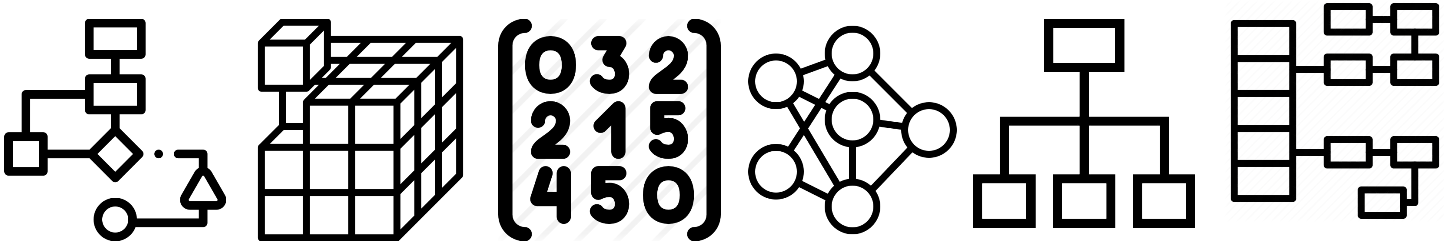
CS211-Algorithms & Data Structures

Taibah University

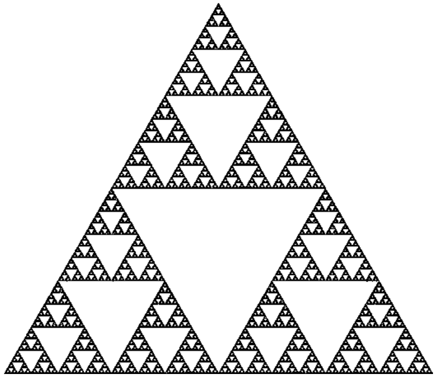
Dr. Sameer M. Alrehaili

College of Science and Computer Engineering, Yanbu



What is Recursion?

- Something whose definition includes itself.
- Self referencing.
- Dreams within your dreams.
- Recursion is useful for big problems to broke down into smaller ones.
- Recursive is used when the problem is naturally recursive (e.g. Fibonacci).
- Recursive is used when the data is naturally recursive (e.g. filesystem).



Recursive algorithms

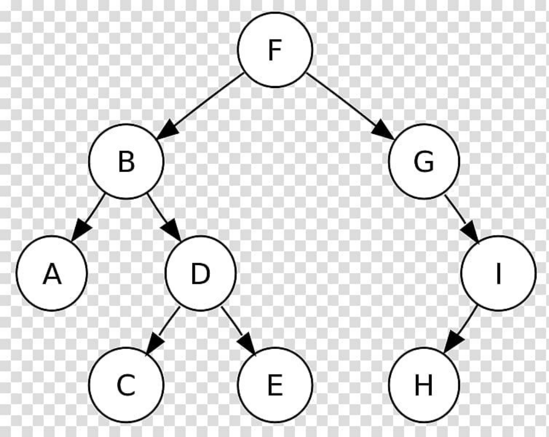
- Any algorithm which calls it self to do part of its work is called a recursive algorithm.
- It is important to ensure that the recursive algorithm terminates. Otherwise, stack overflow error occurs.
- When a problem is defined in terms of similar subtasks, then it is useful to apply recursive methods.

