Algoritmos e Estrutura de Dados II

Aula 09

Recursão

Prof. Dr. Dilermando Piva Jr

2º Semestre - CDN

Recursividade...

Uma função que chama a si mesma...

Você já sonhou que estava sonhando?

Recursividade...

- Propriedade de uma função chamar a si mesma.
- Necessariamente, toda função recursiva deve ter uma ou mais condições de parada.

```
Exemplo clássico:
   - Cálculo FATORIAL

Fatorial de 5
5 * 4 * 3 * 2 * 1 = 120

Fatorial de 0 = 1

Fatorial de 1 = 1
```

Recursividade

- Cálculo Fatorial

```
def fatorial(n):
    if n > 1:
        return n * fatorial(n-1)
    else:
        return 1
```

Recursividade

- Cálculo Fatorial

N = 5

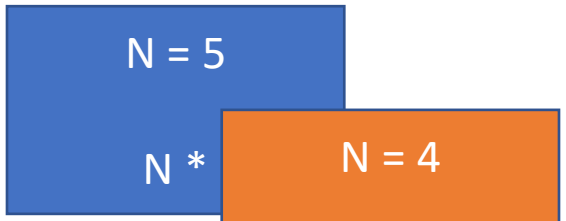
N *

```
def fatorial(n):
    if n > 1:
        return n * fatorial(n-1)
    else:
        return 1
```

CONDIÇÃO DE PARADA !!

Recursividade

- Cálculo Fatorial



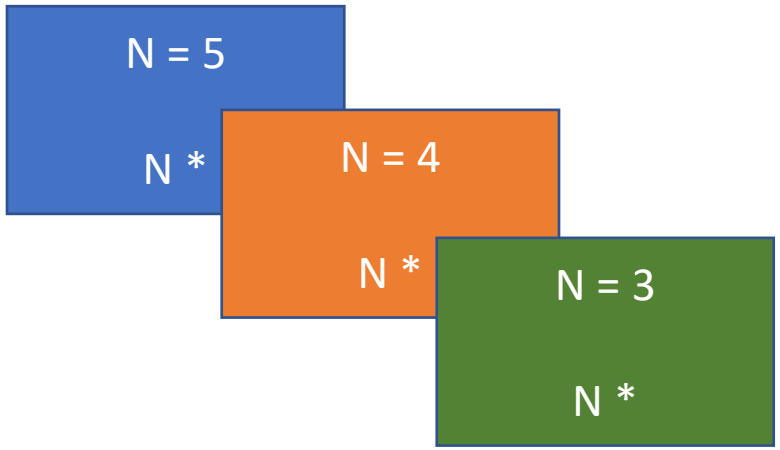

```
def fatorial(n):
    if n > 1:
        return n * fatorial(n-1)
    else:
        return 1
```

CONDIÇÃO DE PARADA !!

Recursividade

- Cálculo Fatorial

```
def fatorial(n):
    if n > 1:
        return n * fatorial(n-1)
    else:
        return 1
```

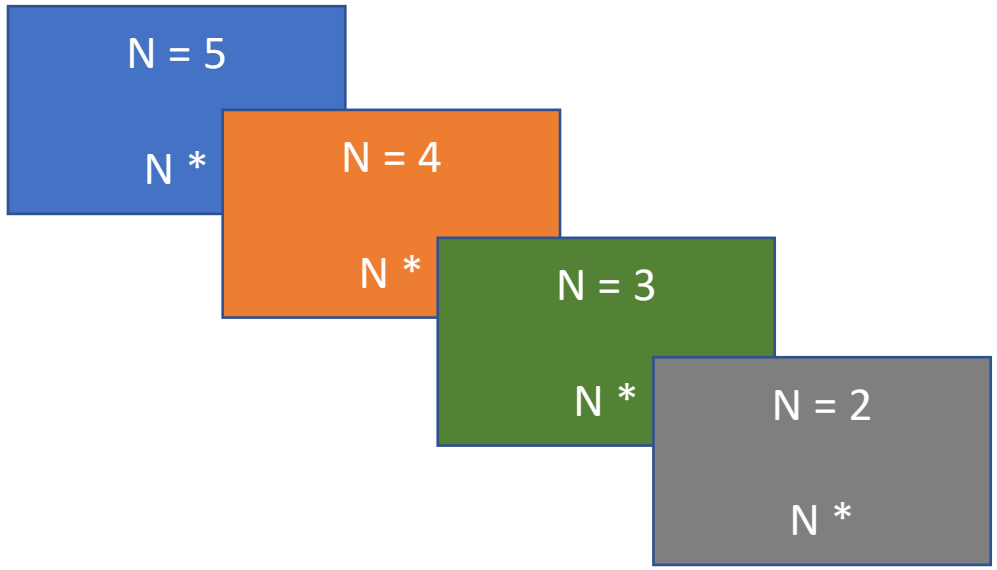


CONDIÇÃO DE PARADA !!

