Recursividade

VISA

PROUD SPONSOR

Recursividade

VISA

PROUD SPONSOR

Recursividade

VISA

PROUD SPONSOR

VISA

VISA

VISA

International

PROUD SPONSOR

VISA

International

Service

Association

International

Service

Association

VISA

International

Service

Association

International

Service

Association

VISA

International

Service

International

Service

Association

VISA

International

Service

Association

International

Service

Association

ZERO
DE UTILIDADE
imgflip.com

caso **recursivo**

caso **base**

caso **recursivo**

(sub-caso ou caso anterior)

---

caso **base**

caso **recursivo**

(sub-caso ou caso anterior)

---

caso **base**

(caso atômico ou inicial)

caso **recursivo**

(o que fazer a cada nível da recursão)

---

caso **base**

caso **recursivo**

(o que fazer a cada nível da recursão)

---

caso **base**

(o que fazer ao atingir o caso base)

cálculo **fatorial**

cálculo **fatorial**

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

cálculo fatorial

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

cálculo fatorial

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

cálculo **fatorial**

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

cálculo fatorial

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

cálculo fatorial

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

cálculo fatorial

```
int fat_iter(int n)
{
   int fat = 1;
   while (n > 1) { fat *= n; n—; }
   return fat;
}
```

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

cálculo fatorial

cálculo fatorial

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

cálculo fatorial

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

cálculo fatorial

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

- 5! = 5 * 4!

cálculo fatorial

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

- 5! = 5 * 4!

- 4! = 4 * 3 * 2 * 1

cálculo fatorial

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

- 5! = 5 * 4!

- 4! = 4 * 3 * 2 * 1
- 3! = 3 * 2 * 1

cálculo fatorial

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

- 5! = 5 * 4!

- 4! = 4 * 3 * 2 * 1
- 3! = 3 * 2 * 1

- 4! = 4 * 3!

cálculo fatorial

caso recursivo

caso base

cálculo fatorial

caso recursivo

(sub-caso ou caso anterior)

caso base

cálculo fatorial

caso recursivo

$n! = n * (n-1)!$

---

caso base

cálculo **fatorial**

caso **recursivo**

n! = n * (n-1)!

caso **base**

(caso atômico ou inicial)

cálculo **fatorial**

caso **recursivo**

$n! = n * (n-1)!$

---

caso **base**

$0! = 1! = 1$

cálculo fatorial

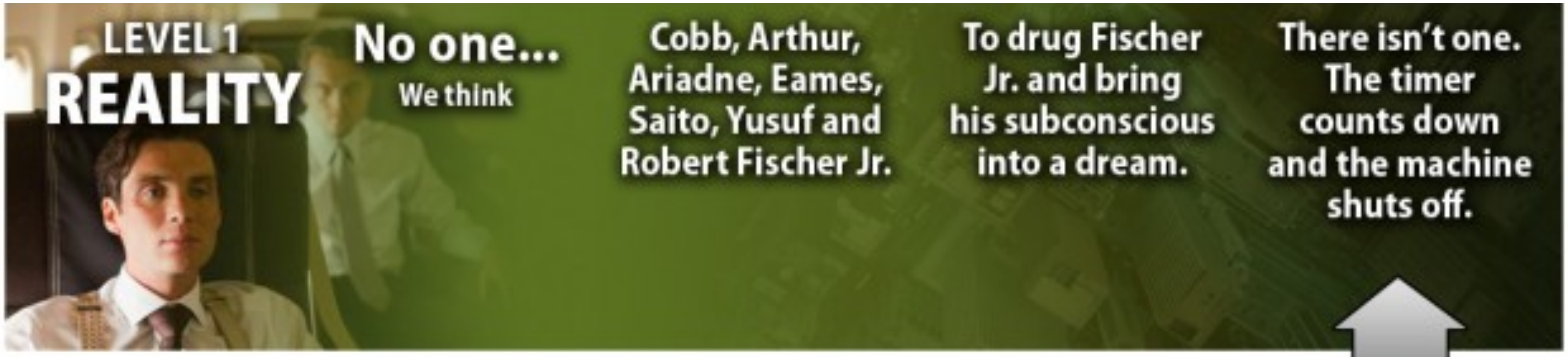
cálculo fatorial

```
int fat_rec(int n)
{
   if (n <= 1) return 1;
   return n * fat_rec(n-1);
}
```

cálculo fatorial

```
int fat_iter(int n)
{
   int fat = 1;
   while (n > 1) { fat *= n; n—; }
   return fat;
}
```

```
int fat_rec(int n)
{
   if (n <= 1) return 1;
   return n * fat_rec(n-1);
}
```



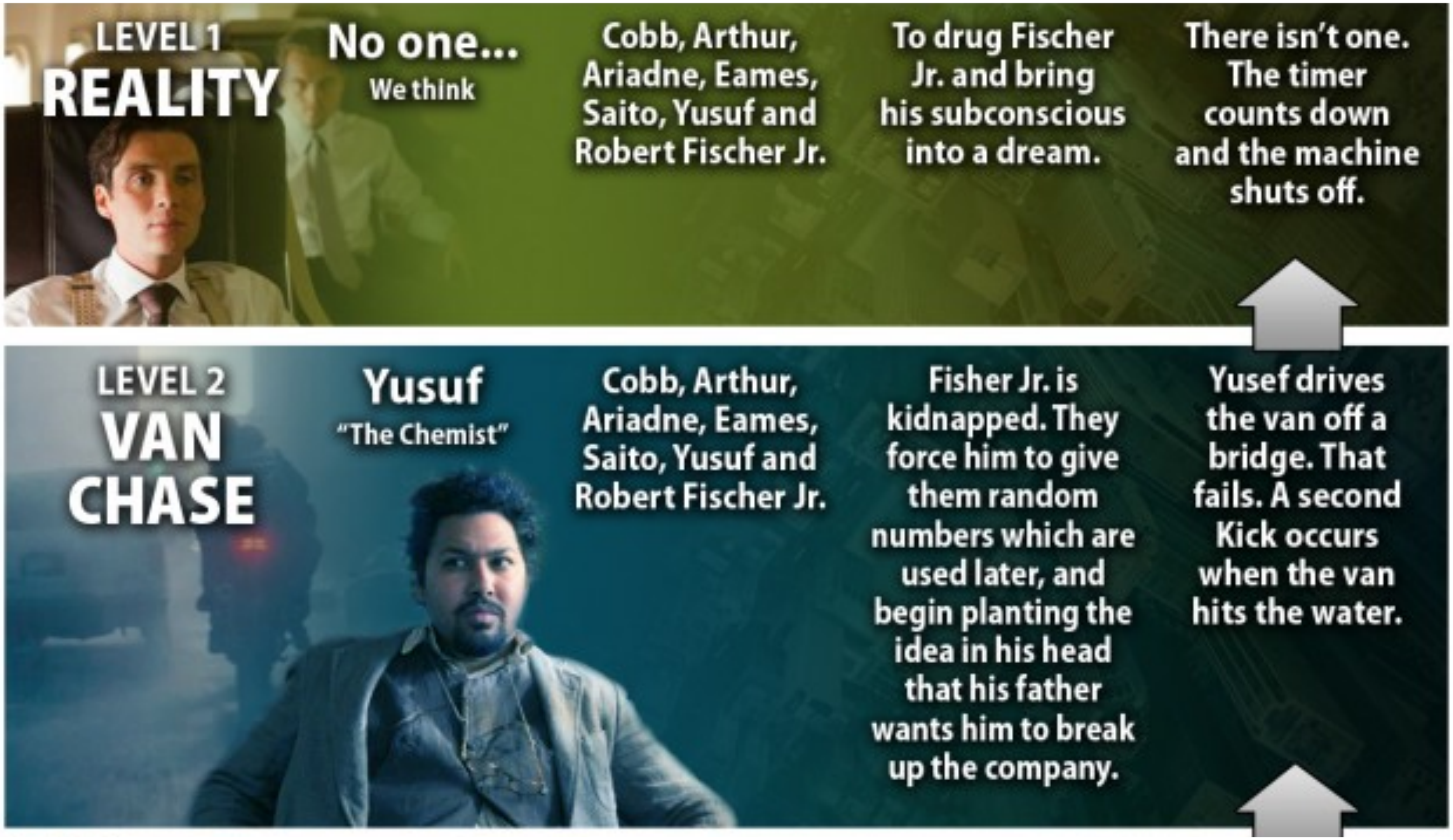
LEVEL 1
REALITY

No one...
We think

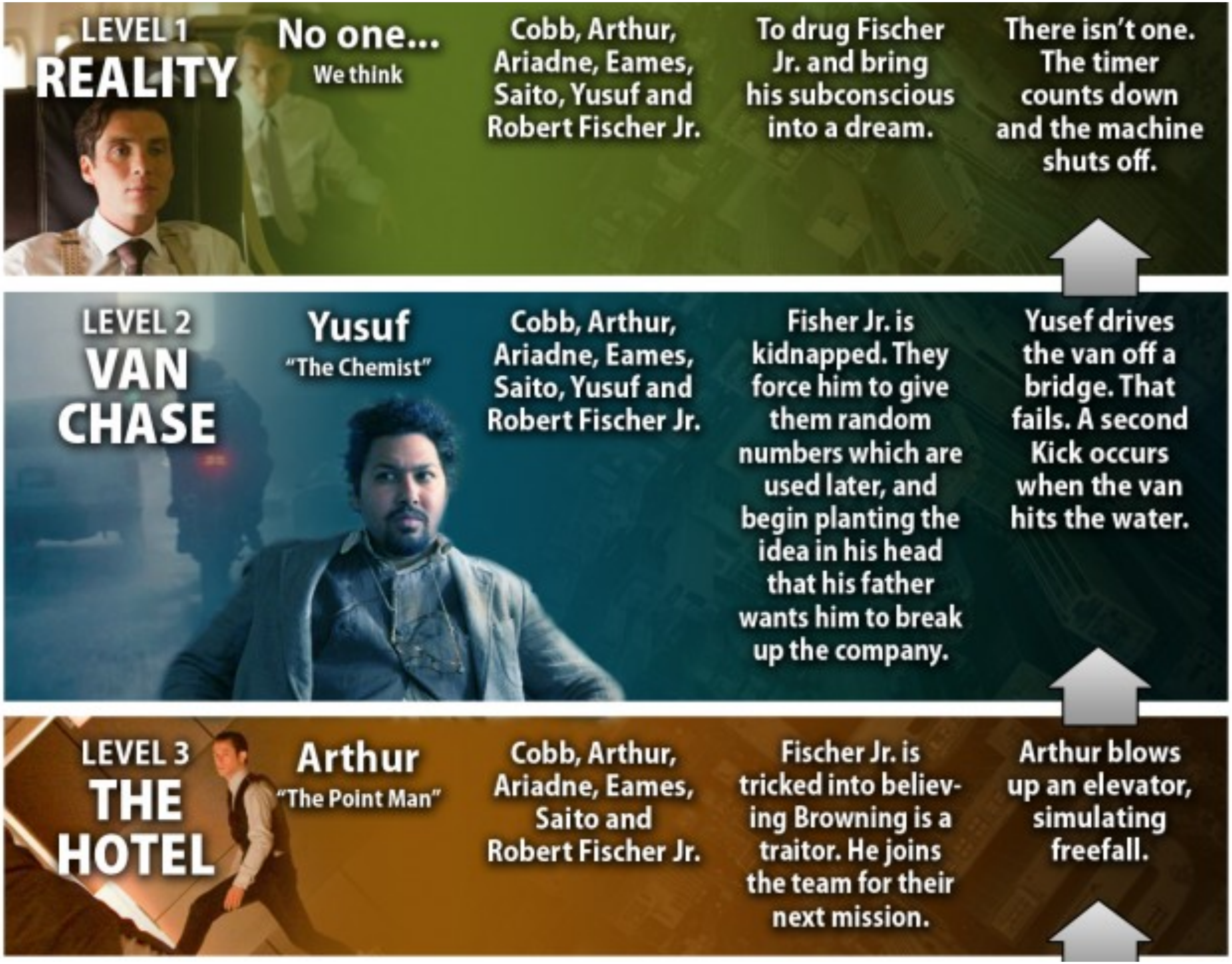
Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.

To drug Fischer Jr. and bring his subconscious into a dream.

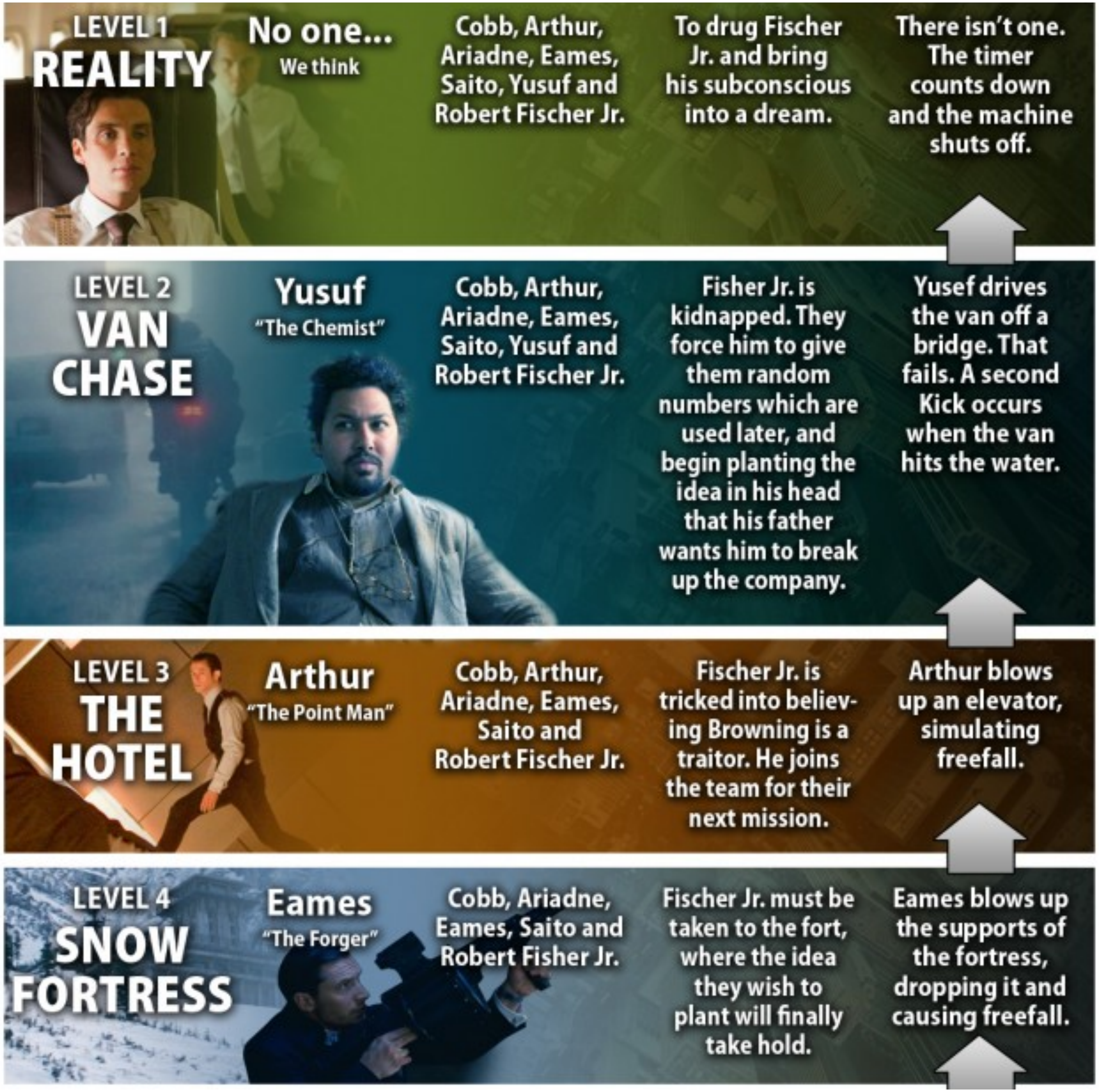
There isn't one. The timer counts down and the machine shuts off.



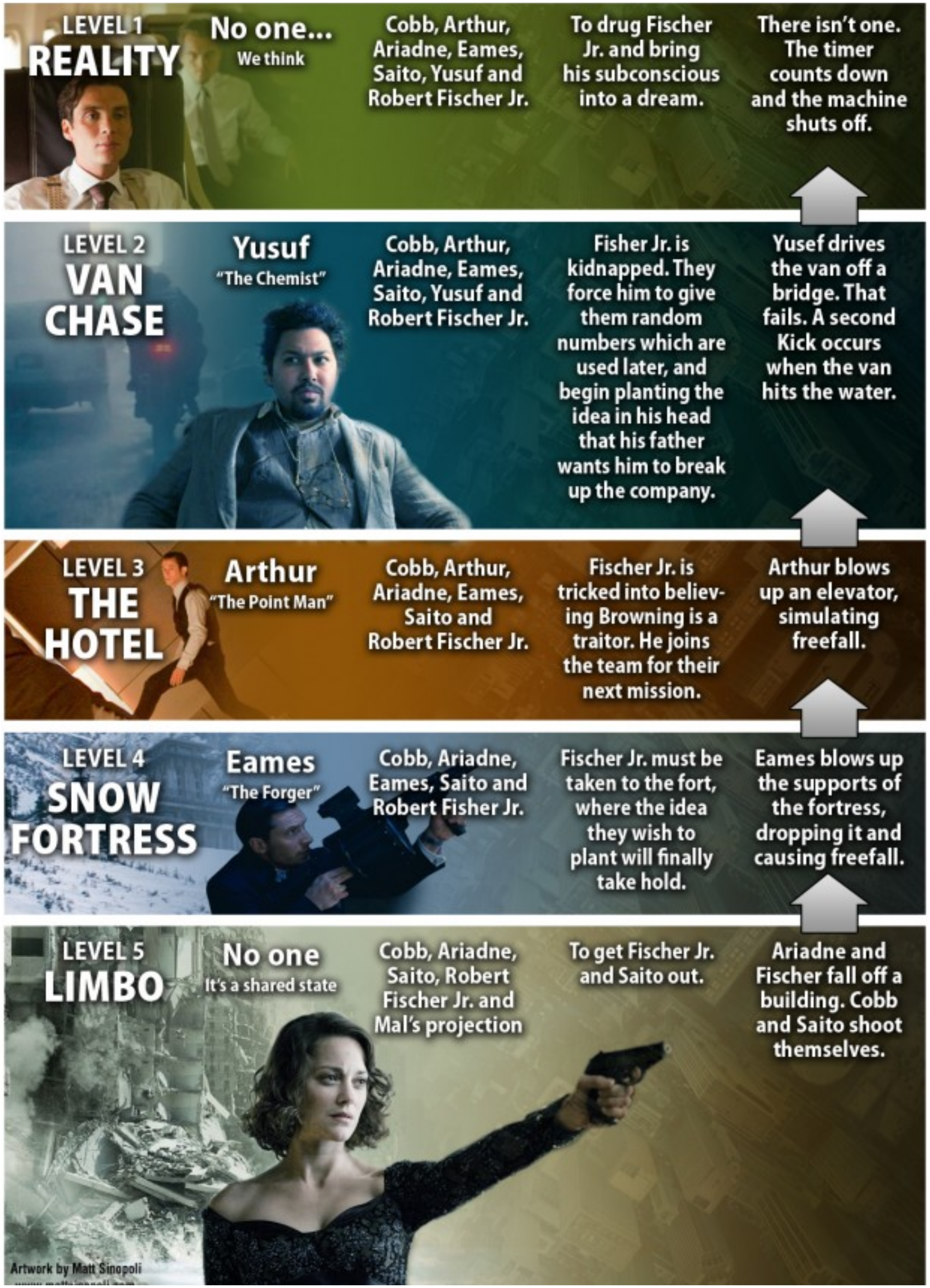
LEVEL 1 REALITY	No one... We think	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	To drug Fischer Jr. and bring his subconscious into a dream.	There isn't one. The timer counts down and the machine shuts off.
LEVEL 2 VAN CHASE	Yusuf "The Chemist"	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	Fisher Jr. is kidnapped. They force him to give them random numbers which are used later, and begin planting the idea in his head that his father wants him to break up the company.	Yusef drives the van off a bridge. That fails. A second Kick occurs when the van hits the water.



LEVEL 1 REALITY	No one... We think	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	To drug Fischer Jr. and bring his subconscious into a dream.	There isn't one. The timer counts down and the machine shuts off.
LEVEL 2 VAN CHASE	Yusuf "The Chemist"	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	Fisher Jr. is kidnapped. They force him to give them random numbers which are used later, and begin planting the idea in his head that his father wants him to break up the company.	Yusef drives the van off a bridge. That fails. A second Kick occurs when the van hits the water.
LEVEL 3 THE HOTEL	Arthur "The Point Man"	Cobb, Arthur, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. is tricked into believing Browning is a traitor. He joins the team for their next mission.	Arthur blows up an elevator, simulating freefall.

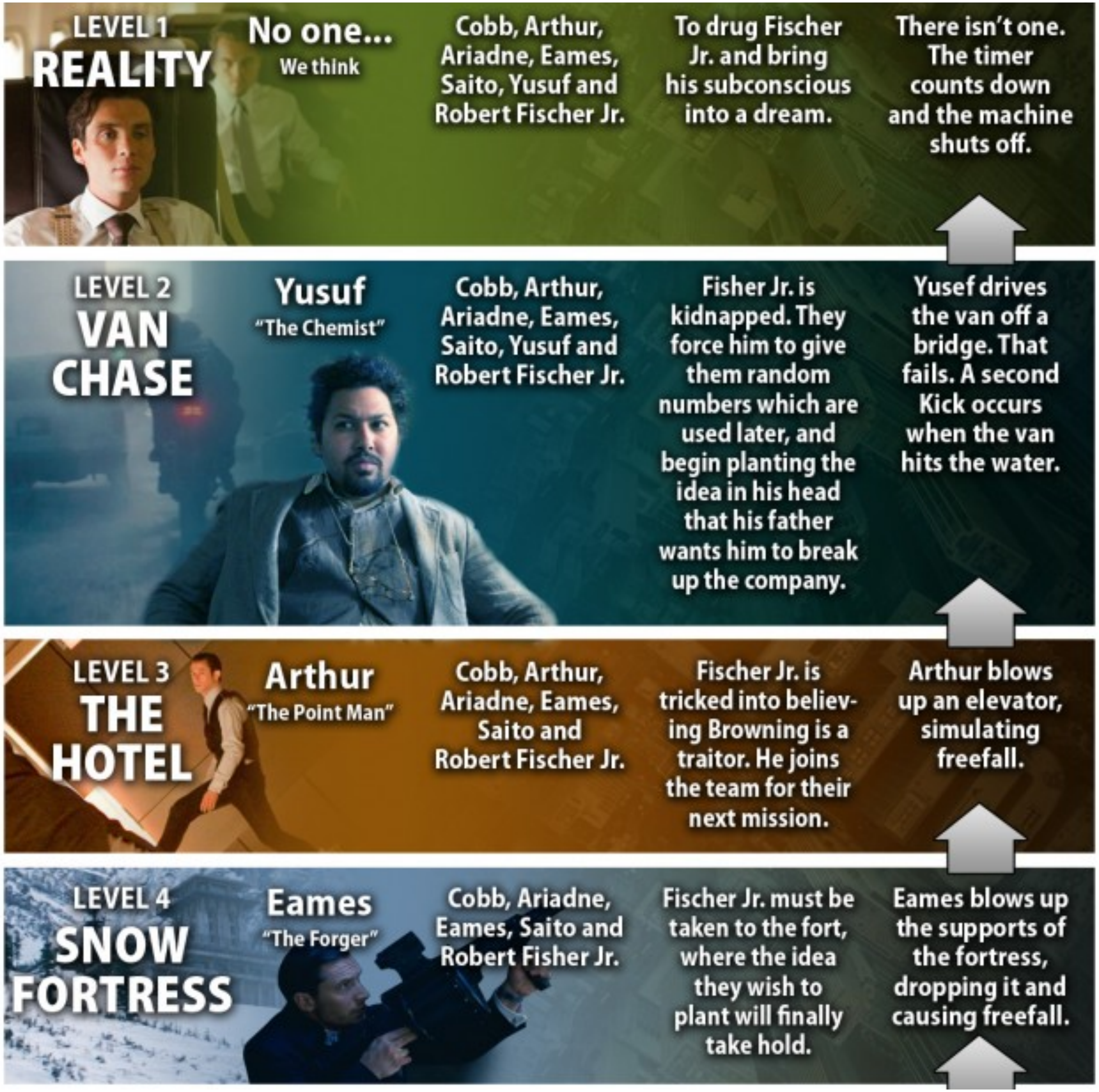


LEVEL 1 REALITY	No one... We think	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	To drug Fischer Jr. and bring his subconscious into a dream.	There isn't one. The timer counts down and the machine shuts off.
LEVEL 2 VAN CHASE	Yusuf "The Chemist"	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	Fisher Jr. is kidnapped. They force him to give them random numbers which are used later, and begin planting the idea in his head that his father wants him to break up the company.	Yusef drives the van off a bridge. That fails. A second Kick occurs when the van hits the water.
LEVEL 3 THE HOTEL	Arthur "The Point Man"	Cobb, Arthur, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. is tricked into believing Browning is a traitor. He joins the team for their next mission.	Arthur blows up an elevator, simulating freefall.
LEVEL 4 SNOW FORTRESS	Eames "The Forger"	Cobb, Ariadne, Eames, Saito and Robert Fisher Jr.	Fischer Jr. must be taken to the fort, where the idea they wish to plant will finally take hold.	Eames blows up the supports of the fortress, dropping it and causing freefall.

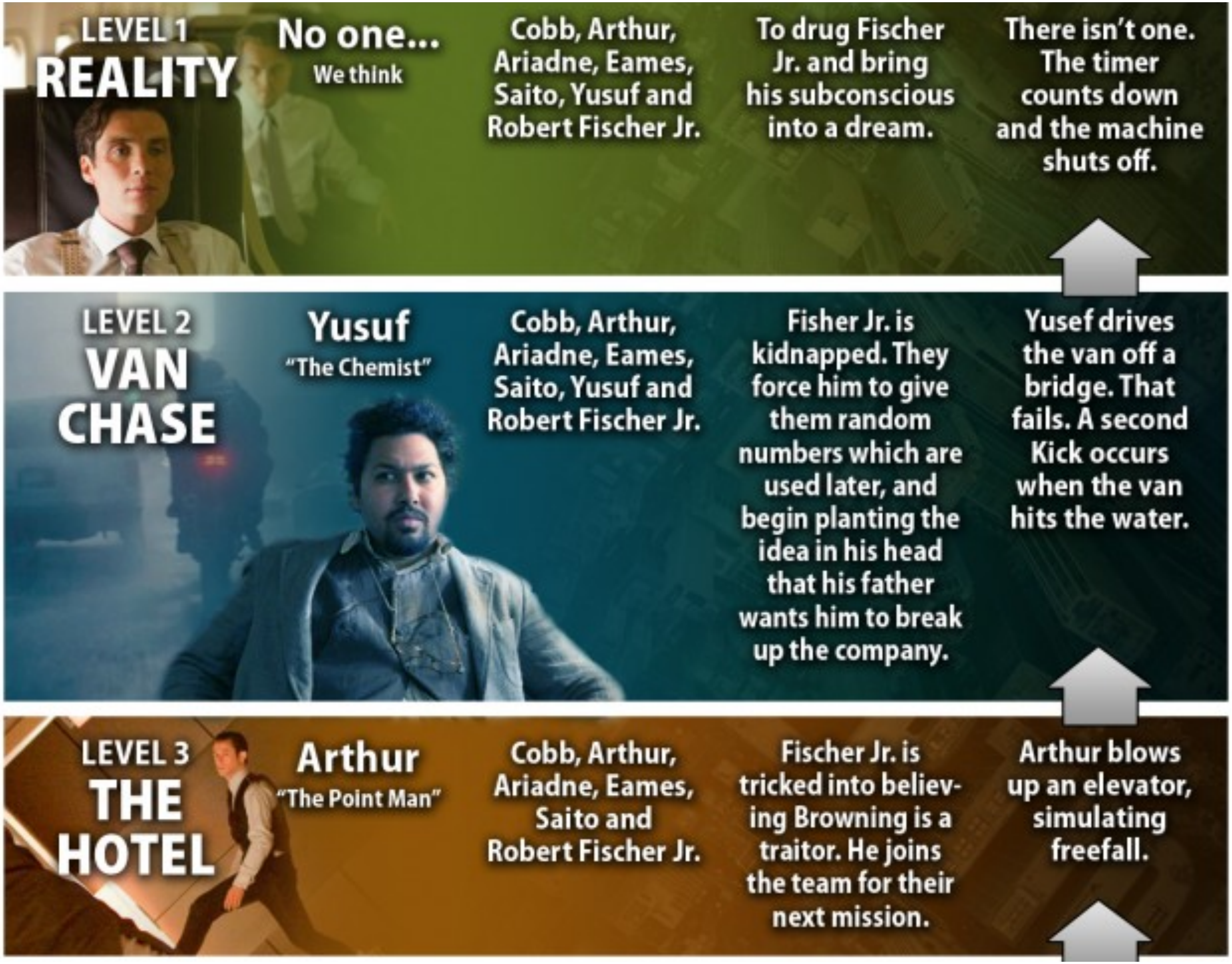


LEVEL 1 REALITY	No one... We think	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	To drug Fischer Jr. and bring his subconscious into a dream.	There isn't one. The timer counts down and the machine shuts off.
LEVEL 2 VAN CHASE	Yusuf "The Chemist"	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	Fisher Jr. is kidnapped. They force him to give them random numbers which are used later, and begin planting the idea in his head that his father wants him to break up the company.	Yusef drives the van off a bridge. That fails. A second Kick occurs when the van hits the water.
LEVEL 3 THE HOTEL	Arthur "The Point Man"	Cobb, Arthur, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. is tricked into believing Browning is a traitor. He joins the team for their next mission.	Arthur blows up an elevator, simulating freefall.
LEVEL 4 SNOW FORTRESS	Eames "The Forger"	Cobb, Ariadne, Eames, Saito and Robert Fisher Jr.	Fischer Jr. must be taken to the fort, where the idea they wish to plant will finally take hold.	Eames blows up the supports of the fortress, dropping it and causing freefall.
LEVEL 5 LIMBO	No one It's a shared state	Cobb, Ariadne, Saito, Robert Fischer Jr. and Mal's projection	To get Fischer Jr. and Saito out.	Ariadne and Fischer fall off a building. Cobb and Saito shoot themselves.

