FAKULTAS
ILMU
KOMPUTER

Recursion

Dasar – Dasar Pemrograman 2

Dinial Utami Nurul Qomariah

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia

FAKULTAS ILMU KOMPUTER

Credits

- Liang, Introduction to Java Programming, 11th Edition, Ch. 2
- Downey & Mayfield, Think Java: How to Think Like a Computer Scientist, Ch. 2
- Slide Kuliah Dasar-dasar Pemrograman 2 Semester Genap 2021/2022

Perulangan (Loop) di Java

- for
- While
- do – while
- if – else (using recursion)

Perulangan (Loop) di Java

Recursive method → method yang memanggil dirinya sendiri baik secara langsung maupun tidak langsung.

direct recursive

```
void directRec(int n) {
//  . . .
directRec(n - 1);
//  . . .
}
```

```
void indirectRec(int n) {
//  . . .
bridge(n - 1);
//  . . .
}
void bridge(int n) {
//  . . .
indirectRec(n);
//  . . .
}
```

indirect recursive

Computing Factorial

Create a method to compute the factorial of a non-negative integer *n*, with **no loops**.

factorial(0) = 1;

factorial(n) = n*factorial(n-1);

n! = n * (n-1)!

Solution = Recursion

```
public static int faktor(int n){
    if (n==0){
        return 1;
    }
    else{
        return n * faktor(n-1);
    }
}
```

Recursion = loops with no loops!

Solution = Recursion TIP

```
public static int faktor(int n){
        return n == 0 ? 1 : n * factorialRec(n-1);
    }
```

Short if-else: condition ? trueCase : falseCase

Recursion: Base case + recursive case

```
public static int faktor(int n){
    if (n==0){
        return 1;
    }
    else{
        return n * faktor(n-1);
    }
}
```

Base CASE

Recursion: Base case + recursive case

```
public static int faktor(int n){
    if (n==0){
        return 1;
    }
    else{
        return n * faktor(n-1);
    }
}
```

Base CASE

Recursive CASE

Recursion: Base case + recursive case

```
public static int faktor(int n){
    if (n==0){
        return 1;
    }
    else{
        return n * faktor(n-1);
    }
}
```

Base CASE

Recursive CASE

One or more base cases (the simplest case) are used to stop recursion.

Recursion vs. Iteration

- Recursion is an alternative form of program control. It is essentially repetition without a loop.
- Recursion is a technique that leads to elegant solutions to problems that are difficult to program using simple loops.

Recursion Keuntungan dan Kerugian

Advantages : Recursion is good for solving the problems that are inherently recursive.

Disadvantages : Recursive programs can run out of memory, causing a **StackOverflowError**.

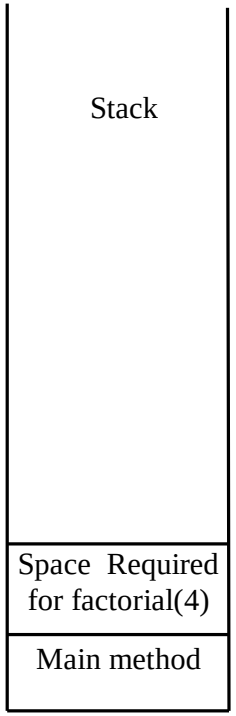
Don't do this anywhere!

```
public static void forever() {
    forever();
}
```

