Fakultas Informatika
School of Computing
Telkom University

CSG2A3
ALGORITMA dan STRUKTUR DATA

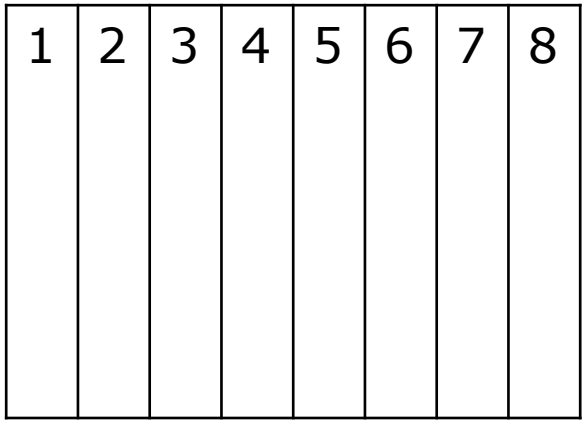
Recursive Algorithm

What is recursion?

- Sometimes, the best way to solve a problem is by solving a smaller version of the exact same problem first
- Example problem:
  - Try to tear a sheet of paper into the same 8 pieces

Tear paper into the same 8 pieces

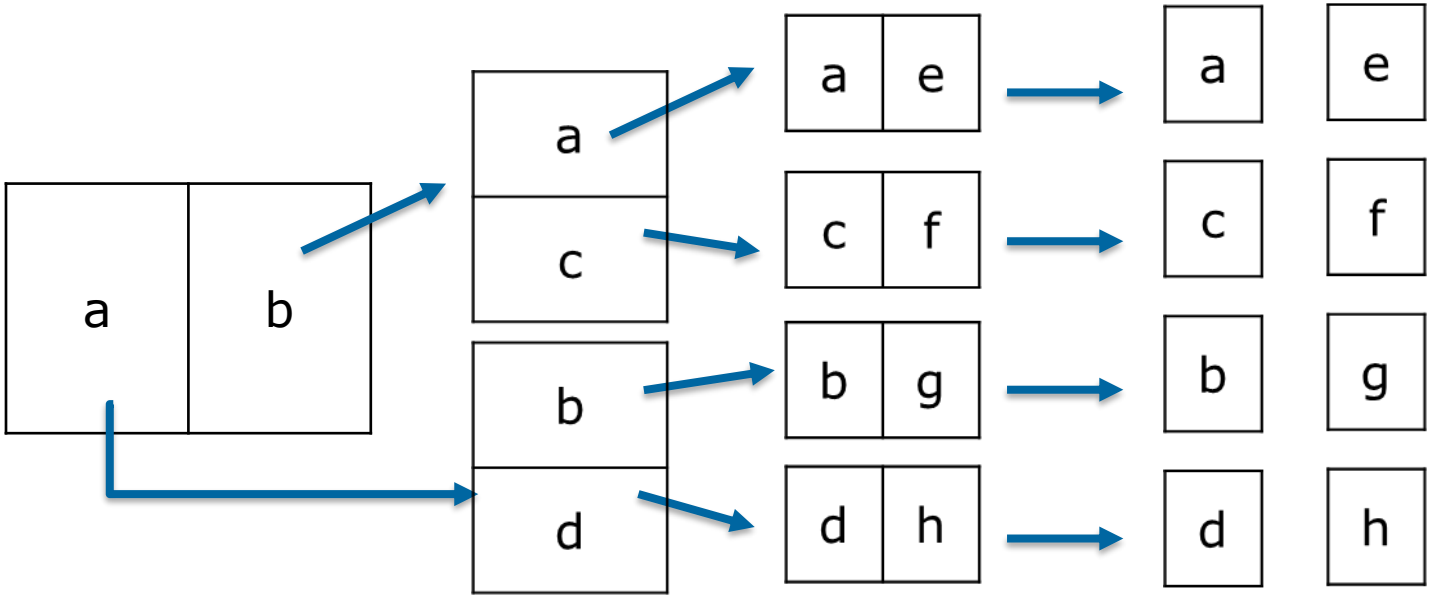
- To solve this, one solution is that we can just tear it 7 times as follows:



- That was an example of the application of looping

Tear paper into the same 8 pieces

- Or we can tear it into 2, and repeat the process for each pieces 2 times



- That is an example of the application of recursive

Recursive Function

- Function that calls itself within the program text

Have you seen this movie?