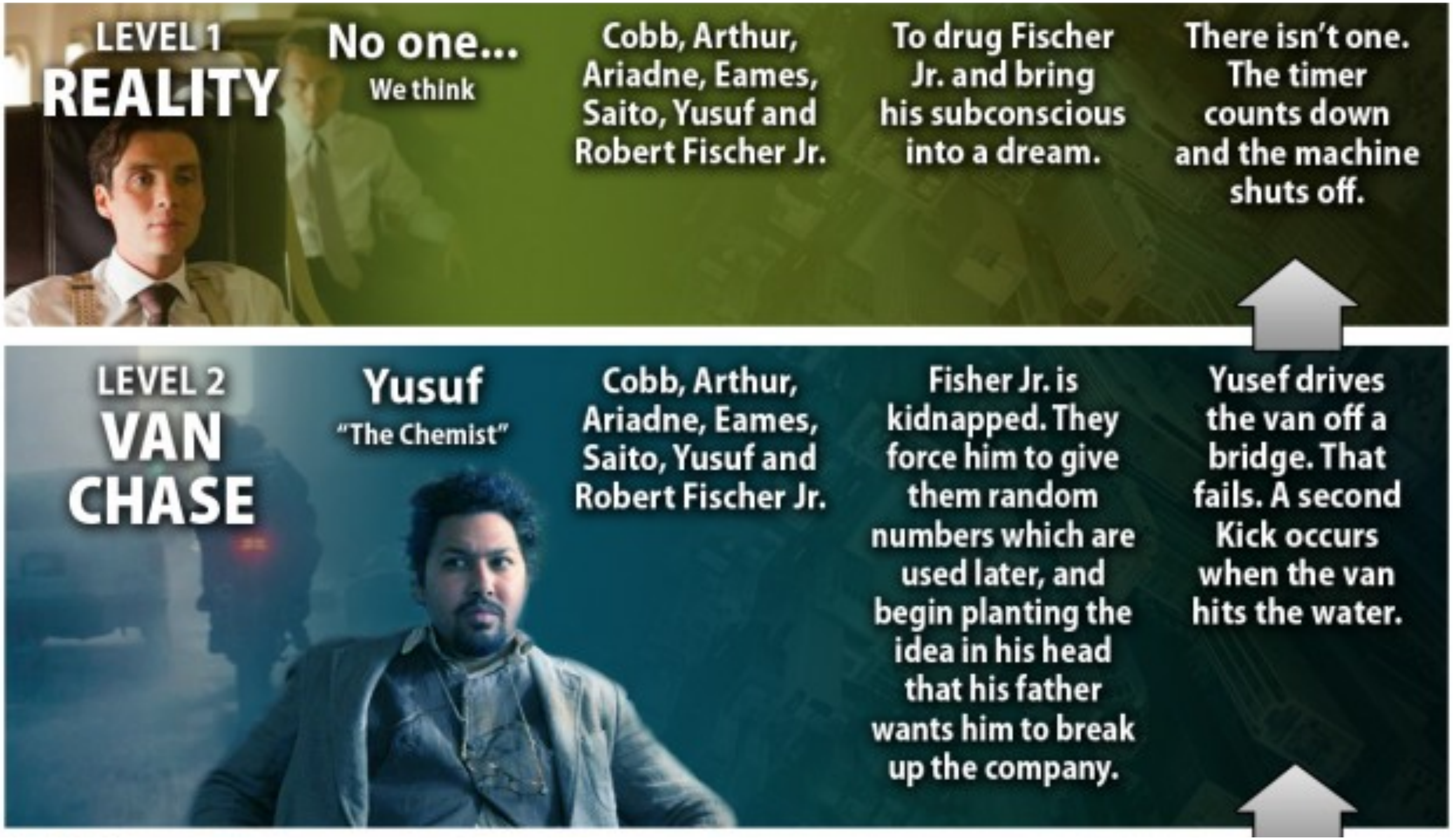
Artwork by Matt Sinopoli
www.mattsinopoli.com



LEVEL 1 REALITY	No one... We think	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	To drug Fischer Jr. and bring his subconscious into a dream.	There isn't one. The timer counts down and the machine shuts off.
LEVEL 2 VAN CHASE	Yusuf "The Chemist"	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	Fisher Jr. is kidnapped. They force him to give them random numbers which are used later, and begin planting the idea in his head that his father wants him to break up the company.	Yusef drives the van off a bridge. That fails. A second Kick occurs when the van hits the water.
LEVEL 3 THE HOTEL	Arthur "The Point Man"	Cobb, Arthur, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. is tricked into believing Browning is a traitor. He joins the team for their next mission.	Arthur blows up an elevator, simulating freefall.
LEVEL 4 SNOW FORTRESS	Eames "The Forger"	Cobb, Ariadne, Eames, Saito and Robert Fisher Jr.	Fischer Jr. must be taken to the fort, where the idea they wish to plant will finally take hold.	Eames blows up the supports of the fortress, dropping it and causing freefall.



LEVEL 1 REALITY	No one... We think	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	To drug Fischer Jr. and bring his subconscious into a dream.	There isn't one. The timer counts down and the machine shuts off.
LEVEL 2 VAN CHASE	Yusuf "The Chemist"	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	Fisher Jr. is kidnapped. They force him to give them random numbers which are used later, and begin planting the idea in his head that his father wants him to break up the company.	Yusef drives the van off a bridge. That fails. A second Kick occurs when the van hits the water.
LEVEL 3 THE HOTEL	Arthur "The Point Man"	Cobb, Arthur, Ariadne, Eames, Saito and Robert Fischer Jr.	Fischer Jr. is tricked into believing Browning is a traitor. He joins the team for their next mission.	Arthur blows up an elevator, simulating freefall.



LEVEL 1 REALITY	No one... We think	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	To drug Fischer Jr. and bring his subconscious into a dream.	There isn't one. The timer counts down and the machine shuts off.
LEVEL 2 VAN CHASE	Yusuf "The Chemist"	Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.	Fisher Jr. is kidnapped. They force him to give them random numbers which are used later, and begin planting the idea in his head that his father wants him to break up the company.	Yusef drives the van off a bridge. That fails. A second Kick occurs when the van hits the water.

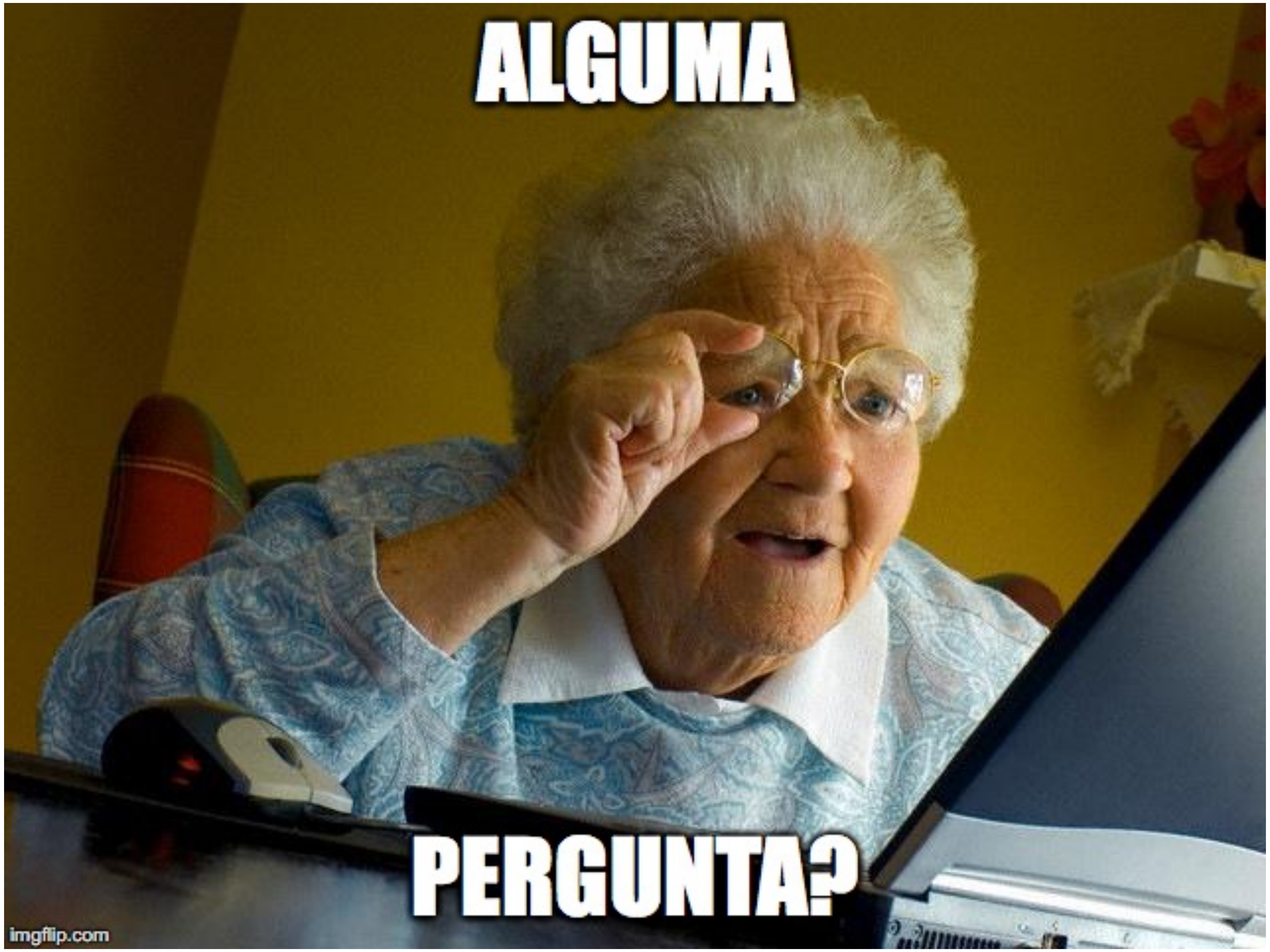
LEVEL 1
REALITY

No one...
We think

Cobb, Arthur, Ariadne, Eames, Saito, Yusuf and Robert Fischer Jr.

To drug Fischer Jr. and bring his subconscious into a dream.

There isn't one. The timer counts down and the machine shuts off.


ALGUMA
PERGUNTA?
imgflip.com

cálculo fatorial

cálculo fatorial

```
int fat_rec(int n)
{
   if (n <= 1) return 1;
   return n * fat_rec(n-1);
}
```

cálculo fatorial

```
int fat_iter(int n)
{
   int fat = 1;
   while (n > 1) { fat *= n; n—; }
   return fat;
}
```

```
int fat_rec(int n)
{
   if (n <= 1) return 1;
   return n * fat_rec(n-1);
}
```

cálculo fatorial

```
int fat_iter(int n)
{
   int fat = 1;
   while (n > 1) { fat *= n; n—; }
   return fat;
}
```

cálculo fatorial

```
int fat_iter(int n)
{
   int fat = 1;
   while (n > 1) { fat *= n; n—; }
   return fat;
}
```

Melhor caso

- O(n)

Pior caso

- O(n)

cálculo fatorial

```
int fat_rec(int n)
{
   if (n <= 1) return 1;
   return n * fat_rec(n-1);
}
```

cálculo fatorial

Caso base

- ?

Caso recursivo

- ?

```
int fat_rec(int n)
{
   if (n <= 1) return 1;
   return n * fat_rec(n-1);
}
```

cálculo fatorial

Caso **base**

- 2

Caso **recursivo**

- 2 + T(n-1)

```
int fat_rec(int n)
{
   if (n <= 1) return 1;
   return n * fat_rec(n-1);
}
```

cálculo fatorial

cálculo **fatorial**

$$T(n) = T(n-1) + 2$$

cálculo **fatorial**

$$T(n) = T(n-1) + 2$$

$$T(n) = (T(n-2) + 2) + 2$$

cálculo **fatorial**

$$T(n) = T(n-1) \quad + 2$$

$$T(n) = (T(n-2) + 2) \quad + 2$$

$$T(n) = T(n-2) + 4$$

cálculo **fatorial**

$$T(n) = T(n-1) + 2$$

$$T(n) = (T(n-2) + 2) + 2$$

$$T(n) = T(n-2) + 4$$

$$T(n) = (T(n-3) + 2) + 4$$

cálculo **fatorial**

$$T(n) = T(n-1) + 2$$

$$T(n) = (T(n-2) + 2) + 2$$

$$T(n) = T(n-2) + 4$$

$$T(n) = (T(n-3) + 2) + 4$$

$$T(n) = T(n-3) + 6$$

cálculo fatorial

cálculo **fatorial**

$$T(n) = T(n-1) + 2$$

cálculo **fatorial**

$$T(n) = T(n-1) + 2$$

$$T(n) = T(n-2) + 4$$

cálculo **fatorial**

$$T(n) = T(n-1) + 2$$

$$T(n) = T(n-2) + 4$$

$$T(n) = T(n-3) + 6$$

cálculo **fatorial**

$$T(n) = T(n-1) + 2$$

$$T(n) = T(n-2) + 4$$

$$T(n) = T(n-3) + 6$$

$$\dots$$

cálculo **fatorial**

$$T(n) = T(n-1) + 2$$

$$T(n) = T(n-2) + 4$$

$$T(n) = T(n-3) + 6$$

$$\ldots$$

$$T(n) = T(n-k) + 2k$$

cálculo fatorial

cálculo **fatorial**

$$T(n) = T(n-k) + 2k$$

cálculo **fatorial**

$$T(n) = T(n-k) + 2k$$

Fazendo $k = n$:

cálculo **fatorial**

$$T(n) = T(n-k) + 2k$$

Fazendo $k = n$:

$$T(n) = T(n-n) + 2n$$

cálculo **fatorial**

$$T(n) = T(n-k) + 2k$$

Fazendo $k = n$:

$$T(n) = T(n-n) + 2n$$

$$T(n) = T(0) + 2n$$

cálculo **fatorial**

$$T(n) = T(n-k) + 2k$$

Fazendo $k = n$:

$$T(n) = T(n-n) + 2n$$

$$T(n) = T(0) + 2n$$

$$T(n) = 2 + 2n$$

cálculo fatorial

- $T(n) = 2 + 2n$
  - $O(n)$

```
int fat_rec(int n)
{
   if (n <= 1) return 1;
   return n * fat_rec(n-1);
}
```

cálculo **fatorial**

cálculo fatorial

iterativo	

cálculo fatorial

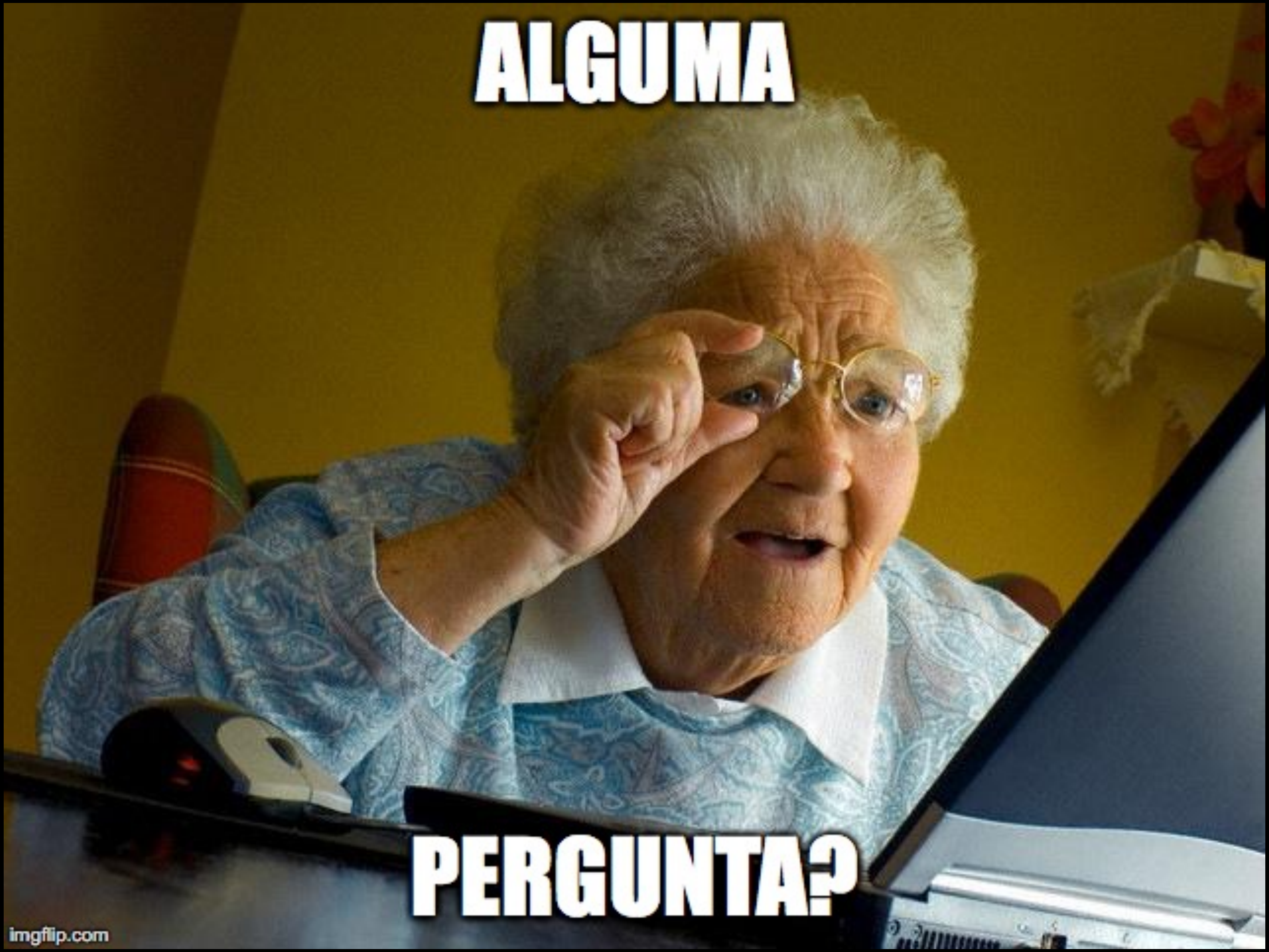
iterativo	recursivo

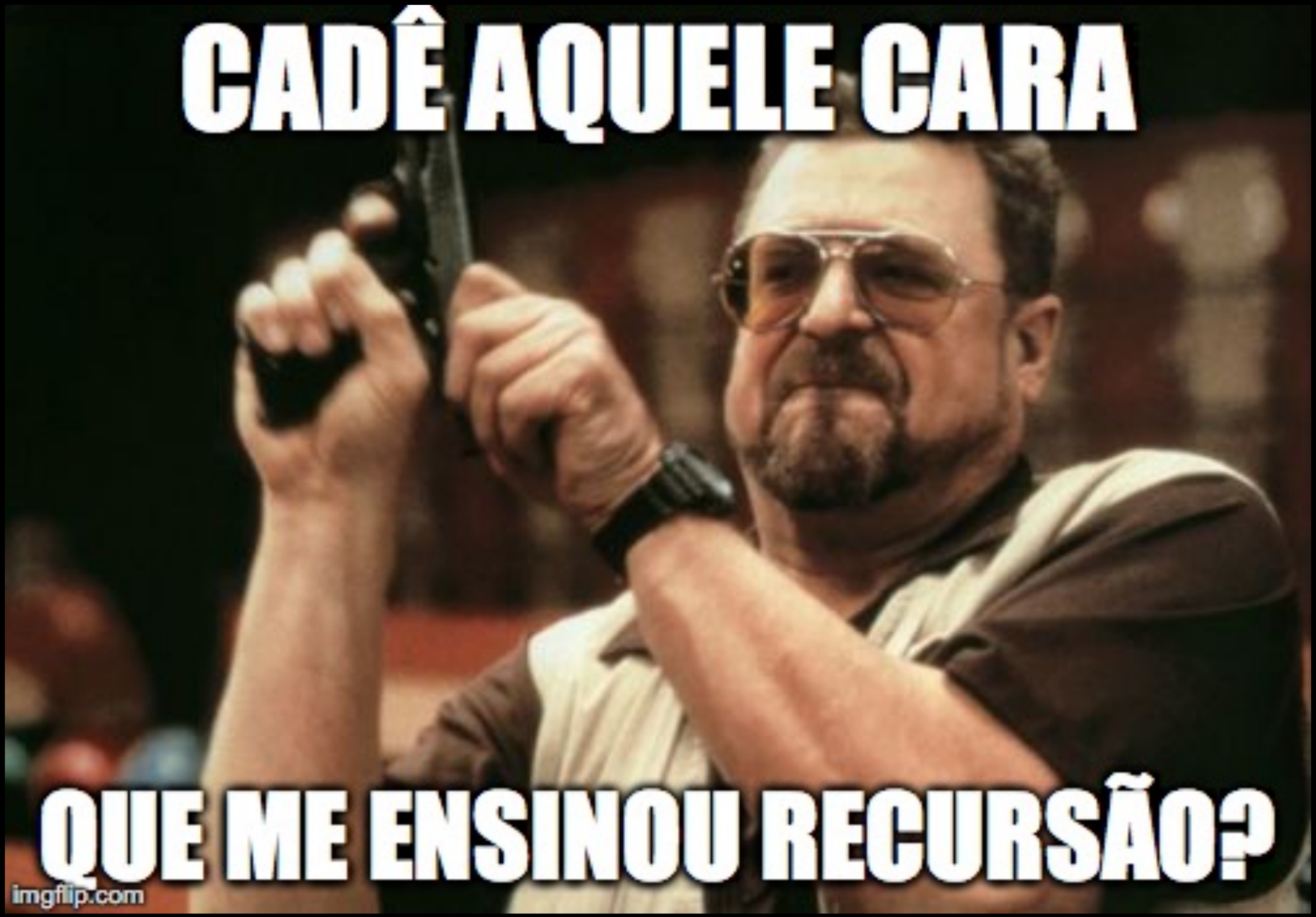
cálculo **fatorial**

iterativo	recursivo
O(n)	

cálculo **fatorial**

iterativo	recursivo
$O(n)$	$O(n)$


ALGUMA
PERGUNTA?
imgflip.com


CADÊ AQUELE CARA
QUE ME ENSINOU RECURSÃO?
imgflip.com

dividir para conquistar

dividir para conquistar	intersecções entre subproblemas

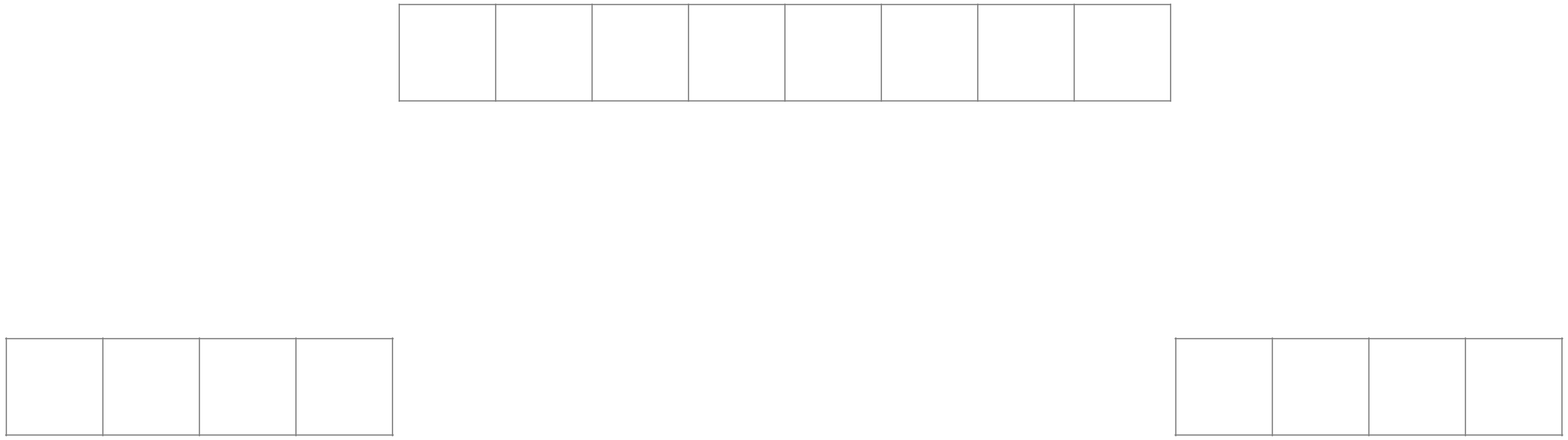
dividir para conquistar	intersecções entre subproblemas
recursão de cauda	

dividir para conquistar	intersecções entre subproblemas
recursão de cauda	demais casos

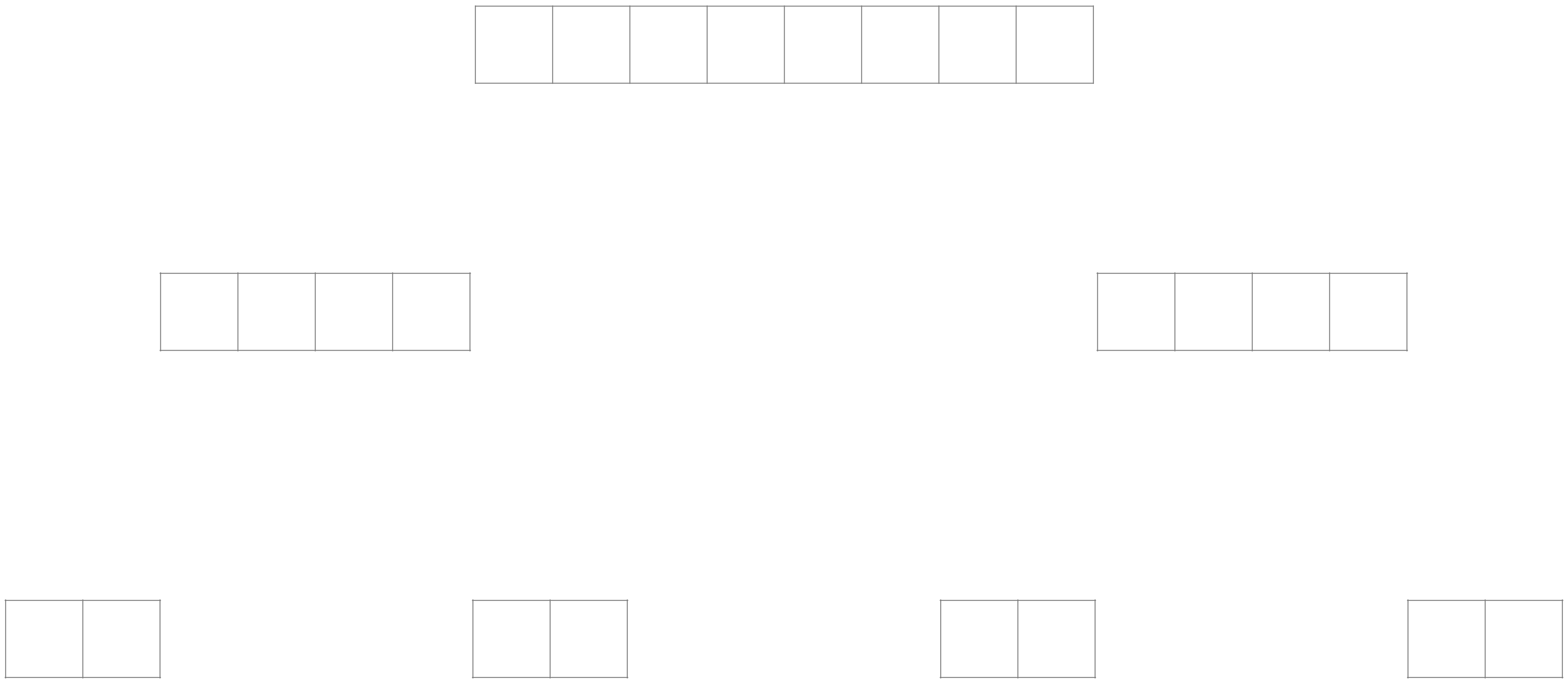
dividir para conquistar	intersecções entre subproblemas
recursão de cauda	demais casos

dividir para conquistar

dividir para conquistar



dividir para conquistar



dividir para conquistar

1	4	7	10	15	16	18	21

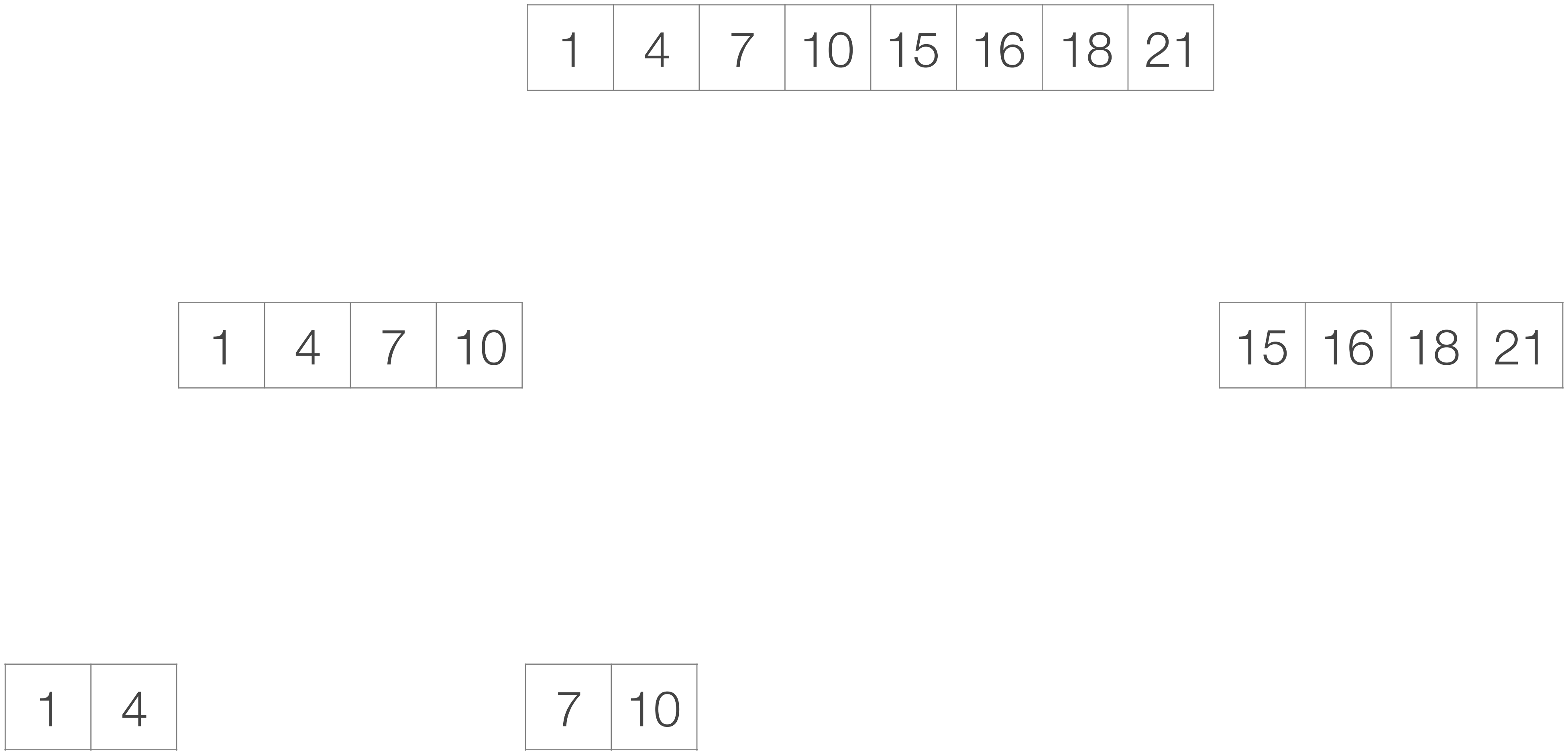
dividir para conquistar

1	4	7	10	15	16	18	21

1	4	7	10

15	16	18	21

dividir para conquistar



1	4	7	10	15	16	18	21

1	4	7	10

15	16	18	21

1	4

7	10

dividir para conquistar

1	4	7	10	15	16	18	21

1	4	7	10

15	16	18	21

1	4

7	10

15	16

18	21

dividir para conquistar

busca(3)

