CS112

Recursion (Part 1)

Chapter 18

Lecture 13

الفصل الدراسي الثاني 1443- Spring 2022

College of Computer Science and Engineering

Introduction

- Suppose you want to find all the files under a directory that contains a particular word. How do you solve this problem?
- There are several ways to solve this problem. Can you give me examples?
- **Recursion**: An intuitive solution is to use recursion by searching the files in the subdirectories recursively.

Case Study – Computing Factorial

factorial(0) = 1;

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

n! = n * (n-1)!

- See ComputeFactorial.java

Computing Factorial (1/10)

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

factorial(4)

Computing Factorial (2/10)

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

factorial(4) = 4 * factorial(3)

Computing Factorial (3/10)

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

factorial(4) = 4 * factorial(3)
= 4 * 3 * factorial(2)

Computing Factorial (4/10)

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

factorial(4) = 4 * factorial(3)
= 4 * 3 * factorial(2)
= 4 * 3 * (2 * factorial(1))

Computing Factorial (5/10)

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

factorial(4) = 4 * factorial(3)
= 4 * 3 * factorial(2)
= 4 * 3 * (2 * factorial(1))
= 4 * 3 * (2 * (1 * factorial(0)))

Computing Factorial (6/10)

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

factorial(4) = 4 * factorial(3)
= 4 * 3 * factorial(2)
= 4 * 3 * (2 * factorial(1))
= 4 * 3 * (2 * (1 * factorial(0)))
= 4 * 3 * (2 * (1 * 1)))

Computing Factorial (7/10)

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

factorial(4) = 4 * factorial(3)
= 4 * 3 * factorial(2)
= 4 * 3 * (2 * factorial(1))
= 4 * 3 * (2 * (1 * factorial(0)))
= 4 * 3 * (2 * (1 * 1)))
= 4 * 3 * (2 * 1)

Computing Factorial (8/10)

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

factorial(4) = 4 * factorial(3)
= 4 * 3 * factorial(2)
= 4 * 3 * (2 * factorial(1))
= 4 * 3 * (2 * (1 * factorial(0)))
= 4 * 3 * (2 * (1 * 1)))
= 4 * 3 * (2 * 1)
= 4 * 3 * 2

Computing Factorial (9/10)

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

factorial(4) = 4 * factorial(3)
= 4 * 3 * factorial(2)
= 4 * 3 * (2 * factorial(1))
= 4 * 3 * (2 * (1 * factorial(0)))
= 4 * 3 * (2 * (1 * 1)))
= 4 * 3 * (2 * 1)
= 4 * 3 * 2
= 4 * 6

Computing Factorial (10/10)

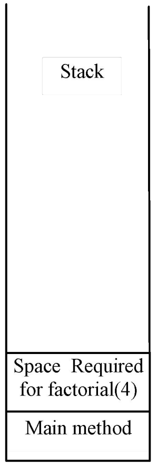
factorial(0) = 1;
factorial(n) = n*factorial(n-1);

```
factorial(4) = 4 * factorial(3)
             = 4 * 3 * factorial(2)
             = 4 * 3 * (2 * factorial(1))
             = 4 * 3 * ( 2 * (1 * factorial(0)))
             = 4 * 3 * ( 2 * ( 1 * 1)))
             = 4 * 3 * ( 2 * 1)
             = 4 * 3 * 2
             = 4 * 6
             = 24
```

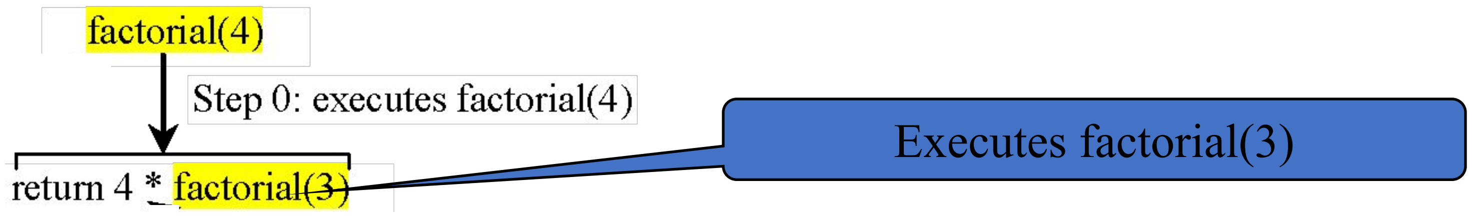
Trace Recursive factorial (1/11)

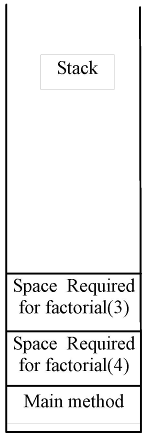
Executes factorial(4)

factorial(4)

