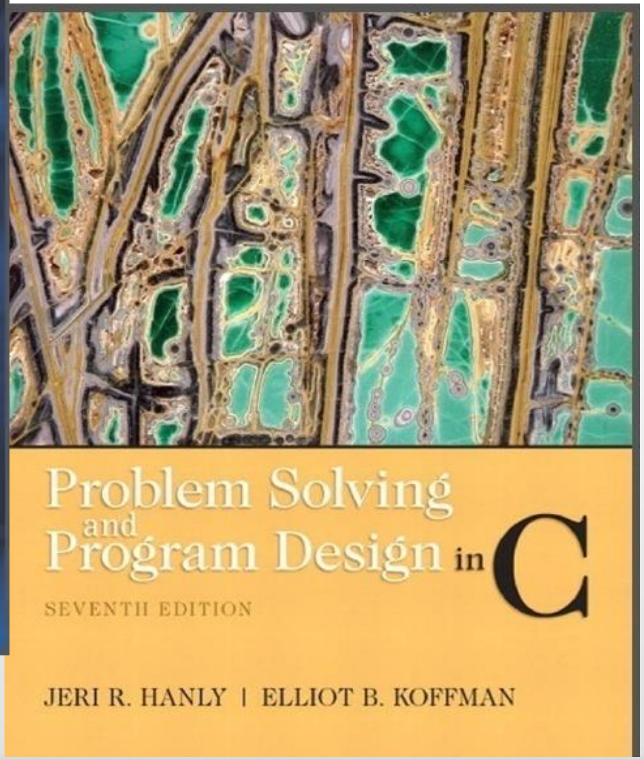
(Hint: What should the base case be?)

Question?

“Success is the sum of small efforts, repeated day in and day out.”
Robert Collier

Reference


INTRODUCTION TO
JAVA
PROGRAMMING
COMPREHENSIVE VERSION
Y. Daniel Liang
Covers
JAVA™ 7
and JAVA™ 8


Problem Solving and Program Design in C
SEVENTH EDITION
JERI R. HANLY | ELLIOT B. KOFFMAN