1	4	7	10	15	16	18	21

dividir para conquistar

busca(3)

1	4	7	10	15	16	18	21

1	4	7	10

dividir para conquistar

busca(3)

1	4	7	10	15	16	18	21

1	4	7	10

1	4

dividir para conquistar

busca(3)

O(log n)

1	4	7	10	15	16	18	21

1	4	7	10

1	4

dividir para conquistar

busca(3)

1	7	4	15	10	21	18	16

dividir para conquistar

busca(3)

1	7	4	15	10	21	18	16

1	7	4	15

10	21	18	16

dividir para conquistar

busca(3)

1	7	4	15	10	21	18	16

1	7	4	15

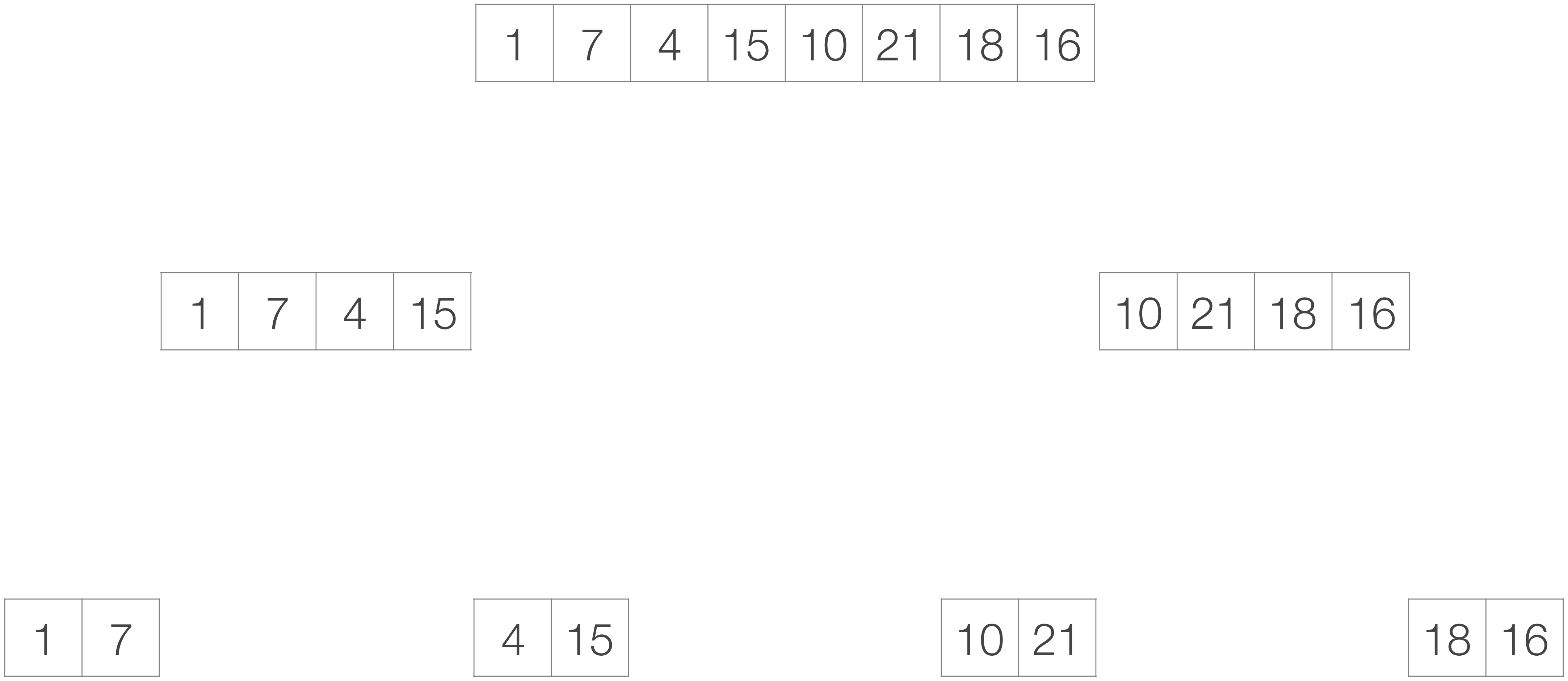
10	21	18	16

1	7

4	15

dividir para conquistar

busca(3)



1	7	4	15	10	21	18	16

1	7	4	15

10	21	18	16

1	7

4	15

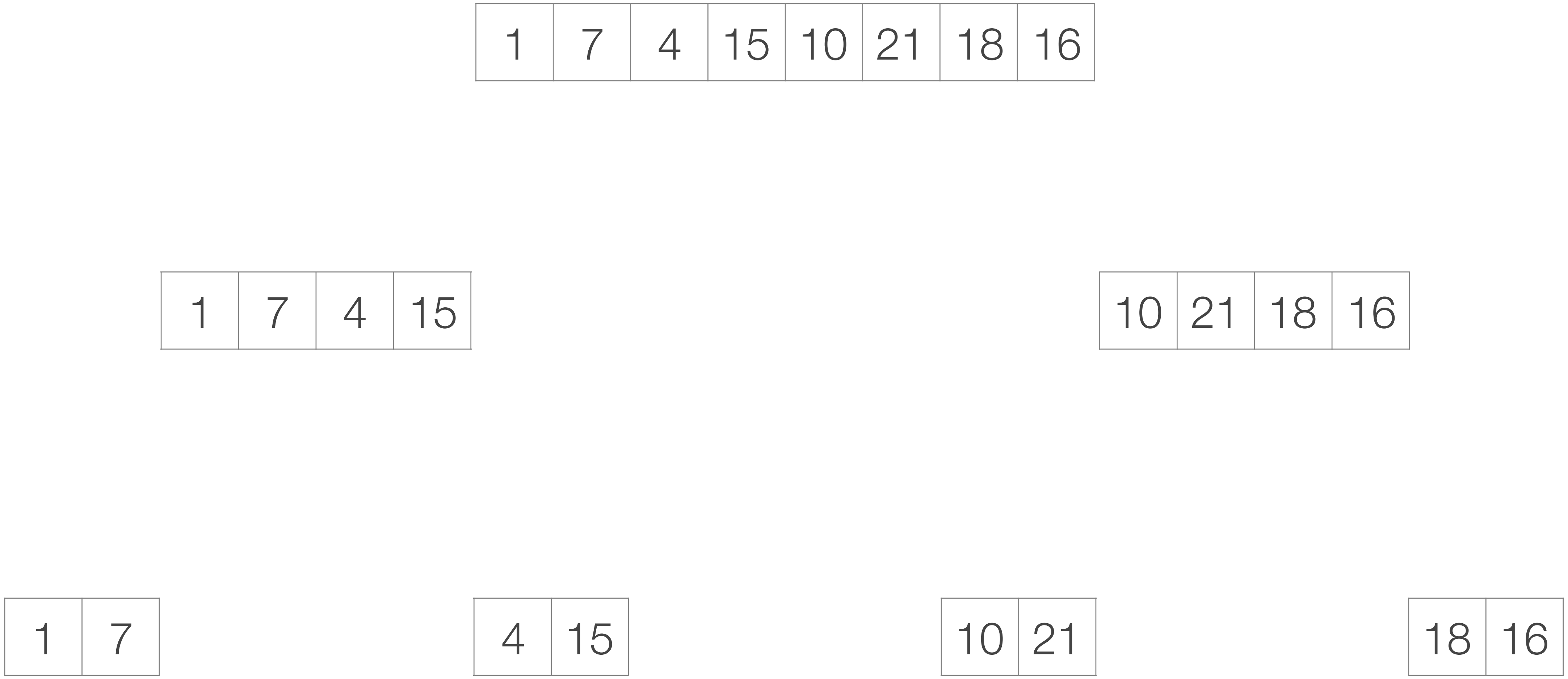
10	21

18	16

dividir para conquistar

busca(3)

O(n)



1	7	4	15	10	21	18	16

1	7	4	15

10	21	18	16

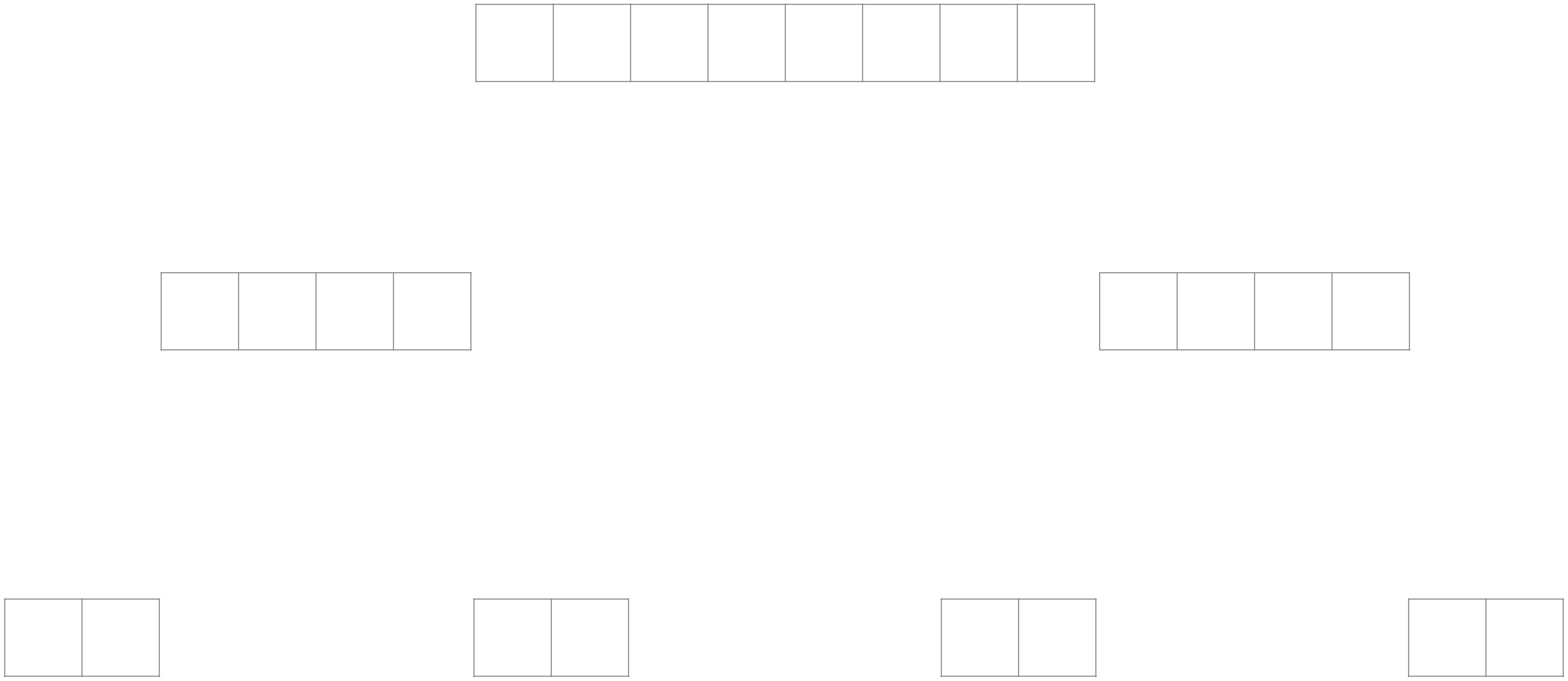
1	7

4	15

10	21

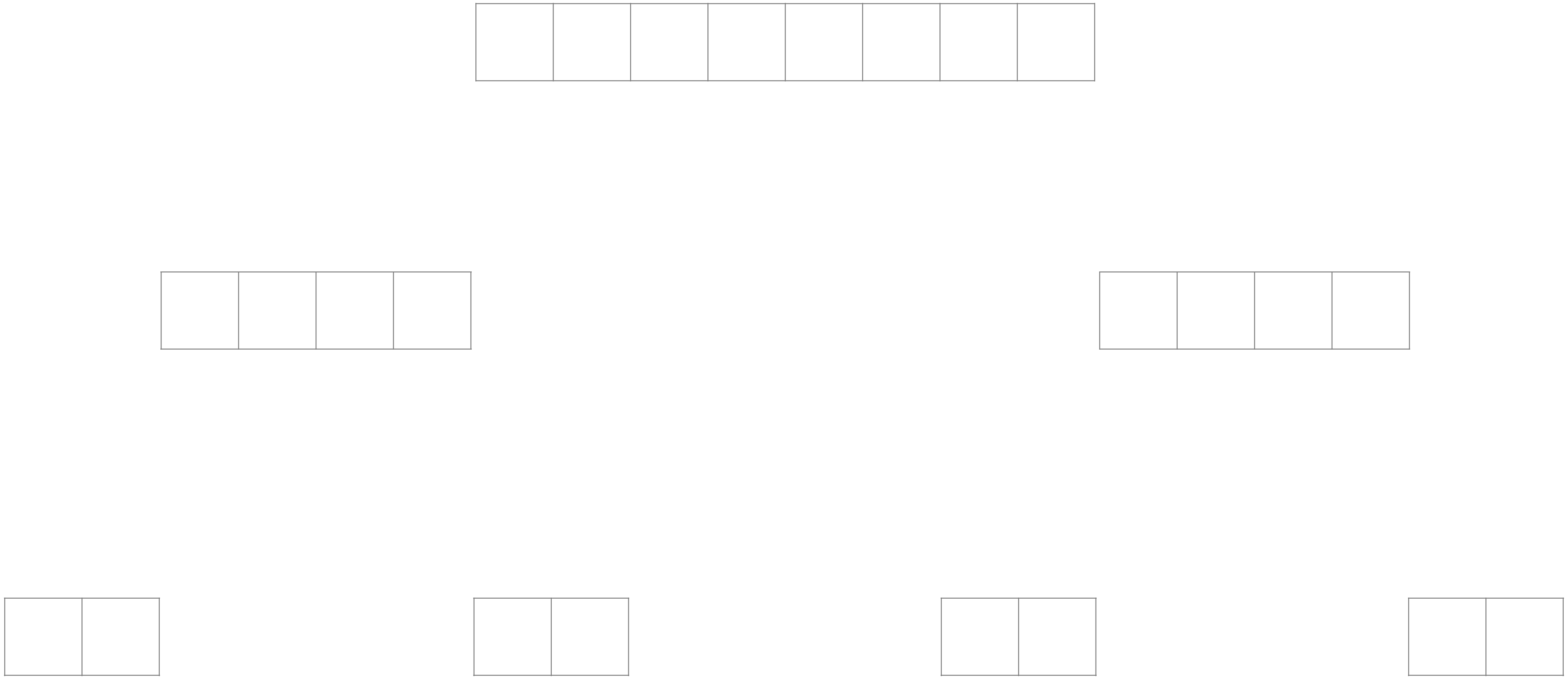
18	16

dividir para conquistar



dividir para conquistar

$aT(n/b)$



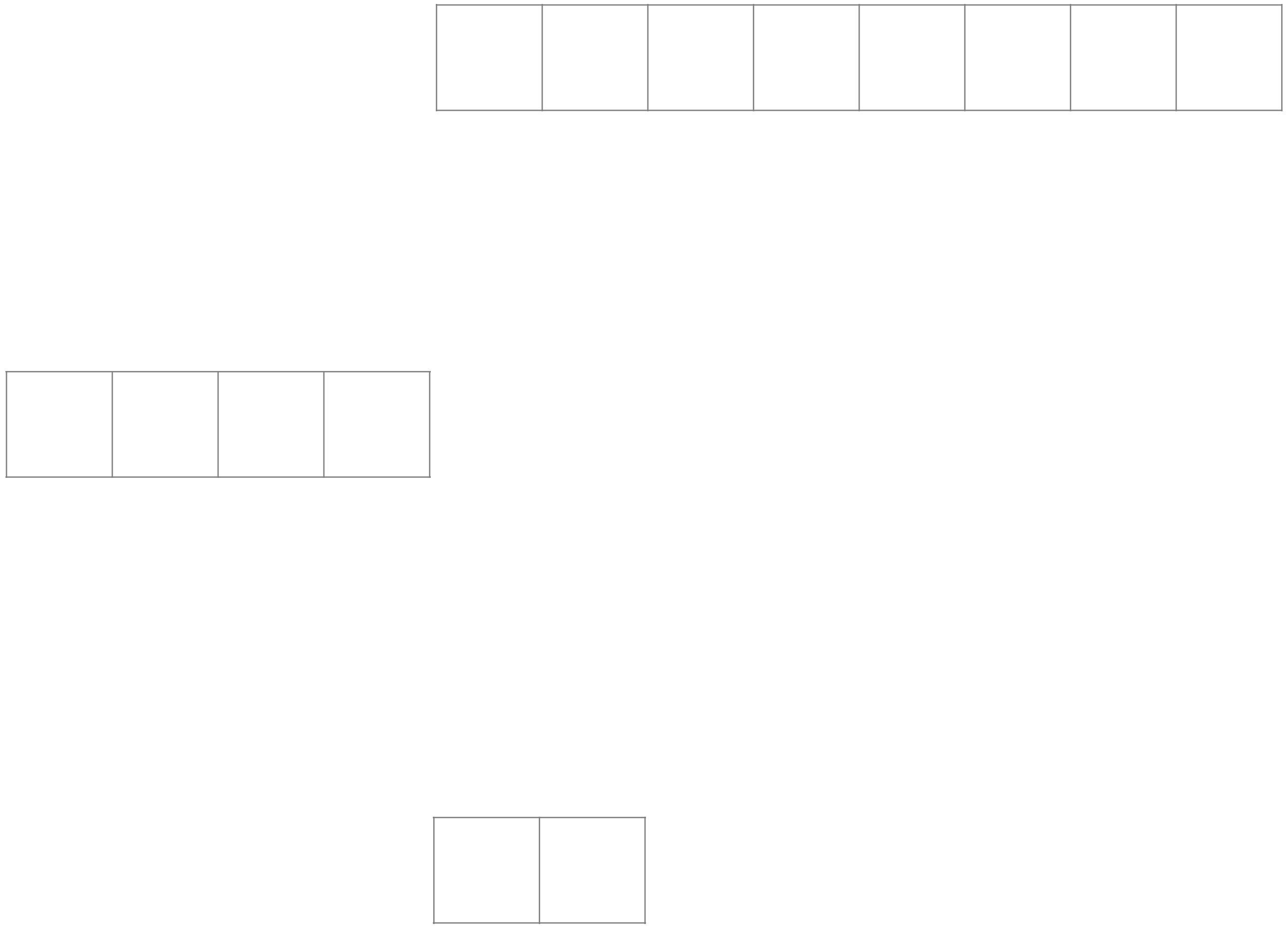
dividir para conquistar

$$aT(n/b)$$

a	fator de arborescência
b	fator de divisão

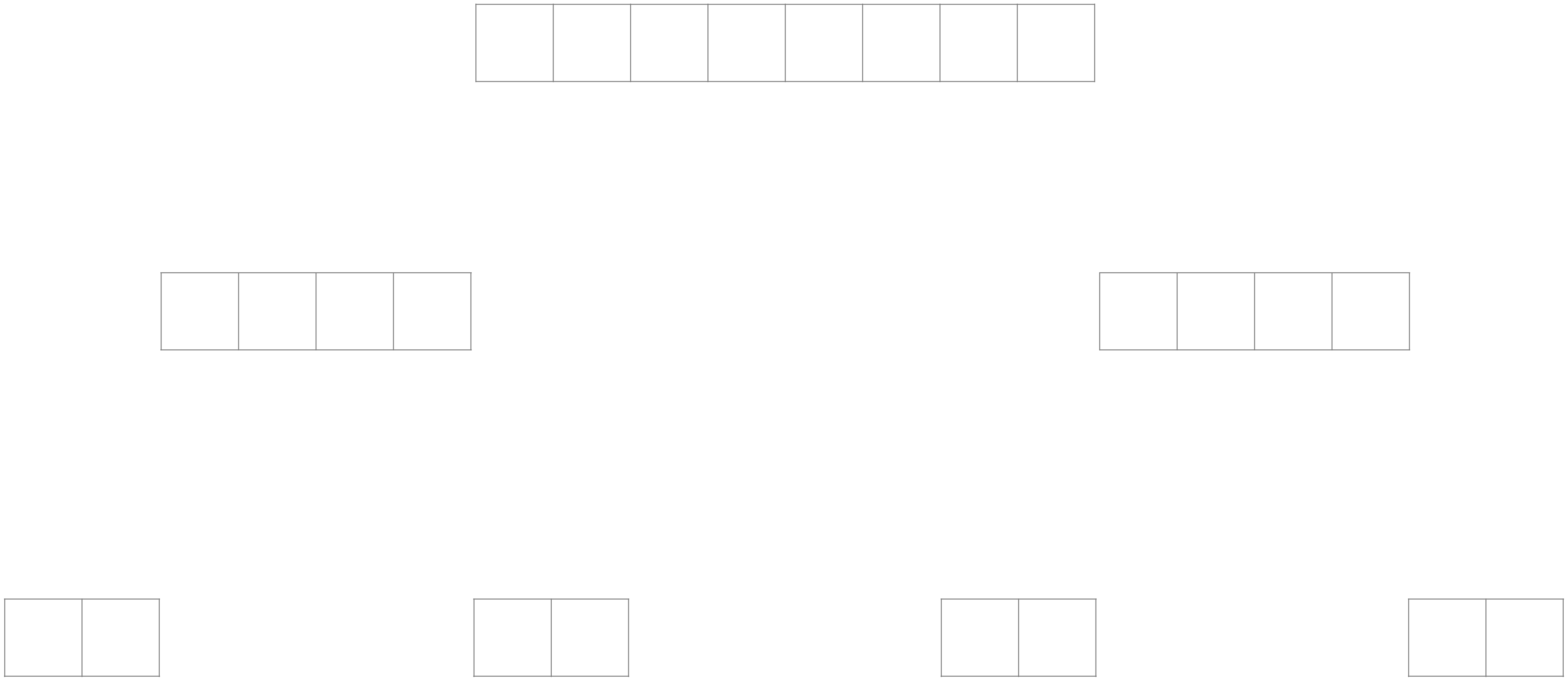
dividir para conquistar

$T(n/2)$



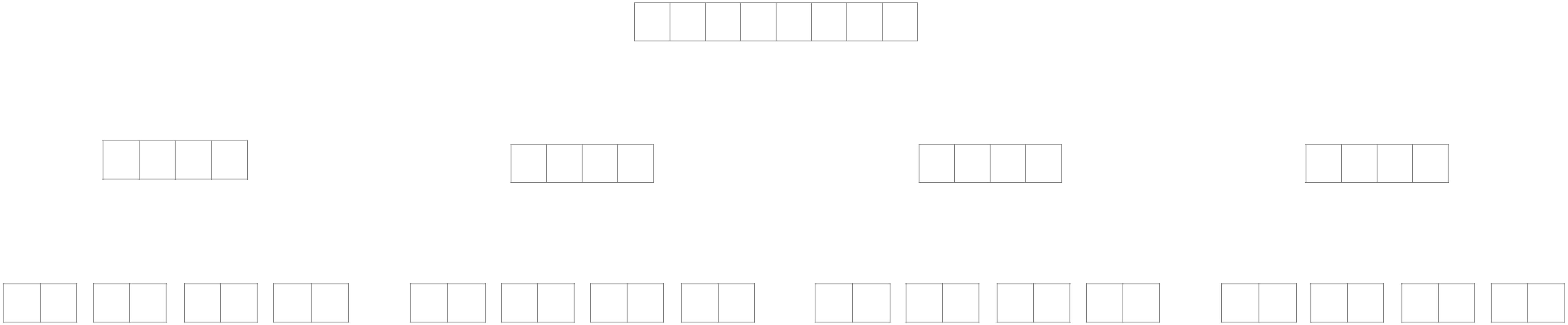
dividir para conquistar

$2T(n/2)$



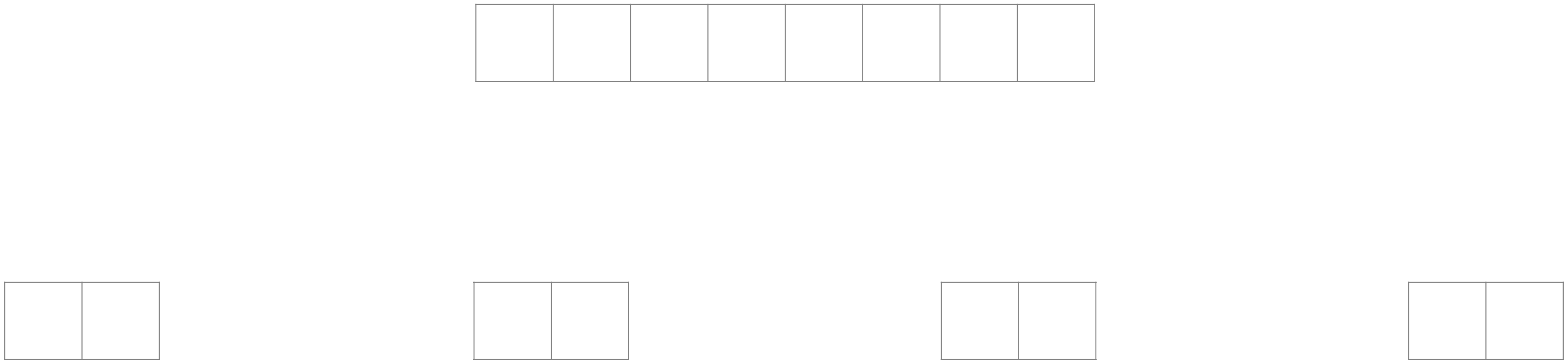
dividir para conquistar

$4T(n/2)$



dividir para conquistar

$4T(n/4)$



dividir para conquistar

dividir para conquistar

$$T(n) = aT(n/b) + f(n)$$

dividir para conquistar

$$T(n) = a\underbrace{T(n/b)}_{\text{dividir}} + f(n)$$

dividir para conquistar

$$T(n) = a\underbrace{T(n/b)}_{\text{dividir}} + \overbrace{f(n)}^{\text{conquistar}}$$

busca **binária**

busca binária

```
int bin_rec(int v[], int chave, int inicio, int fim)
```

busca **binária**

```
int bin_rec(int v[], int chave, int inicio, int fim)
{
```

busca **binária**

```
int bin_rec(int v[], int chave, int inicio, int fim)
{
   int tamanho = fim - inicio;
```

busca **binária**

```
int bin_rec(int v[], int chave, int inicio, int fim)
{
    int tamanho = fim - inicio;
    if (tamanho == 0) { return -1; }
```

busca binária

```
int bin_rec(int v[], int chave, int inicio, int fim)
{
   int tamanho = fim - inicio;
   if (tamanho == 0) { return -1; }
   int meio = inicio + floor(tamanho / 2);
```

busca binária

```
int bin_rec(int v[], int chave, int inicio, int fim)
{
   int tamanho = fim - inicio;
   if (tamanho == 0) { return -1; }
   int meio = inicio + floor(tamanho / 2);
   if (chave == v[meio]) { return meio; }
```

busca **binária**

```
int bin_rec(int v[], int chave, int inicio, int fim)
{
    int tamanho = fim - inicio;
    if (tamanho == 0) { return -1; }
    int meio = inicio + floor(tamanho / 2);
    if (chave == v[meio]) { return meio; }
    else if (chave < v[meio]) {
```

busca **binária**

```
int bin_rec(int v[], int chave, int inicio, int fim)
{
    int tamanho = fim - inicio;
    if (tamanho == 0) { return -1; }
    int meio = inicio + floor(tamanho / 2);
    if (chave == v[meio]) { return meio; }
    else if (chave < v[meio]) {
        return bin_rec(v, chave, inicio, meio);
```

busca **binária**

```
int bin_rec(int v[], int chave, int inicio, int fim)
{
   int tamanho = fim - inicio;
   if (tamanho == 0) { return -1; }
   int meio = inicio + floor(tamanho / 2);
   if (chave == v[meio]) { return meio; }
   else if (chave < v[meio]) {
      return bin_rec(v, chave, inicio, meio);
   }
```

busca **binária**

```
int bin_rec(int v[], int chave, int inicio, int fim)
{
    int tamanho = fim - inicio;
    if (tamanho == 0) { return -1; }
    int meio = inicio + floor(tamanho / 2);
    if (chave == v[meio]) { return meio; }
    else if (chave < v[meio]) {
        return bin_rec(v, chave, inicio, meio);
    }
    return bin_rec(v, chave, meio + 1, fim);
```

busca **binária**

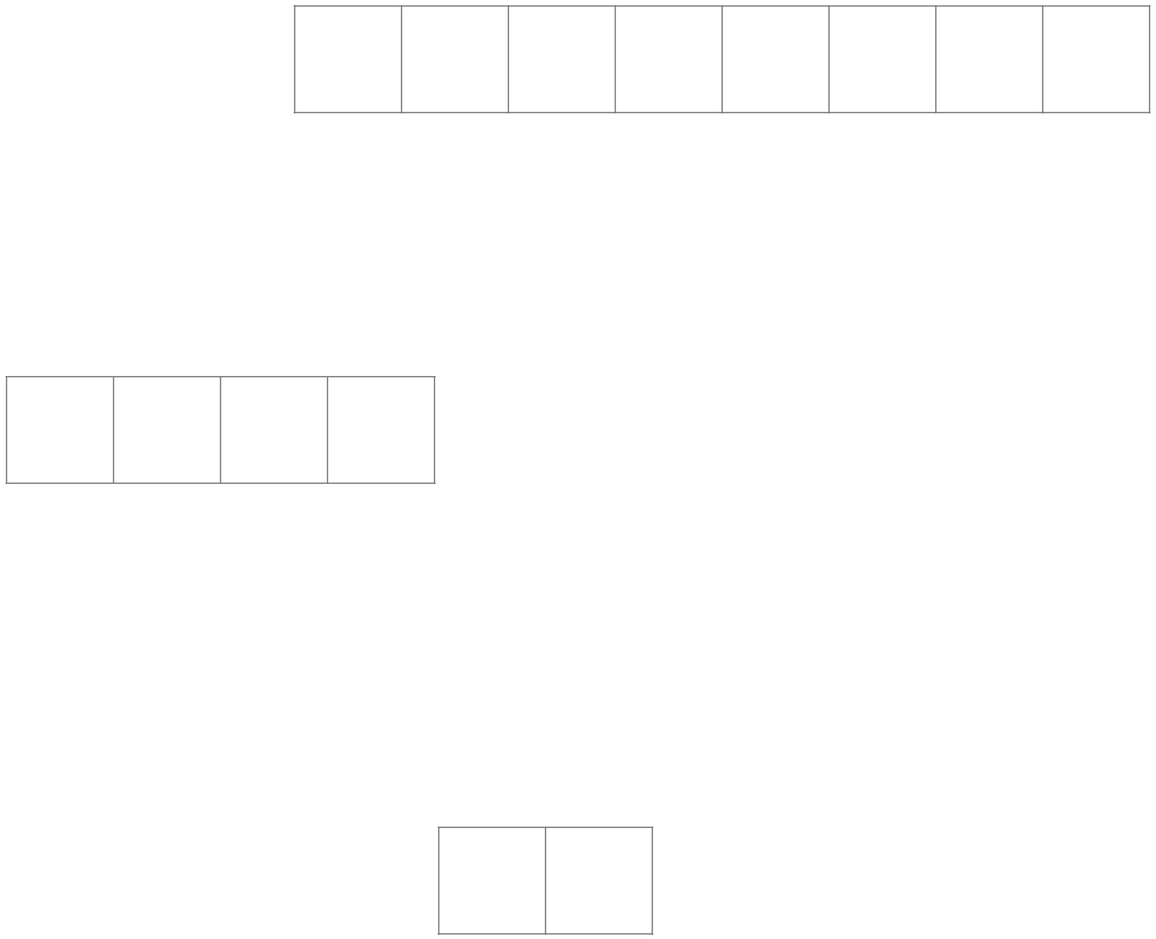
```
int bin_rec(int v[], int chave, int inicio, int fim)
{
    int tamanho = fim - inicio;
    if (tamanho == 0) { return -1; }
    int meio = inicio + floor(tamanho / 2);
    if (chave == v[meio]) { return meio; }
    else if (chave < v[meio]) {
        return bin_rec(v, chave, inicio, meio);
    }
    return bin_rec(v, chave, meio + 1, fim);
}
```

dividir para conquistar

$$T(n) = a\underbrace{T(n/b)}_{\text{dividir}} + \overbrace{f(n)}^{\text{conquistar}}$$