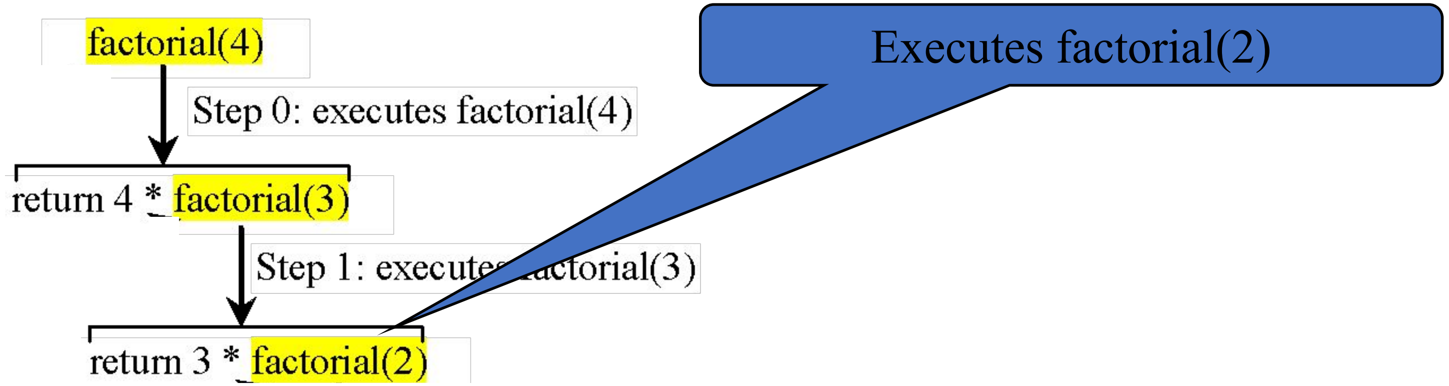


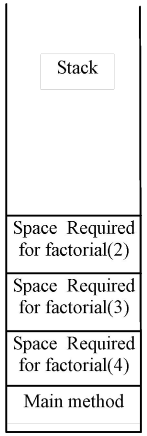
Trace Recursive factorial (2/11)



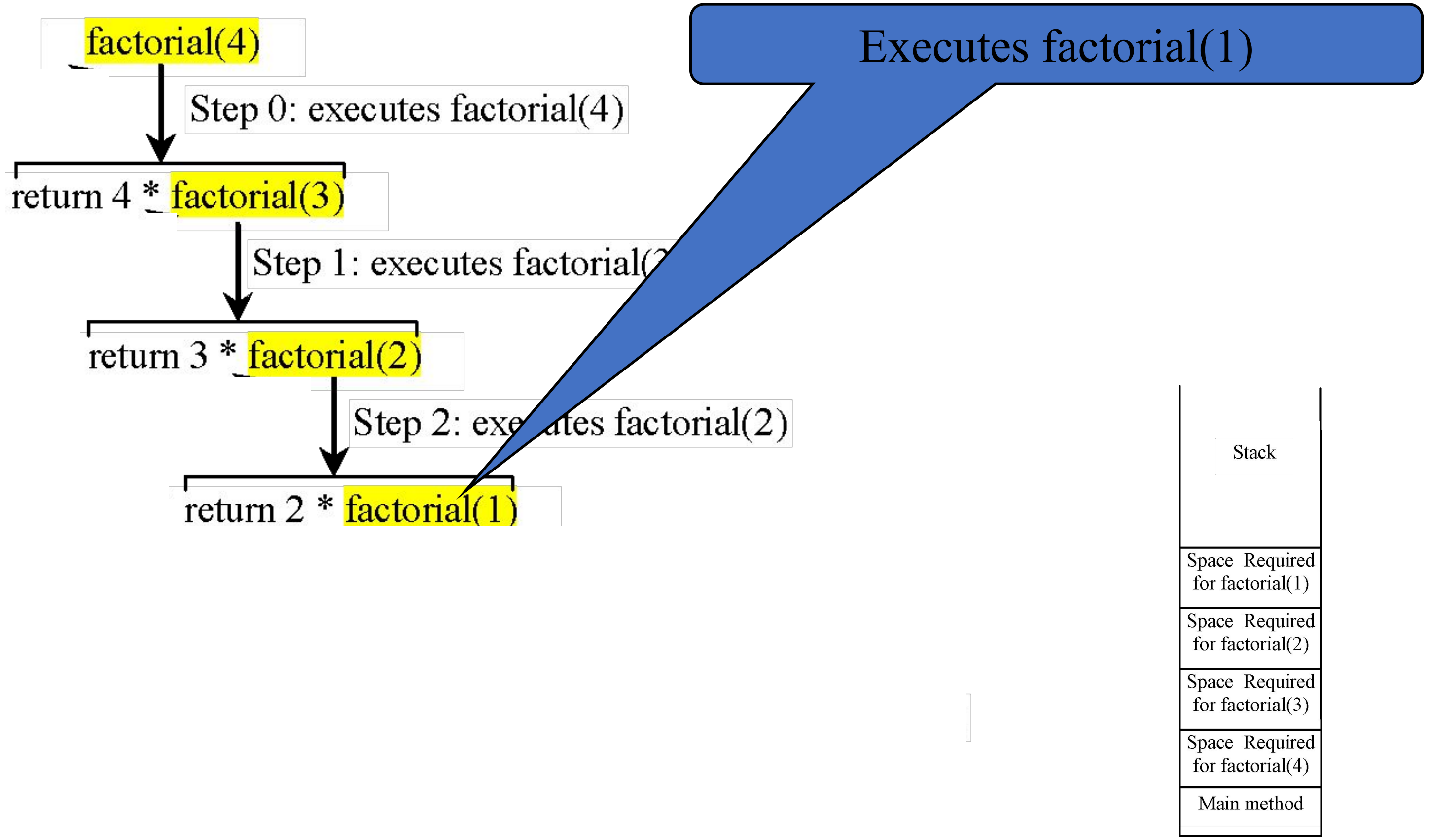


Trace Recursive factorial (3/11)