Recursion

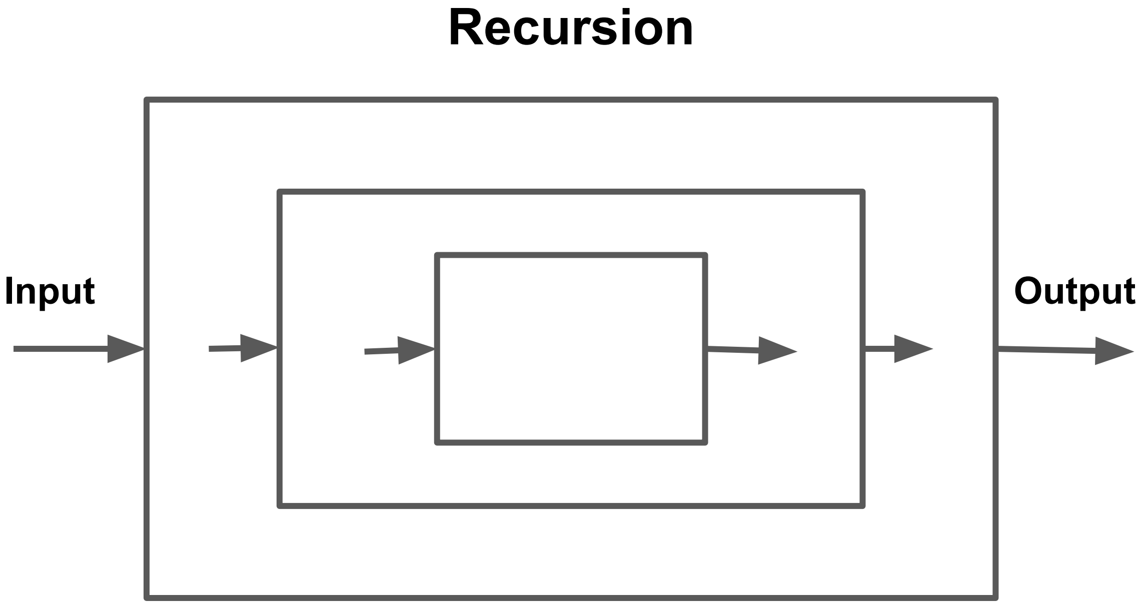
- Recursion is a way of solving problems by having a function call itself.
- Recursion is also a way in which we break down a problem into one or more subproblems.
- A recursive function always is defined by two parts:
    - **Base case** : compute the result immediately given the inputs to the function call.
    - **Recursive case** or **recursive formula** : compute the result with the help of one or more recursive calls to the same function, but with the inputs somehow reduced in size or complexity, closer to a base case.