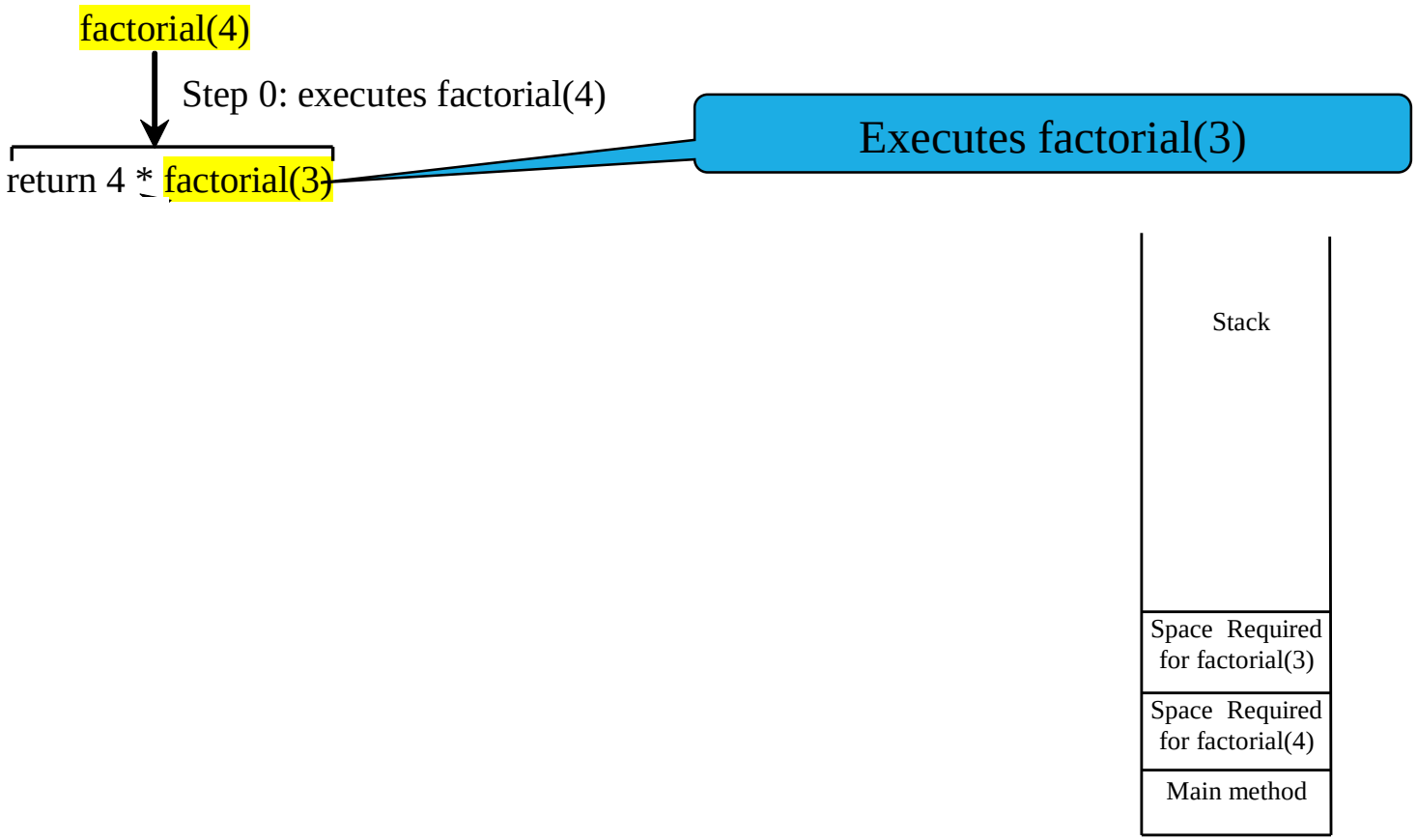
Trace Recursive factorial

Executes factorial(4)

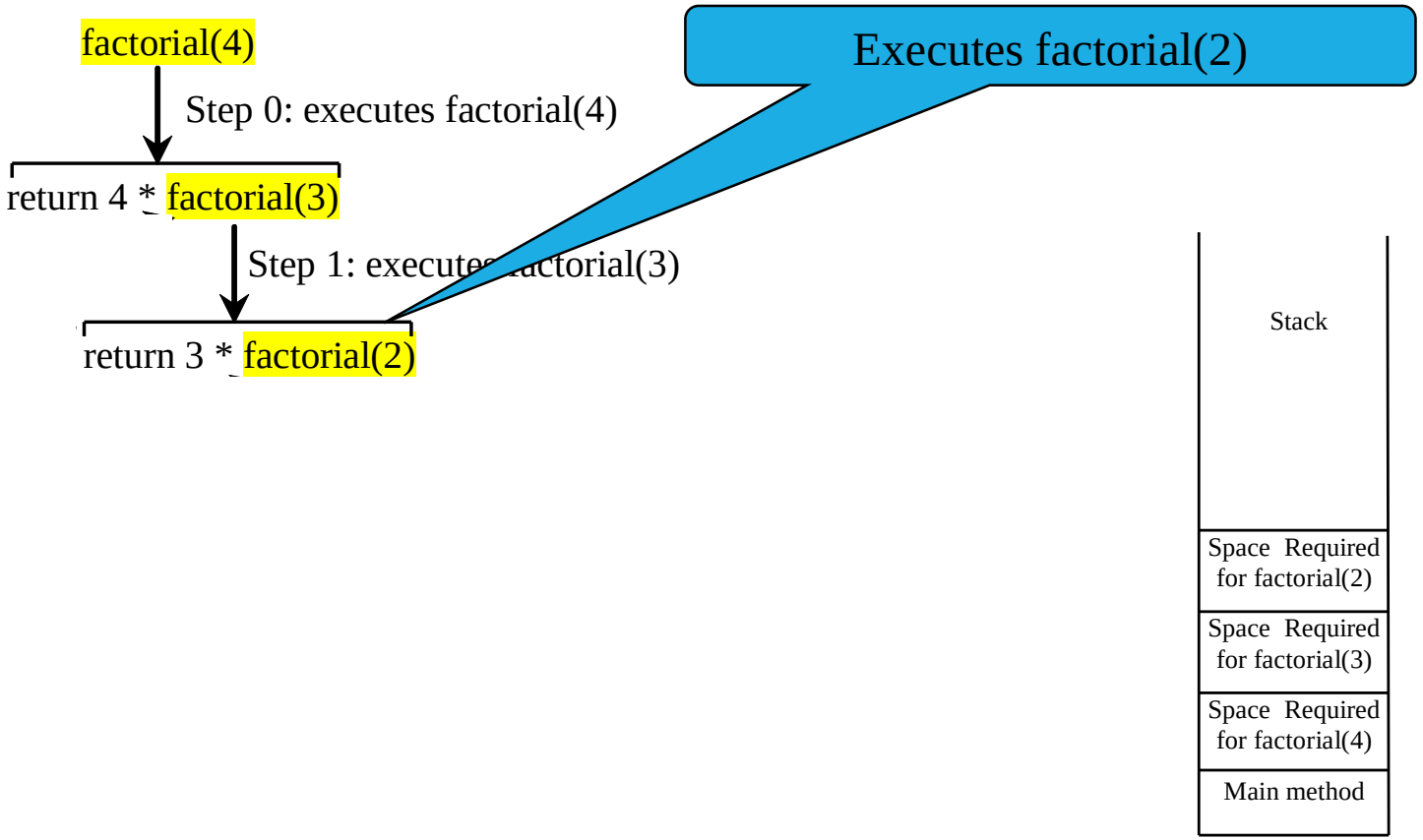
factorial(4)