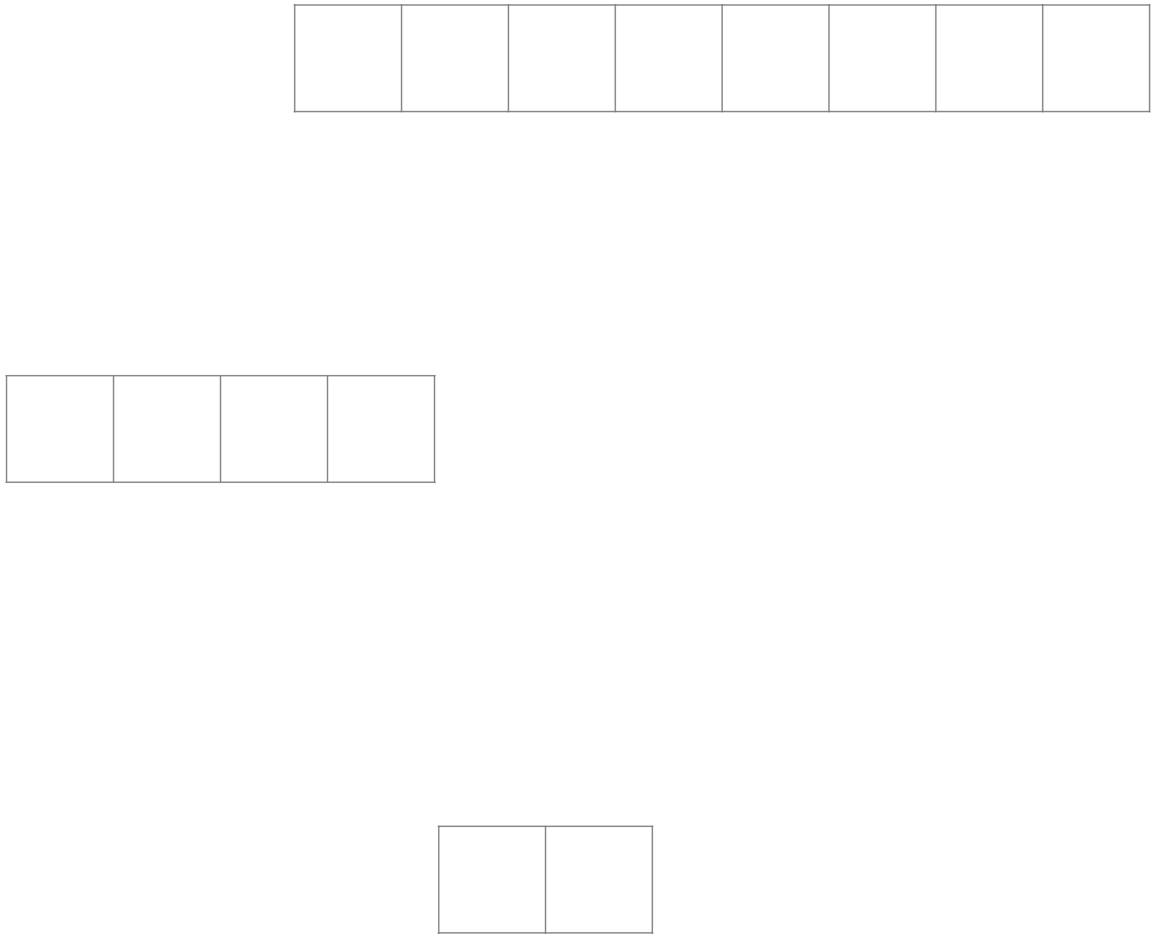
dividir para conquistar

$$T(n/2) + 1$$



dividir para conquistar

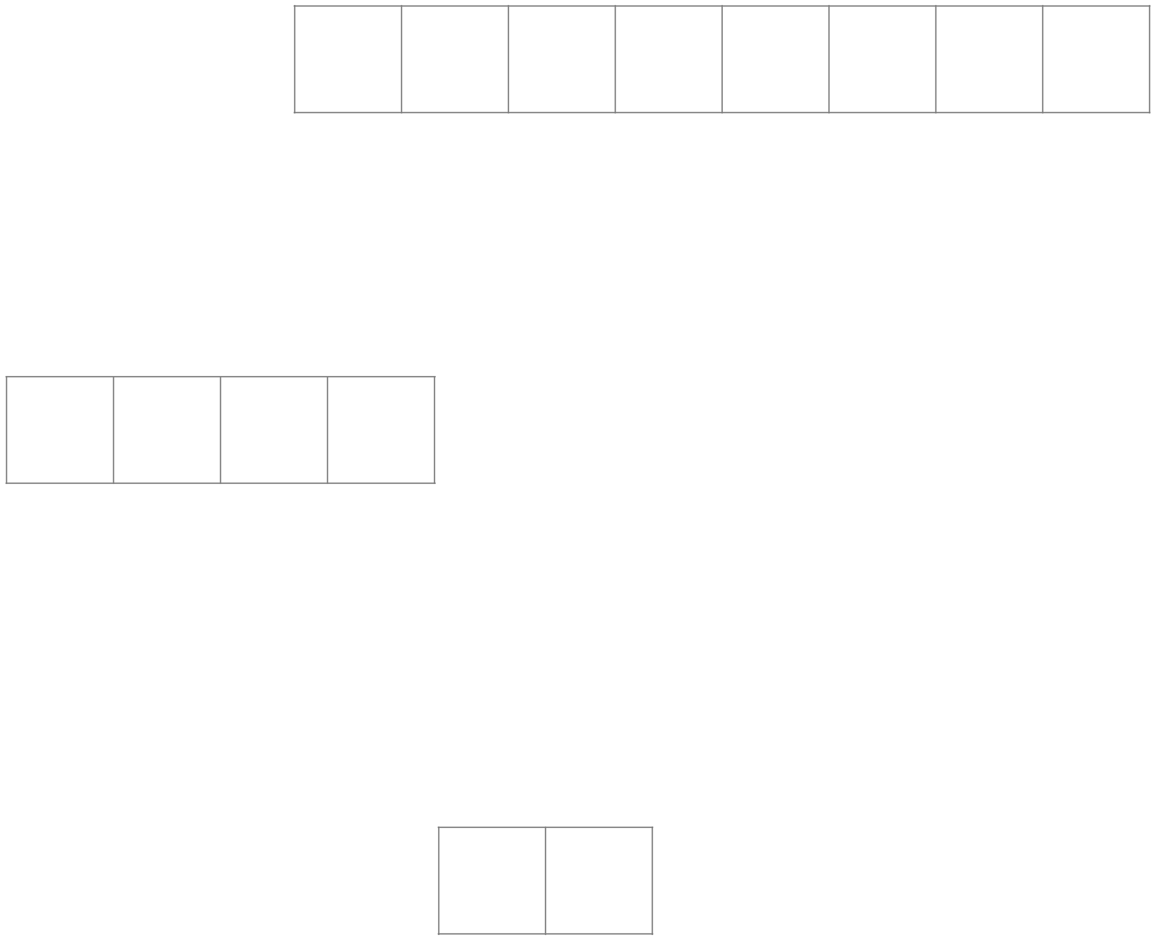
$T(n/2) + 1$



1

dividir para conquistar

$T(n/2) + 1$

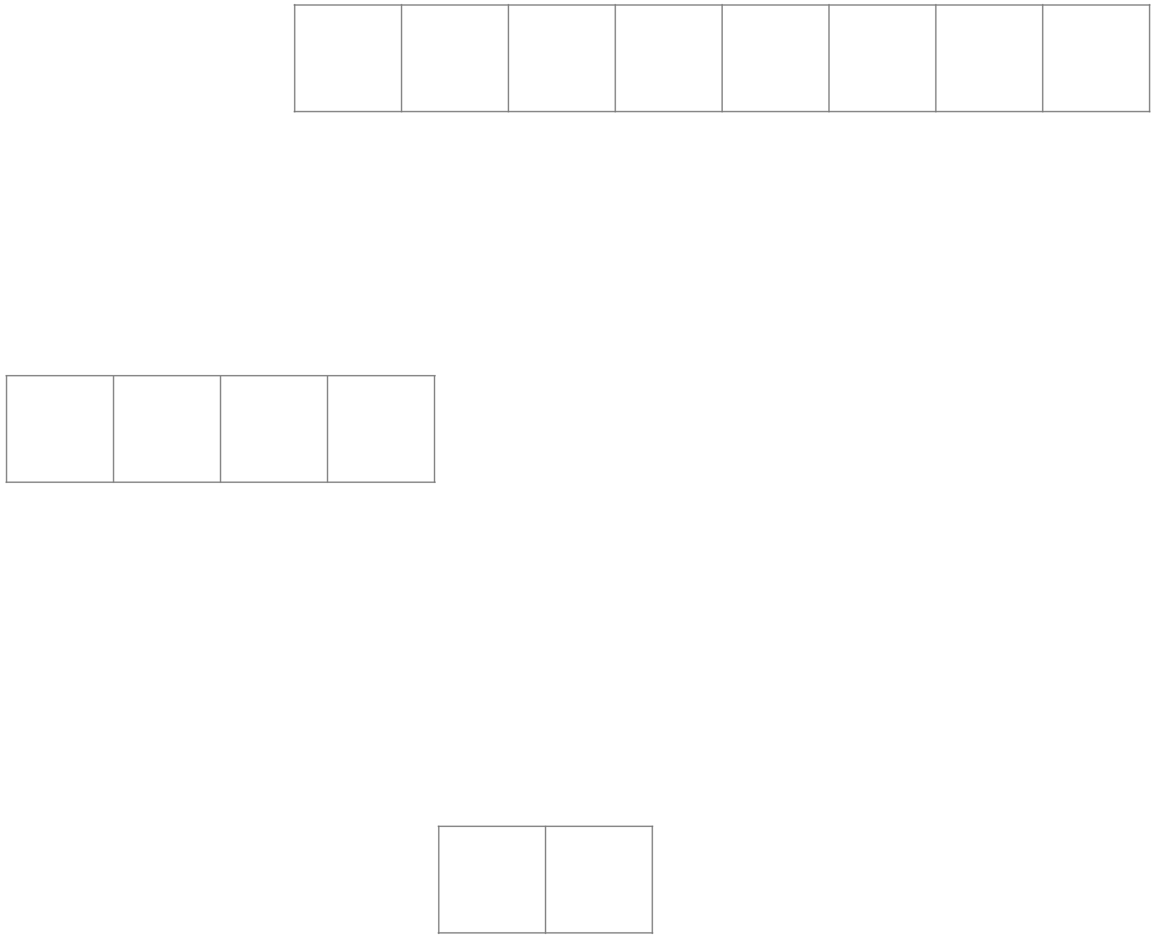


1

1

dividir para conquistar

$T(n/2) + 1$



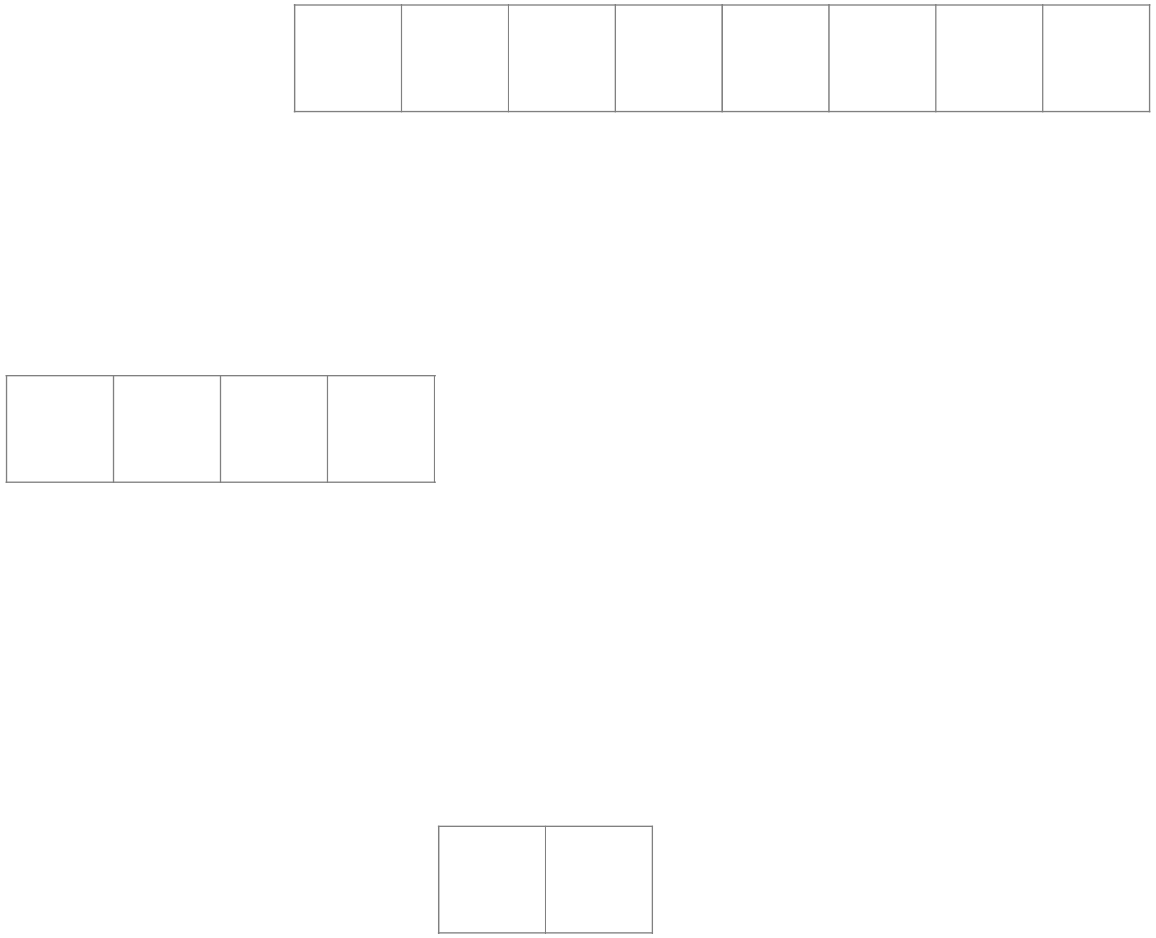
1

1

1

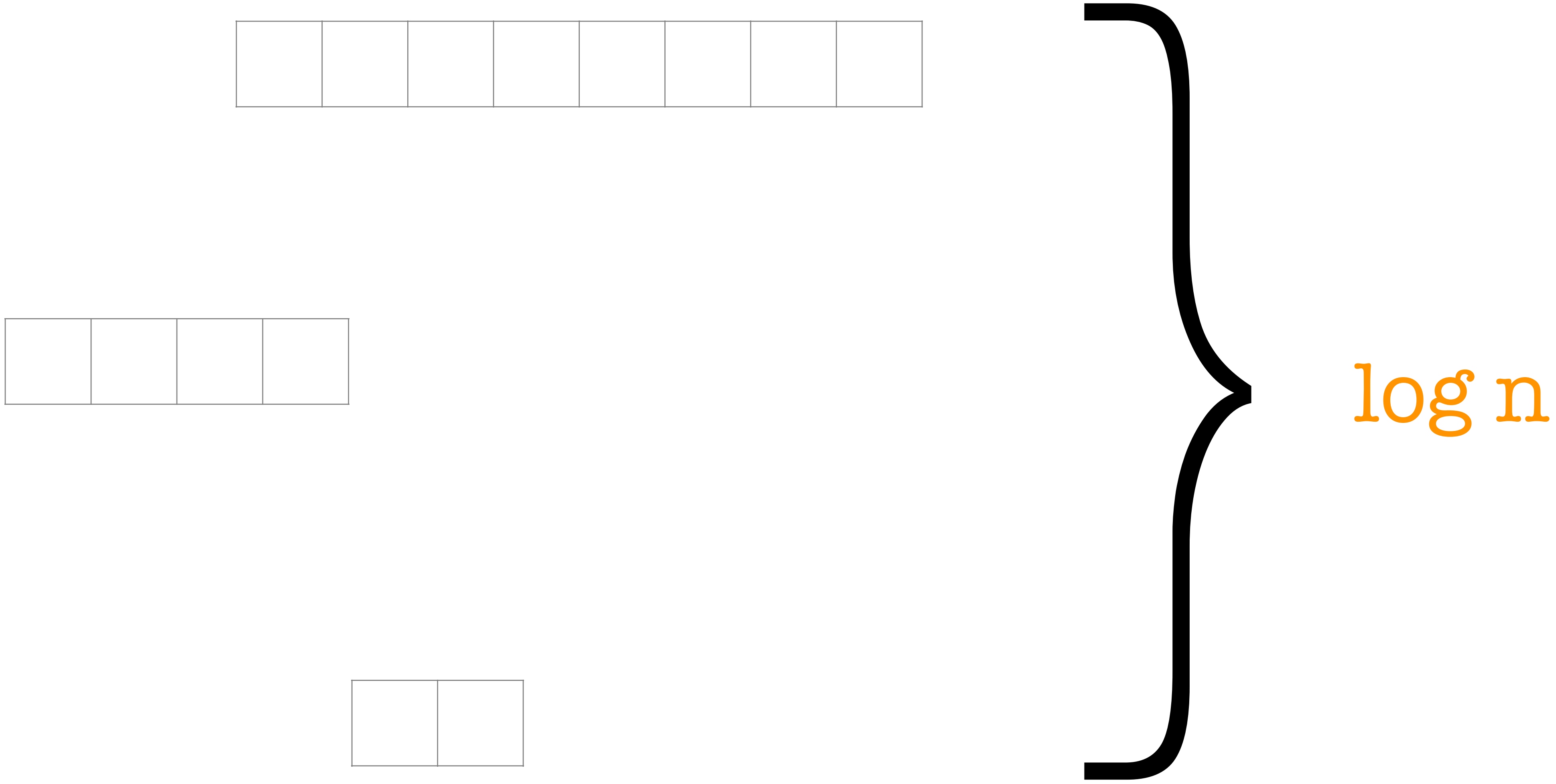
dividir para conquistar

$$T(n/2) + 1$$



dividir para conquistar

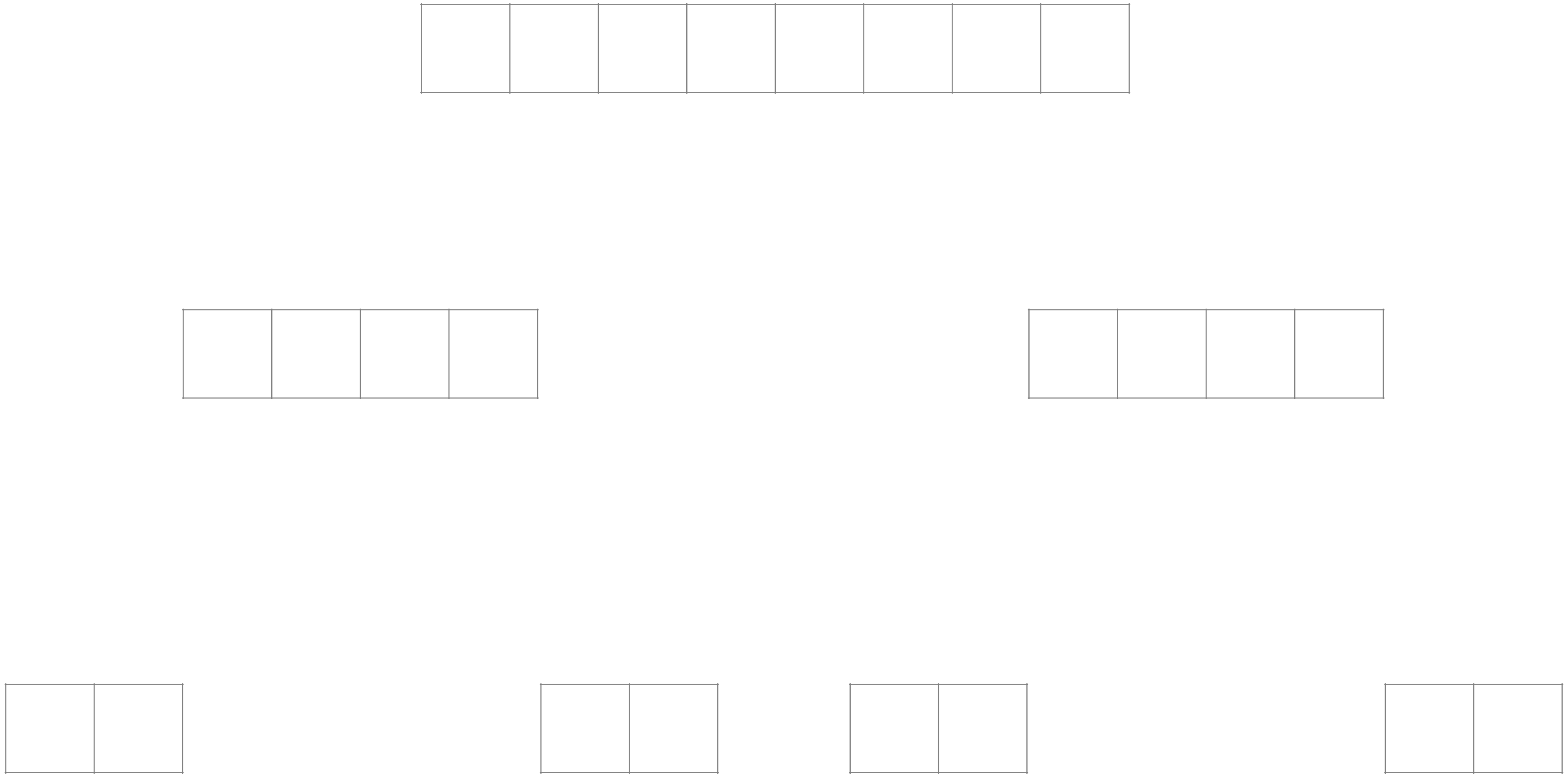
$T(n/2) + 1$



log n

dividir para conquistar

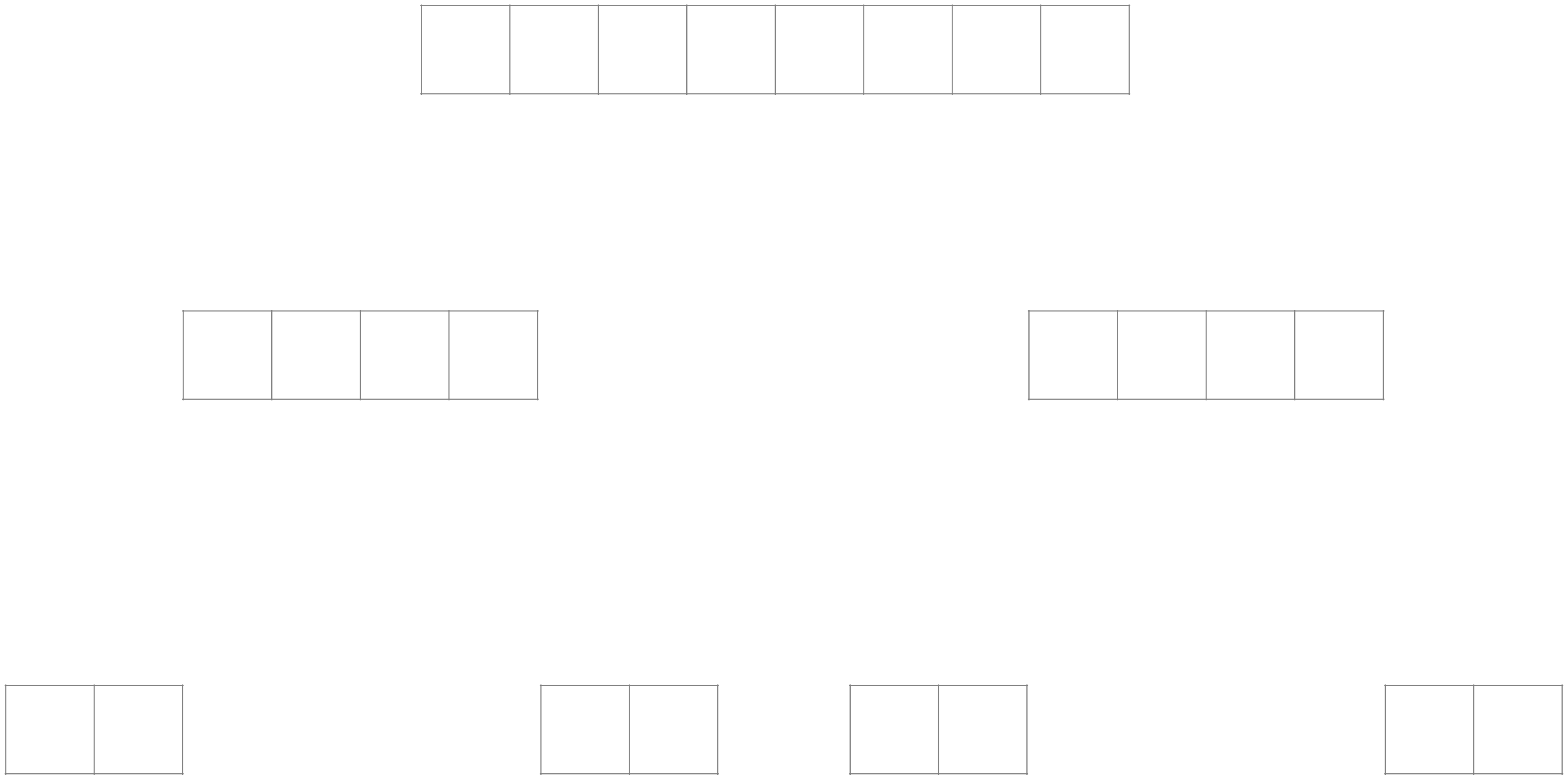
$2T(n/2) + 1$



dividir para conquistar

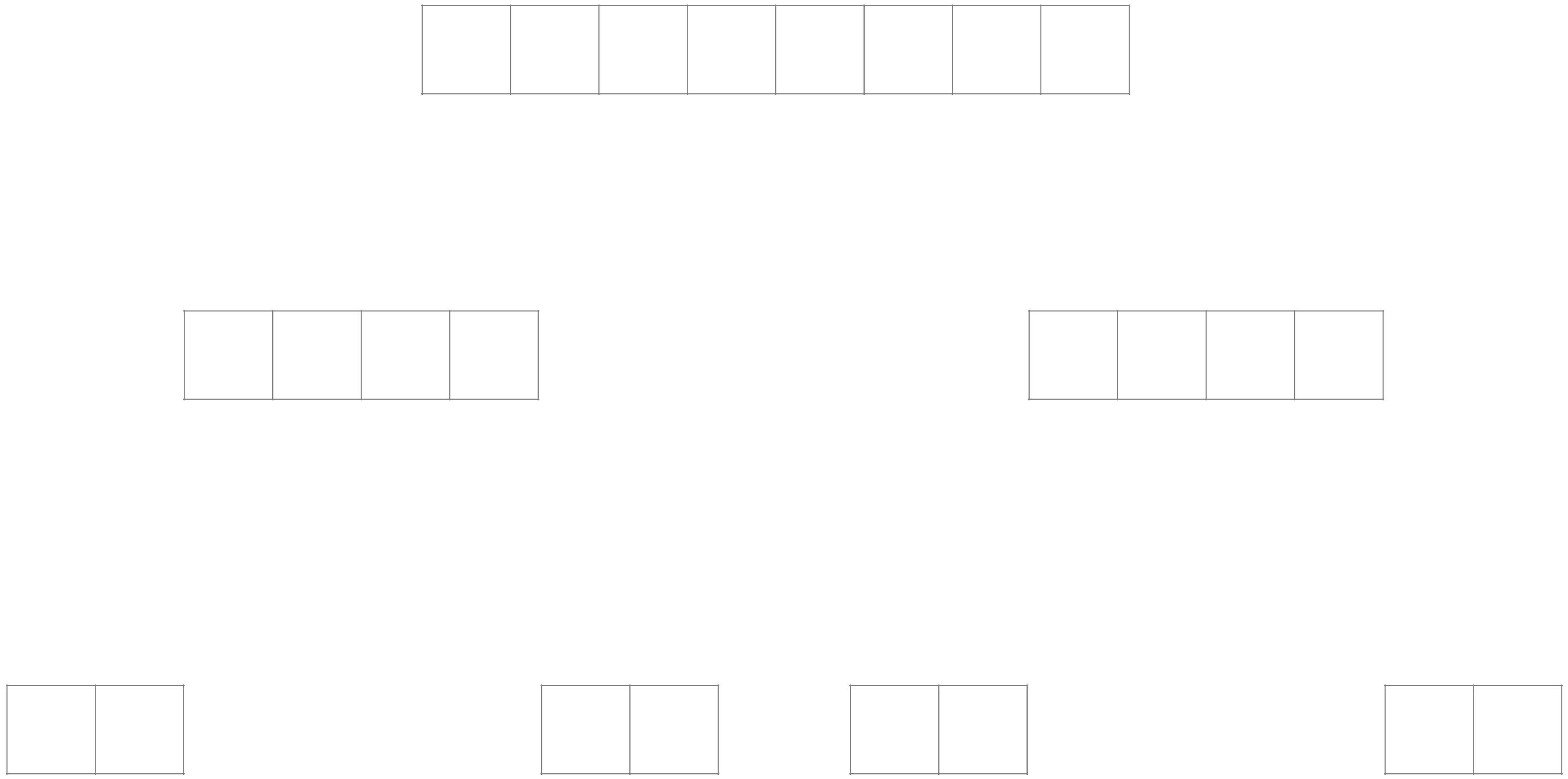
$2T(n/2) + 1$

1



dividir para conquistar

$2T(n/2) + 1$

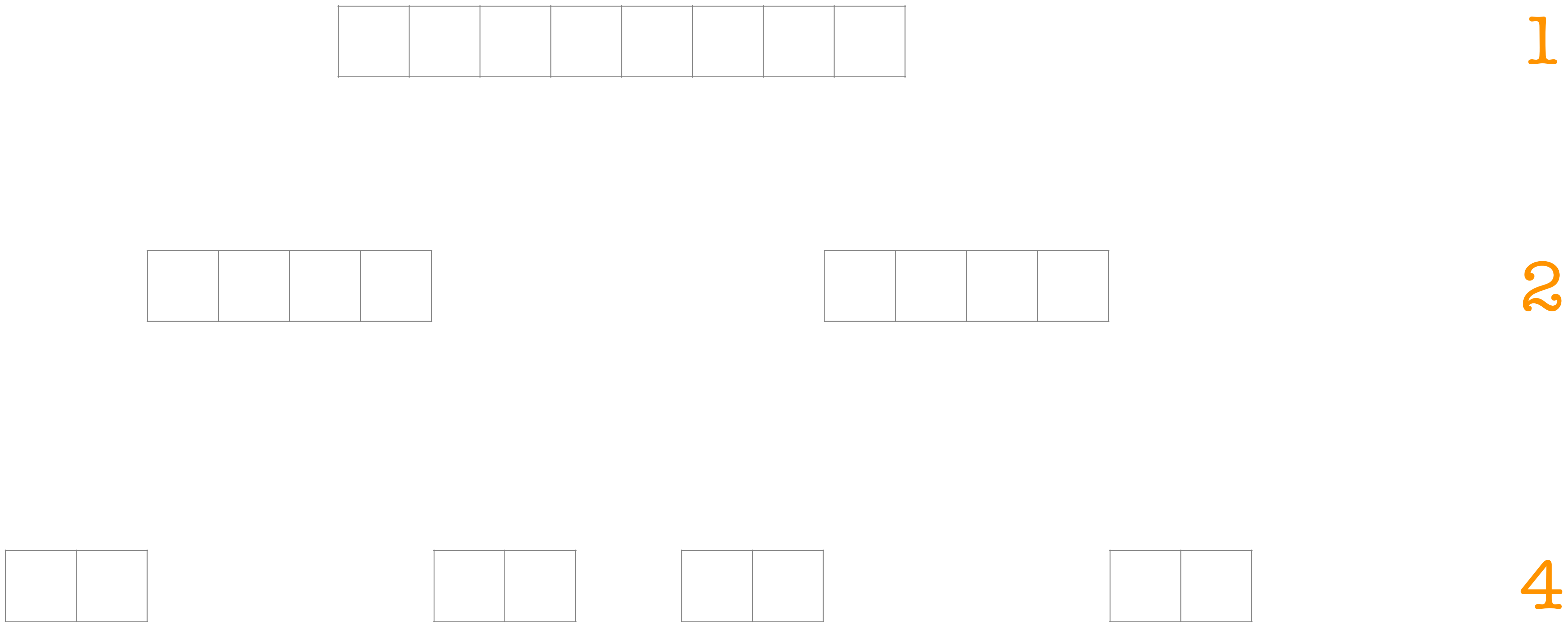


1

2

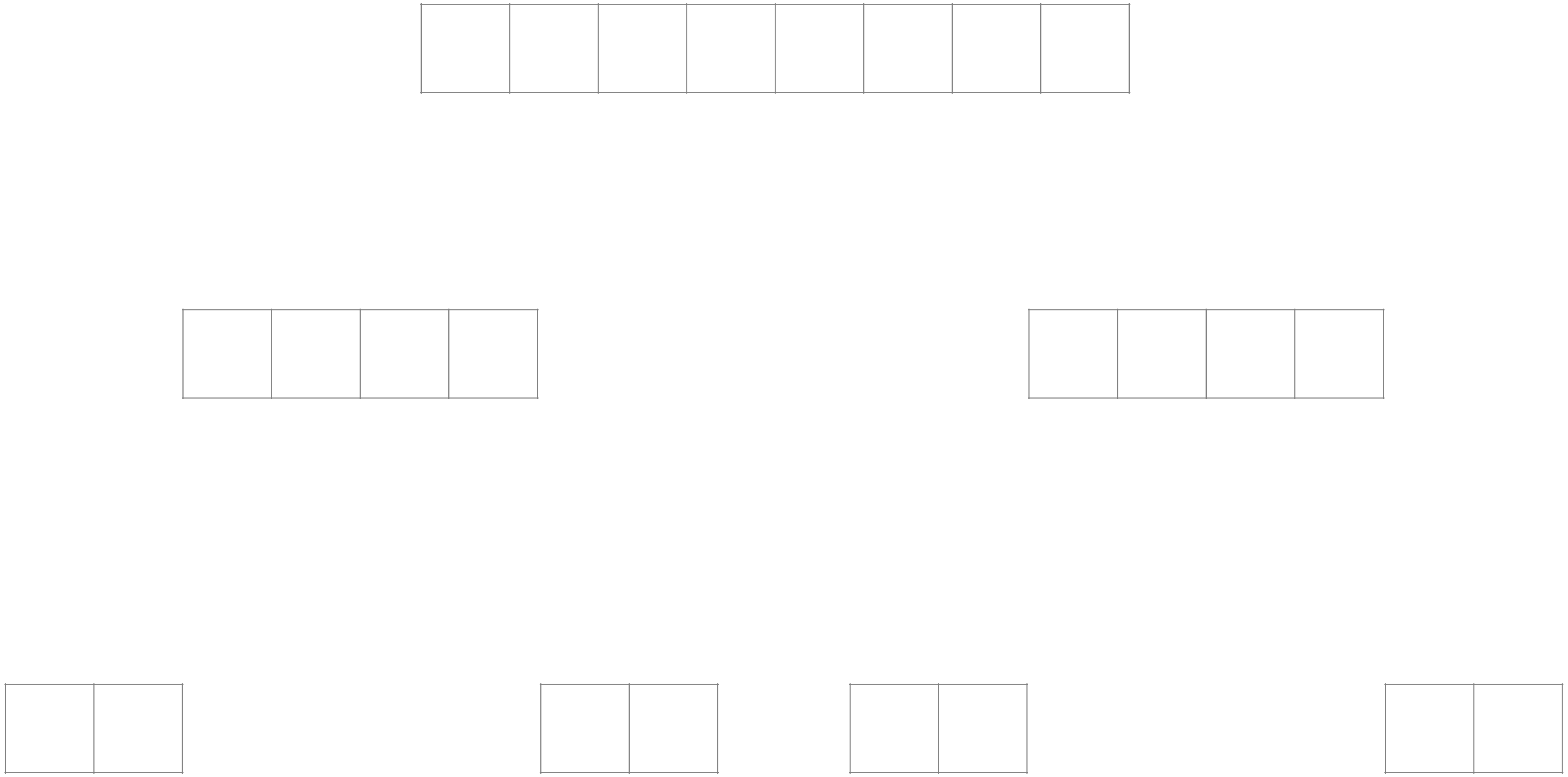
dividir para conquistar

$$2T(n/2) + 1$$



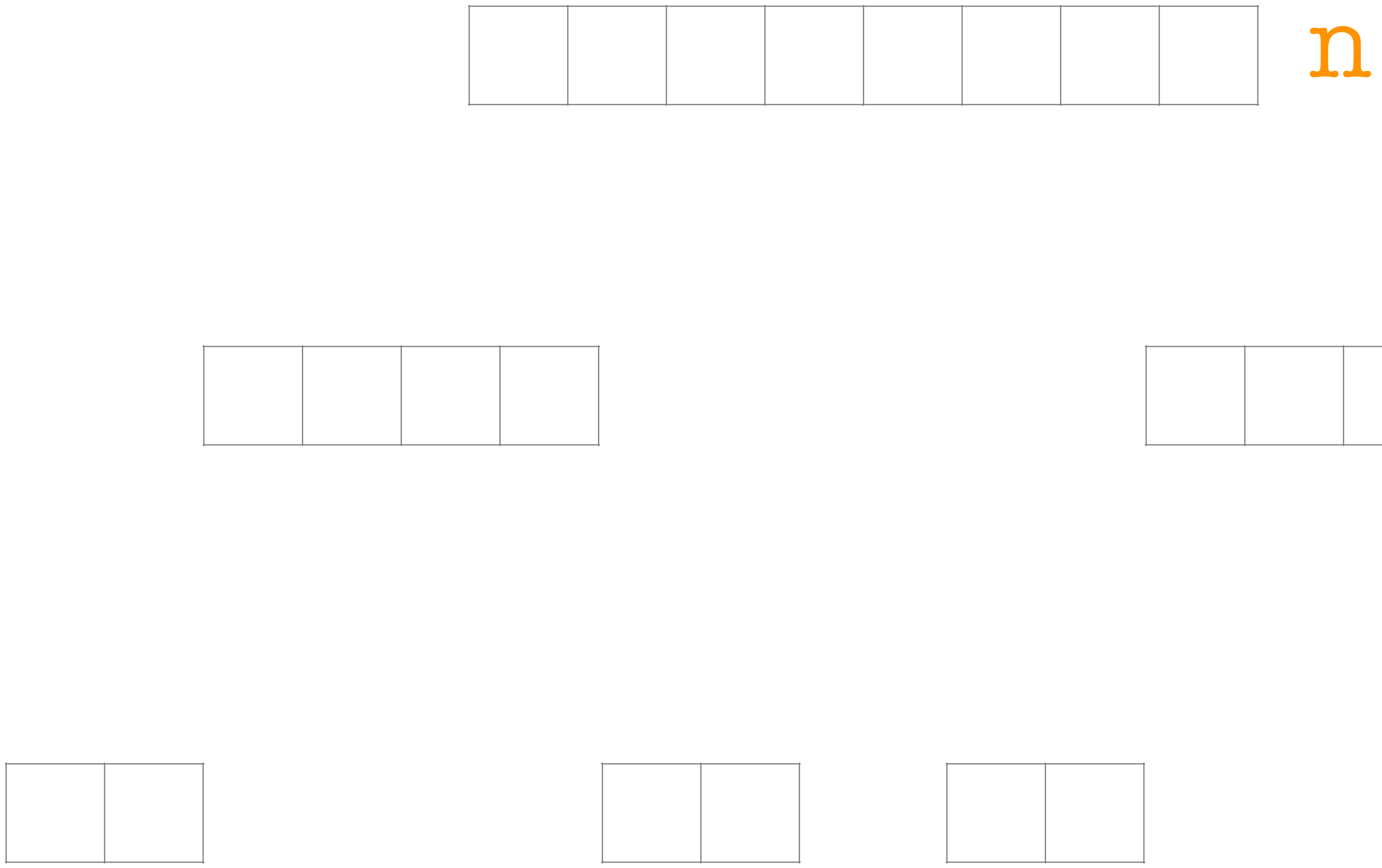
