

crie_lista()

```
def crie_lista(n, valor):  
    '''(int,objeto) -> list  
    Recebe um inteiro n e um objeto valor  
    e cria e retorna uma lista de n itens  
    que fazem referência a valor.  
    '''  
    lista = [ ] # lista vazia  
  
    for i in range(0,n,1):  
        lista.append(valor)  
  
    return lista
```

◀ ▶ ↻ 🔍

crie_matriz()

```
def crie_matriz(n, m, valor):  
    '''(int,int,objeto) -> list  
    Recebe um inteiros n e m e um objeto  
    valor e cria e retorna uma matriz  
    de dimensão n x n com itens  
    que fazem referência a valor.  
    '''  
    matriz = [ ] # lista vazia  
    for i in range(0,n,1):  
        linha = crie_lista(m,valor)  
        matriz.append(linha)  
    return matriz
```

◀ ▶ ↻ 🔍

Qual é mais eficiente?

```
meu_prompt> time ./binomialI.py 30 2  
binom(30,2)=435  
real 0m0.023s  
user 0m0.020s  
sys 0m0.000s
```

```
meu_prompt> time ./binomialR0.py 30 2  
binom(30,2)=435  
real 0m0.024s  
user 0m0.020s  
sys 0m0.000s
```

◀ ▶ ↻ 🔍

Qual é mais eficiente?

```
meu_prompt> time ./binomialI.py 30 20  
binom(30,20)=30045015  
real 0m0.023s  
user 0m0.020s  
sys 0m0.000s
```

```
meu_prompt> time ./binomialR0.py 30 20  
binom(30,20)=30045015  
real 6m44.702s  
user 6m44.436s  
sys 0m0.092s
```

◀ ▶ ↻ 🔍

Resolve subproblemas muitas vezes

```
binomialR0(3,2)  
binomialR0(2,2)  
binomialR0(1,2)  
binomialR0(0,2)  
binomialR0(0,1)  
binomialR0(1,1)  
binomialR0(0,1)  
binomialR0(0,0)  
binomialR0(2,1)  
binomialR0(1,1)  
binomialR0(0,1)  
binomialR0(0,0)  
binomialR0(1,0)  
binom(3,2)=3.
```

◀ ▶ ↻ 🔍

Resolve subproblemas muitas vezes

```
binomialR0(5,4) binomialR0(0,2) binomialR0(1,2)  
binomialR0(4,4) binomialR0(1,2) binomialR0(0,2)  
binomialR0(3,4) binomialR0(0,2) binomialR0(0,1)  
binomialR0(2,4) binomialR0(0,1) binomialR0(0,1)  
binomialR0(1,4) binomialR0(2,2) binomialR0(0,1)  
binomialR0(0,4) binomialR0(1,2) binomialR0(0,0)  
binomialR0(0,3) binomialR0(0,2) binomialR0(3,2)  
binomialR0(1,3) binomialR0(0,1) binomialR0(2,2)  
binomialR0(0,3) binomialR0(1,1) binomialR0(1,2)  
binomialR0(0,2) binomialR0(0,1) binomialR0(0,2)  
binomialR0(2,3) binomialR0(0,1) binomialR0(0,1)  
binomialR0(1,3) binomialR0(4,3) binomialR0(1,1)  
binomialR0(0,3) binomialR0(3,3) binomialR0(0,1)  
binomialR0(0,2) binomialR0(2,3) binomialR0(0,0)  
binomialR0(1,2) binomialR0(1,3) binomialR0(2,1)  
binomialR0(0,2) binomialR0(0,3) binomialR0(1,1)  
binomialR0(0,1) binomialR0(0,3) binomialR0(0,1)  
binomialR0(3,3) binomialR0(1,2) binomialR0(0,1)  
binomialR0(2,3) binomialR0(0,2) binomialR0(0,0)  
binomialR0(1,3) binomialR0(0,2) binomialR0(1,0)  
binomialR0(0,3) binomialR0(0,1) binomialR0(0,0)  
binomialR0(2,2) binomialR0(2,2) binom(5,4)=5.
```

◀ ▶ ↻ 🔍

Número de adições

linha	número de adições
1	= 0
2	= 0
3	= $T(n-1, k)$
4	= $T(n-1, k-1) + 1$

$$T(n, k) = T(n-1, k) + T(n-1, k-1) + 1$$

Relação de recorrência

$$T(n, k) = \begin{cases} 0, & n < k, \\ 0, & n = k \text{ ou } k = 0, \\ T(n-1, k) + T(n-1, k-1) + 1, & n, k > 0. \end{cases}$$

Quanto vale $T(n, k)$?