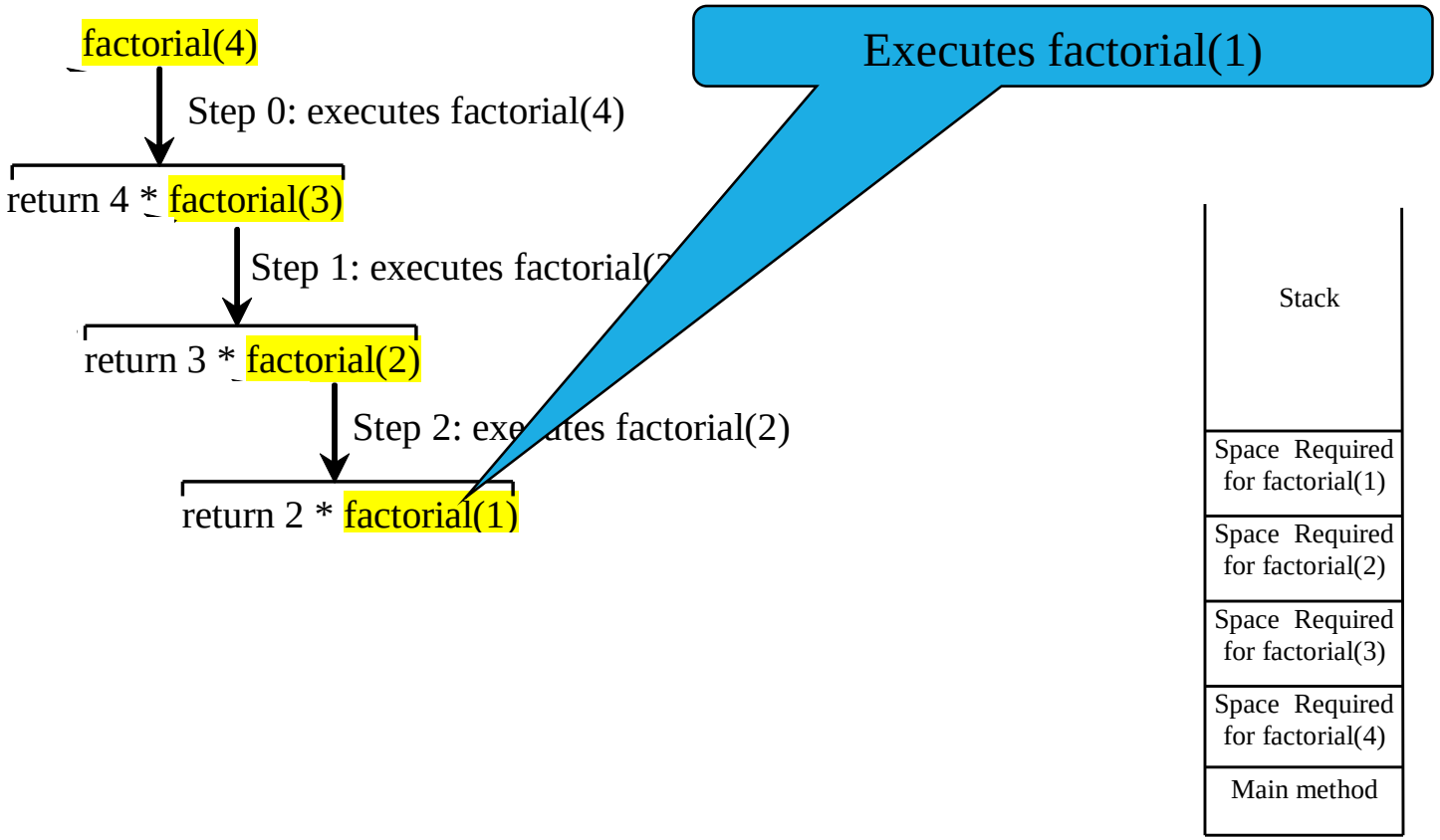
Trace Recursive factorial



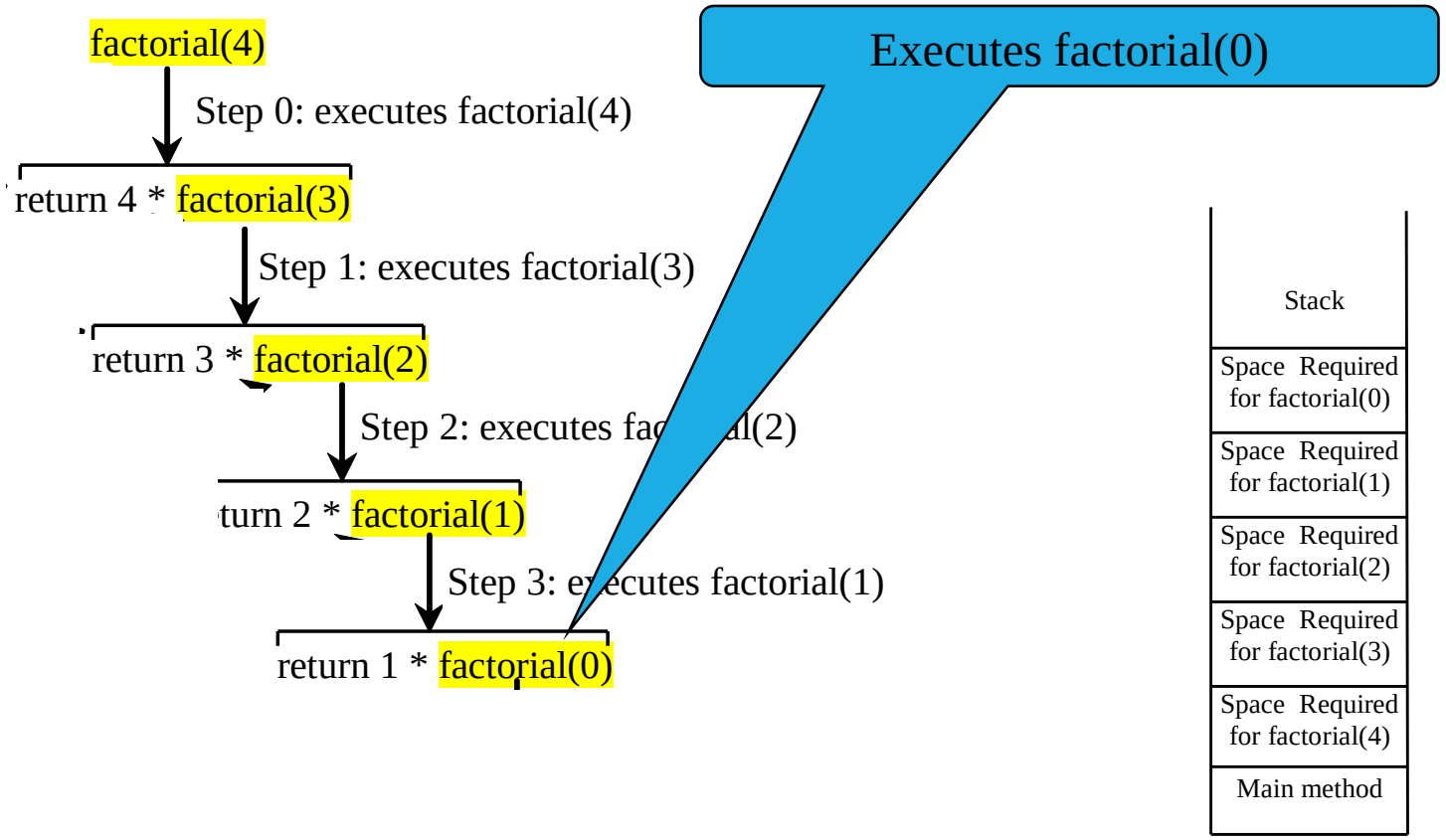
Trace Recursive factorial



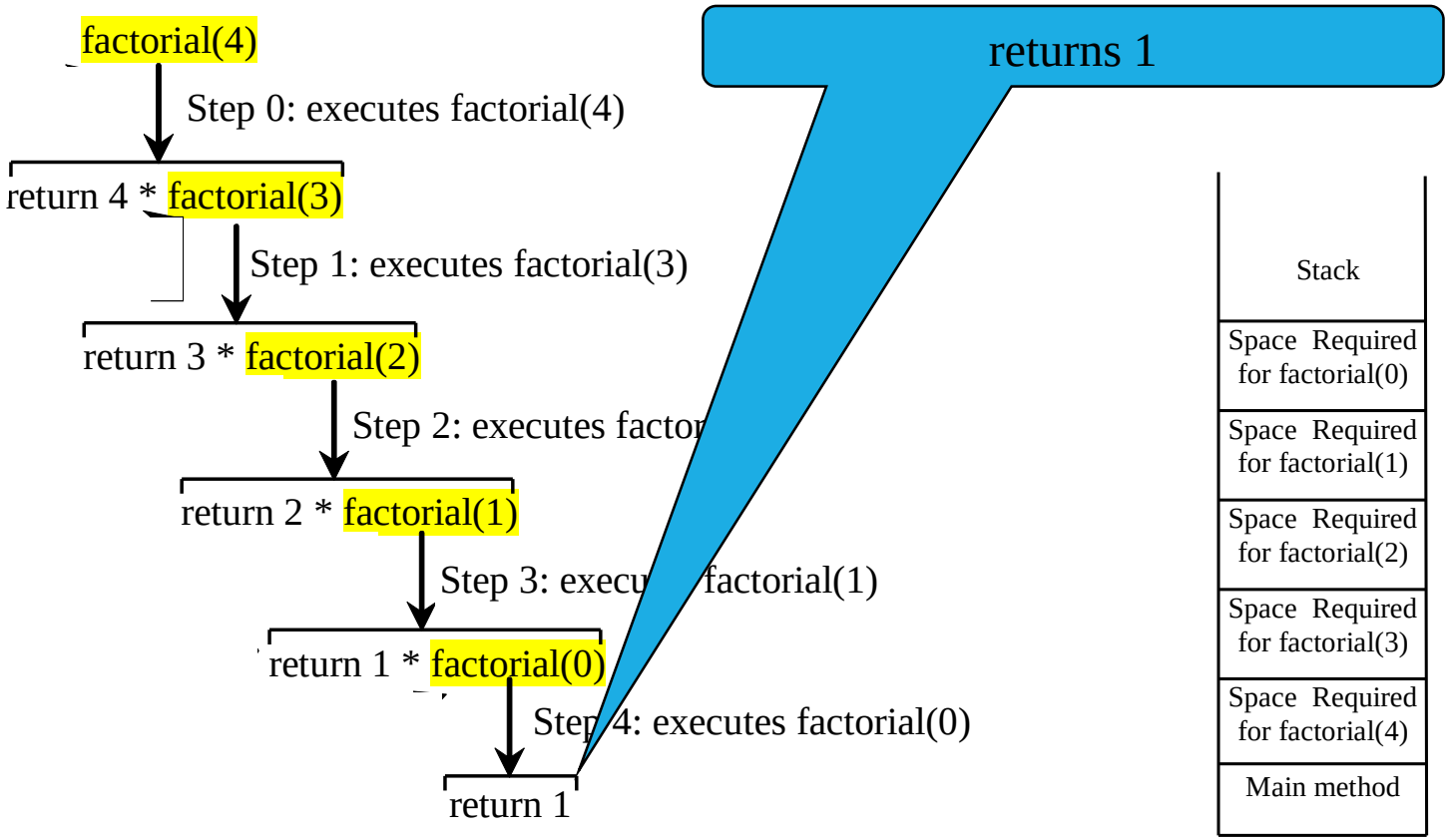
Trace Recursive factorial



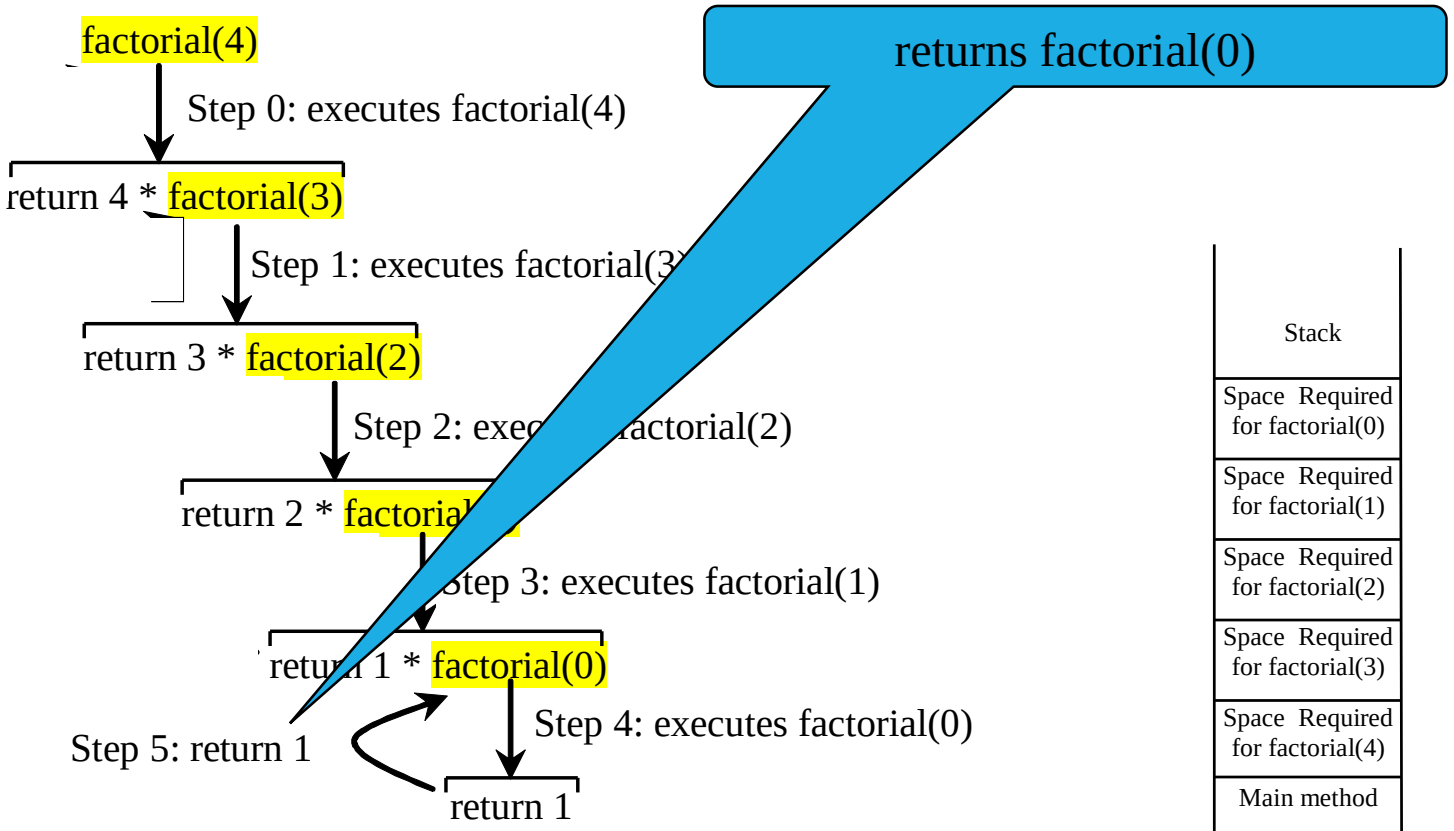
Trace Recursive factorial



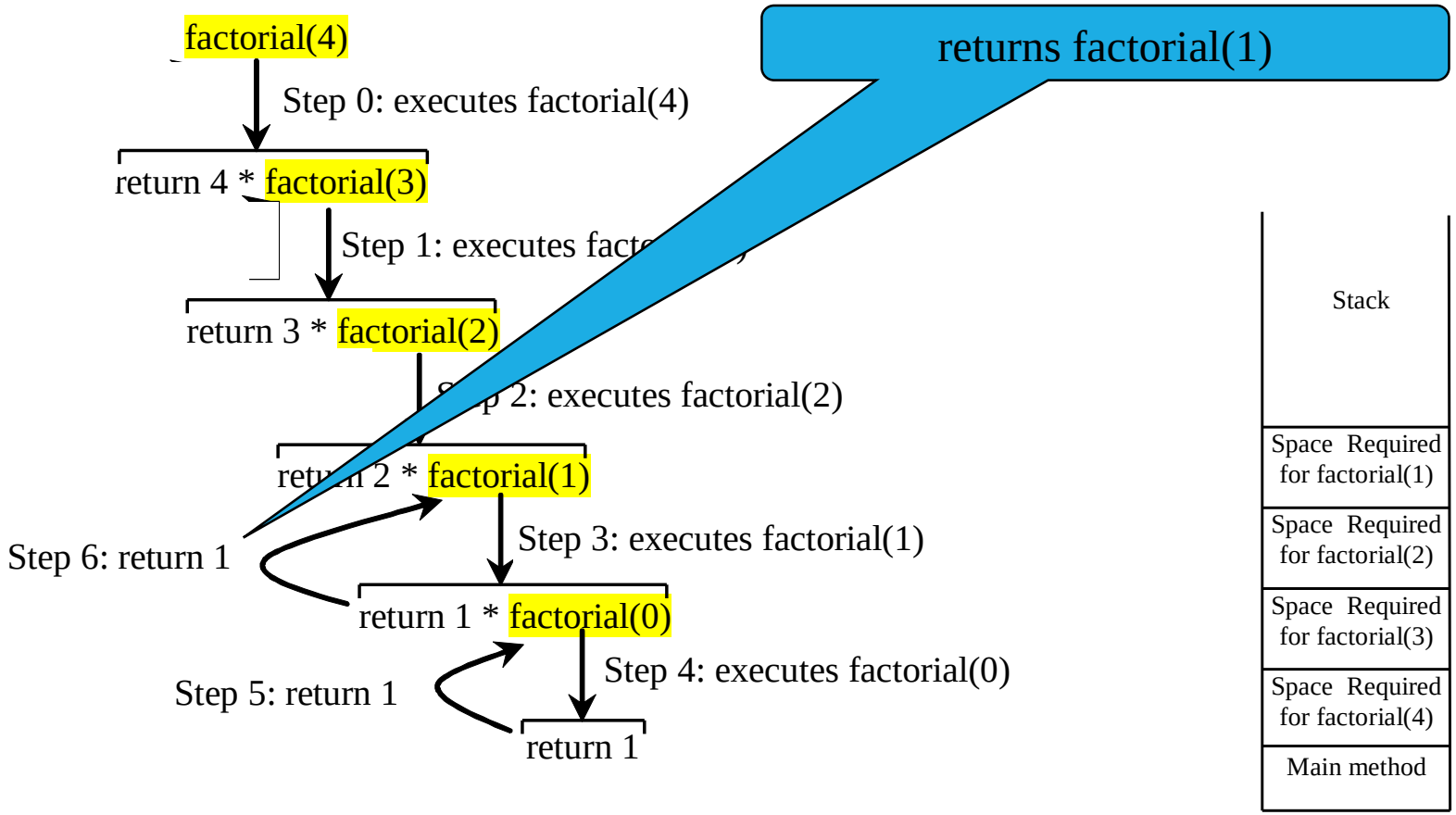
Trace Recursive factorial



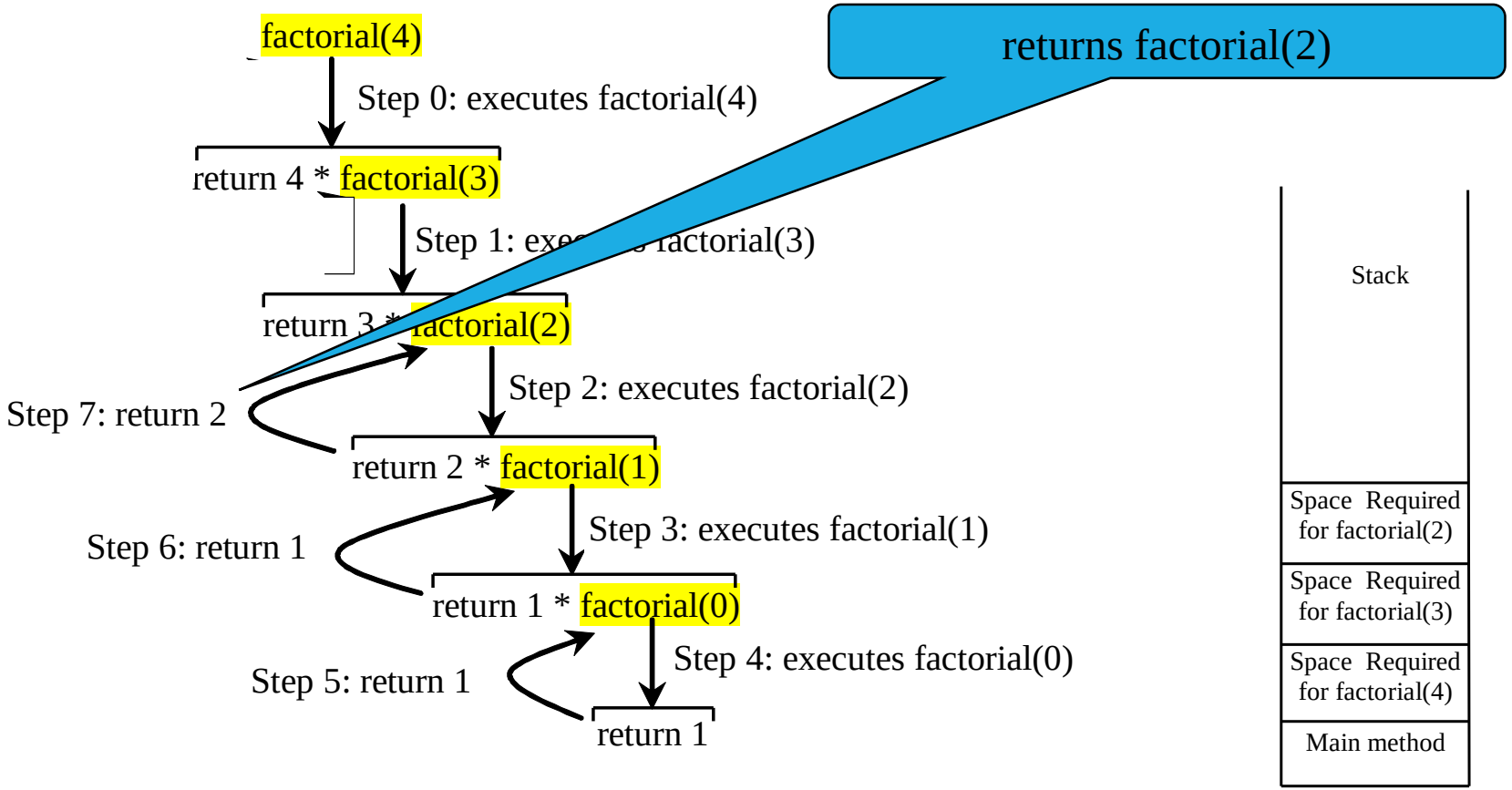
Trace Recursive factorial



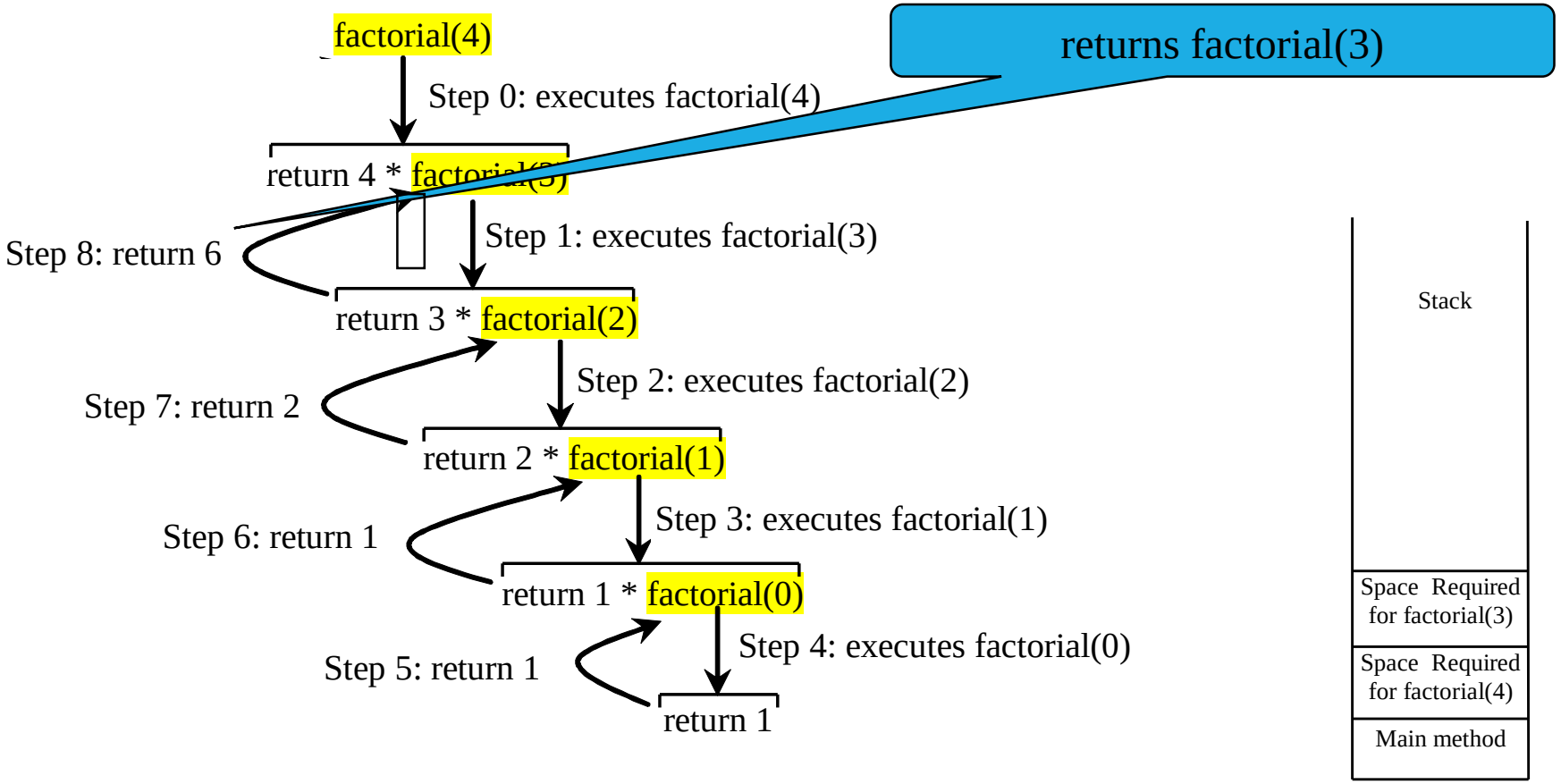
Trace Recursive factorial



