BIRZEIT UNIVERSITY

Recursion

Dr. Abdallah Karakra

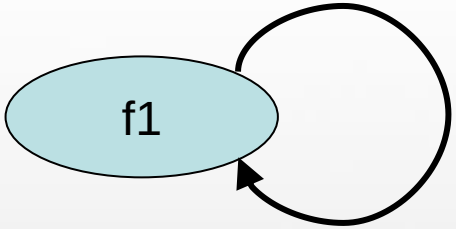
Computer Science Department

COMP242

Monday, April 3, 2023

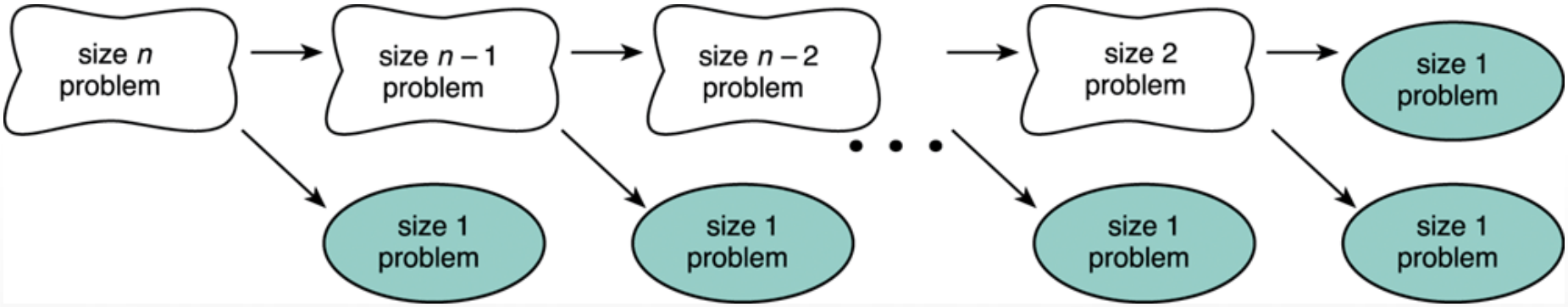
Introduction to Recursion

- A recursive function is one that calls itself.



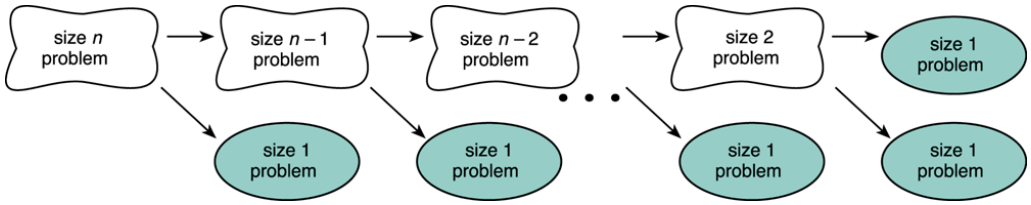

```
public void message()
{
        System.out.println("This is a recursive function");
        message();
}
```

Splitting a Problem into Smaller Problems



- Assume that the problem of size 1 can be solved easily (i.e., the simple case).
- We can recursively split the problem into a problem of size 1 and another problem of size n-1.

Splitting a Problem into Smaller



Let f(x)=f(x -1)+3 , f(0)=4 , find f(7)

f(7) = f(7-1)+3 → f(7)=f(6)+3	f(7)=22+2=25
f(6) = f(6-1)+3 → f(6)=f(5)+3	f(6)=19+3=22
f(5) = f(5-1)+3 → f(5)=f(4)+3	f(5)=16+3=19
f(4) = f(4-1)+3 → f(4)=f(3)+3	f(4)=13+3=16
f(3) = f(3-1)+3 → f(3)=f(2)+3	f(3)=10+3=13
f(2) = f(2-1)+3 → f(2)=f(1)+3	f(2)=7+3=10
f(1) = f(1-1)+3 → f(1)=f(0)+3	f(1)=4+3=7
	f(0)=4

Base case

Recursive Problem

The function below displays the string "This is a recursive function.", and then calls itself.

```
public void message()
{
        System.out.println("This is a recursive function");
        message();
}
```