dividir para conquistar

$$2T(n/2) + n$$



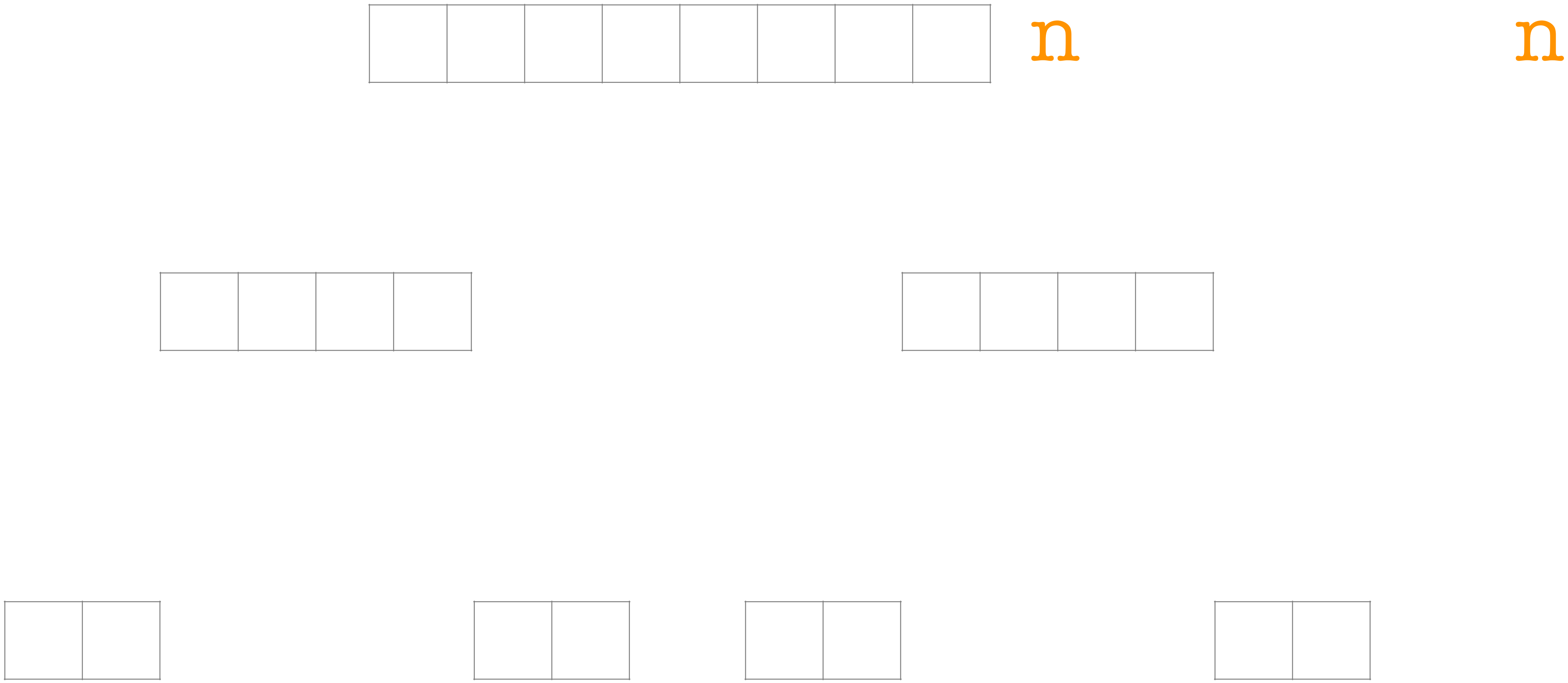
dividir para conquistar

$$2T(n/2) + n$$



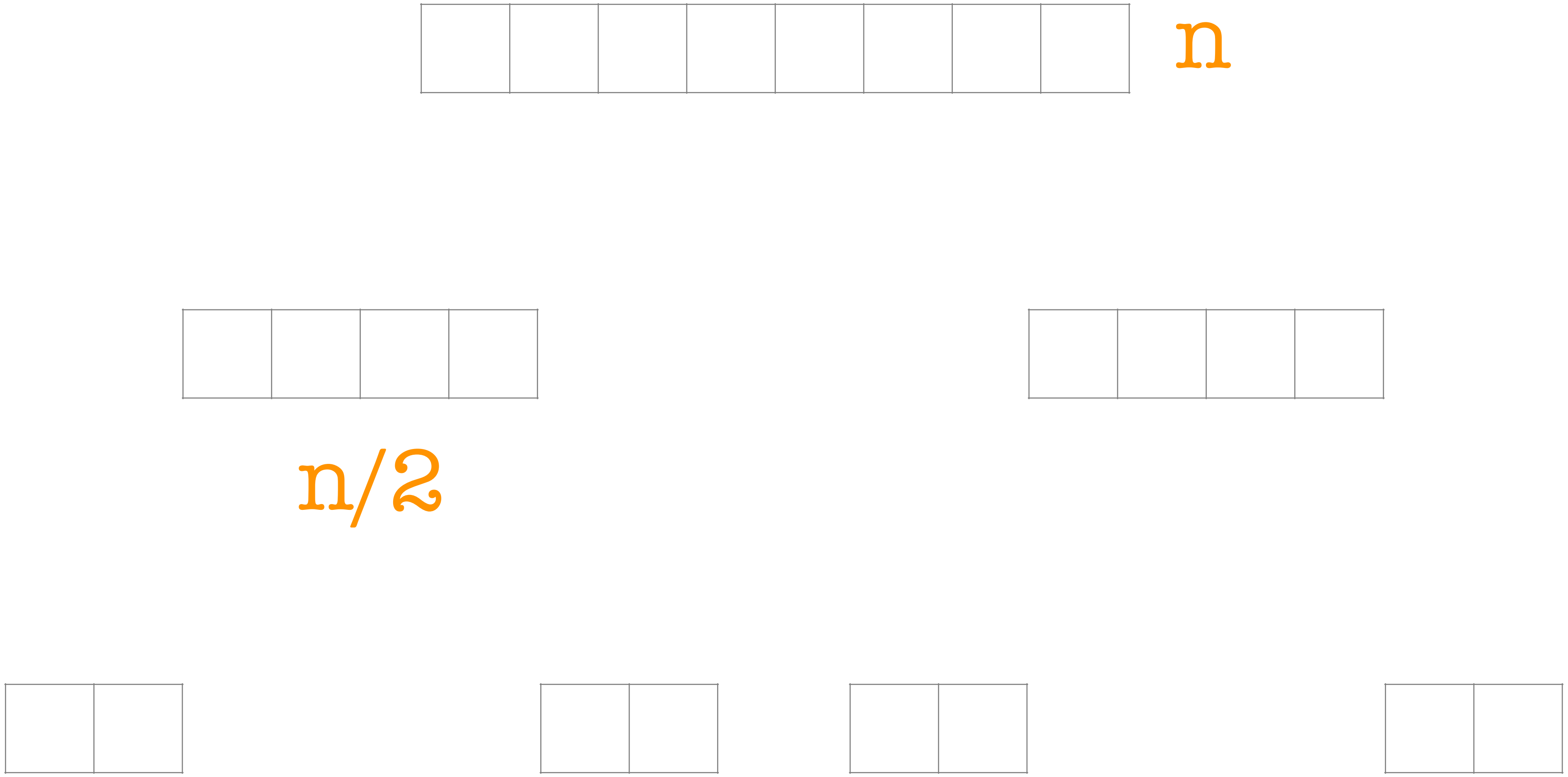
dividir para conquistar

$2T(n/2) + n$



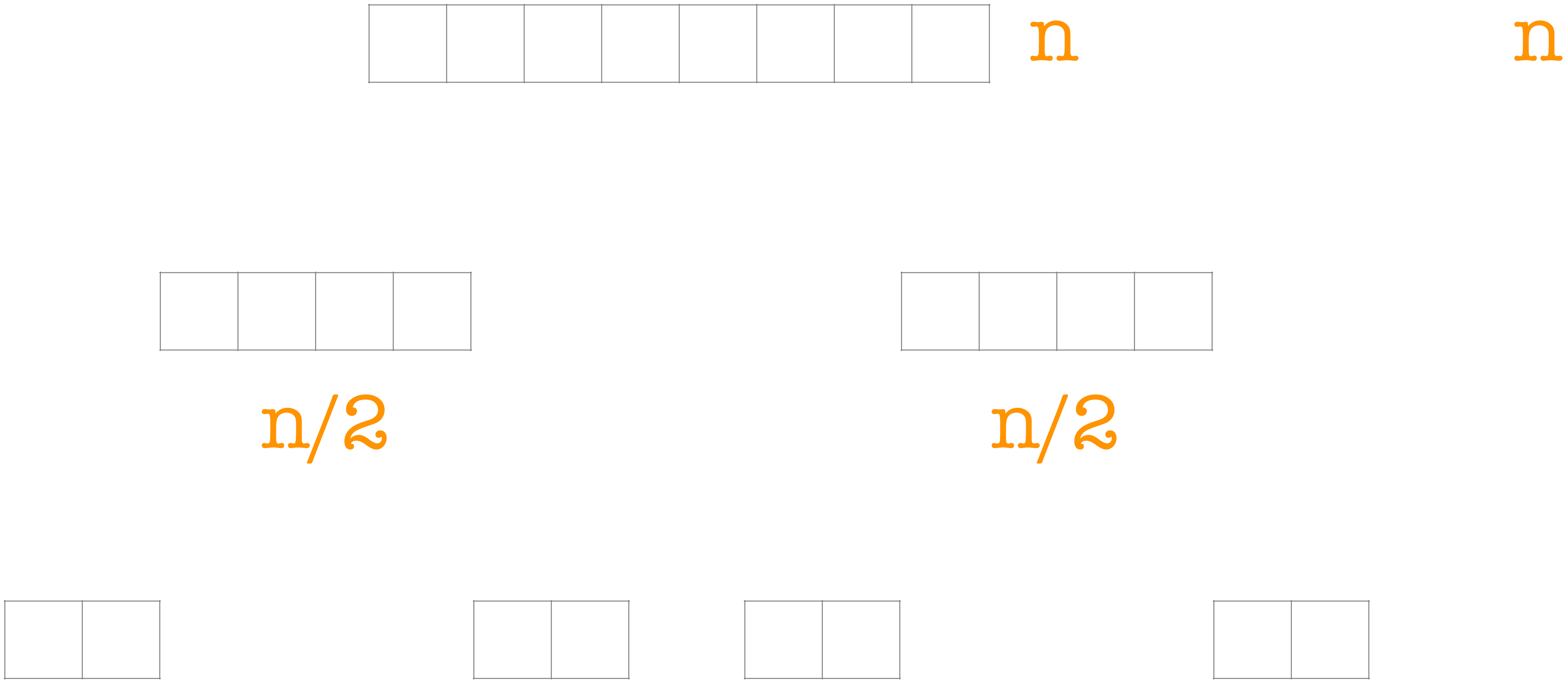
dividir para conquistar

$$2T(n/2) + n$$



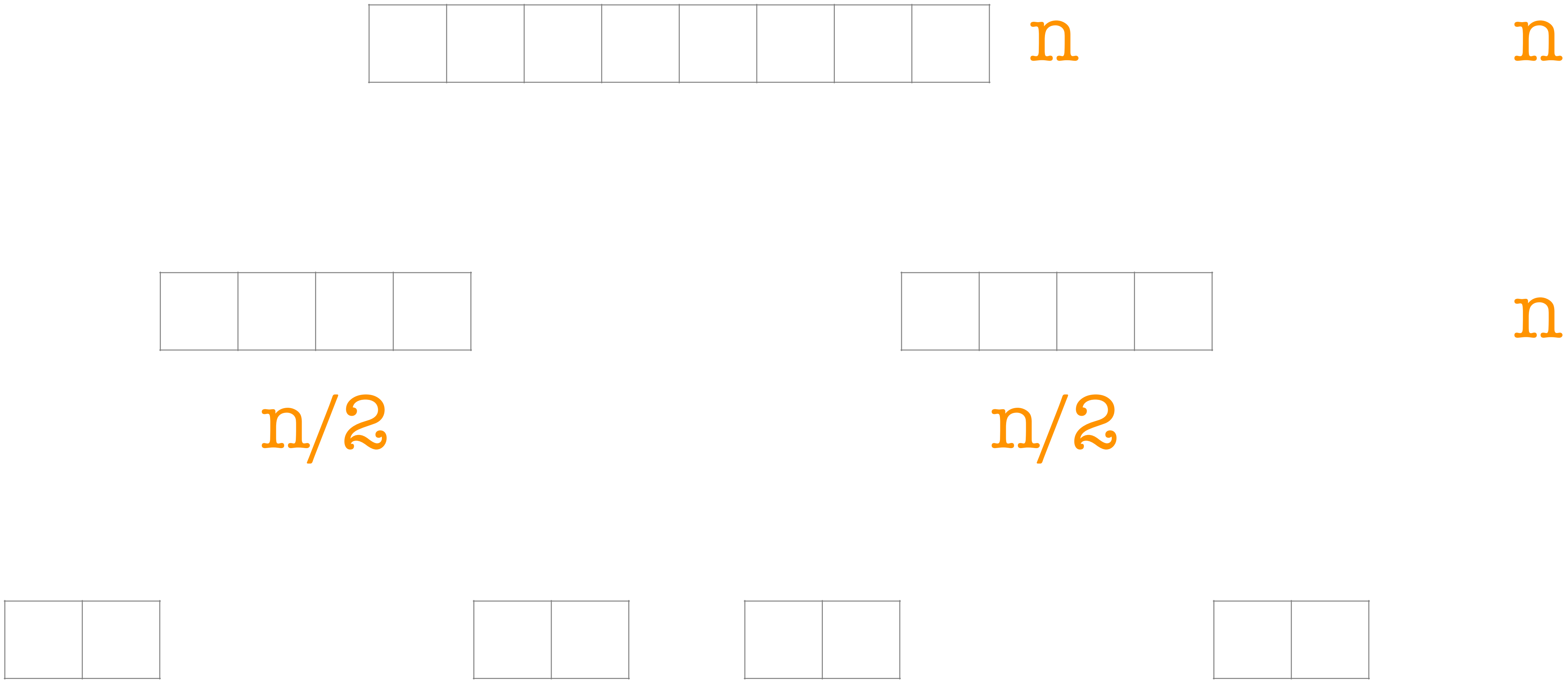
dividir para conquistar

$2T(n/2) + n$



dividir para conquistar

$2T(n/2) + n$


n

n

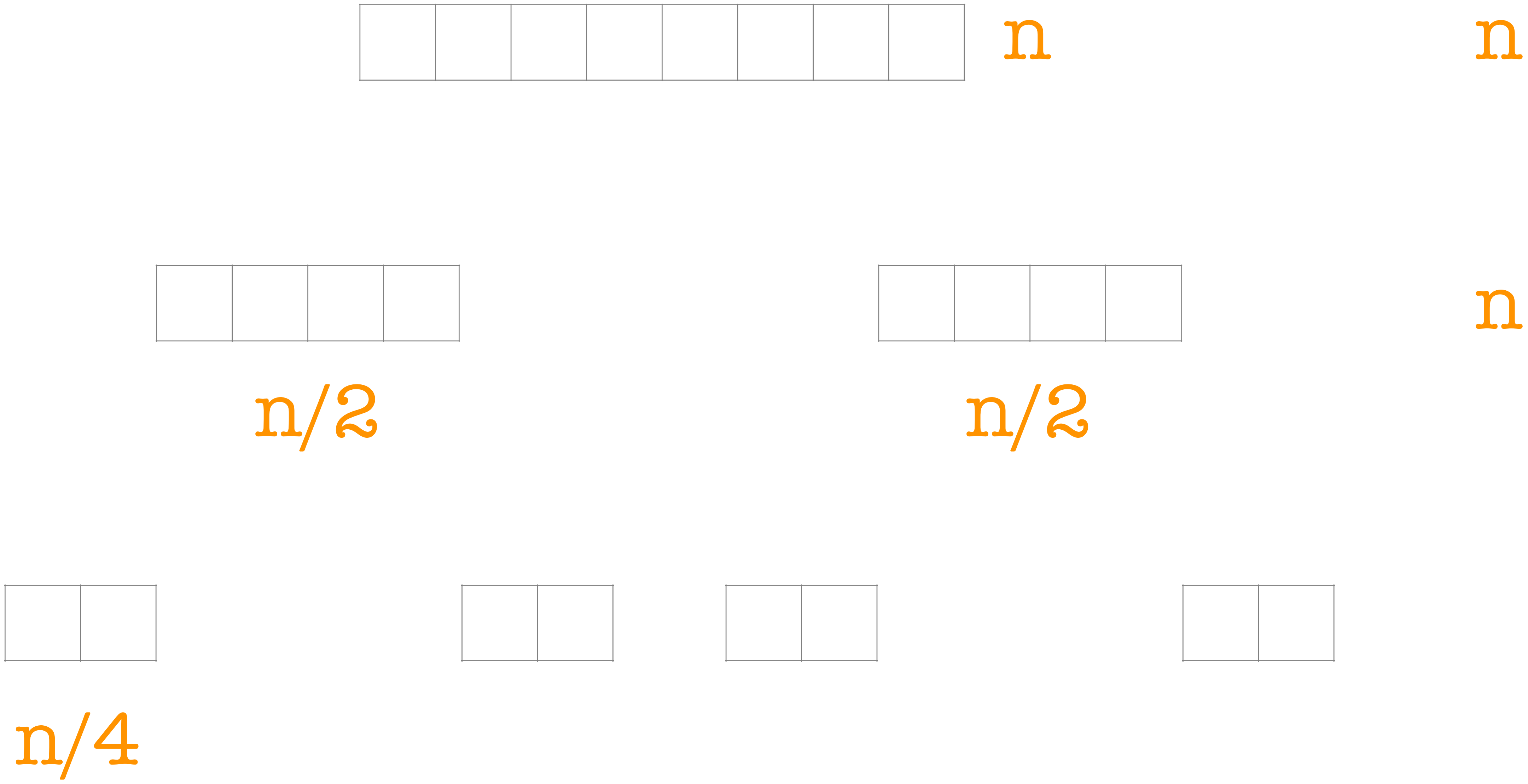
n/2

n/2

n

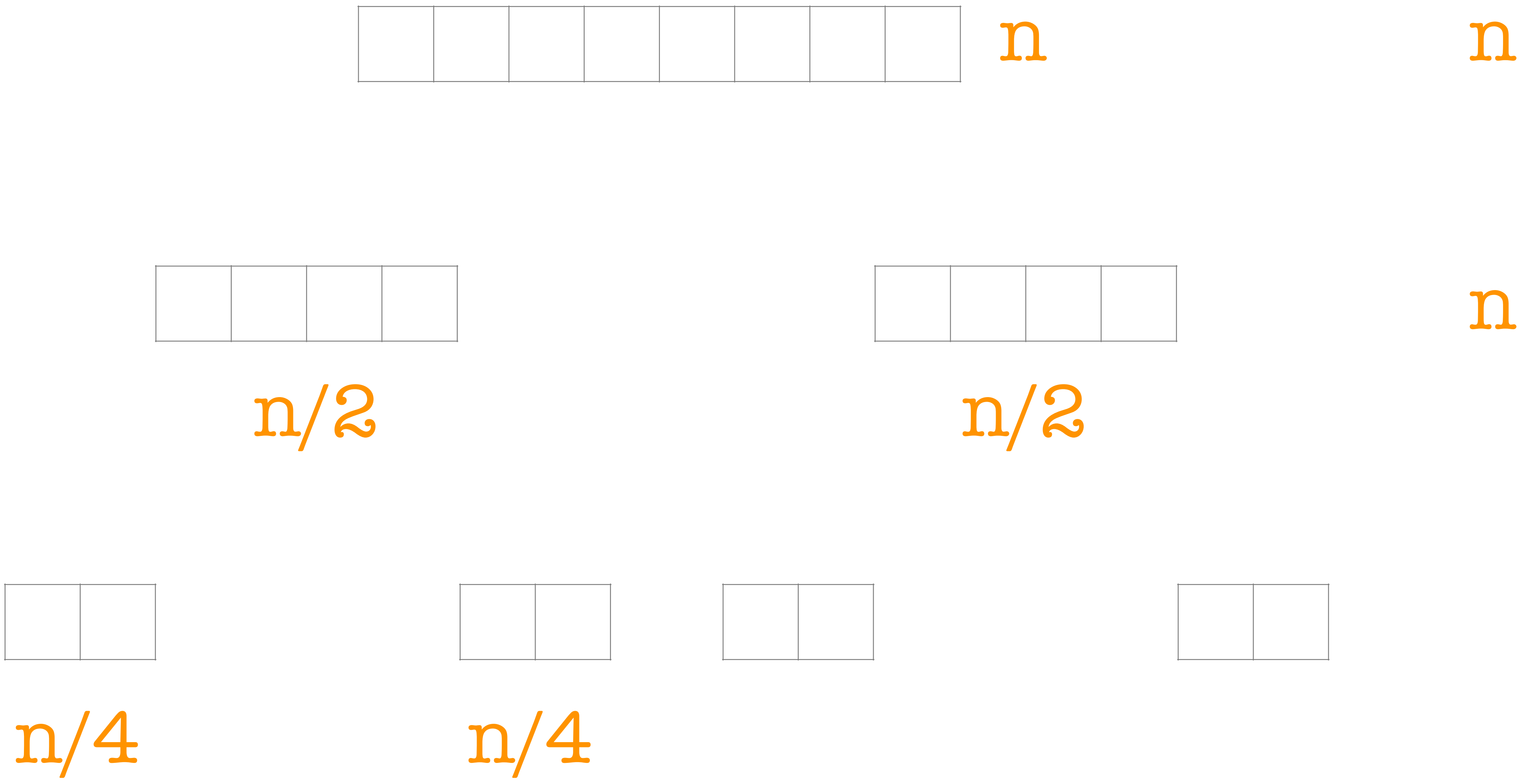
dividir para conquistar

$$2T(n/2) + n$$



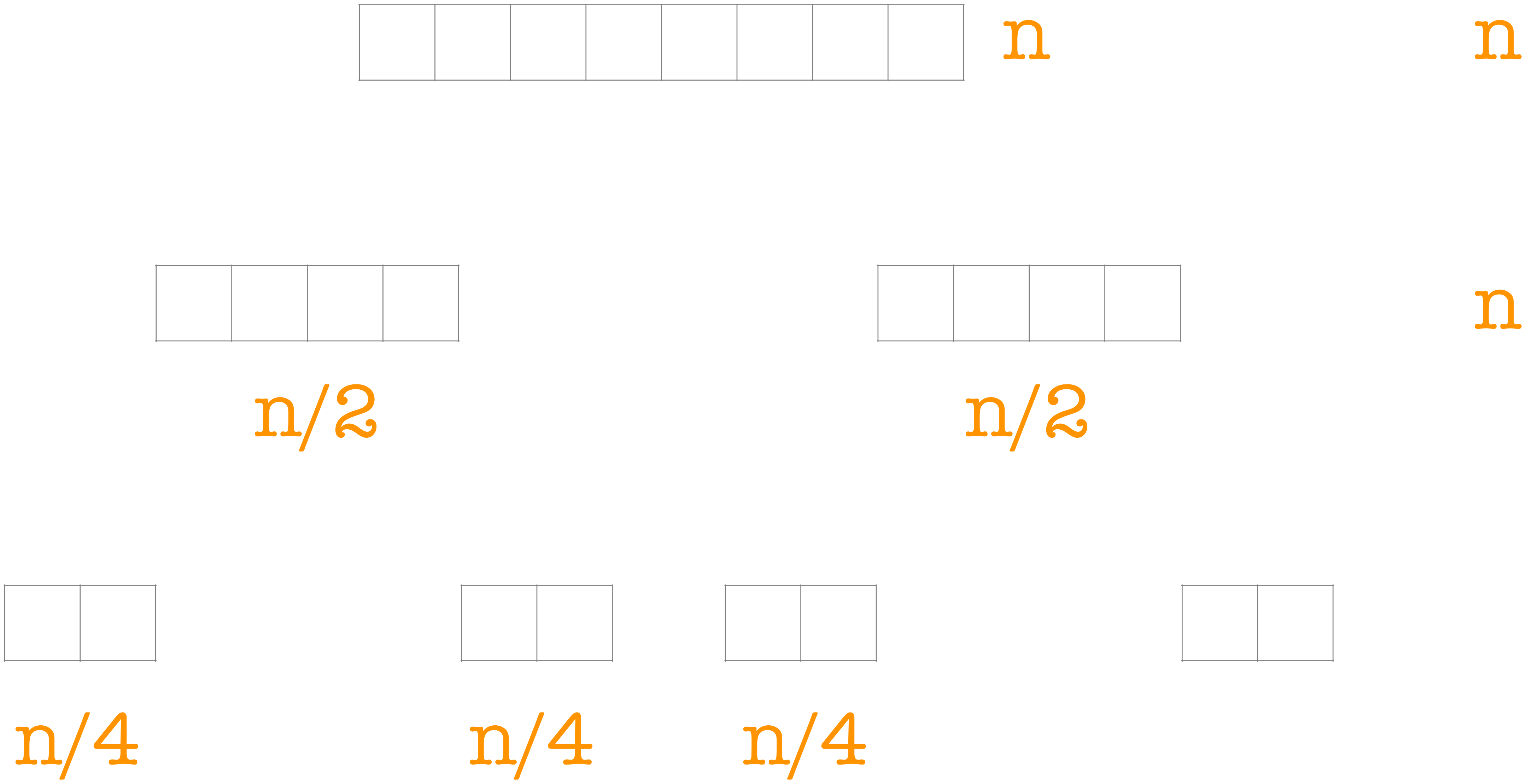
dividir para **conquistar**

$$2T(n/2) + n$$



dividir para conquistar

$2T(n/2) + n$



n

n

n

n/2

n/2

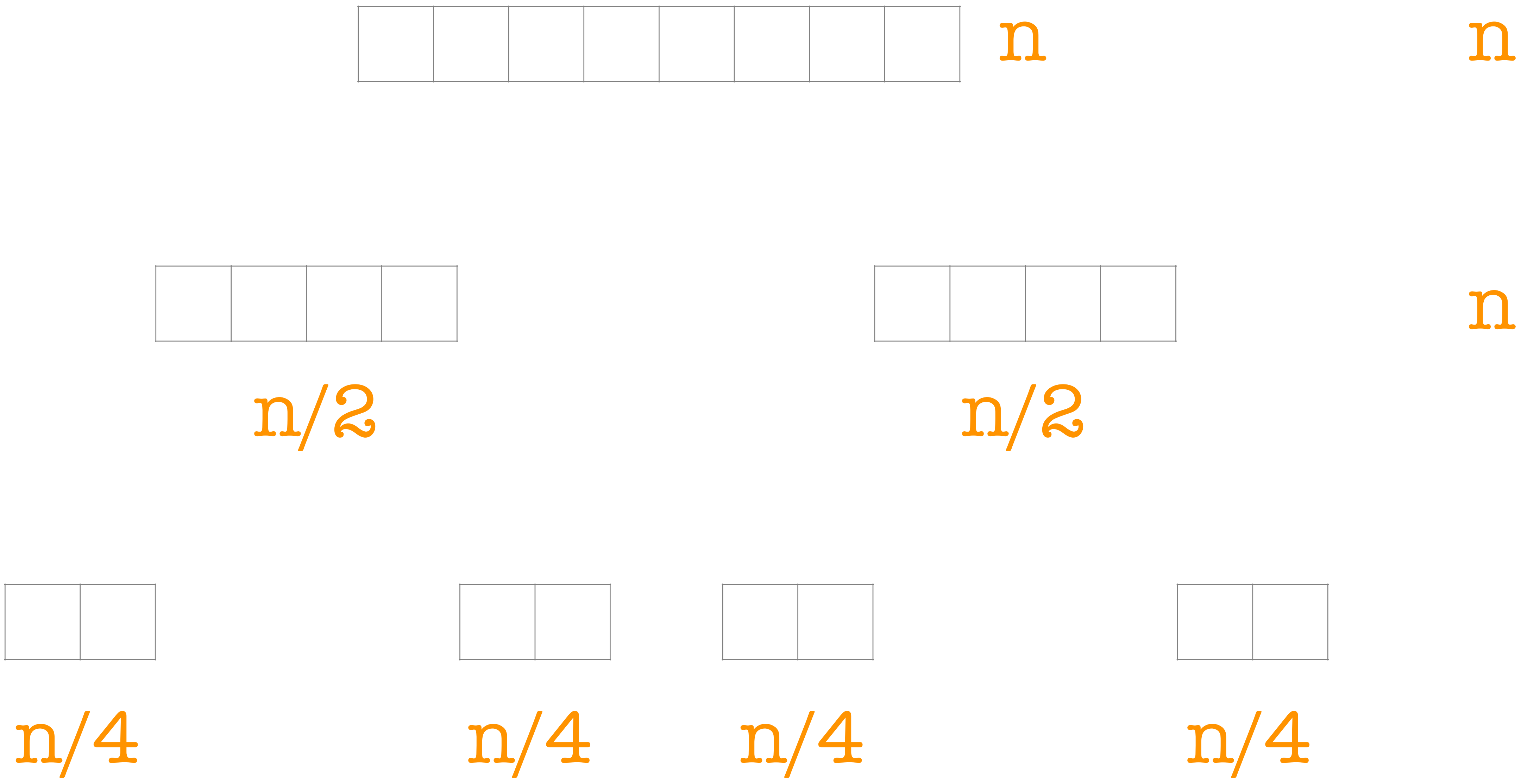
n/4

n/4

n/4

dividir para conquistar

$2T(n/2) + n$



n

n

n

n/2

n/2

n/4

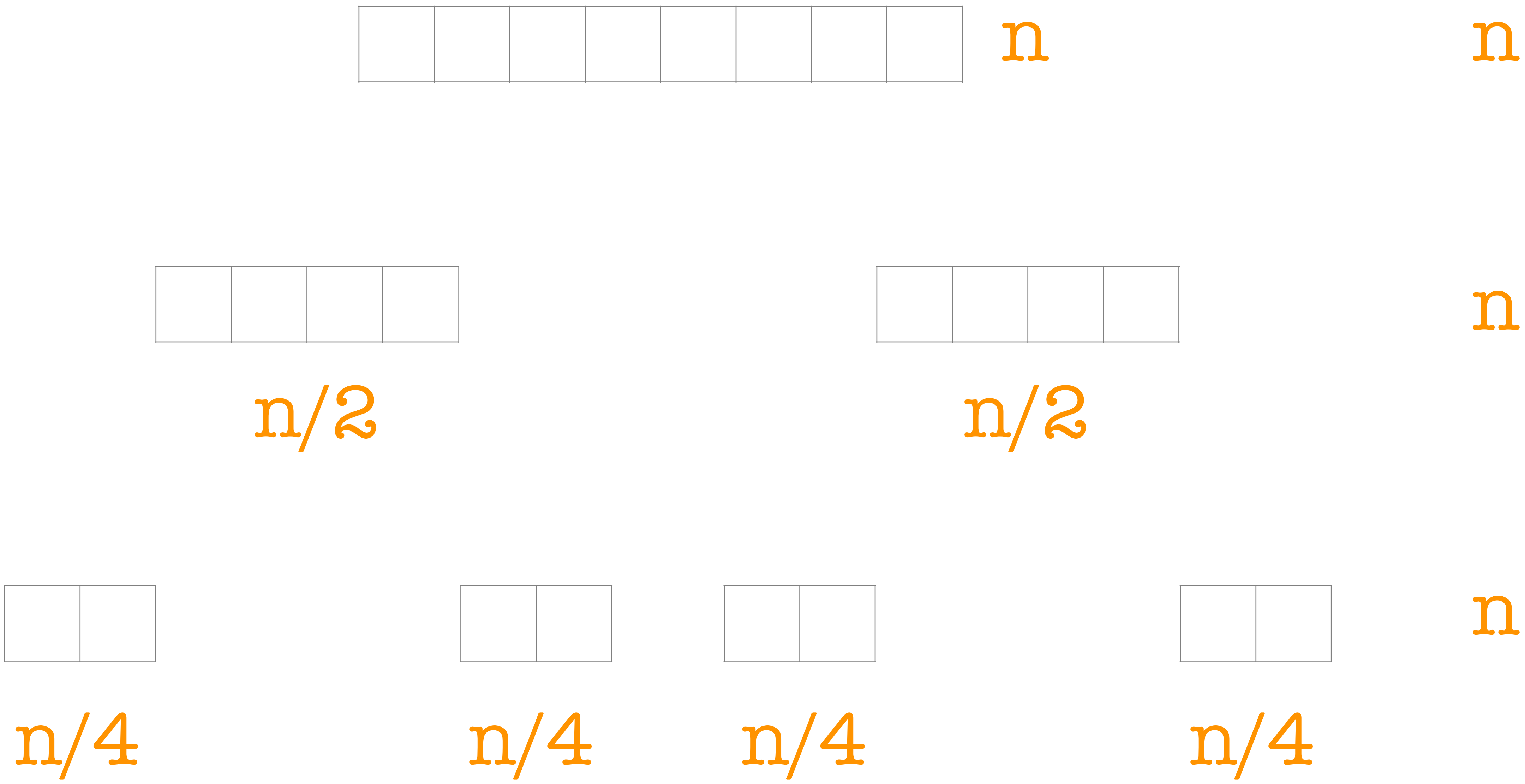