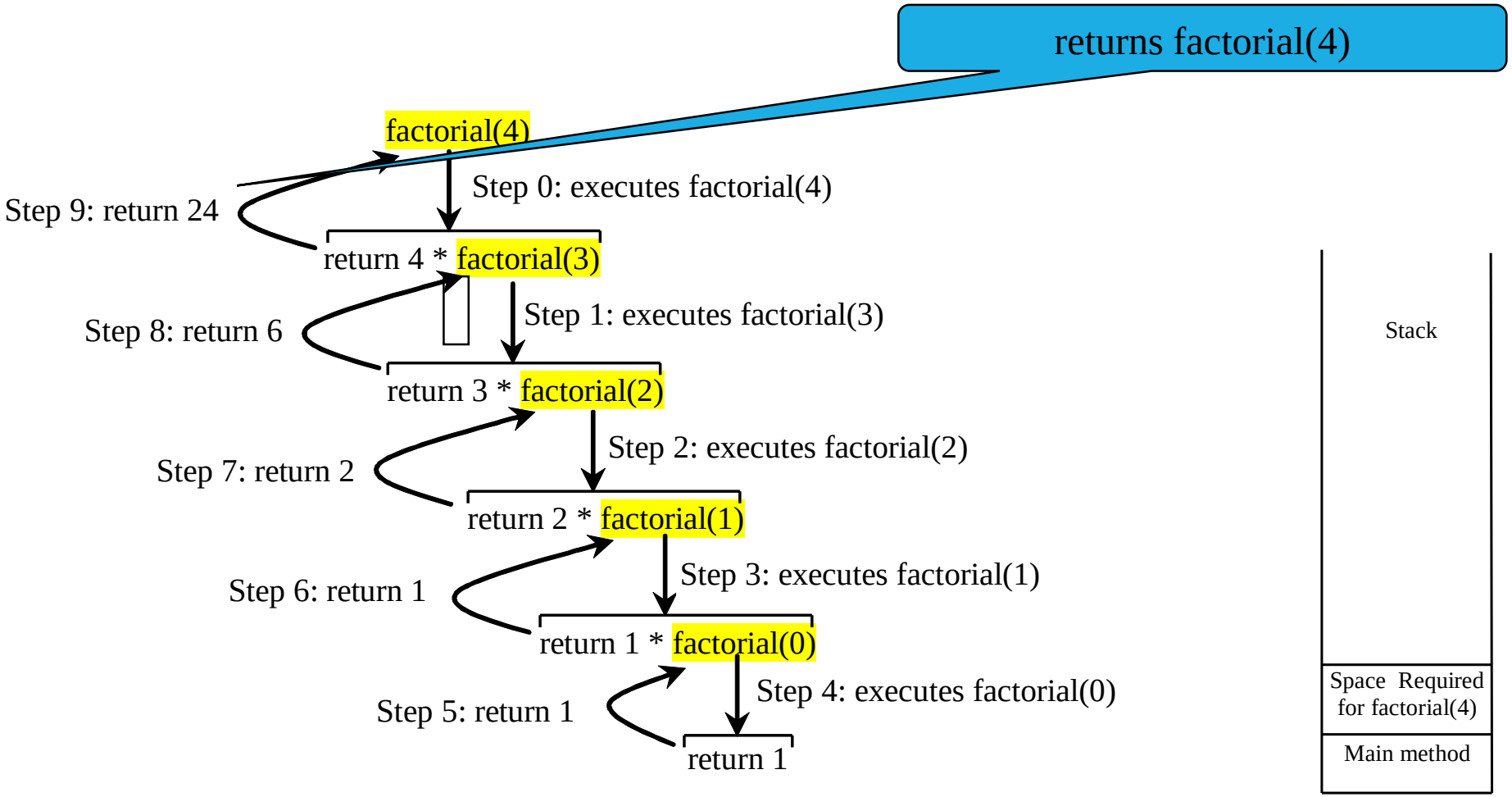
Trace Recursive factorial



Trace Recursive factorial



Trace Recursive factorial



Characteristics of recursive methods:

All recursive methods have the following characteristics:

- One or more base cases (the simplest case) are used to stop recursion.
- Every recursive call reduces the original problem, bringing it increasingly closer to a base case until it becomes that case.