Recursividade

- Cálculo Fatorial

```
def fatorial(n):
    if n > 1:
        return n * fatorial(n-1)
    else:
        return 1
```

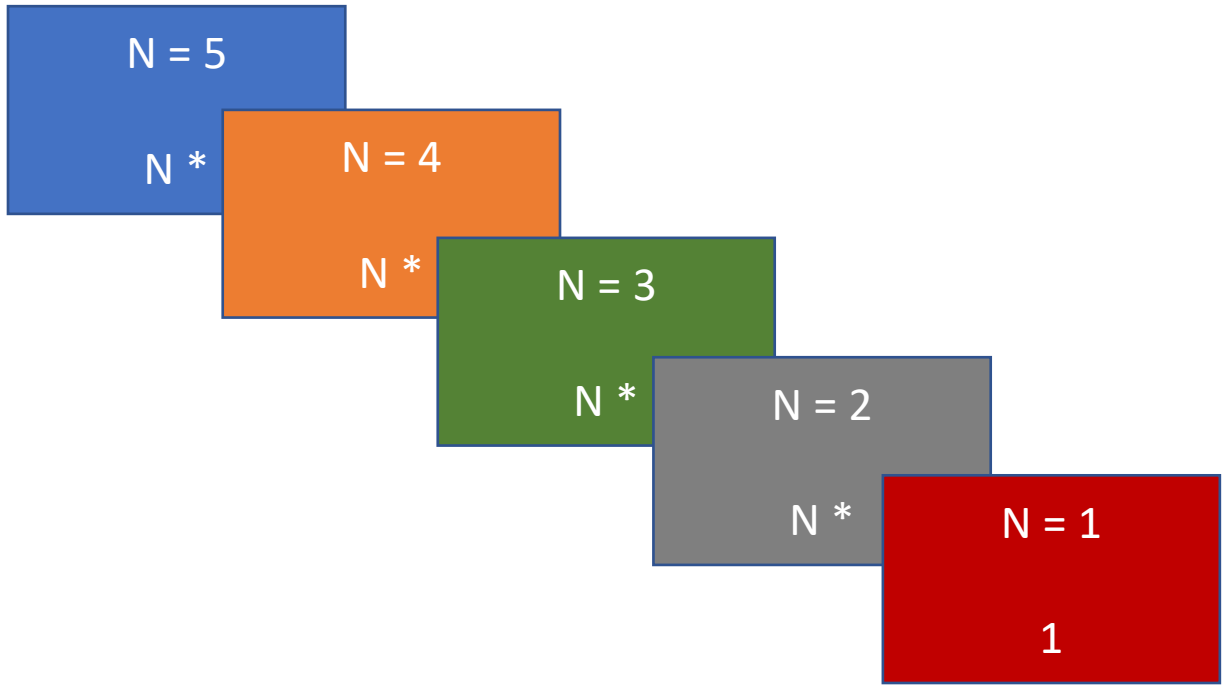


CONDIÇÃO DE PARADA !!

Recursividade

- Cálculo Fatorial

```
def fatorial(n):
    if n > 1:
        return n * fatorial(n-1)
    else:
        return 1
```