isAncestor(F, E) =?

```
FUNCTION isAncestor(x, y):
        IF x is y's parent, THEN:
        return true
        ELSE
        return isAncestor(x, y's mom) OR isAncestor(x, y's dad)
}
```



What is Recursion?



Simple recursive implementation

As an example consider the following function which prints all integer number between 1 and n.

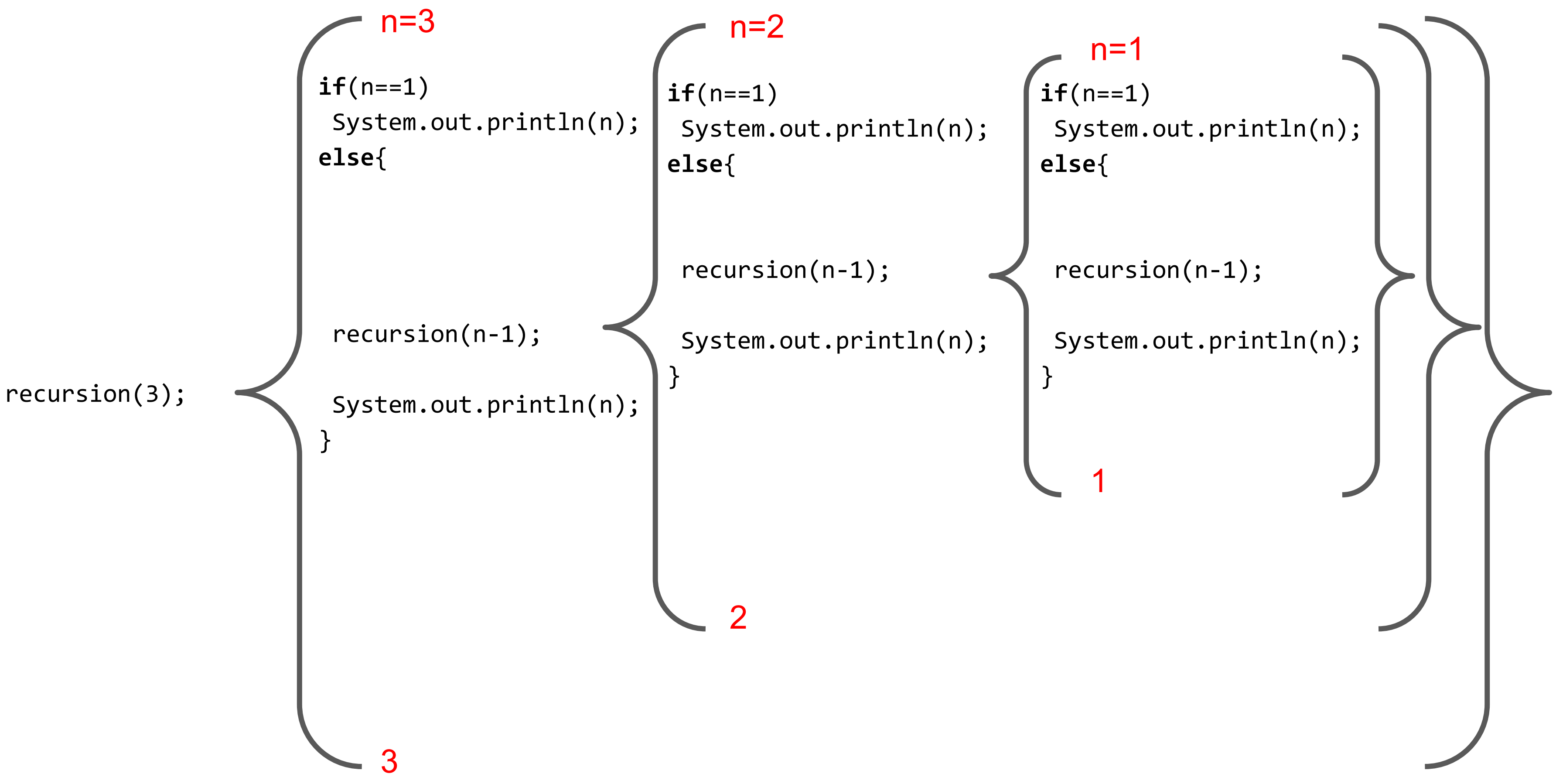
Iterative

```
public static void iterative(int n){
    for(int i=1;i<=n;i++)
        System.out.println(i);
}
```