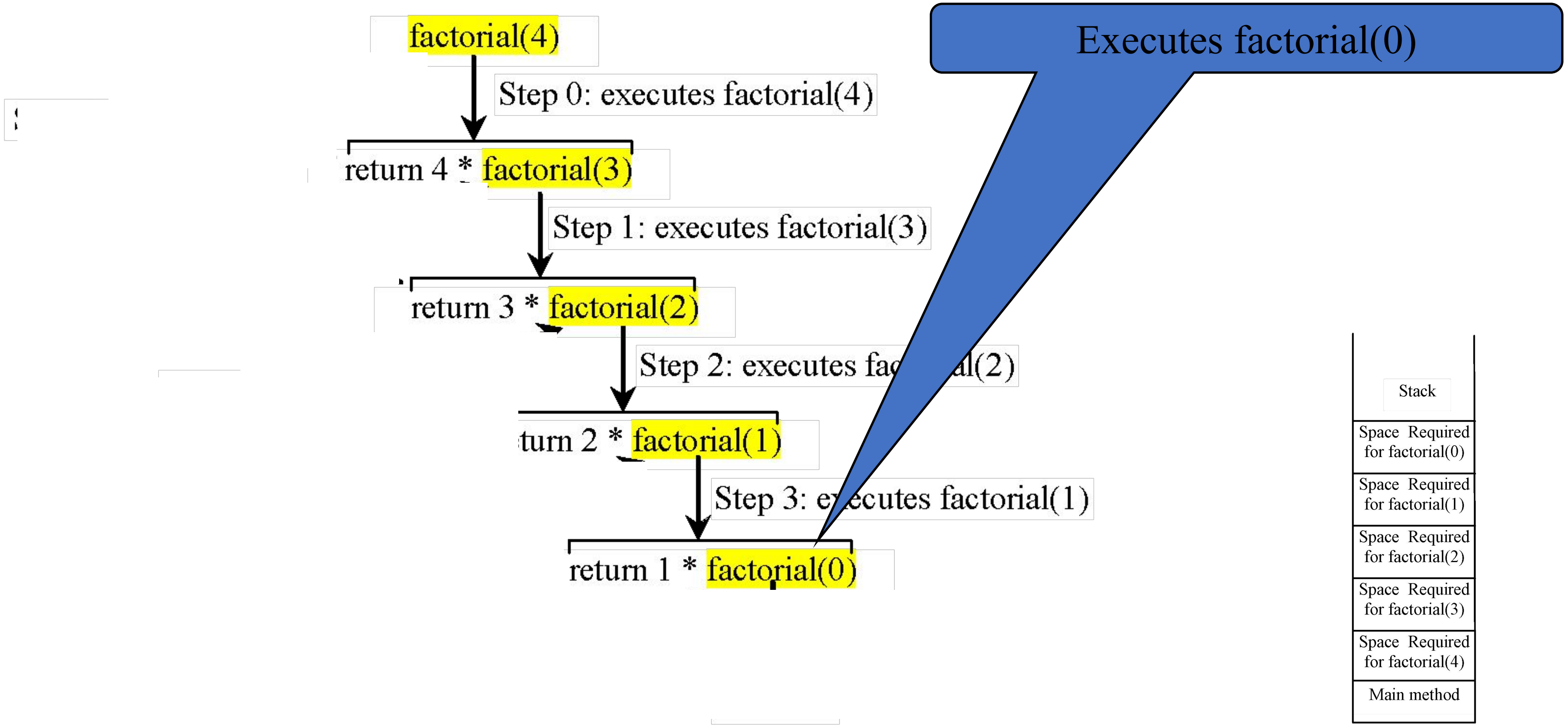




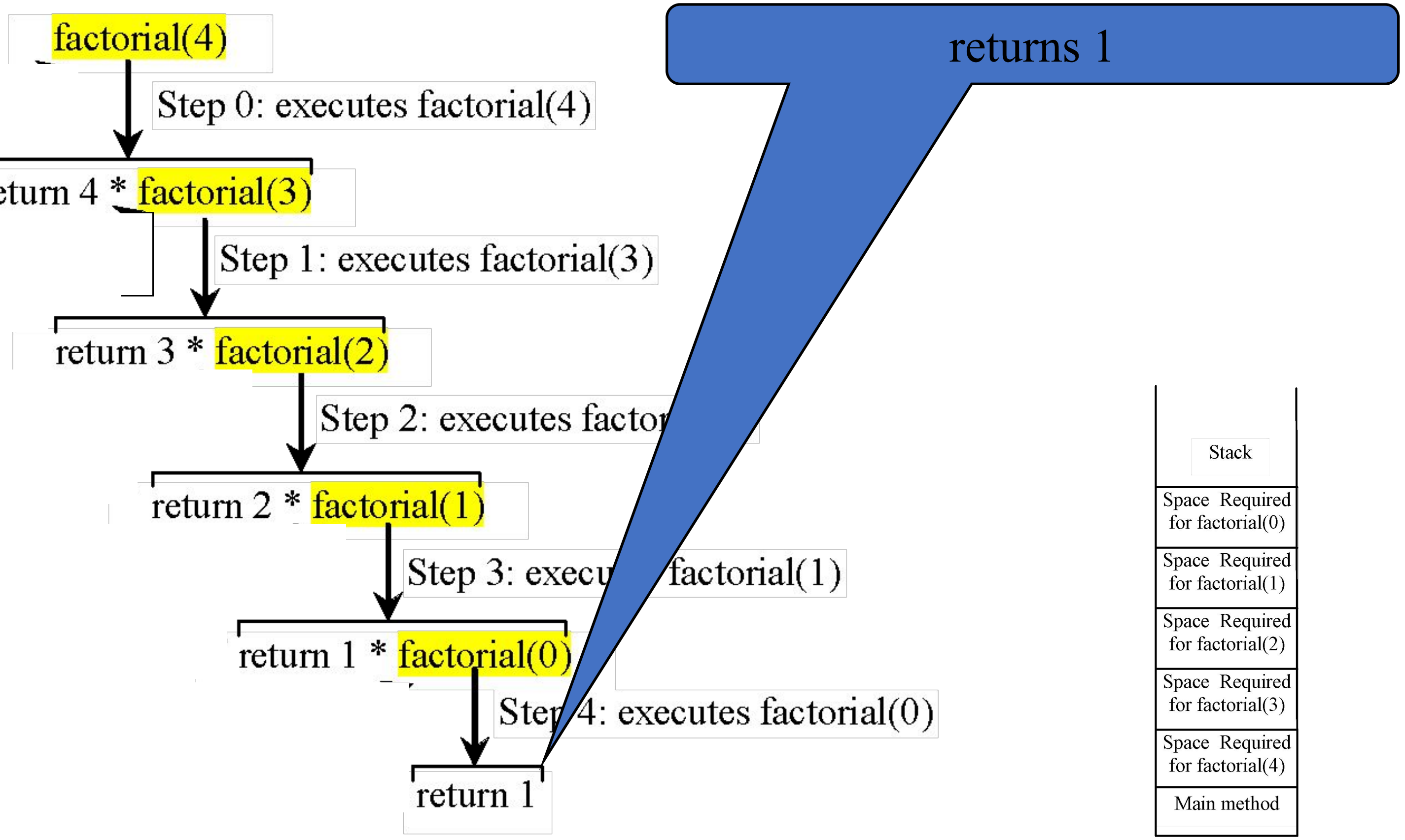
Trace Recursive factorial (4/11)