Recursive Problem

- The function is like an **infinite loop** because there is **no code to stop it from repeating.**
- Like a loop, a recursive function **must have some algorithm to control the number of times it repeats.**

Recursion

- Like a loop, a recursive function must have some algorithm to control the number of times it repeats. Shown below is a modification of the **`message`** function. It passes an integer argument, which holds the number of times the function is to call itself.

```
Public void message(int times)
{
        if (times > 0)
        {
                System.out.println("This is a recursive function");
                message(times - 1);
        }

}
```

Recursion

- The function contains an `if/else` statement that controls the repetition.
- As long as the `times` argument is greater than zero, it will display the message and call itself again. Each time it calls itself, it passes `times - 1` as the argument.

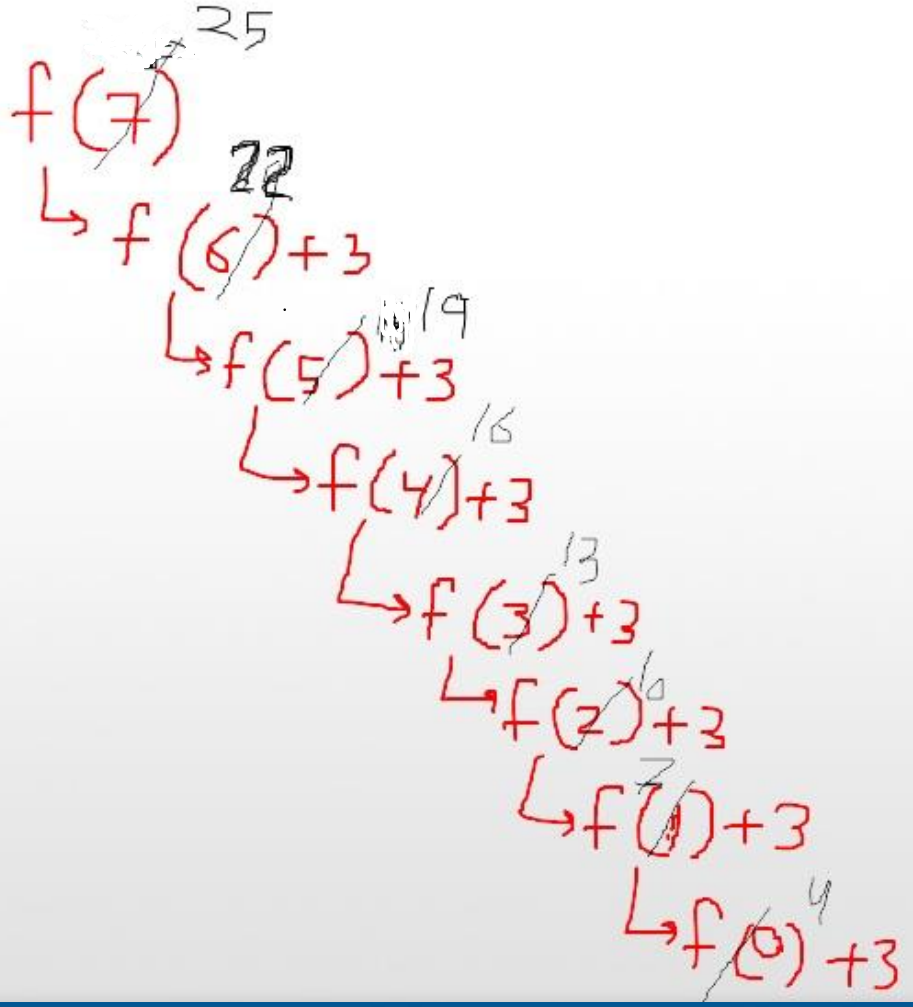
Recursive Function

Let f(x)=f(x -1)+3 , f(0)=4 , find f(7)

```
public int f(int x)
{
  if (x == 0)
       return 4; //base case
  else
       return f(x-1)+3;

}
```

Recursive function terminates when a base case is met.