Recursive

```
public static void recursion(int n){
   if(n==1)
       System.out.println(n);
   else
   {
       recursion(n-1);
       System.out.println(n);
   }
}
```

Printing from 1 to 3 using recursive methods



```
recursion(3);
```

```
if(n==1)
 System.out.println(n);
else{


 recursion(n-1);

 System.out.println(n);
}
```

```
if(n==1)
 System.out.println(n);
else{


 recursion(n-1);

 System.out.println(n);
}
```

```
if(n==1)
 System.out.println(n);
else{


 recursion(n-1);

 System.out.println(n);
}
```

Factorial

- n! is the product of all integers between 1 and n.

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ (n-1)! \times n & \text{if } n > 0 \end{cases}$$

- The problem definition is n!, and the subproblem (n-1)!

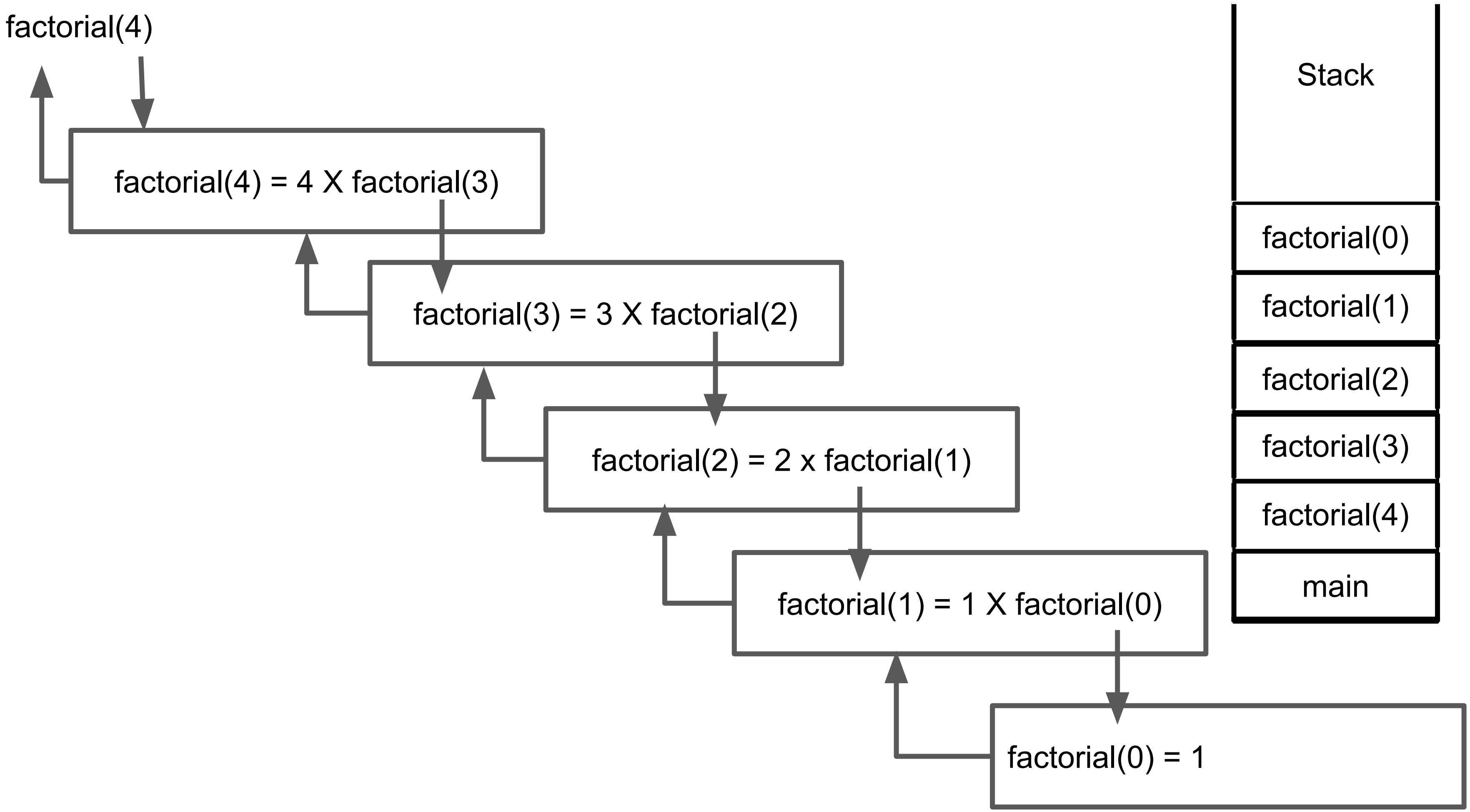
n! = n*(n-1)*(n-2)...3*2*1
5! = 5*4*3*2*1 = 120
3! = 3*2*1 = 6
2! = 2*1 =2
1! = 1
0! = 1

5! = 5*(4*(3*(2*(1*(1)))))

5! = 5*4!
4! = 4* 3!
3! = 3 * 2!
2! = 2 * 1!
1!= 1* 0!
0! = 1

F(5) = (5+F(4+F(3+F(2+F(1+F(0))))))

An example of the implementation of factorial of 4


factorial(4)
factorial(4) = 4 X factorial(3)
factorial(3) = 3 X factorial(2)
factorial(2) = 2 x factorial(1)
factorial(1) = 1 X factorial(0)
factorial(0) = 1
Stack
factorial(0)
factorial(1)
factorial(2)
factorial(3)
factorial(4)
main

Factorial algorithms

Iterative

```
class factorial{
     public static void main(String[] args){
           System.out.println(f(5));
     }

    public static int factorial(int n){
        int f=1;
        for(int i =2; i<=n;i++)
            f*=i;
        return f;
    }
}
```

Recursive

```
class factorial_recursion{
     public static void main(String[] args){
           System.out.println(factorial(5));
     }

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

Sum elements of an array

Fibonacci

0	1	1	2	3	5	8

$$F_n = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ F_{n-1} + F_{n-2} & n > 1 \end{cases}$$

Fi = Fi-1 + Fi-2 i>=2

F0=0

F1=1

Pow(n, a)

2^2 = 2*2

2^3 = 2*2*2

2^4 = 2*2*2*2

Homework

Use iterative and recursion

On tuesday

Recursion vs Iteration

Iterative function

- It terminates when a condition is false.
- Each iteration doesn’t require any extra space.

Recursive function

- It terminates when a base case is reached.
- Each recursive requires extra space on the memory.
- Shorter and easier to formulate complex problems.
-

Tail and non-tail recursion

- A recursive method is tail when there are no pending operations to be performed on return from the recursive call.

- Non-tail recursive method

```
public static void recursion(int n){
   if(n==1)
       System.out.println(n);
   else
   {
       recursion(n-1);
       System.out.println(n);
   }
}
```

- Tail recursive method

```
public static void recursion(int n){
   if(n==1)
       System.out.println(n);
   else
   {
       System.out.println(n);
       recursion(n-1);
   }
}
```