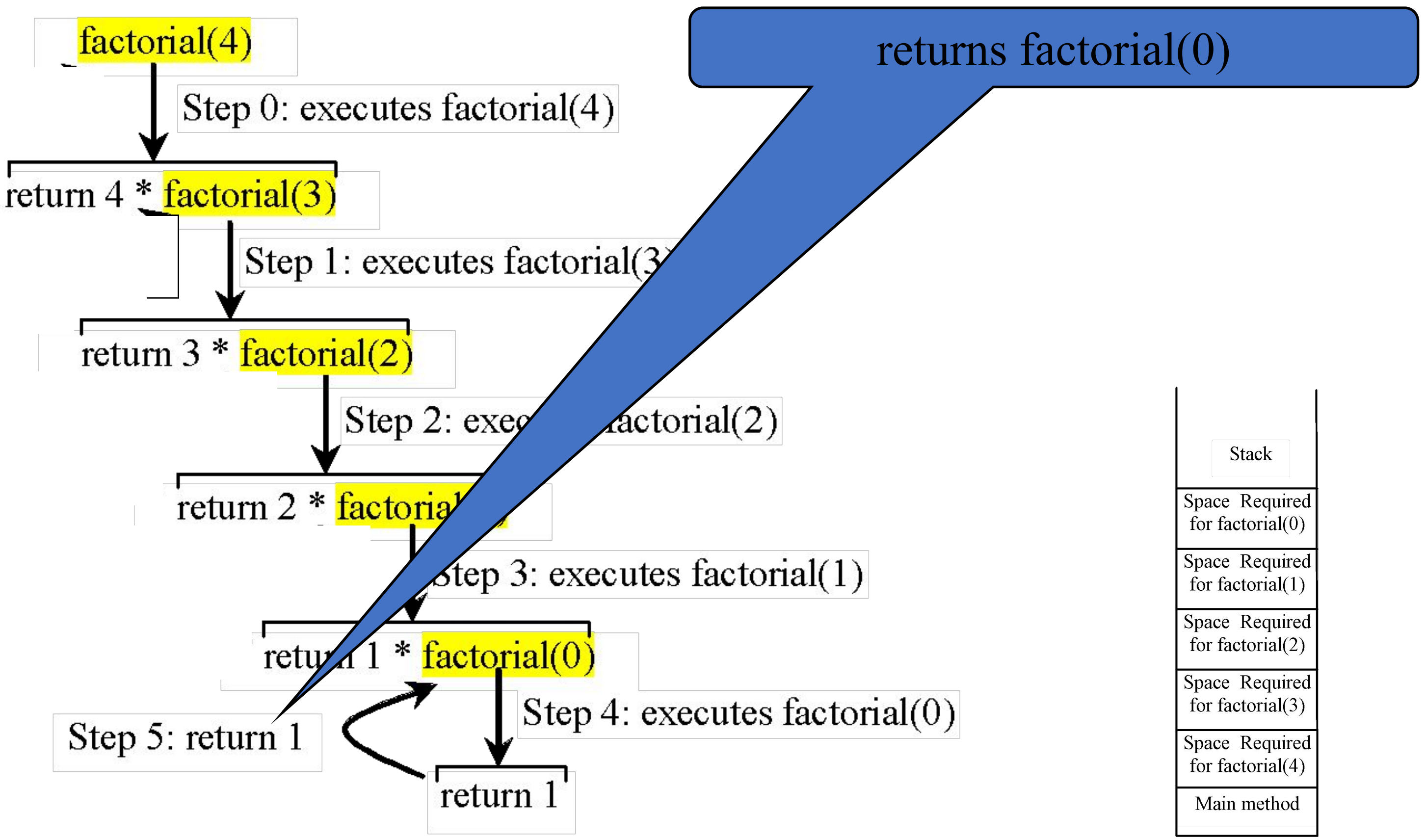
Trace Recursive factorial (5/11)