The movie tells about someone who can dream inside a dream

If you think that the "dream" in the movie is a function, then you'll find it really similar concept to the recursive function

Some Definitions

- Recursion is a technique that solves a problem by solving a smaller problem of the same type
- Recursion is a principle closely related to mathematical induction.

- F(0) = 0
- F(x) = F(x-1) + 2

$$F(x) = \begin{cases} 0 \\ F(x-1) + 2 \end{cases}$$

Example

- Power of two
- $2^n = 2 * 2^{n-1}$
- $2^0 = 1$

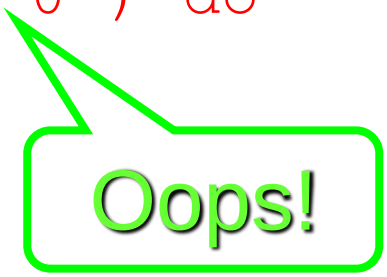
- Factorial
- $X! = X * (X-1)!$
- $0! = 1$

Careful when writing

- If we use iteration, we must be careful not to create an infinite loop by accident:

```
While ( result > 0 ) do

   result ++
```



Careful when writing

- Similarly, if we use recursion we must be careful not to create an infinite chain of function calls

Remember the Rule!

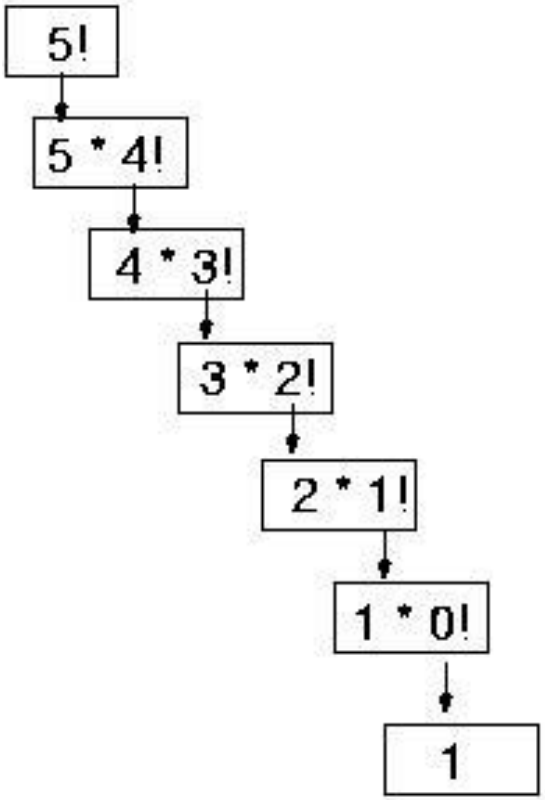
- An Algorithm must stop!
- Define a rule that will stop the recursion

  (initial set / base case)
  - X! = X * (X-1)!
  - 0! = 1
- Define a rule to reach the next iteration