In general, to solve a problem using recursion, you break it into subproblems. If a subproblem resembles the original problem, you can apply the same approach to solve the subproblem recursively. This subproblem is almost the same as the original problem in nature with a smaller size.

Revisiting this factorial recursion

```
public static int factorialRec(int n) {
    if (n == 0){
        return 1;
    }
    else{
        return n * factorialRec(n-1);
    }
}
```

Try it with n = 13, what happens?

Revisiting this factorial recursion

```
public static int factorialRec(int n) {
    if (n == 0){
        return 1;
    }
    else{
        return n * factorialRec(n-1);
    }
}
```

Try it with n = 13, what happens?
It returns 1,932,053,504

Revisiting this factorial recursion

```
public static int factorialRec(int n) {
    if (n == 0){
        return 1;
    }
    else{
        return n * factorialRec(n-1);
    }
}
```

Try it with n = 13, what happens?
It returns 1,932,053,504 (should've been 6,227,020,800). But why?

Revisiting this factorial recursion

```
public static int factorialRec(int n) {
    if (n == 0){
        return 1;
    }
    else{
        return n * factorialRec(n-1);
    }
}
```

Max value of int is 2,147,483,647

Try it with n = 13, what happens?
It returns 1,932,053,504 (should've been 6,227,020,800). But why?

Solution

```
public static long factorialRec(int n) {
    if (n == 0){
        return 1;
    }
    else{