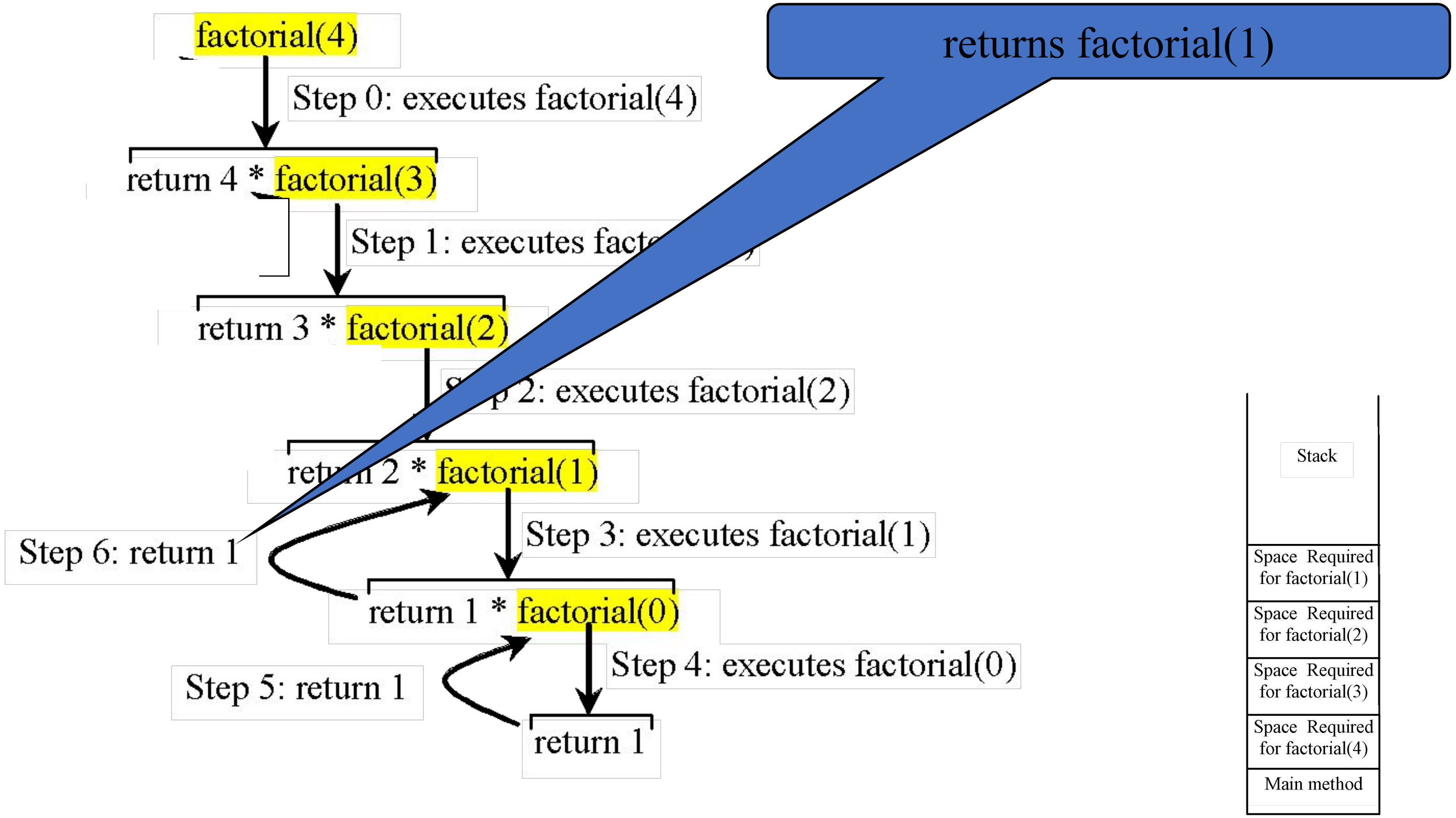
Trace Recursive factorial (6/11)



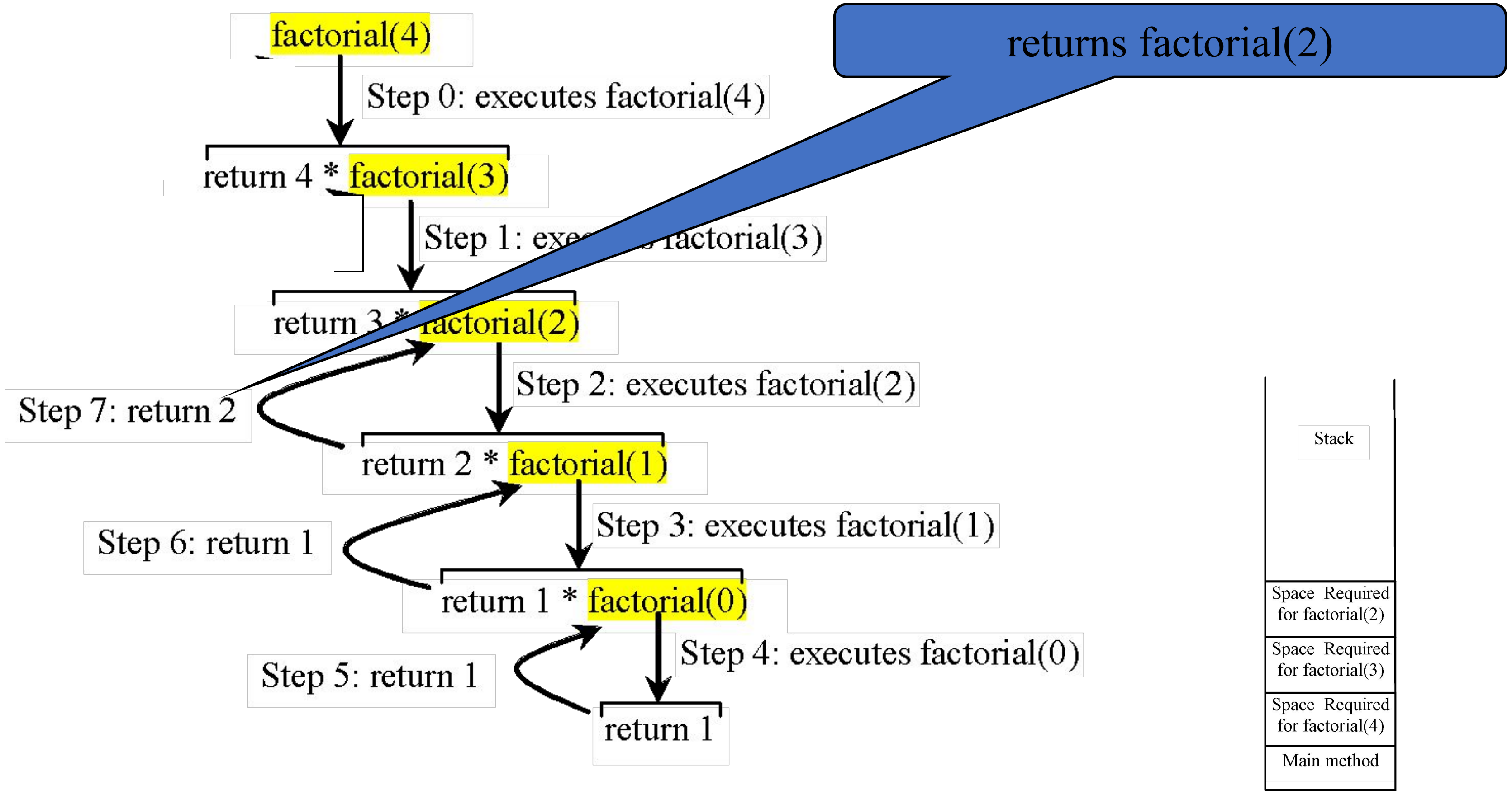
Trace Recursive factorial (7/11)