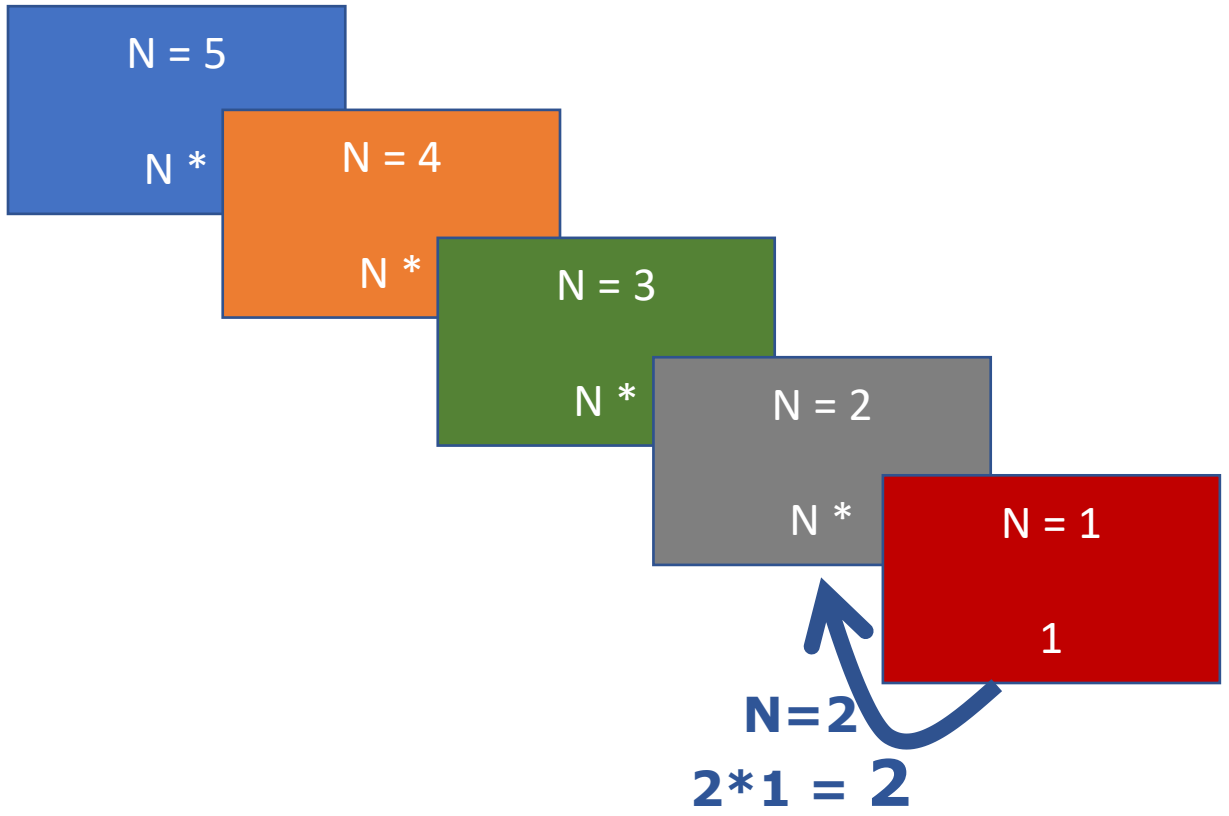
CONDIÇÃO DE PARADA !!

Recursividade

- Cálculo Fatorial

```
def fatorial(n):
    if n > 1:
        return n * fatorial(n-1)
    else:
        return 1
```

CONDIÇÃO DE PARADA !!

