RECURSION

DIVIDE AND CONQUER ALGORITHM DESIGN

Reference: http://introcs.cs.princeton.edu/java/home/

DEFINITION

- **A method that calls itself is recursive.**
- **Example:**

```
public void infinity (){
     infinity();
}
```

- **This method calls itself indefinitely. Not very useful.**

RECURSIVE METHODS

Properties of good recursive methods

- **Identify a base case:**
  - When should the recursion stop?
  - The base case is a condition that makes the method return without calling itself.
- **Method arguments**
  - Recursive methods must have at least one argument as a means of communication.
- **Recursive calls must lead to the base case.**
  - Each call to the same function must modify the argument such that reaching the base case is guaranteed.

RECURSIVE ALGORITHMS

How to write a recursive algorithm:

- **Break down the original problem into smaller parts**
  - The parts should be of the same type as the original problem.
- **Identify the cases where the problem can be solved directly:**
  - When it can't get any simpler
- **Combine the solutions of the smaller problems.**
  - This should be the solution to the original problem

EXAMPLE 1

Factorials :

N! = N × (N-1) × (N-2) × ... × 2 × 1

Argument : N

Base case: 1! = 1

Recursion: N! = N * (N-1)!

```
public int factorial(int N) {
   if (N == 1) return 1;
   return (N * factorial(N-1));
}
```

EXAMPLE 1

4 !

Factorial(4);

```
public int factorial(int N) {
    if (N == 1) return 1;
    return (N * factorial(N-1));
}
```

EXAMPLE 1

4 !

Factorial(4);

Return (4 * factorial (3));

```
public int factorial(int N) {
   if (N == 1) return 1;
   return (N * factorial(N-1));
}
```

EXAMPLE 1

4 !

Factorial(4);

Return (4 * factorial (3));

return (3 * factorial (2));

```
public int factorial(int N) {
   if (N == 1) return 1;
   return (N * factorial(N-1));
}
```

EXAMPLE 1

4 !

Factorial(4);

Return (4 * factorial (3));

return (3 * factorial (2));

return (2 * factorial (1));

```
public int factorial(int N) {
   if (N == 1) return 1;
   return (N * factorial(N-1));
}
```

EXAMPLE 1

4 !

Factorial(4);

Return (4 * factorial (3));

return (3 * factorial (2));

return (2 * factorial (1));

return 1;

```
public int factorial(int N) {
   if (N == 1) return 1;
   return (N * factorial(N-1));
}
```

EXAMPLE 1

4 !

Factorial(4);

1

Return (4 * factorial (3));

return (3 * factorial (2));

return (2 * factorial (1));

return

1;

```
public int factorial(int N) {
   if (N == 1) return 1;
   return (N * factorial(N-1));
}
```

EXAMPLE 1

4 !

Factorial(4);

2*1

1

Return (4 * factorial (3));

return (3 * factorial (2));

return (2 * factorial (1));

return 1;

```
public int factorial(int N) {
   if (N == 1) return 1;
   return (N * factorial(N-1));
}
```

EXAMPLE 1

4 !

Factorial(4);

3*2

2*1

1

Return (4 * factorial (3));

return (3 * factorial (2));

return (2 * factorial (1));

return 1;

```
public int factorial(int N) {
   if (N == 1) return 1;
   return (N * factorial(N-1));
}
```

EXAMPLE 1

24

4 !

3*2

Factorial(4);

2*1

1

Return (4 * factorial (3));

return (3 * factorial (2));

return (2 * factorial (1));

return

1;

```
public int factorial(int N) {
   if (N == 1) return 1;
   return (N * factorial(N-1));
}
```

EXAMPLE 2

Calculate a^b

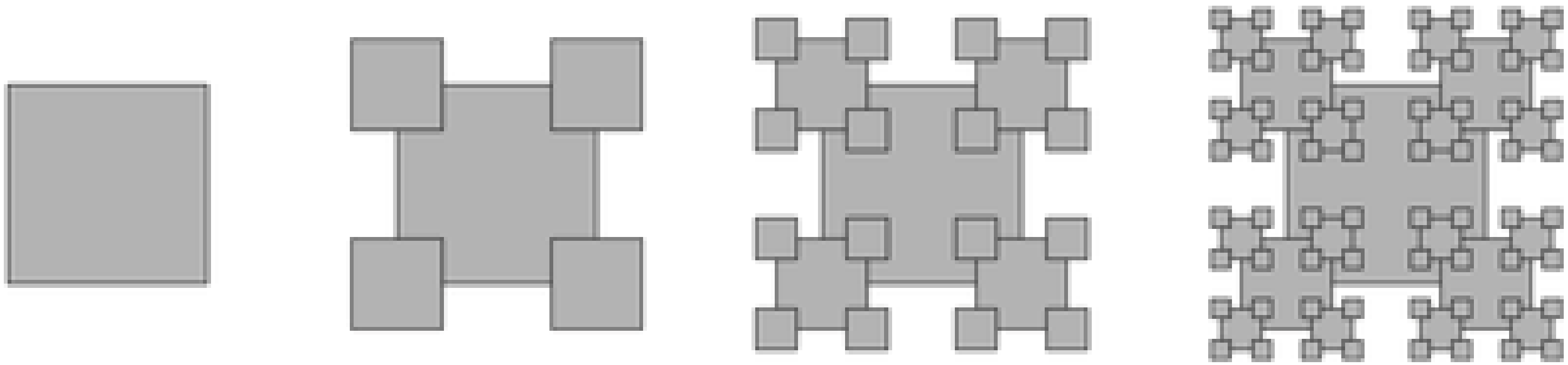
Arguments : a, b

Base case: $a^0 = 1$

Recursion: $a^b = a * a^{b-1}$

```
public int power(int a, int b) {
   if (b == 0) return 1;
   return (a * power(a, b-1));
}
```

RECURSIVE GRAPHICS



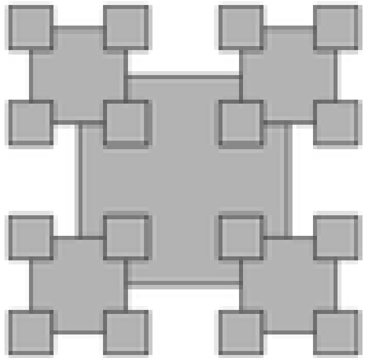
- Recursion can be used to produce intricate graphics with simple shapes
- A simple example of a fractal; a geometric shape whose parts are a smaller copy of the original shape.

RECURSIVE GRAPHICS

```
public void drawRecursiveSquare(int n, double x, double y, double size) {
    if (n == 0) return;
    drawSquare(x, y, size);

    // compute x- and y-coordinates of the 4 half-size squares
    double x0 = x - size/2;
    double x1 = x + size/2;
    double y0 = y - size/2;
    double y1 = y + size/2;

    // recursively draw 4 half-size recursive squares of order n-1
    drawRecursiveSquare(n-1, x0, y0, size/2);    // lower left square
    drawRecursiveSquare(n-1, x0, y1, size/2);    // upper left square
    drawRecursiveSquare(n-1, x1, y0, size/2);    // lower right square
    drawRecursiveSquare(n-1, x1, y1, size/2);    // upper right square
}
```



EXERCISE (3)