Trace of f(x)=f(x -1)+3



Recursive Function Factorial

In mathematics, the notation n! represents the factorial of the number n. The factorial of a number is defined as:

$$fact\ (n) = \begin{cases} 1 & , \quad n = 0 \\ n*fact\ (n-1) & , \quad n>0 \end{cases}$$

In other words,

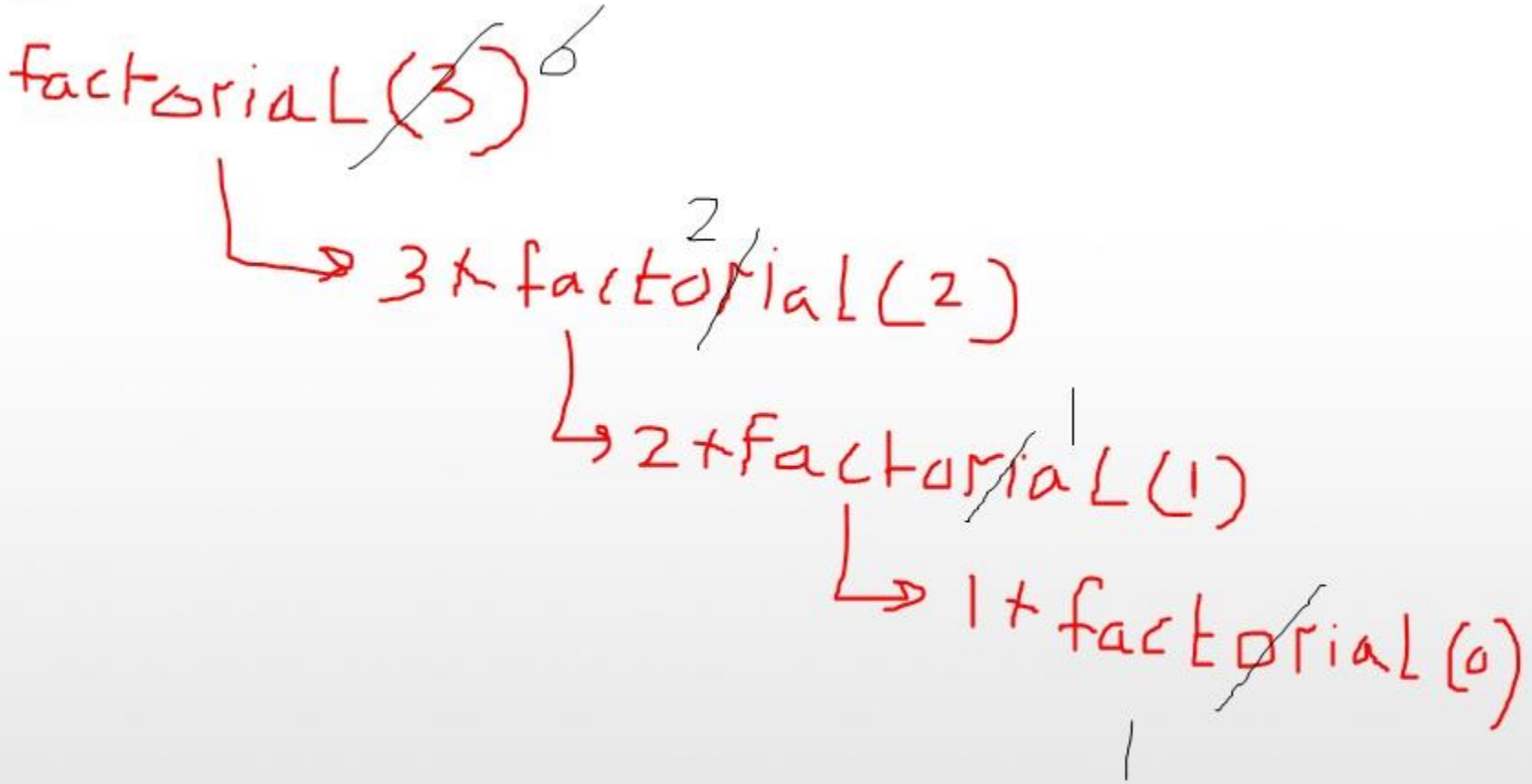
$$n! = \begin{cases} 1 * 2 * 3 * \ldots * n & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$$

Recursive Function Factorial

The following Java function implements the recursive definition of factorial:

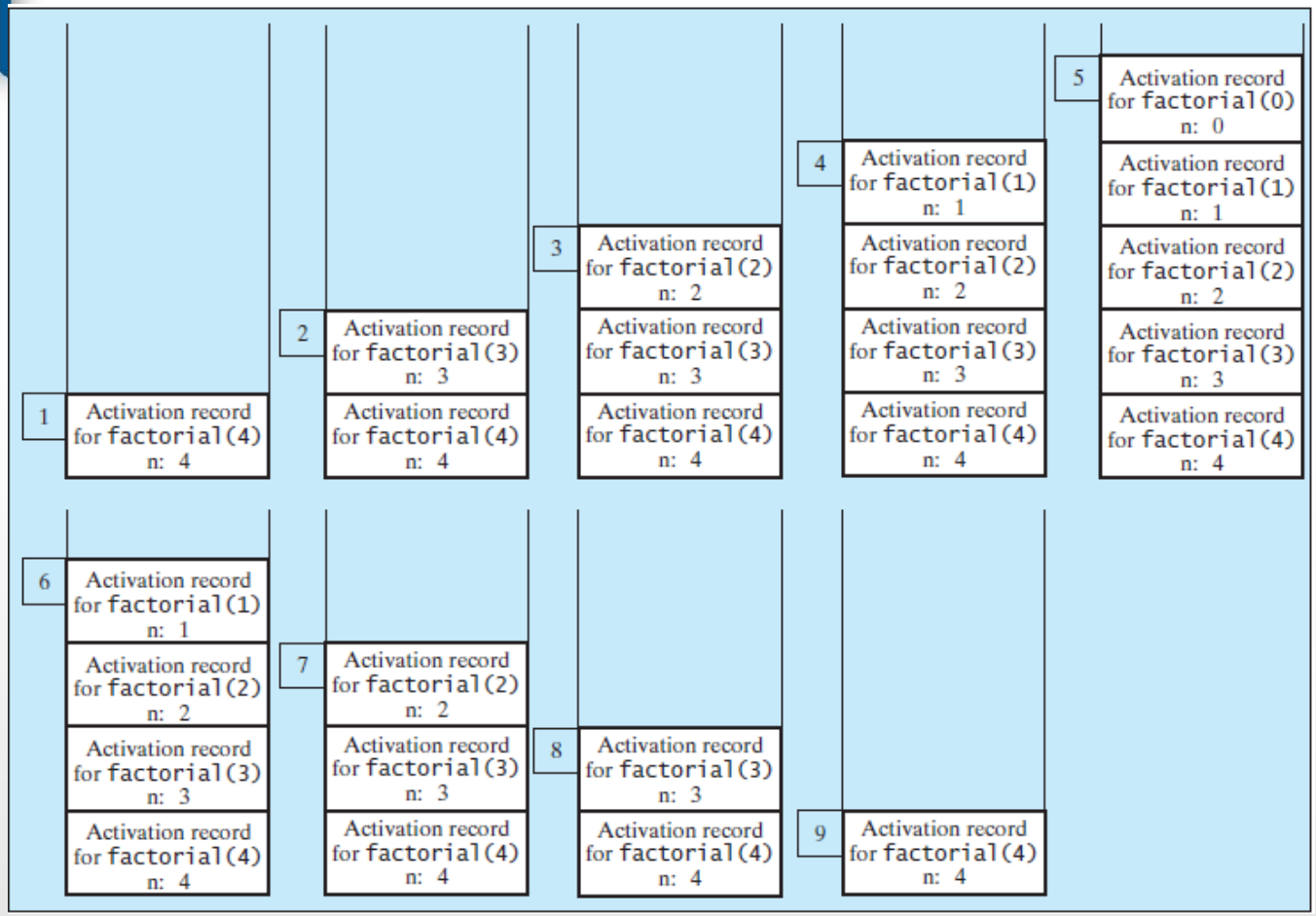
```
/** Return the factorial for the specified number */
public static long factorial(int n) {
  if (n == 0) // Base case                                    base case
    return 1;
  else
    return n * factorial(n - 1); // Recursive call            recursion
}
```

Trace of fact = factorial(3);



Trace of fact = factorial(4);

The Stack "FILO"



When `factorial(4)` is being executed, the `factorial` method is called recursively, causing stack space to dynamically change.