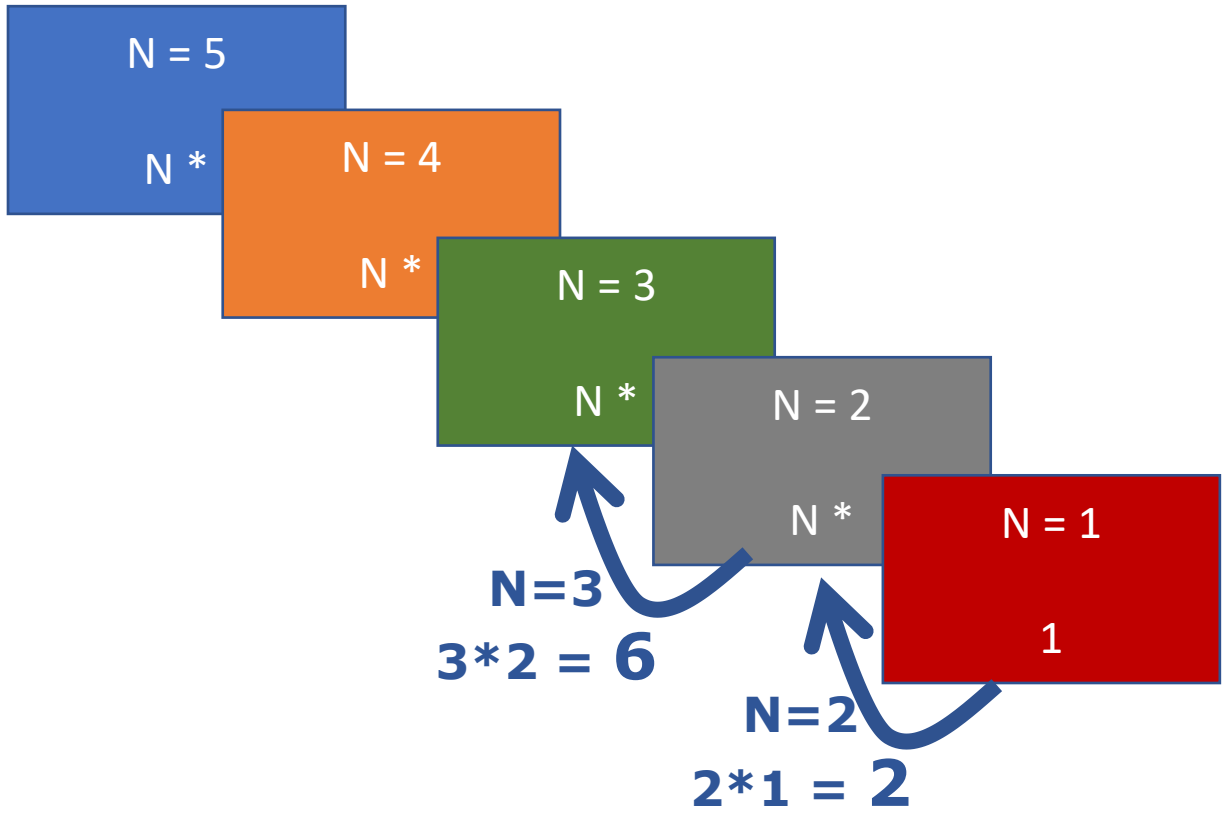


Recursividade

- Cálculo Fatorial

```
def fatorial(n):
    if n > 1:
        return n * fatorial(n-1)
    else:
        return 1
```

CONDIÇÃO DE PARADA !!