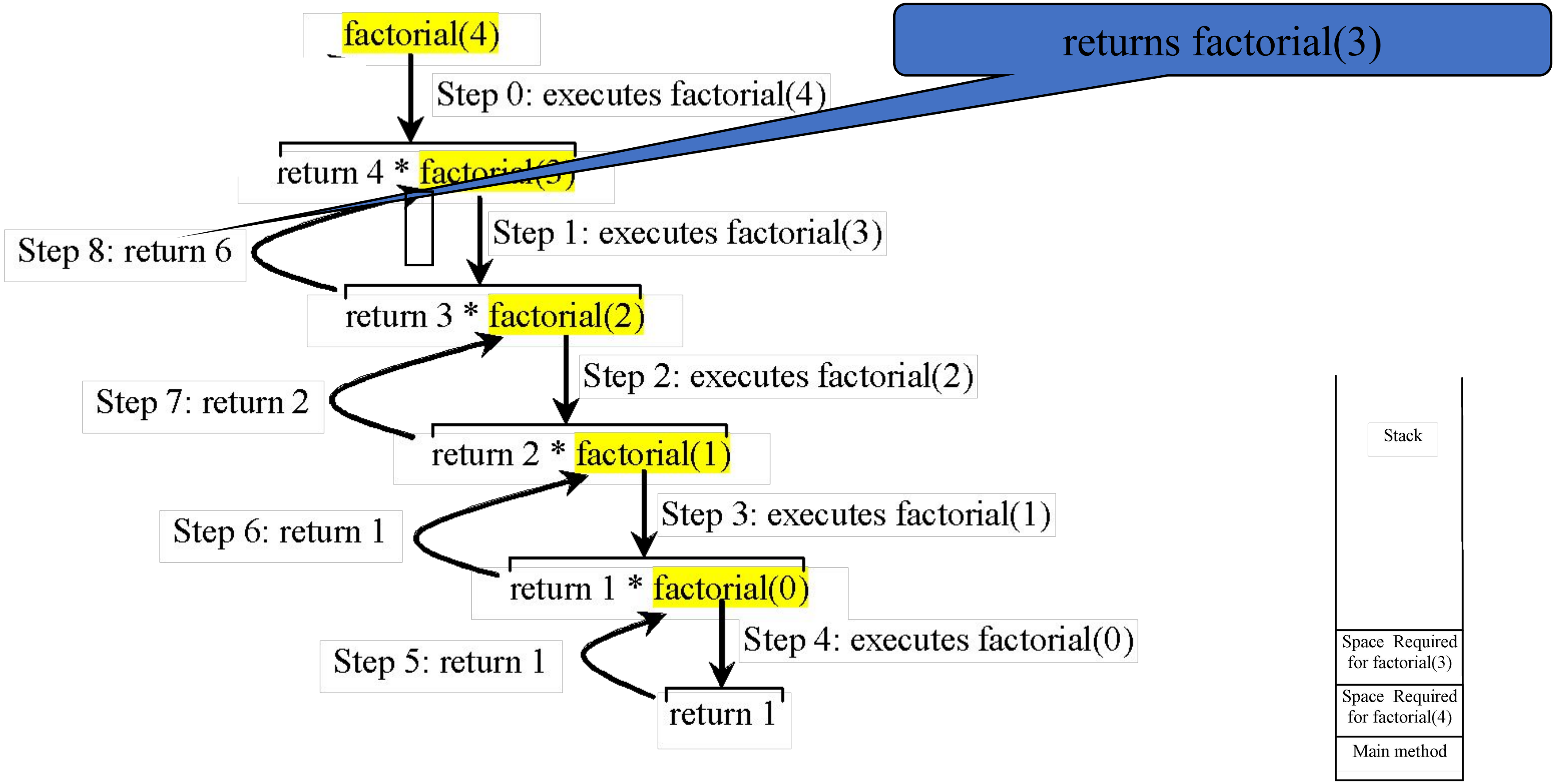
Trace Recursive factorial (8/11)