n/4

n/4

n/4

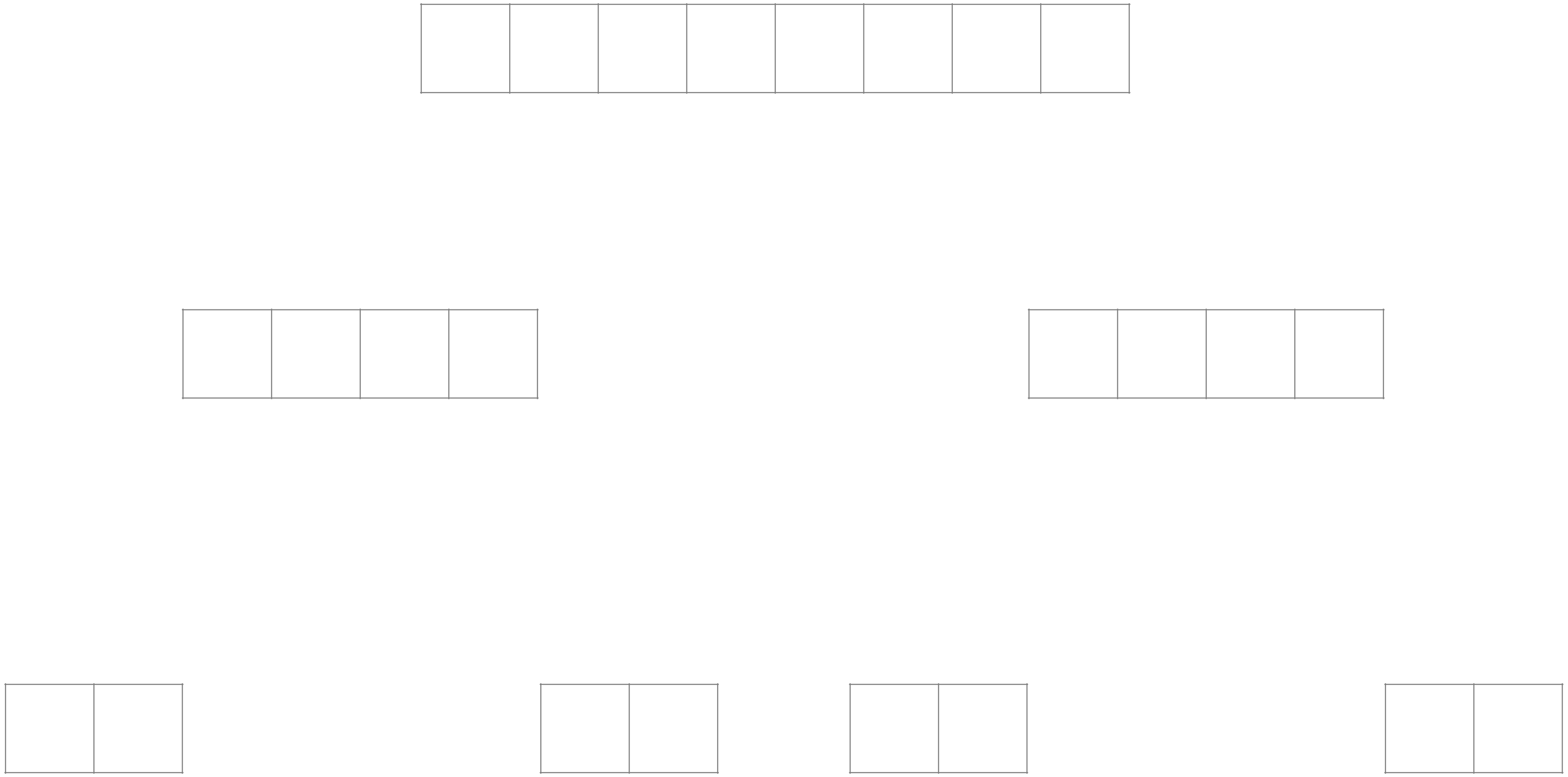
dividir para conquistar

$2T(n/2) + n$



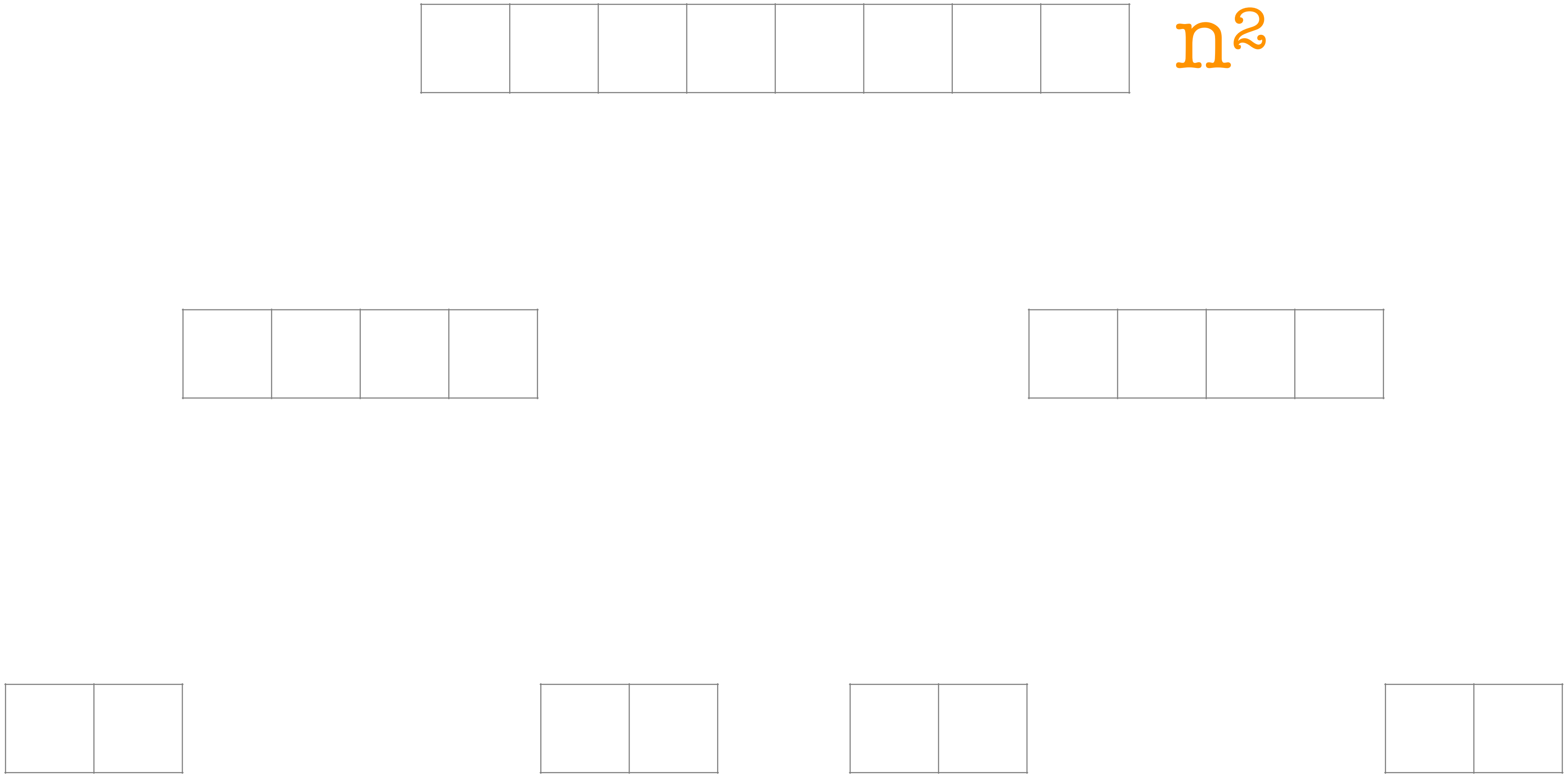
dividir para conquistar

$$2T(n/2) + n^2$$



dividir para conquistar

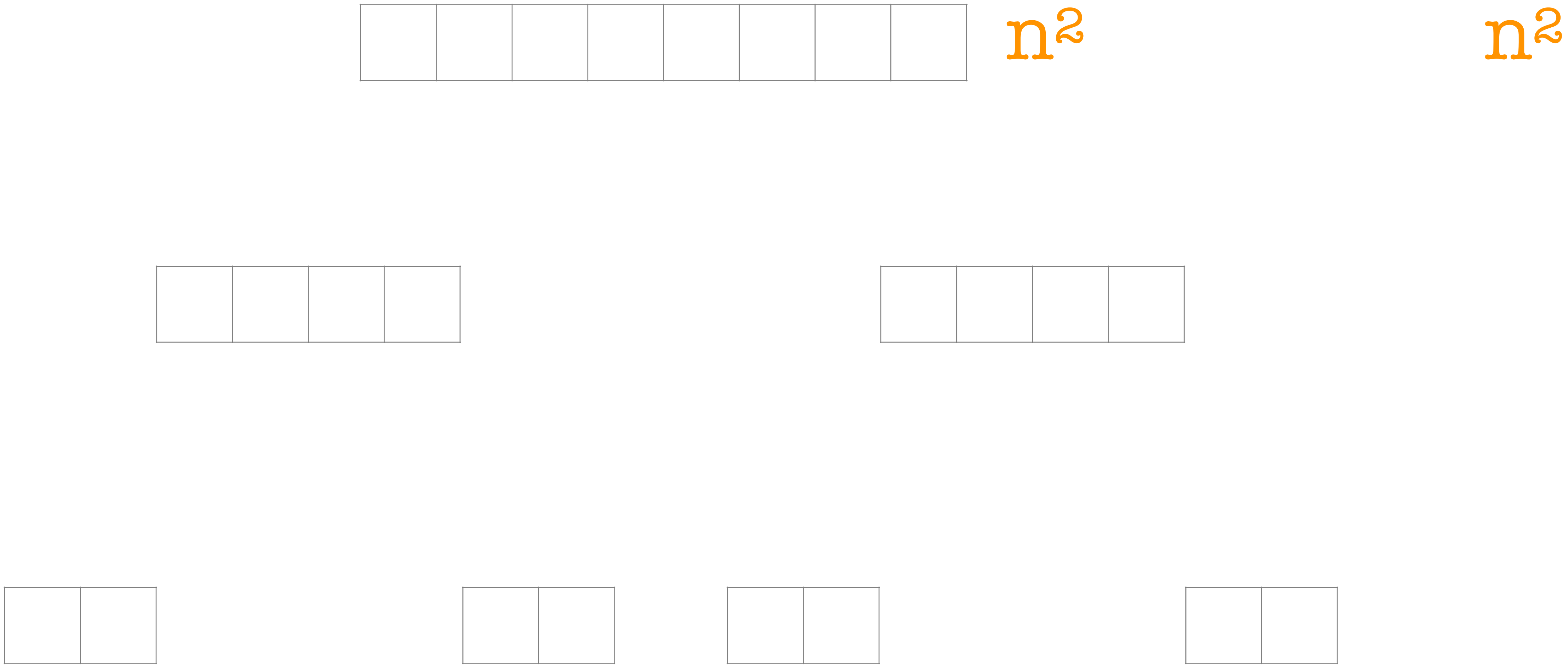
$2T(n/2) + n^2$



n^2

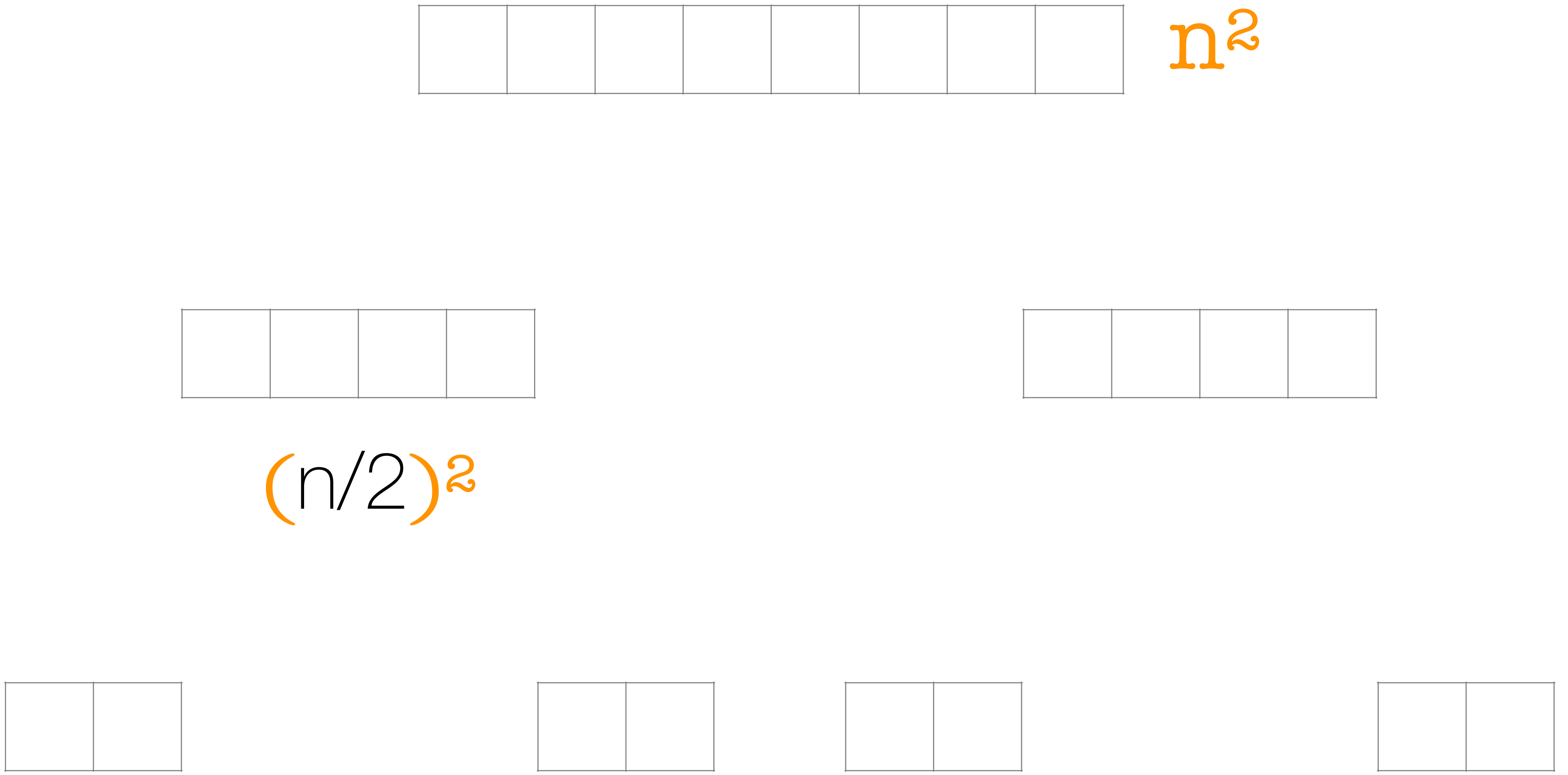
dividir para conquistar

$2T(n/2) + n^2$



dividir para conquistar

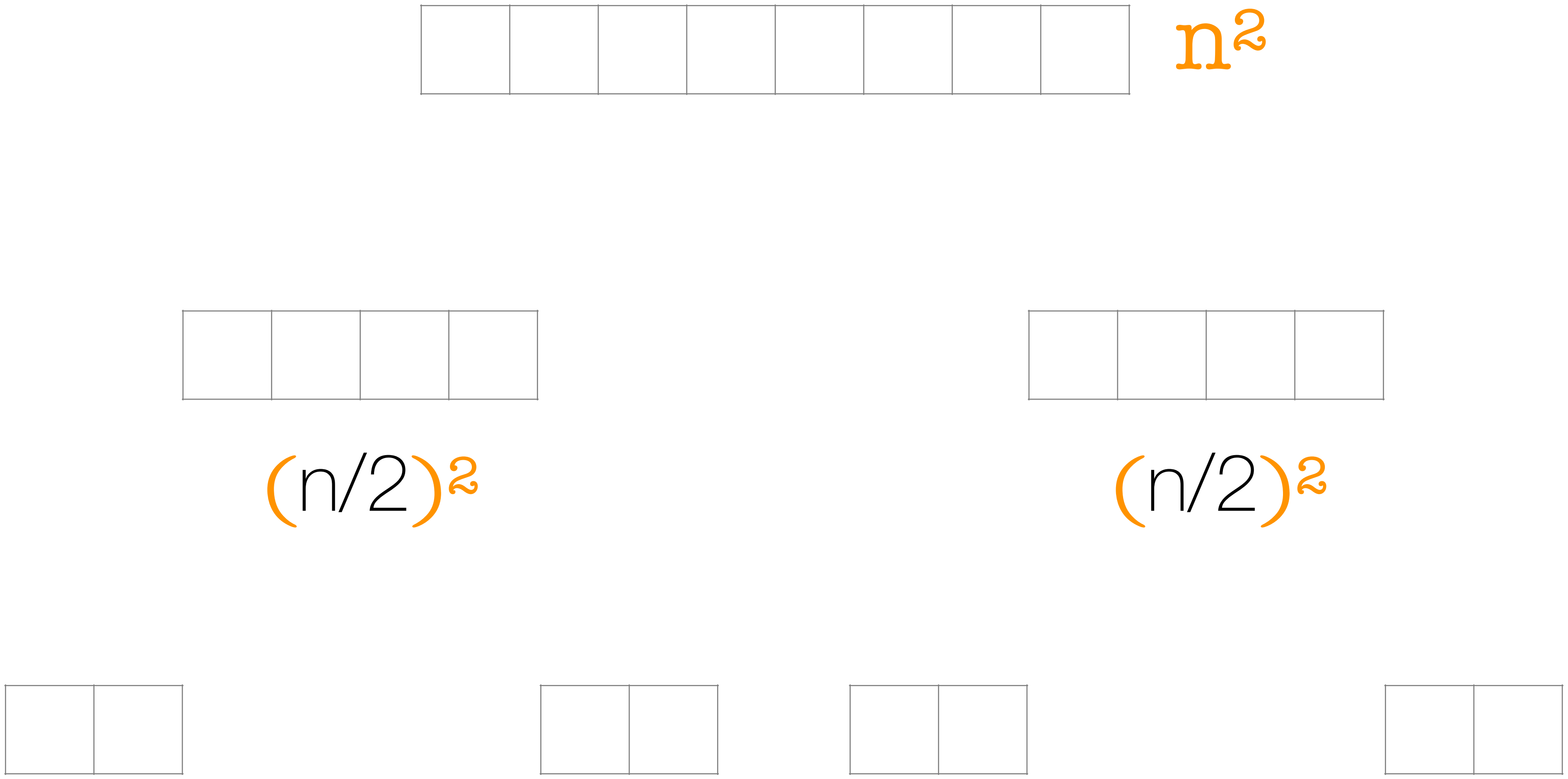
$2T(n/2) + n^2$



n^2

dividir para conquistar

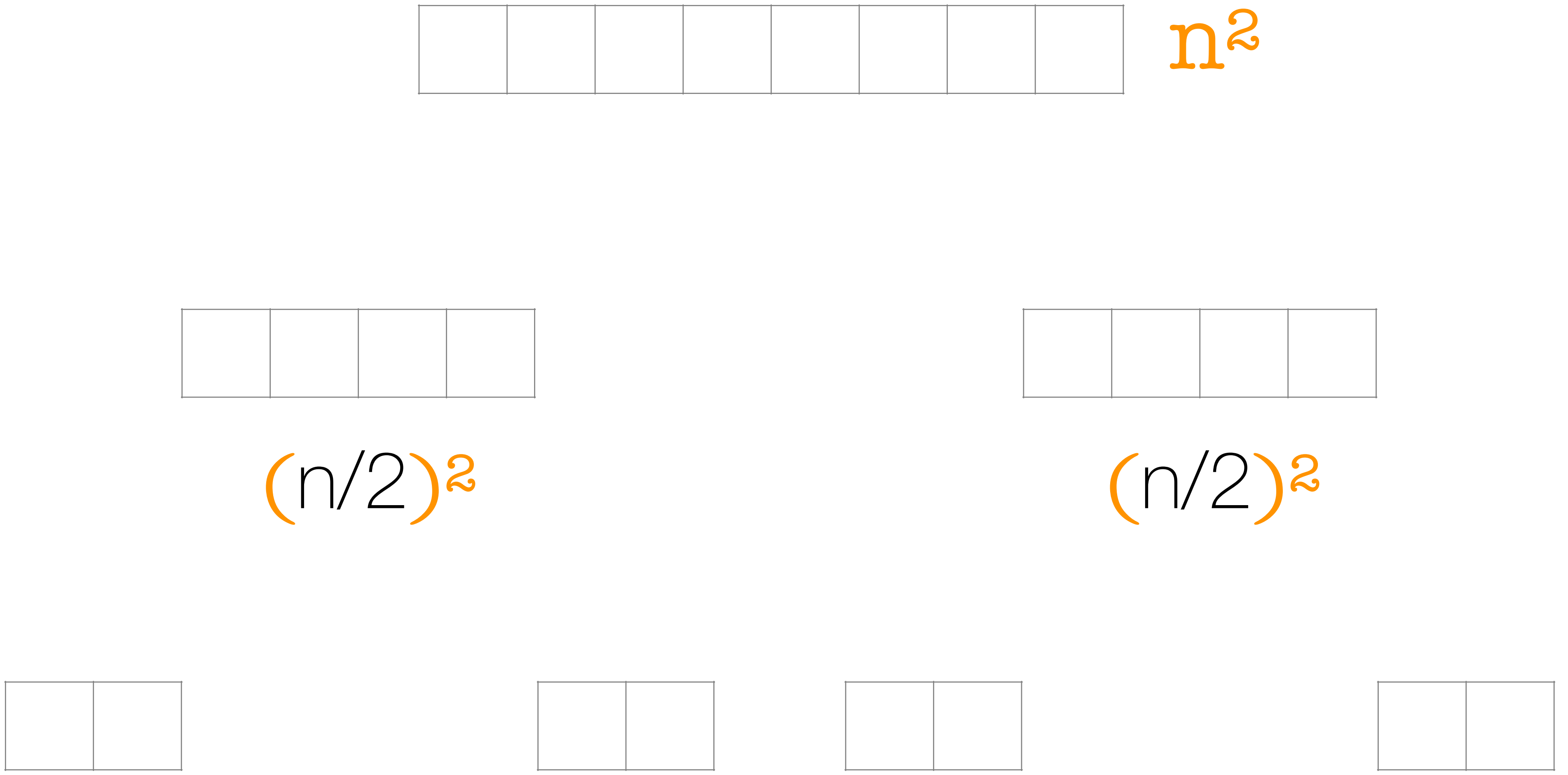
$2T(n/2) + n^2$



n^2

dividir para conquistar

$$2T(n/2) + n^2$$



n^2

$(n/2)^2$

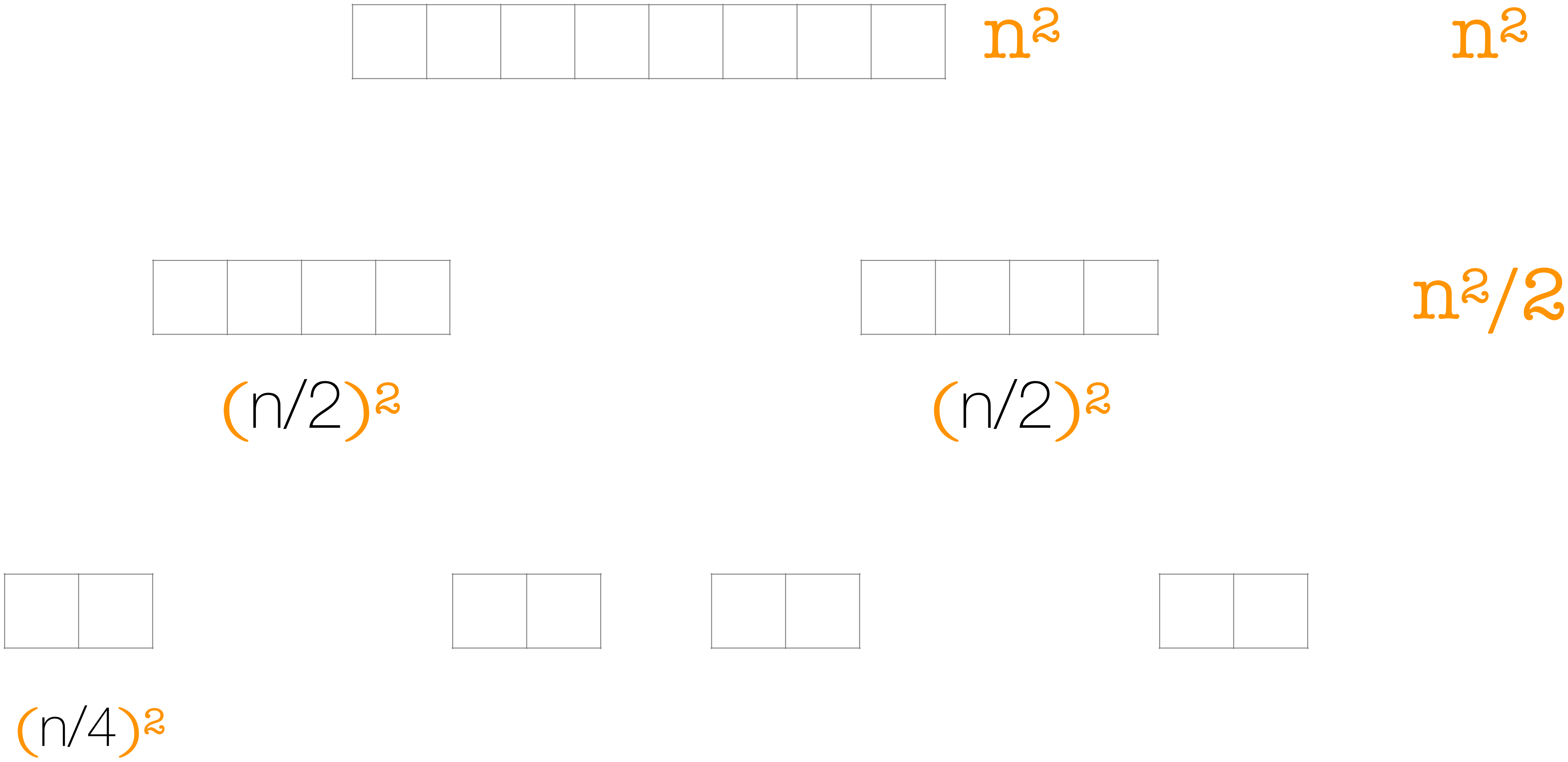
$(n/2)^2$

n^2

$n^2/2$

dividir para conquistar

$$2T(n/2) + n^2$$



n^2

n^2

$n^2/2$

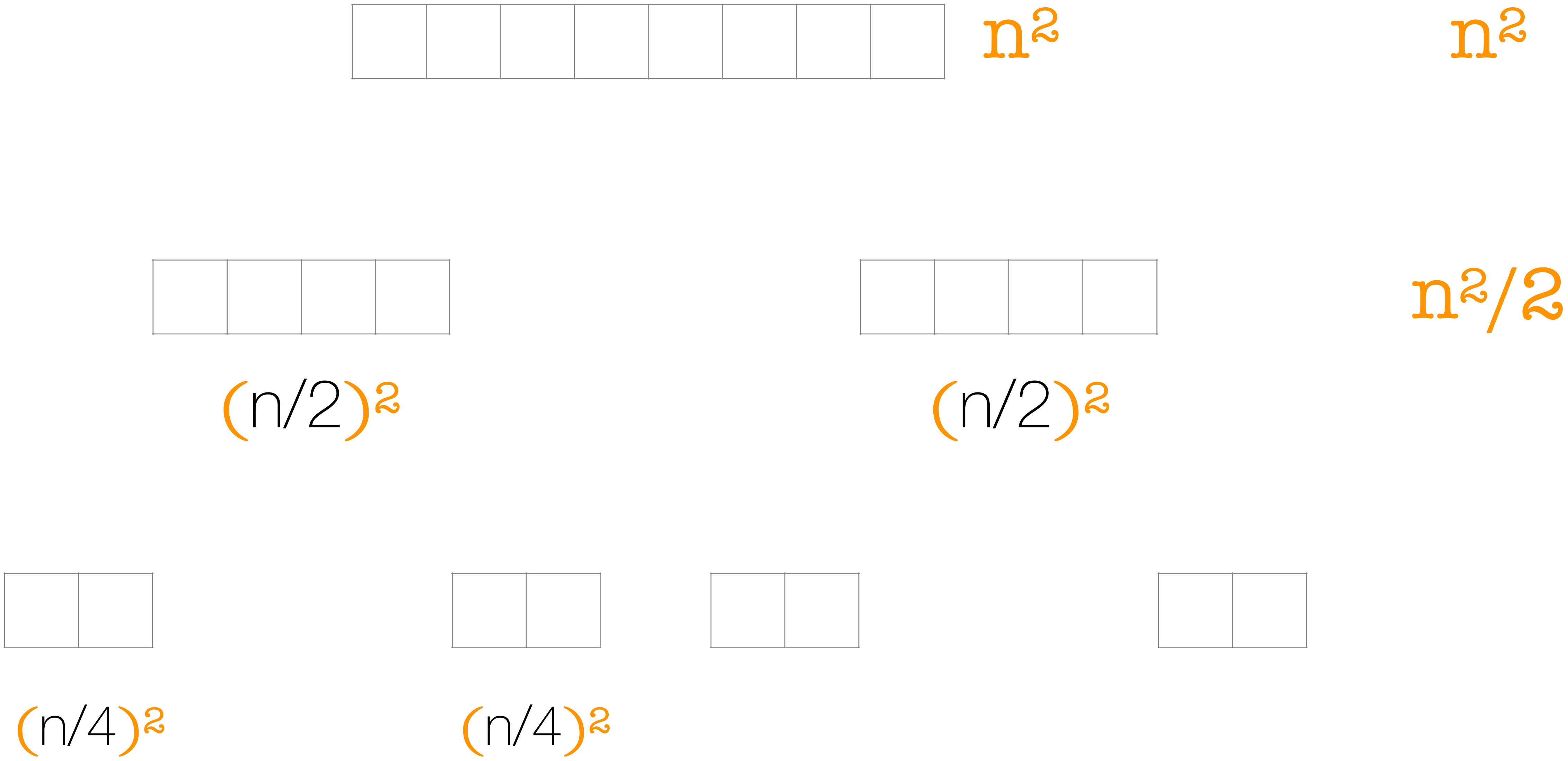
$(n/2)^2$

$(n/2)^2$

$(n/4)^2$

dividir para conquistar

$$2T(n/2) + n^2$$



n^2

n^2

$n^2/2$

$(n/2)^2$