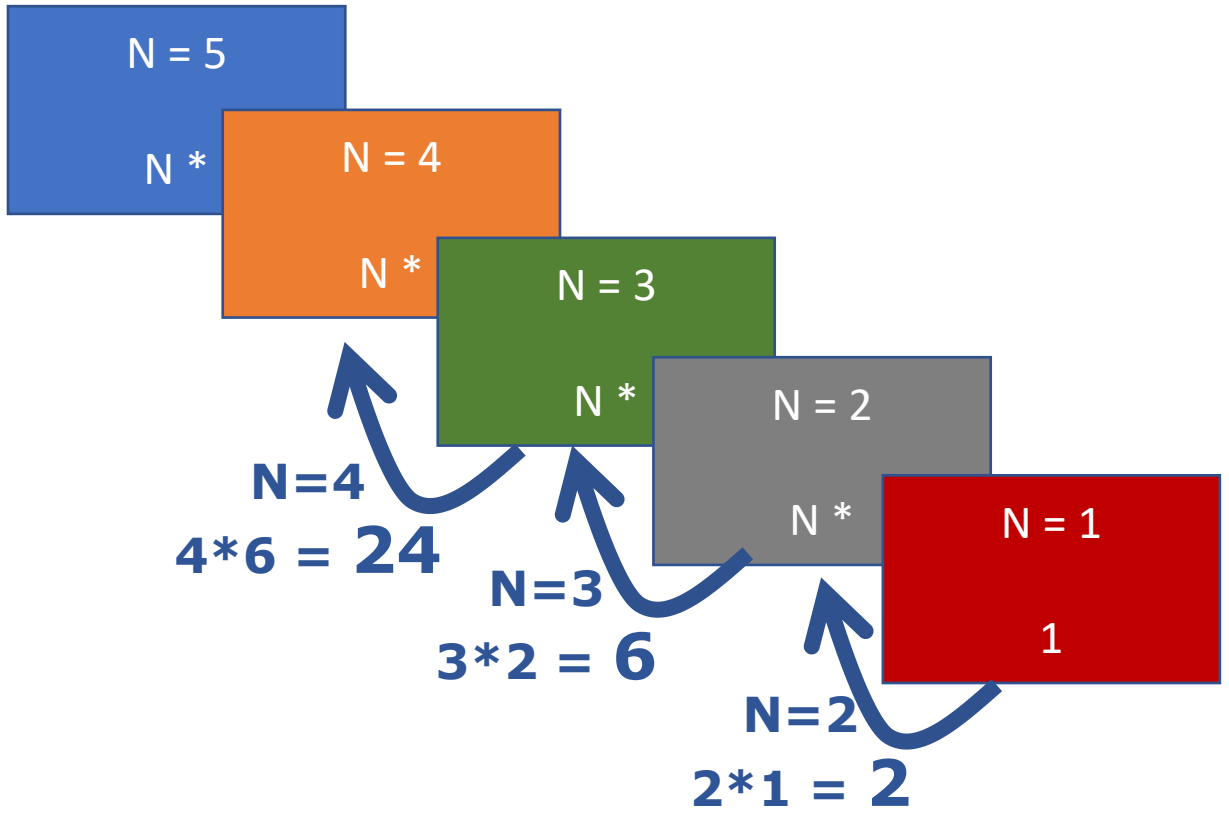


Recursividade

- Cálculo Fatorial

```
def fatorial(n):
    if n > 1:
        return n * fatorial(n-1)
    else:
        return 1
```

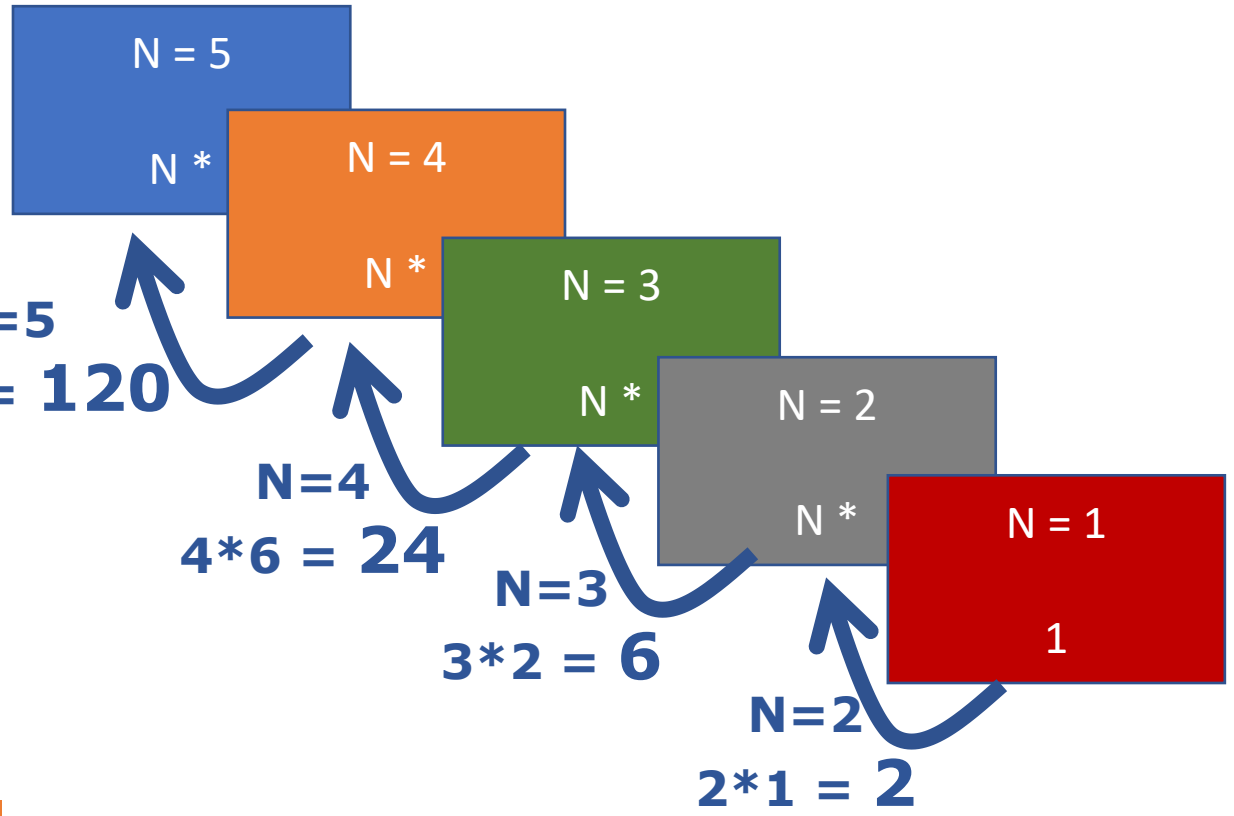
CONDIÇÃO DE PARADA !!