Recursive Function Power

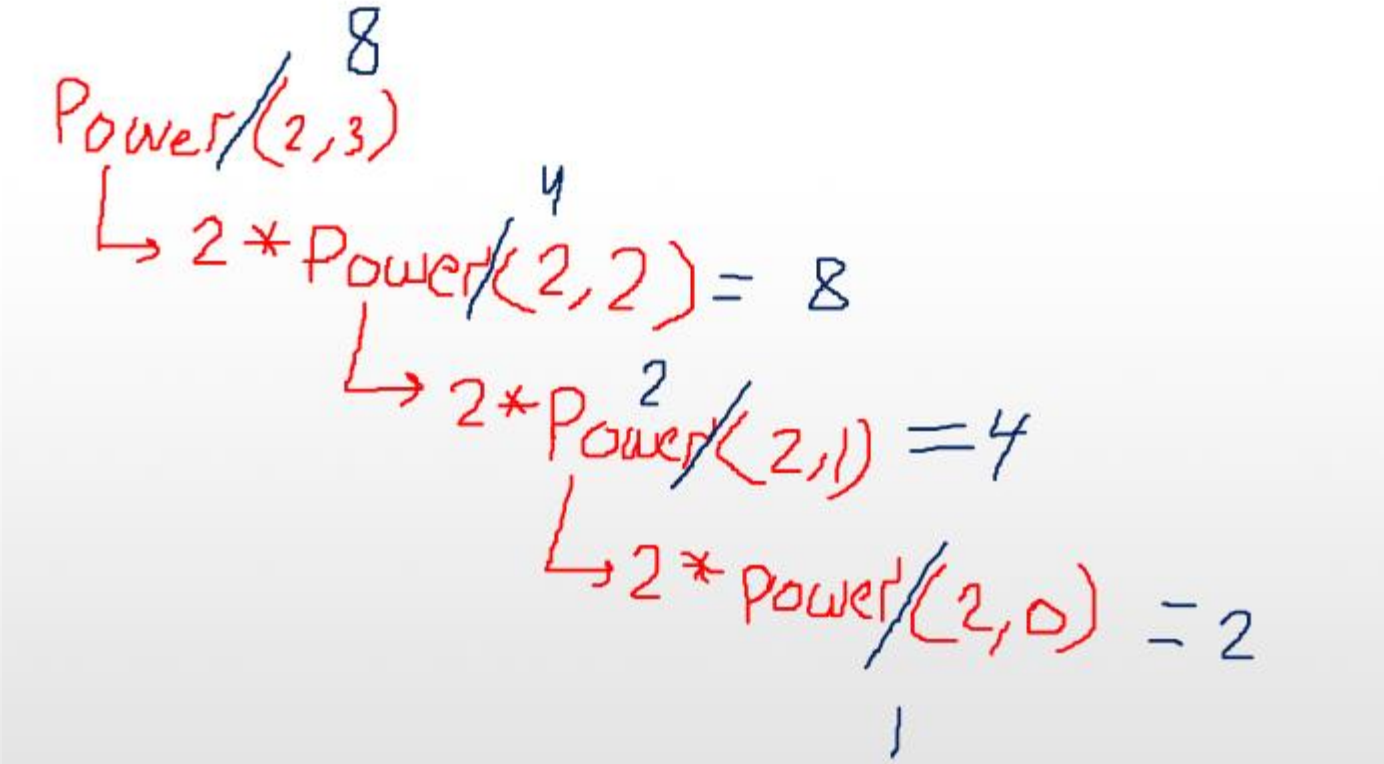
$$\text{Power}(x,y) = \begin{cases} 1 & , \quad y = 0 \\ X * \text{power}(x, y-1) & , \quad y > 0 \end{cases}$$

Recursive Function Power

The following Java function implements the recursive definition of power:

```
public static int power(int x, int y)

{
  if (y == 0)

      return 1;

  else

      return x * power(x, y - 1);
}
```

Recursive Function Power



Recursive Function fibonacci

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

a_n: 1, 1, 2, 3, 5, 8, 13, 21, 34,……
n: 1, 2, 3, 4, 5, 6, 7, 8 , 9,……

$a_1 = 1$, $a_2 = 1$, $a_3 = a_1 + a_2 = 2$, $a_n = a_{n-1} + a_{n-2}$

Recursive Function fibonacci

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

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);


}
```