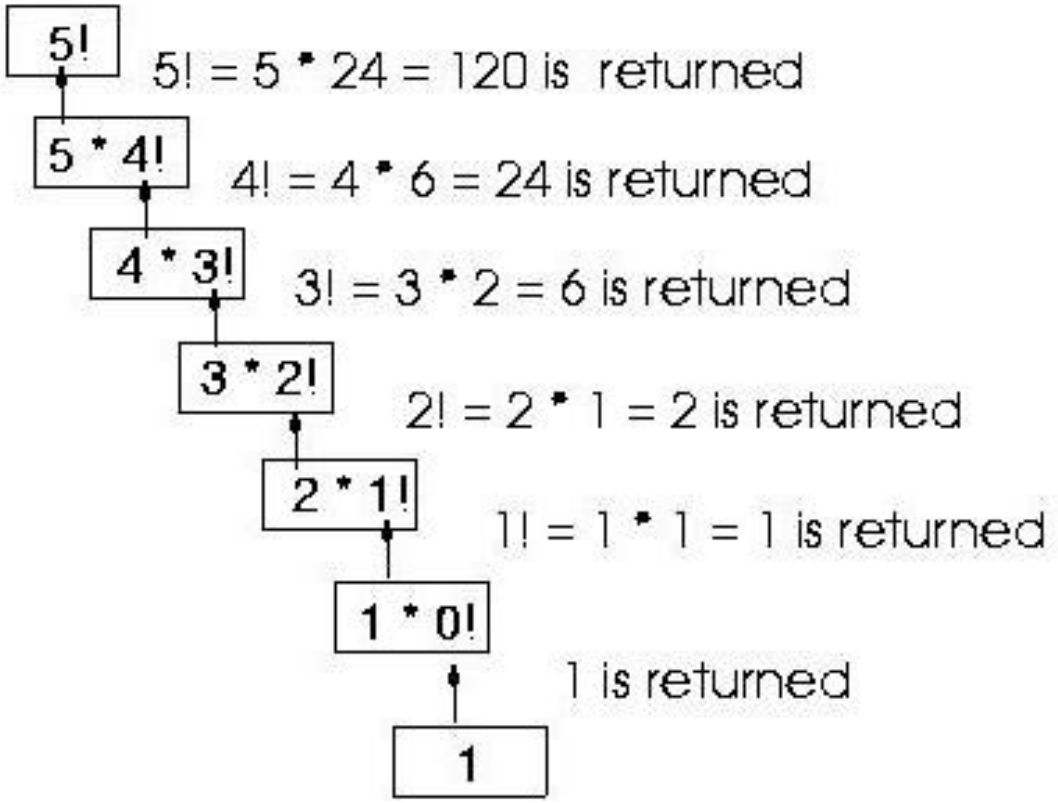
  (construct new element / step)

Algorithm of the factorial function

```
Function Factorial(input : n : integer)
  if (n == 0) then // base case
     → 1
  else
     → n * Factorial(n-1)
```



Final value = 120



A famous example: The Fibonacci numbers

- f(0) = 0, f(1) = 1
- f(n) = f(n – 1) + f(n - 2)

- f(0) = 0
- f(1) = 1
- f(2) = f(1) + f(0) = 1 + 0 = 1
- f(3) = f(2) + f(1) = 1 + 1 = 2
- f(4) = f(3) + f(2) = 2 + 1 = 3
- f(5) = f(4) + f(3) = 3 + 2 = 5
- f(6) = f(5) + f(4) = 5 + 3 = 8

Recursive Algorithm

- An algorithm is called recursive if it solves a problem by reducing it to an instance of the same problem with smaller input
- A recursive function must contain at least one non-recursive branch.
- The recursive calls must eventually lead to a non-recursive branch

Recursion vs. iteration

- For every recursive algorithm, there is an equivalent iterative algorithm
- Iteration can be used in place of recursion
  - An iterative algorithm uses a *looping construct*
  - A recursive algorithm uses a *branching structure*

Recursion vs. iteration

- Recursive solutions are often less efficient, in terms of both *time* and *space*, than iterative solutions
- Recursion can simplify the solution of a problem, often resulting in shorter, more easily understood source code

How do I write a recursive function?

- Determine the size factor
- Determine the base case(s)

  (the one for which you know the answer)
- Determine the general case(s)

  (the one where the problem is expressed as a smaller version of itself)
- Verify the algorithm

  (use the "Three-Question-Method")

Three-Question Verification Method

- The Base-Case Question:
  - Is there a non-recursive way out of the function, and does the routine work correctly for this "base" case?
- The Smaller-Caller Question:
  - Does each recursive call to the function involve a smaller case of the original problem, leading inescapably to the base case?
- The General-Case Question:
  - Assuming that the recursive call(s) work correctly, does the whole function work correctly?

Question?

Write a recursive Algorithm

- Binary Search
- Reverse an Array
- Counts number of zero
- Checking a Palindrome
- Drawing a triangle

  Example :

```
*
**
***
****
```

THANK YOU