Recursividade

- Cálculo Fatorial

```
def fatorial(n):
    if n > 1:
        return n * fatorial(n-1)
    else:
        return 1
```



CONDIÇÃO DE PARADA !!

Recursividade

- Cálculo Fatorial

```
def fatorial(n):
    if n > 1:
        return n * fatorial(n-1)
    else:
        return 1
```

```
def fatorial(n):
    return (n * fatorial(n-1)) if n > 1 else 1
```

Um algoritmo exponencial: Fibonacci recursivo (1 de 3)

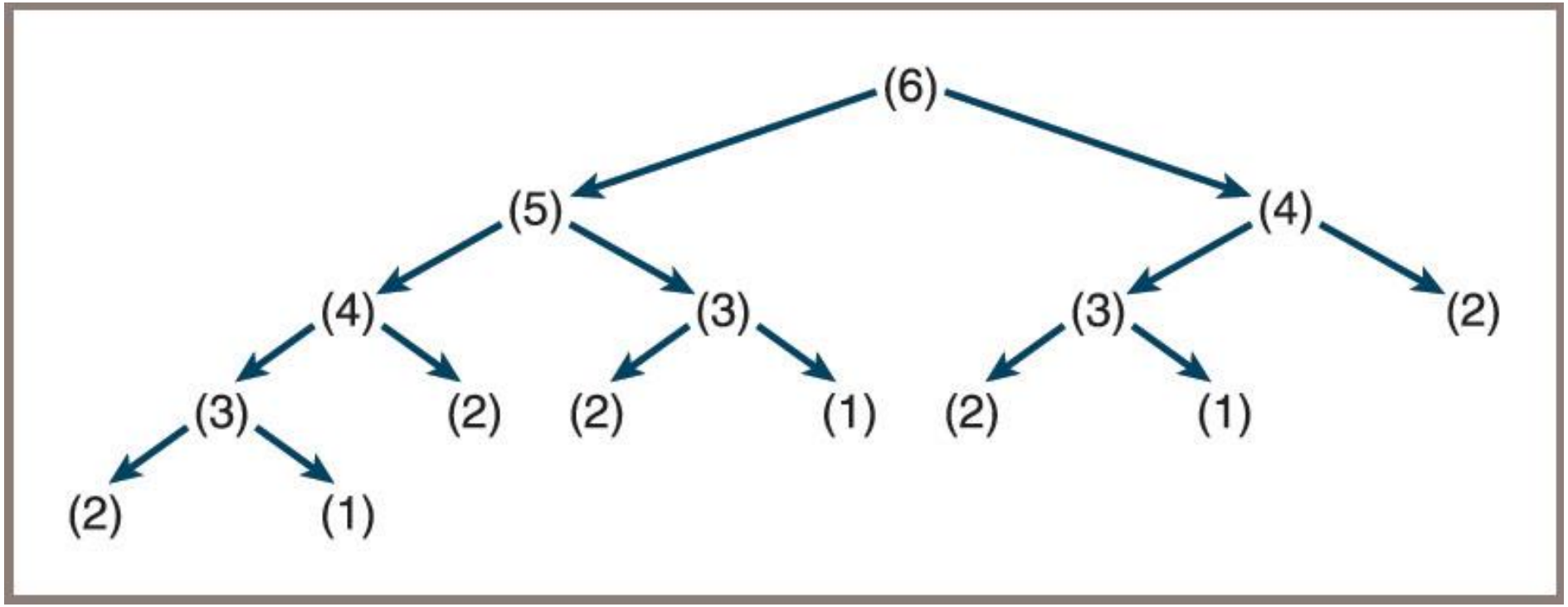
Código para a função Fibonacci:

```
def fib(n):
    """A função Fibonacci recursiva."""
    if n < 3:
        return 1
    else:
        return fib(n - 1) + fib(n - 2)
```

Um algoritmo exponencial: Fibonacci recursivo (2 de 3)

A Figura abaixo mostra as chamadas envolvidas ao se usar a função recursiva para calcular o sexto número de Fibonacci.