Relação de recorrência!

Número $T(n, k)$ de adições

T	0	1	2	3	4	5	6	7	8	...	k
0	0	0	0	0	0	0	0	0	0	...	
1	0	0	0	0	0	0	0	0	0	...	
2	0	1	0	0	0	0	0	0	0	...	
3	0	2	2	0	0	0	0	0	0	...	
4	0	3	5	3	0	0	0	0	0	...	
5	0	4	9	9	4	0	0	0	0	...	
6	0	5	14	19	14	5	0	0	0	...	
7	0	6	20	34	34	20	6	0	0	...	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
n											

Binomial

	0	1	2	3	4	5	6	7	8	...	k
0	1	0	0	0	0	0	0	0	0	...	
1	1	1	0	0	0	0	0	0	0	...	
2	1	2	1	0	0	0	0	0	0	...	
3	1	3	3	1	0	0	0	0	0	...	
4	1	4	6	4	1	0	0	0	0	...	
5	1	5	10	10	5	1	0	0	0	...	
6	1	6	15	20	15	6	1	0	0	...	
7	1	7	21	35	35	21	7	1	0	...	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
n											

Número de adições

O número $T(n, k)$ de adições feitas pela chamada $\text{binomialR1}(n, k)$ é

$$\binom{n}{k} - 1.$$

O consumo de tempo da função é proporcional ao número de iterações e portanto é "proporcional" a $\binom{n}{k}$.

Quando o valor de k é aproximadamente $n/2$

$$\binom{n}{k} \geq 2^{\frac{n}{2}}$$

e o consumo de tempo é dito "exponencial".

Conclusões

Devemos **evitar** resolver o mesmo subproblema várias vezes.

O número de chamadas recursivas feitas por $\text{binomialR1}(n, k)$ é

$$2 \times \binom{n}{k} - 2.$$

Binomial mais eficiente ainda ...

Supondo $n \geq k \geq 1$ temos que

$$\begin{aligned} \binom{n}{k} &= \frac{n!}{(n-k)!k!} \\ &= \frac{(n-1)!}{(n-k)!(k-1)!} \times \frac{n}{k} \\ &= \frac{(n-1)!}{((n-1)-(k-1))!(k-1)!} \times \frac{n}{k} \\ &= \binom{n-1}{k-1} \times \frac{n}{k}. \end{aligned}$$

◀ ▶ ↻ 🔍

`binomialR2(20,10)`

```
binomialR2(20,10)
  binomialR2(19,9)
    binomialR2(18,8)
      binomialR2(17,7)
        binomialR2(16,6)
          binomialR2(15,5)
            binomialR2(14,4)
              binomialR2(13,3)
                binomialR2(12,2)
                  binomialR2(11,1)
binom(20,10)=184756.
```

◀ ▶ ↻ 🔍

E agora, qual é mais eficiente?

```
meu_prompt> time ./binomialI.py 30 20
binom(30,20)=30045015
real                0m0.024s
user                0m0.020s
sys                 0m0.000s
```

```
meu_prompt> time ./binomialR2.py 30 20
binom(30,20)=30045015
real                0m0.022s
user                0m0.020s
sys                 0m0.000s
```

◀ ▶ ↻ 🔍

Binomial mais eficiente ainda ...

Logo, supondo $n \geq k \geq 1$, podemos escrever

$$\binom{n}{k} = \begin{cases} n, & \text{quando } k = 1, \\ \binom{n-1}{k-1} \times \frac{n}{k}, & \text{quando } k > 1. \end{cases}$$

```
def binomialR2(n, k):
    if k == 1: return n
    return binomialR2(n-1, k-1) * n // k
```

A função `binomialR2` faz *recursão de cauda* (*Tail recursion*).

◀ ▶ ↻ 🔍

E agora, qual é mais eficiente?

```
meu_prompt> time ./binomialI.py 30 2
binom(30,2)=435
real                0m0.026s
user                0m0.020s
sys                 0m0.004s
```

```
meu_prompt> time ./binomialR2.py 30 2
binom(30,2)=435
real                0m0.022s
user                0m0.020s
sys                 0m0.000s
```

◀ ▶ ↻ 🔍

Conclusão

O número de chamadas recursivas feitas por `binomialR2(n,k)` é $k - 1$.

◀ ▶ ↻ 🔍