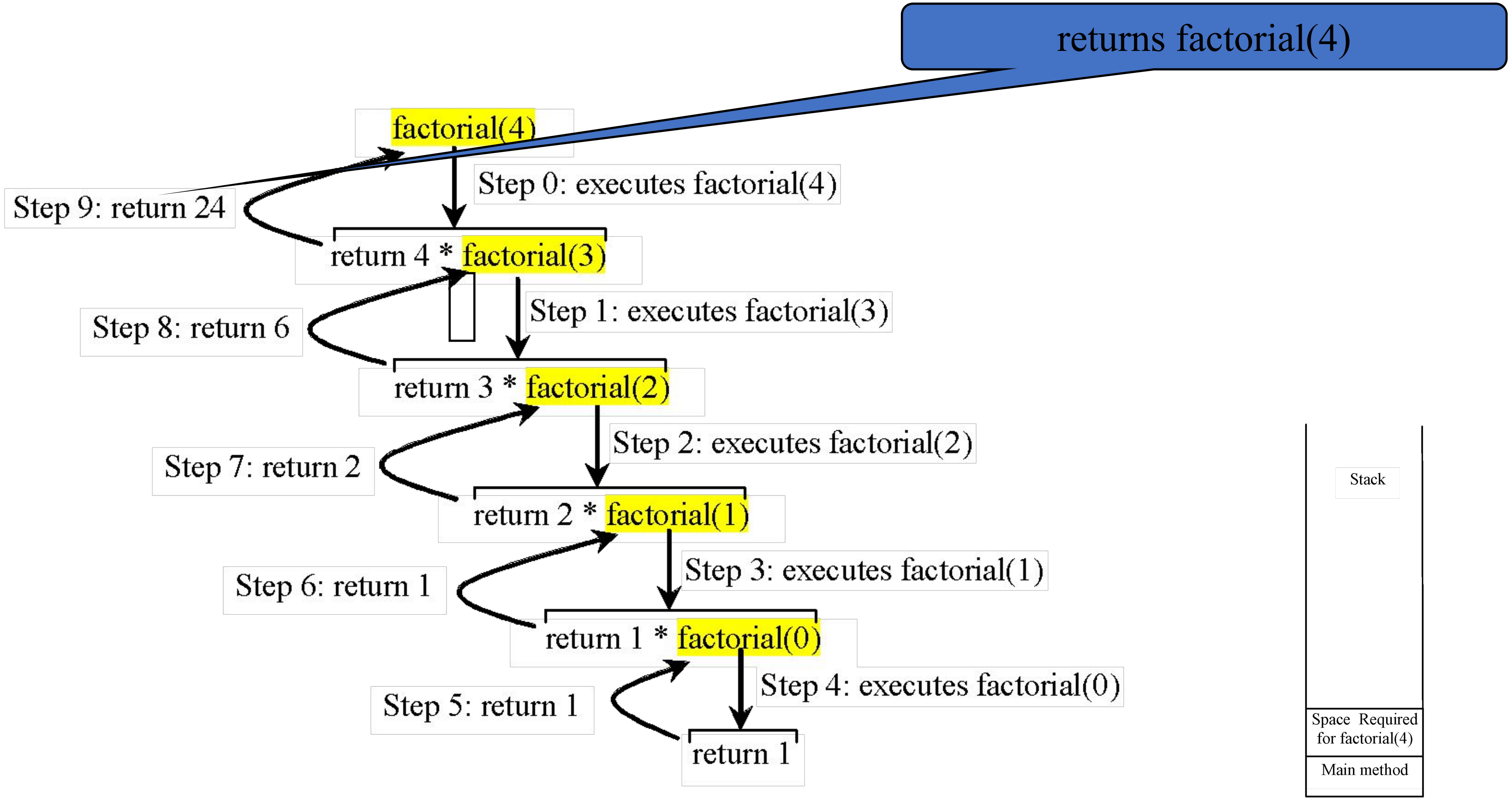
Trace Recursive factorial (9/11)



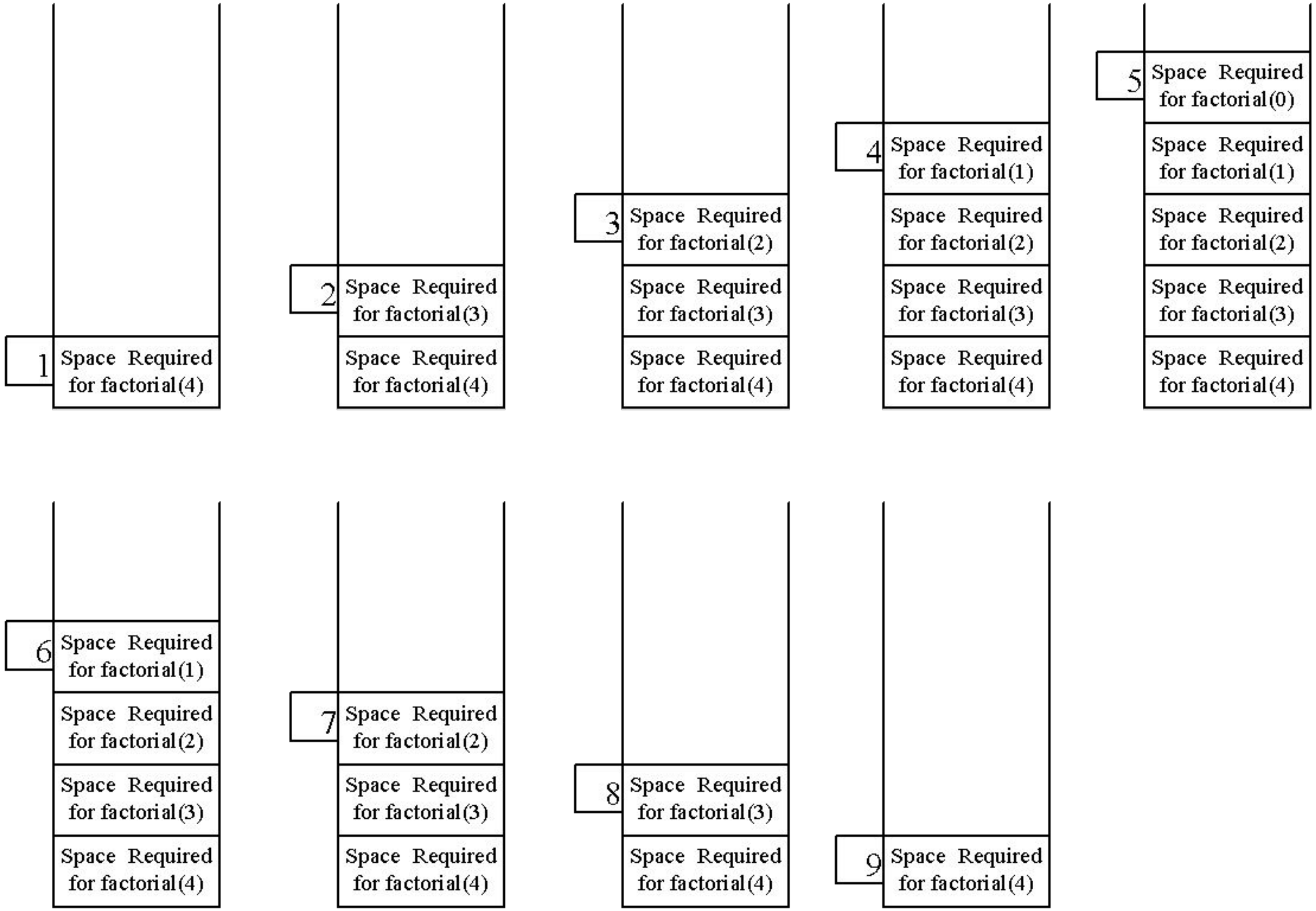
Trace Recursive factorial (10/11)