Uma árvore de chamadas para fib(6)



Um algoritmo exponencial: Fibonacci recursivo (3 de 3)

- Algoritmos exponenciais geralmente são impraticáveis de serem executados com qualquer problema, exceto de tamanhos muito pequenos.
- Embora o design do Fibonacci recursivo seja elegante,
  - Há uma versão menos bela, mas muito mais rápida, que usa um laço para executar em tempo linear. (EM SEGUIDA)
- Como alternativa, funções recursivas que são chamadas repetidamente com os mesmos argumentos, como a função Fibonacci, podem se tornar mais eficientes por meio de uma técnica chamada **memorização***:*
  - O programa mantém uma tabela dos valores para cada argumento usado com a função.
  - Antes de a função calcular recursivamente um valor para determinado argumento, ela verifica na tabela se o argumento já tem um valor
  - Nesse caso, o valor é simplesmente retornado.
  - Do contrário, o cálculo continua e o argumento e o valor são adicionados à tabela posteriormente.

Um algoritmo exponencial: Fibonacci recursivo (3 de 3)

- Algoritmos exponenciais geralmente são impraticáveis de serem executados com qualquer problema, exceto de tamanhos muito pequenos.
- Embora o design do Fibonacci recursivo seja elegante,
  - Há uma versão menos bela, mas muito mais rápida, que usa um laço para executar em tempo linear. (EM SEGUIDA)
- Como alternativa, funções recursivas que são chamadas repetidamente com os mesmos argumentos, como a função Fibonacci, podem se tornar mais eficientes por meio de uma técnica ch
  - O programa mantém uma tal m a função.
  - Antes de a função calcular re nento, ela verifica na tabela se o argum
  - Nesse caso, o valor é simples
  - Do contrário, o cálculo contir abela posteriormente.

Complexidade:
O(n!)

Convertendo Fibonacci em um algoritmo linear (1 de 2)

- o novo algoritmo inicia um laço se *n* é pelo menos o terceiro número de Fibonacci:
  - Esse número será pelo menos a soma dos dois primeiros (1 + 1 = 2).
- O laço calcula essa soma e, em seguida, realiza duas substituições:
  - O primeiro número se torna o segundo e o segundo se torna a soma recém-calculada.
- O laço conta de 3 a *n.*
- A soma no final do laço é o *n*-ésimo número de Fibonacci.

Convertendo Fibonacci em um algoritmo linear (2 de 2)

Pseudocódigo para o algoritmo:

```
Configura sum como 1
Configura first como 1
Configura second como 1
Configura count como 3
Enquanto count <= N
      Configura  sum como first + second
      Configura first como second
      Configura second como sum
      Incrementa count
```

```
def fibonacci(N):
    if N <= 0:
        return "O valor de N deve ser maior que 0"
    elif N == 1:
        return 1
    elif N == 2:
        return 1

    sum, first, second = 1, 1, 1
    count = 3

    while count <= N:
        sum = first + second
        first = second
        second = sum
        count += 1

    return sum
```

```
# Exemplo de uso:
N = 10  # substitua pelo valor desejado
resultado = fibonacci(N)
print(f"O {N}-ésimo número da sequência de Fibonacci é: {resultado}")
```

Convertendo Fibonacci em um algoritmo linear (2 de 2)

Usando o algoritmo, agora linear, para mostrar os n números de fibonacci

```
# Exemplo de uso:
# n = a qtd de números da sequencia
n = int(input("digite o valor de n: "))

for i in range(1, n+1):
    print(fibonacci(i))
```

```
def fibonacci(N):
    if N <= 0:
        return "O valor de N deve ser maior que 0"
    elif N == 1:
        return 1
    elif N == 2:
        return 1

    sum, first, second = 1, 1, 1
    count = 3

    while count <= N:
        sum = first + second
        first = second
        second = sum
        count += 1

    return sum
```

Convertendo Fibonacci em um algoritmo linear (2 de 2)

Usando o algoritmo, agora linear, para mostrar os n números de fibonacci

```
# Exemplo de uso:
# n = a qtd de números da sequencia
n = int(input("digite o valor de n: "))

for i in range(1, n+1):
    print(fibonacci(i))
```

O(n)

Complexidade:
O(n) * (n)
$O(n^2)$

```
def fibonacci(N):
    if N <= 0:
        return "O valor de N deve ser maior que 0"
    elif N == 1:
        return 1
    elif N == 2:
        return 1

    sum, first, second = 1, 1, 1
    count = 3

    while count <= N:
        sum = first + second
        first = second
        second = sum
        count += 1

    return sum
```

O(n)

VAMOS PARA A PRÁTICA ?!!!