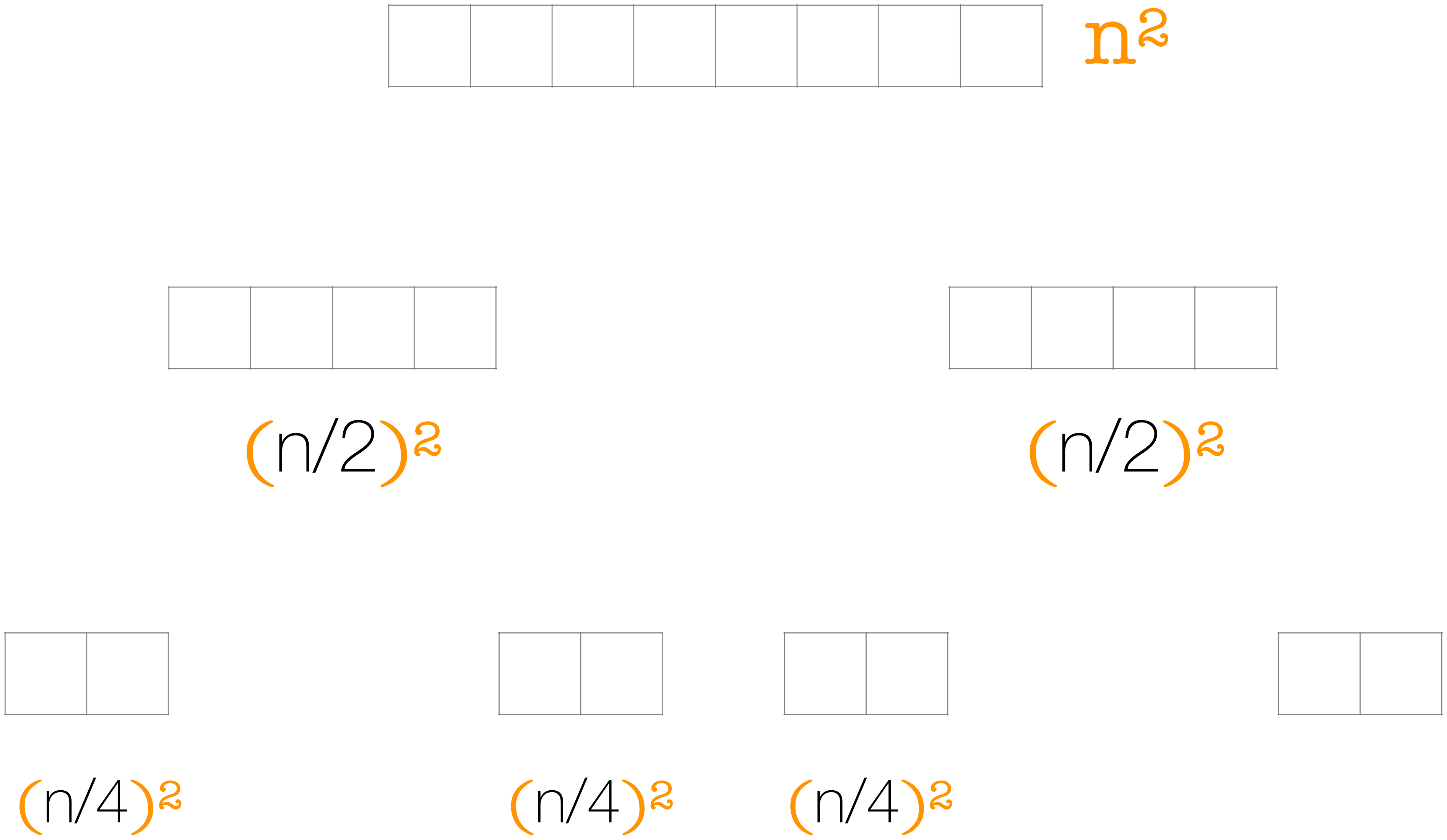
$(n/2)^2$

$(n/4)^2$

$(n/4)^2$

dividir para conquistar

$$2T(n/2) + n^2$$

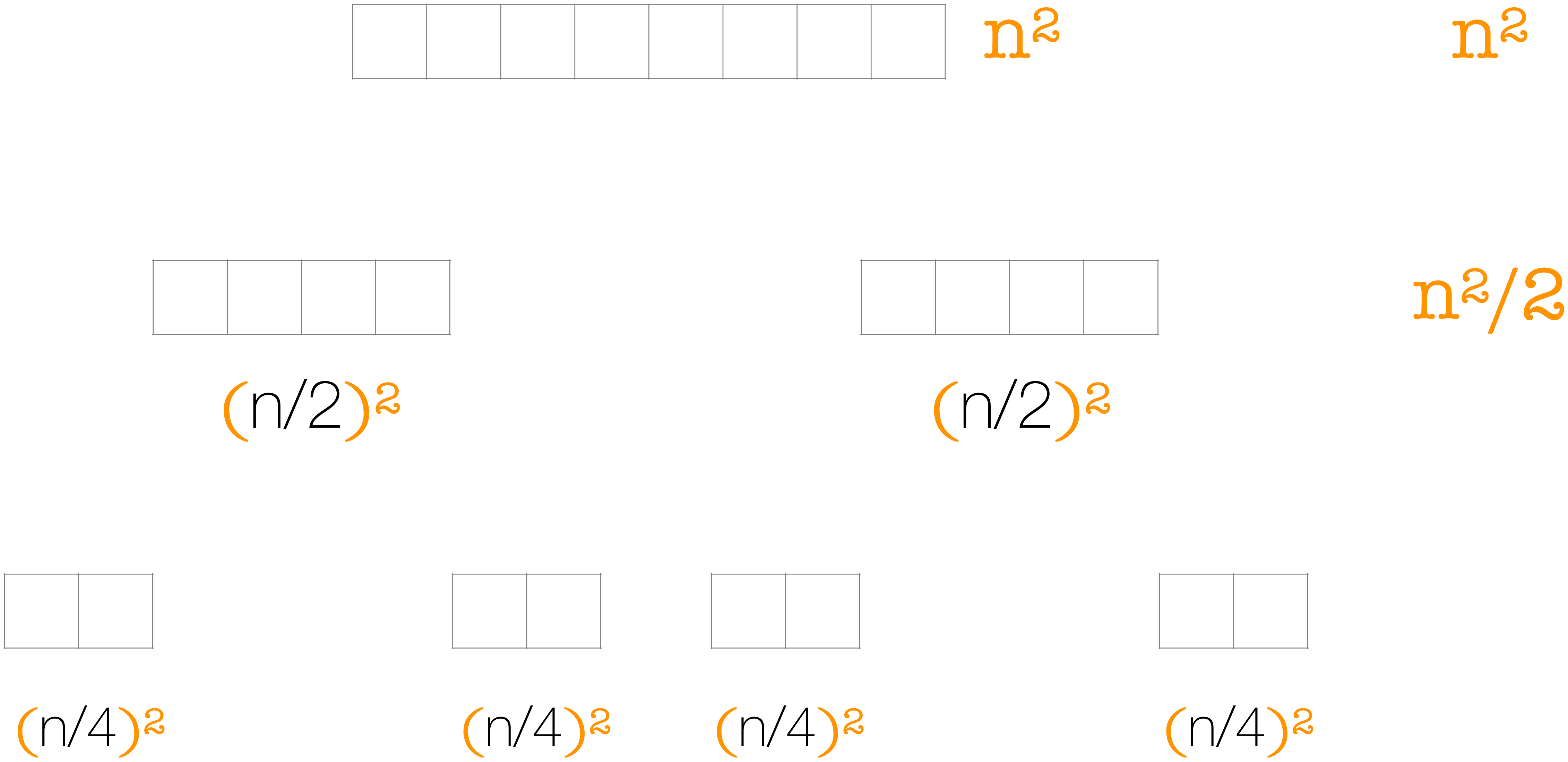


n^2

$n^2/2$

dividir para conquistar

$2T(n/2) + n^2$



n^2 n^2

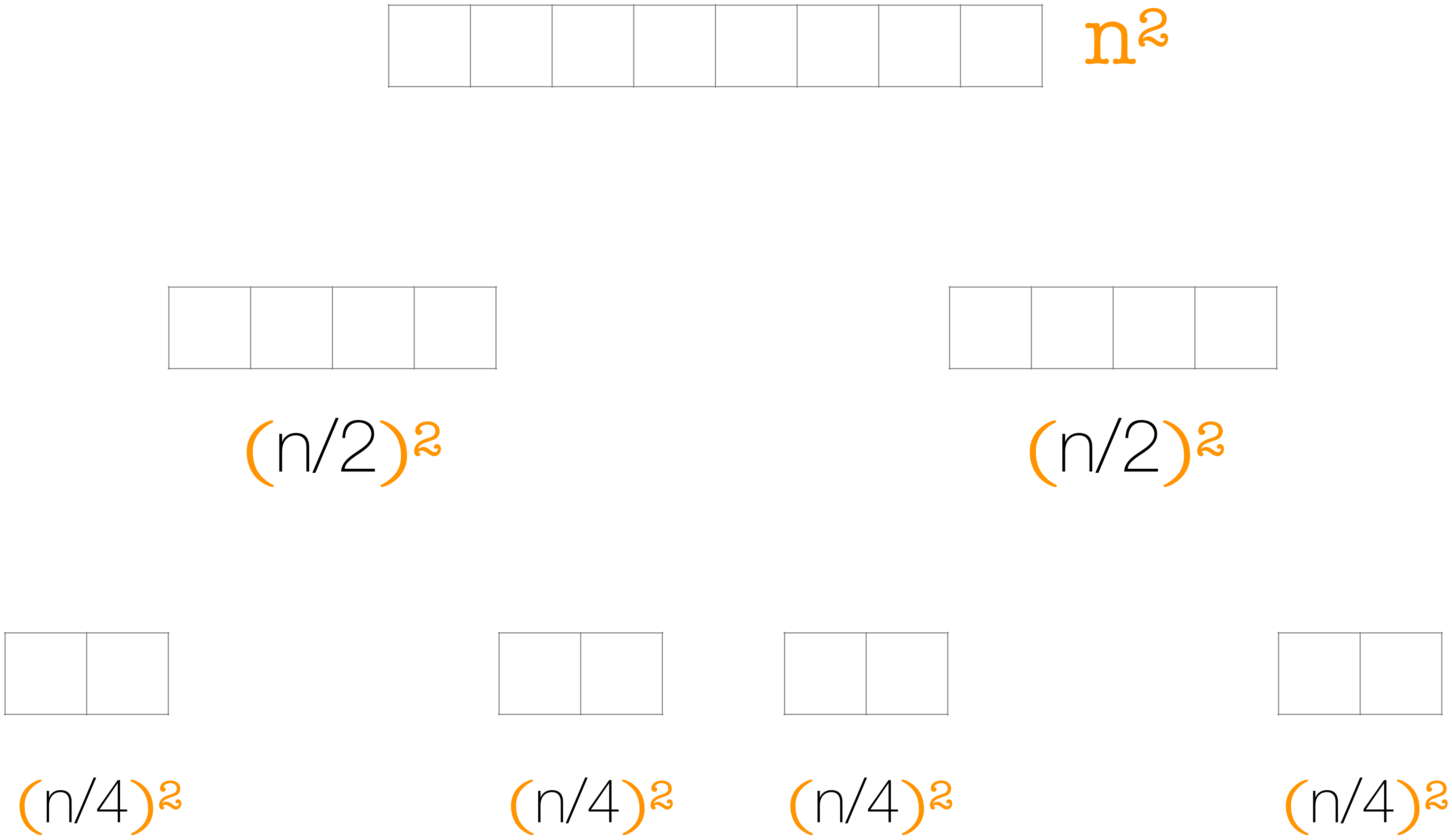
$n^2/2$

$(n/2)^2$ $(n/2)^2$

$(n/4)^2$ $(n/4)^2$ $(n/4)^2$ $(n/4)^2$

dividir para conquistar

$2T(n/2) + n^2$



n^2

$(n/2)^2$ $(n/2)^2$

$(n/4)^2$ $(n/4)^2$ $(n/4)^2$ $(n/4)^2$

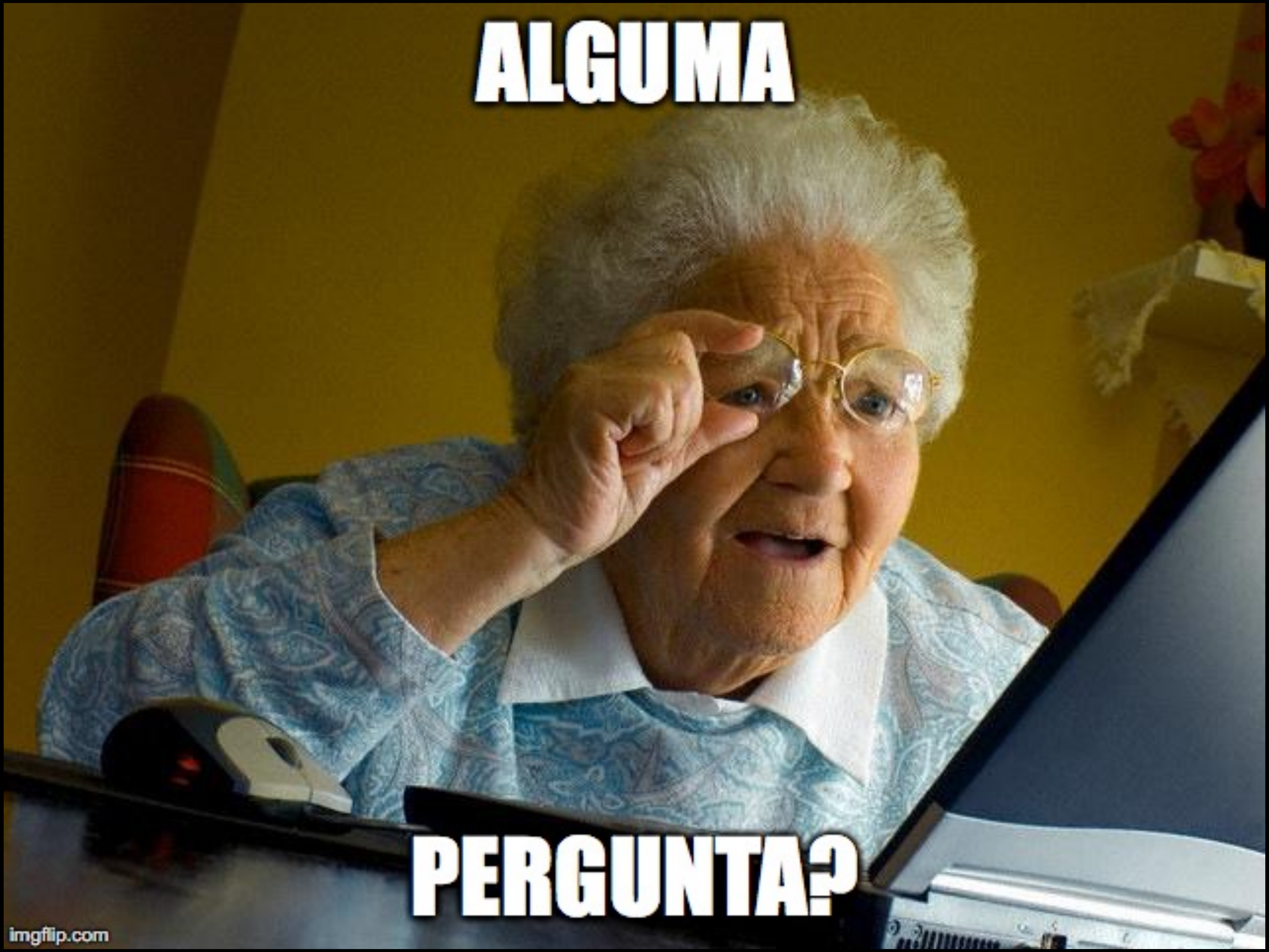
n^2

$n^2/2$

$n^2/4$

dividir para conquistar

$$T(n) = a\underbrace{T(n/b)}_{\text{dividir}} + \overbrace{f(n)}^{\text{conquistar}}$$


ALGUMA
PERGUNTA?
imgflip.com

dividir para conquistar	intersecções entre subproblemas
recursão de cauda	demais casos

sequência de **fibonacci**

sequência de **fibonacci**

- 1 1 2 3 5 8 13 21 34 ...

sequência de **fibonacci**

- 1 1 2 3 5 8 13 21 34 ...