Trace Recursive factorial (11/11)



factorial(4) Stack Trace



Other Examples

f(0) = 0;

f(n) = n + f(n-1);

Case Study - Fibonacci Numbers (1/2)

```
Fibonacci series: 0 1 1 2 3 5 8 13 21 34 55 89…
         indices: 0 1 2 3 4 5 6 7  8  9  10 11
```

fib(0) = 0;

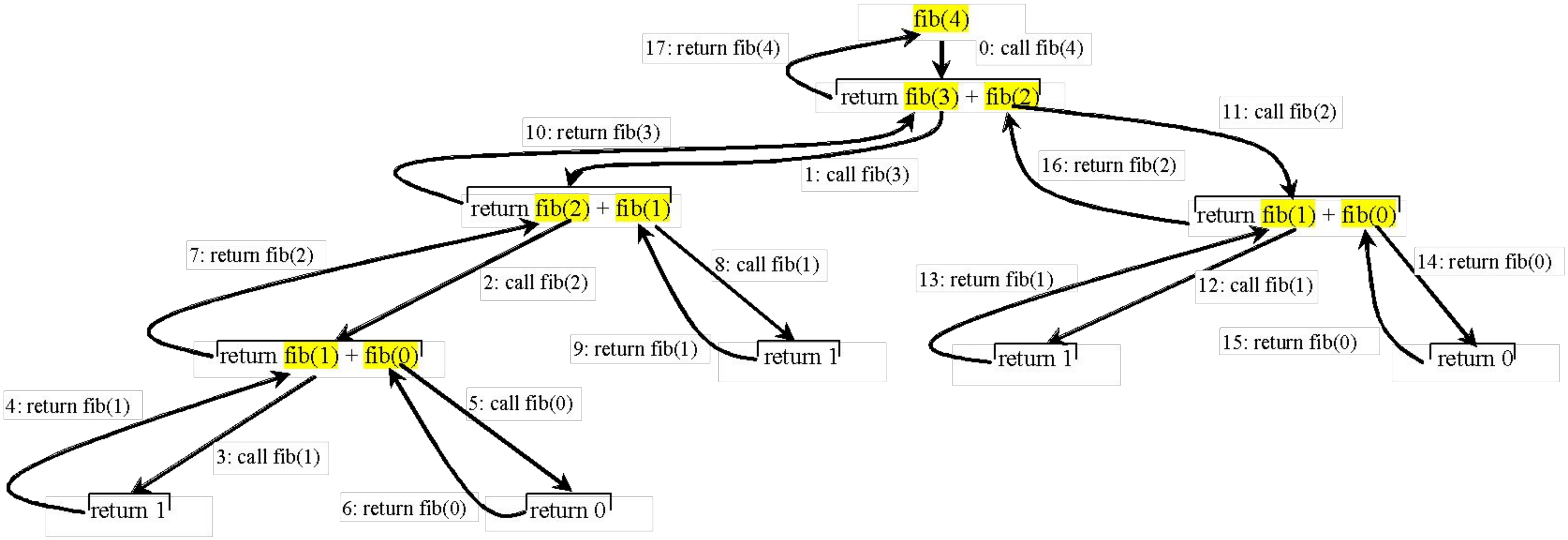
fib(1) = 1;

fib(index) = fib(index -1) + fib(index -2); index >=2

fib(3) = fib(2) + fib(1) = (fib(1) + fib(0)) + fib(1) = (1 + 0) +fib(1) = 1 + fib(1) = 1 + 1 = 2

See ComputeFibonacci.java

Case Study - Fibonacci Numbers (2/2)



Characteristics of Recursion

- 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.

Problem Solving Using Recursion

Let us consider a simple problem of printing a message for n times. You can break the problem into two subproblems: one is to print the message one time and the other is to print the message for n-1 times. The second problem is the same as the original problem with a smaller size. The base case for the problem is n==0. You can solve this problem using recursion as follows:

nPrintln("Welcome", 5);

```
public static void nPrintln(String message, int times) {
  if (times >= 1) {
    System.out.println(message);
    nPrintln(message, times - 1);
  } // The base case is times == 0
}
```