        return n * factorialRec(n-1);
    }
}
```

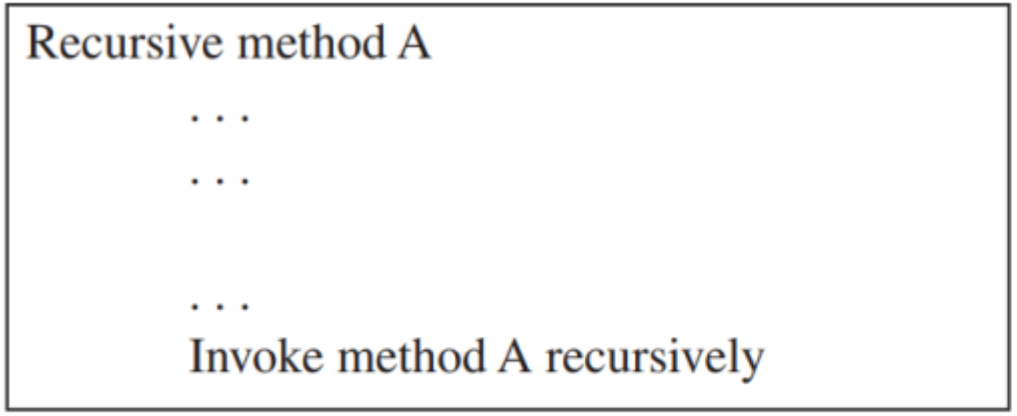
Use **long** instead of **int** as **return type**

Be careful infinite recursion

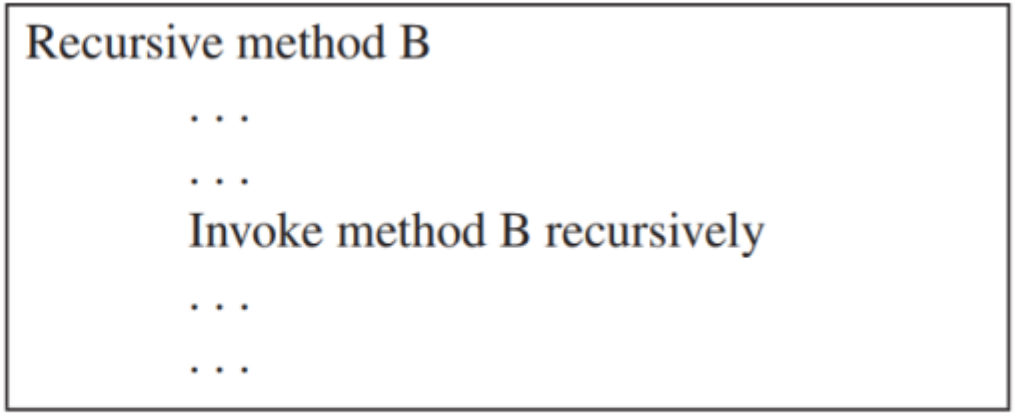
- Check if there is a base case
- Recursive call must move toward the base case

Tail Recursion

- A recursive method is said to be tail recursive if there are no pending operations to be performed on return from a recursive call.



(a) Tail recursion



(b) Nontail recursion