- fib(n) = fib(n-1) + fib(n-2)

sequência de fibonacci

- 1 1 2 3 5 8 13 21 34 ...
- fib(n) = fib(n-1) + fib(n-2)
- fib(0) = 0, fib(1) = 1

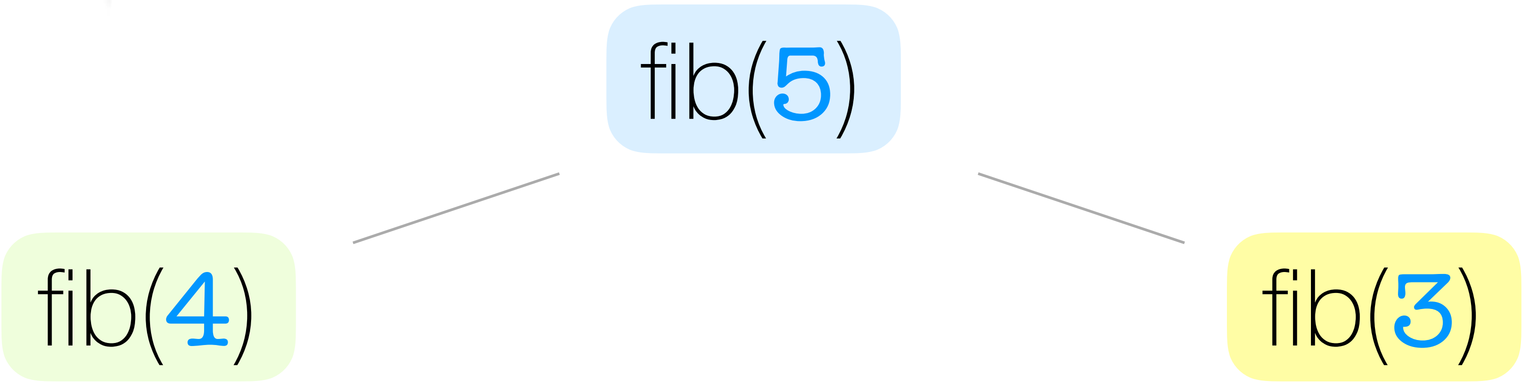
sequência de fibonacci

```
int fib_rec(int n)
{
   if (n <= 1) return n;
   return fib(n-1) + fib(n-2);
}
```

- 1 1 2 3 5 8 13 21 34 ...
- fib(n) = fib(n-1) + fib(n-2)
- fib(0) = 0, fib(1) = 1

intersecções entre subproblemas

intersecções entre subproblemas

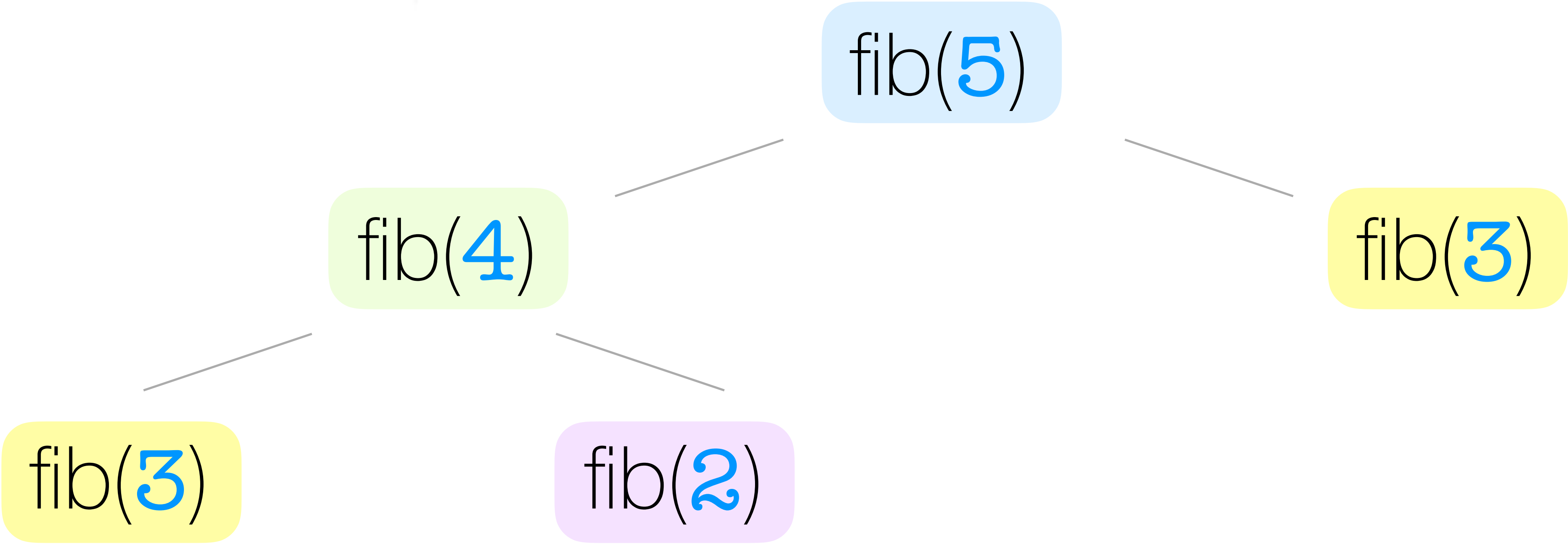


fib(5)

fib(4)

fib(3)

intersecções entre subproblemas



fib(5)

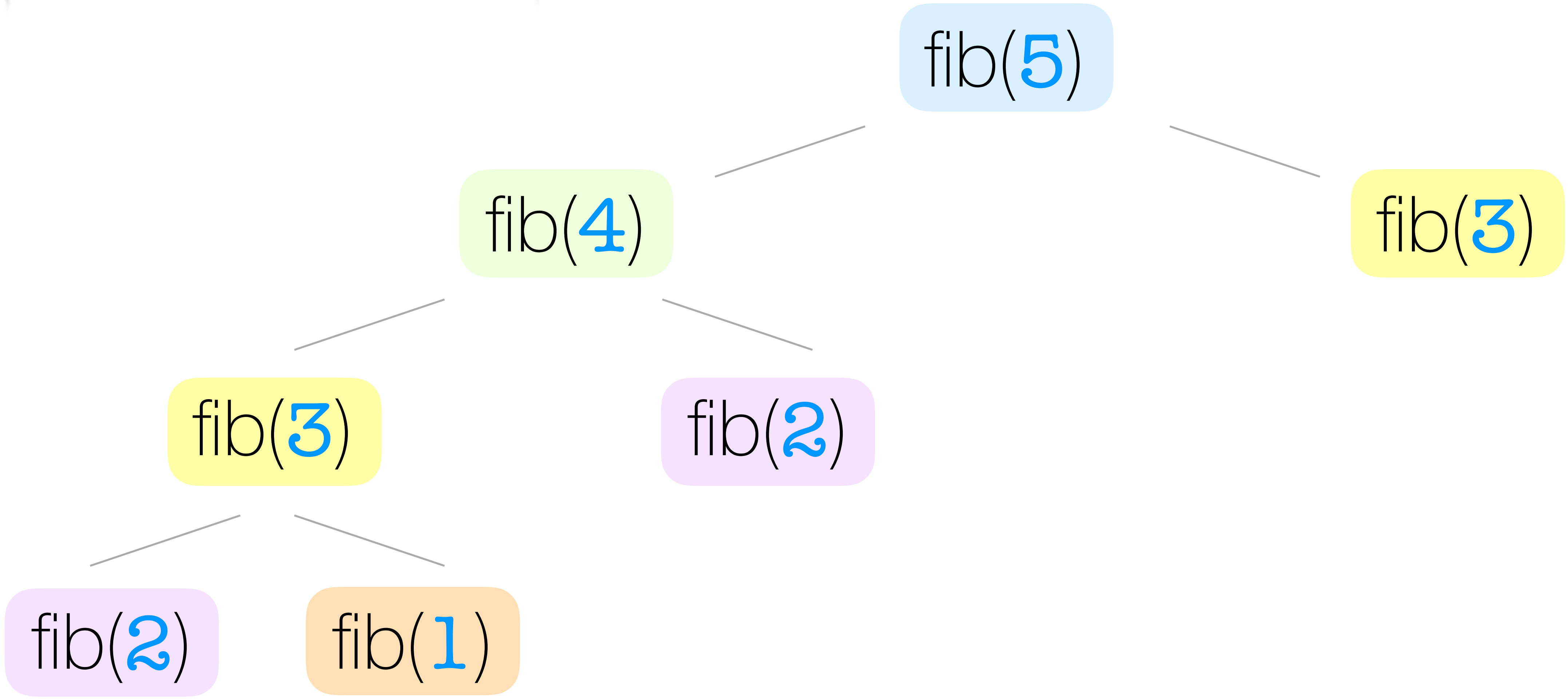
fib(4)

fib(3)

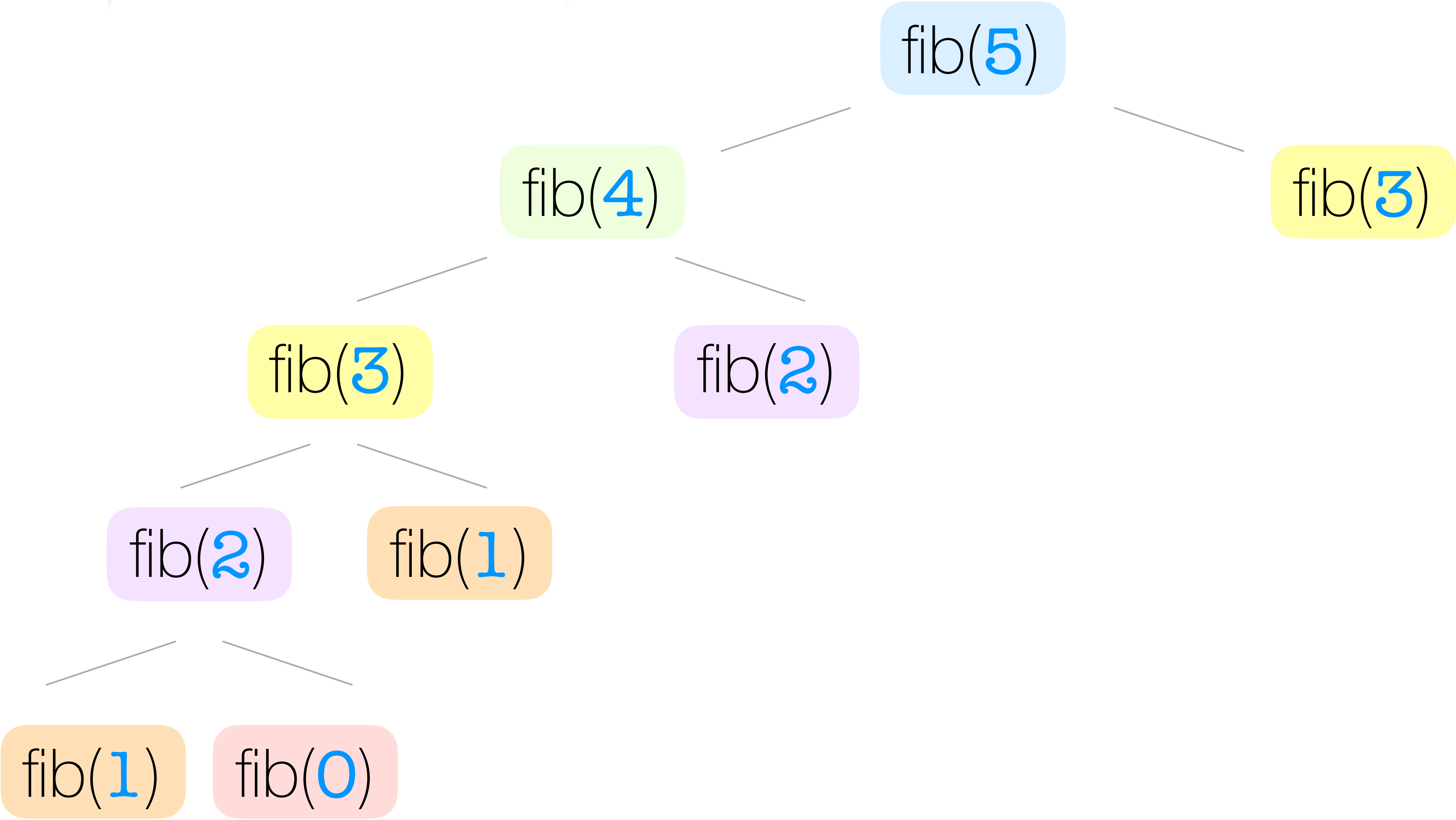
fib(3)

fib(2)

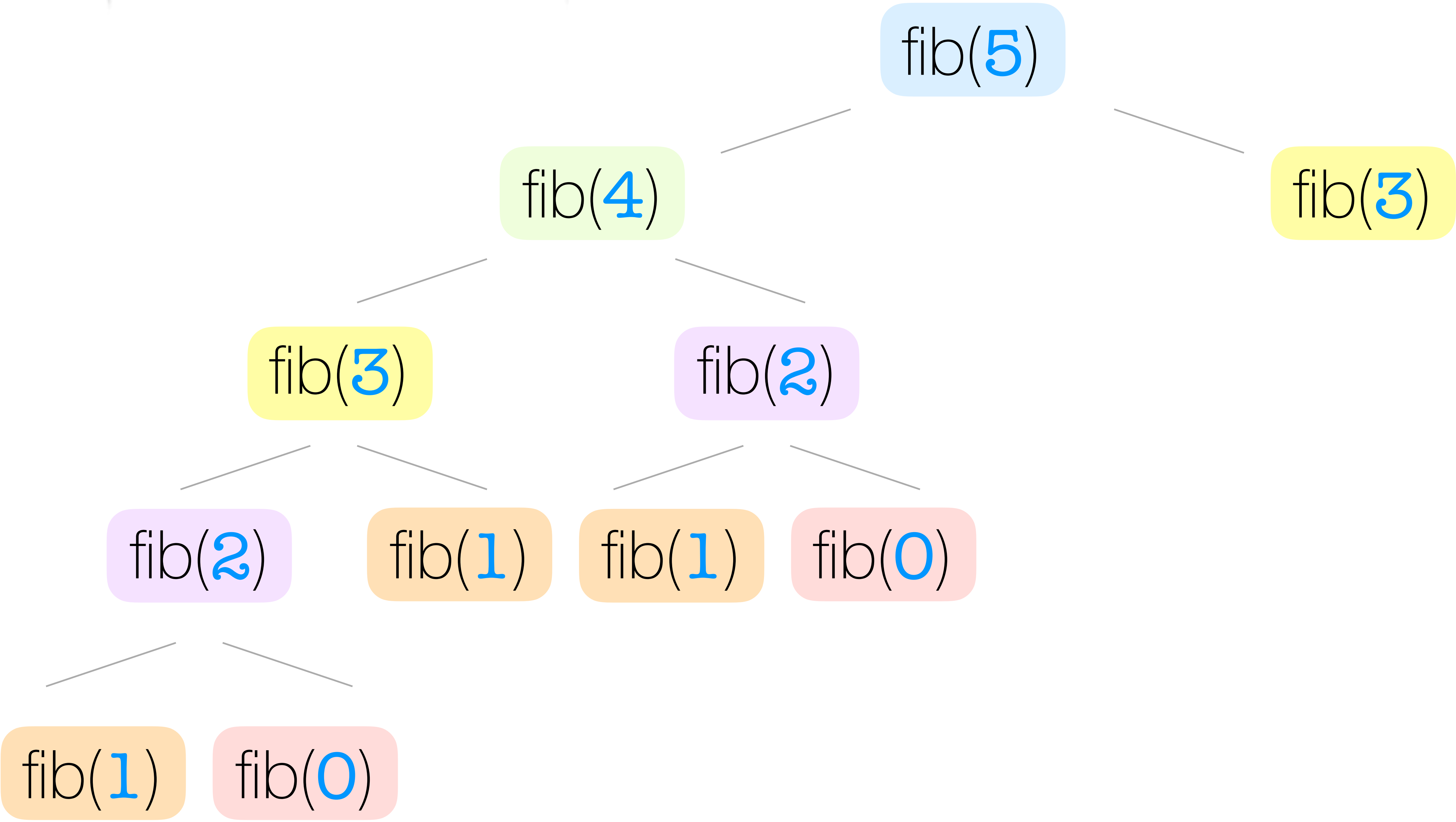
intersecções entre subproblemas



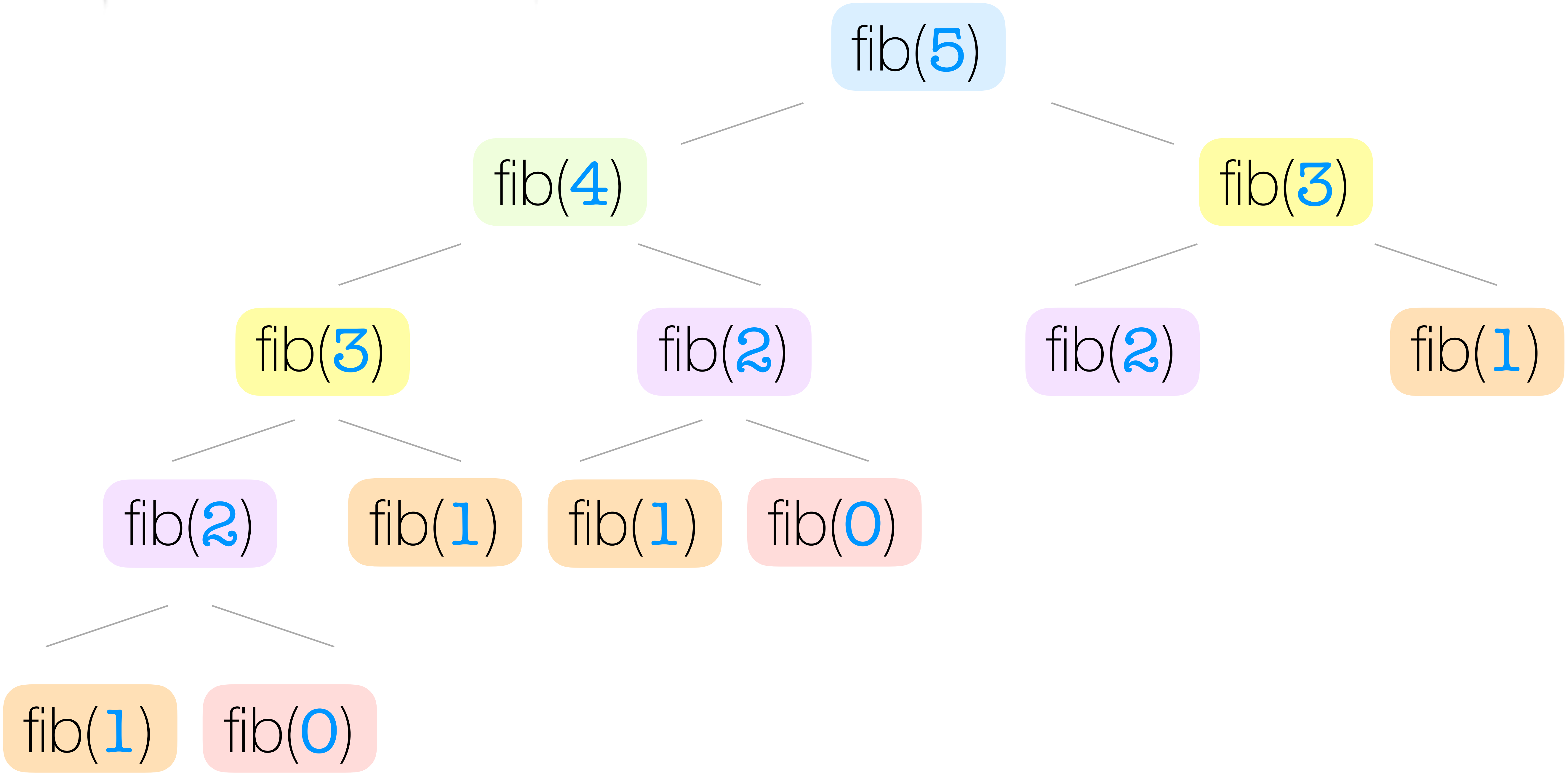
intersecções entre subproblemas



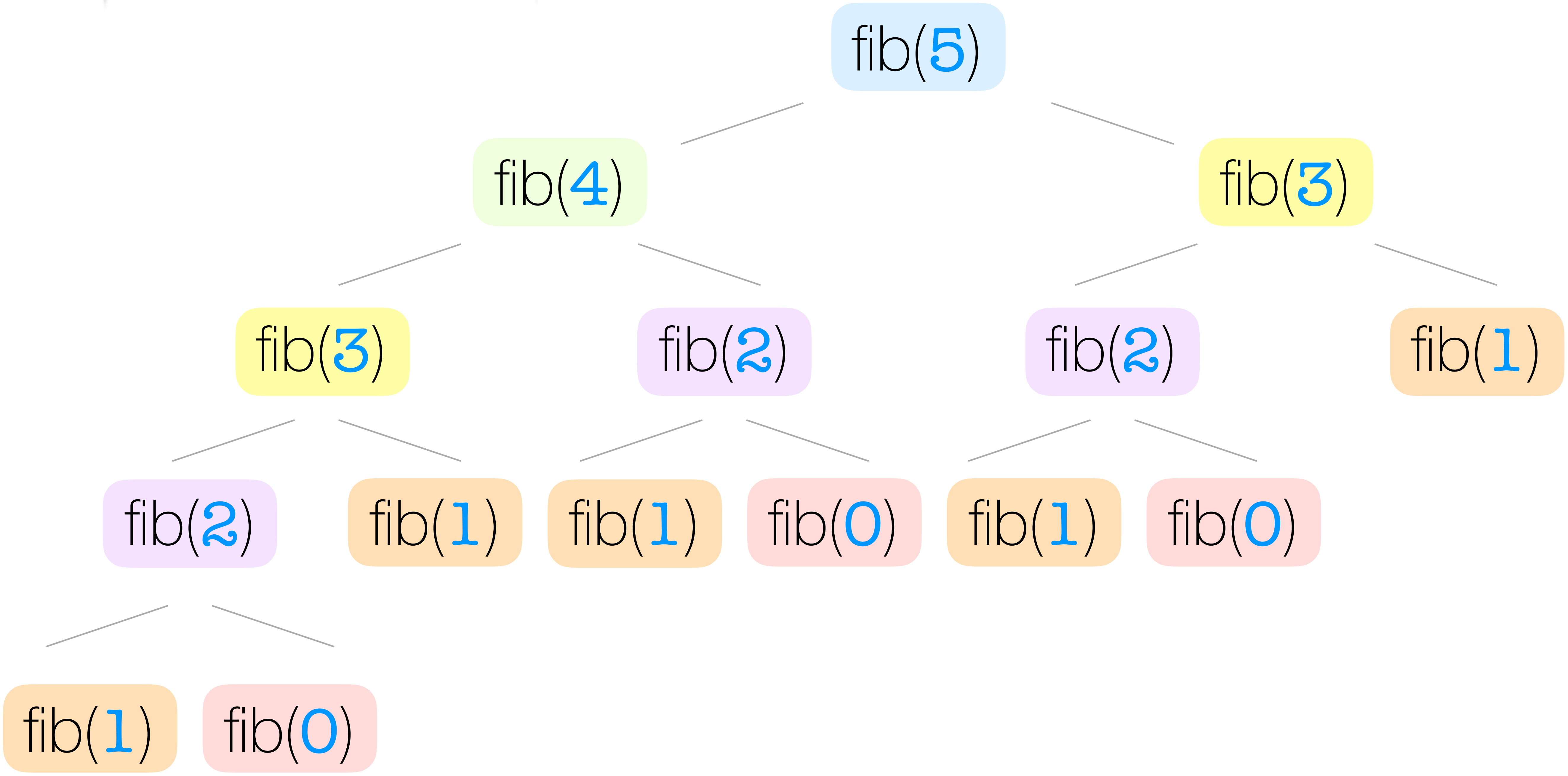
intersecções entre subproblemas



intersecções entre subproblemas



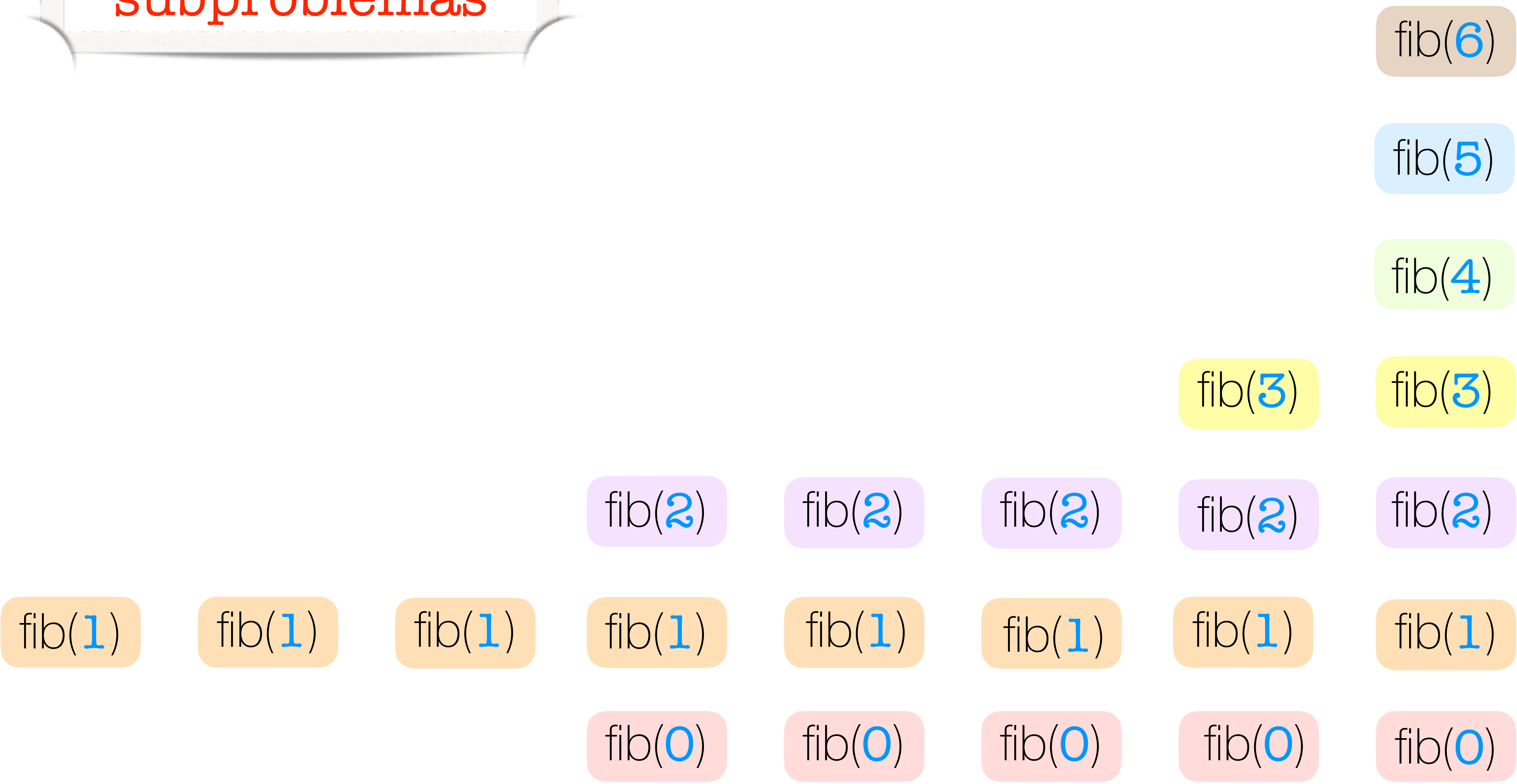
intersecções entre subproblemas



intersecções entre subproblemas

				fib(5)
			fib(4)	fib(4)
		fib(3)	fib(3)	fib(3)
		fib(2)	fib(2)	fib(2)
fib(1)	fib(1)	fib(1)	fib(1)	fib(1)
		fib(0)	fib(0)	fib(0)

intersecções entre subproblemas


fib(6)

fib(5)

fib(4)

fib(3) fib(3)

fib(2) fib(2) fib(2) fib(2) fib(2)

fib(1) fib(1) fib(1) fib(1) fib(1) fib(1) fib(1) fib(1)

fib(0) fib(0) fib(0) fib(0) fib(0)

intersecções entre subproblemas

(programação dinâmica)

intersecções entre subproblemas

fib(0) fib(1)

(programação dinâmica)

intersecções entre subproblemas

fib(0) fib(1) fib(2)

(programação dinâmica)

intersecções entre subproblemas

fib(0) fib(1) fib(2) fib(3)

(programação dinâmica)

intersecções entre subproblemas

fib(0) fib(1) fib(2) fib(3) fib(4)

(programação dinâmica)

intersecções entre subproblemas

fib(0) fib(1) fib(2) fib(3) fib(4) fib(5)

(programação dinâmica)

intersecções entre subproblemas

fib(0) fib(1) fib(2) fib(3) fib(4) fib(5) fib(6)

(programação dinâmica)

(programação dinâmica)

```
int fib_iter(int n)
```

(programação dinâmica)

```
int fib_iter(int n)
{
```

(programação dinâmica)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
```

(programação dinâmica)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
  for (int i = 2; i <= n; i++) {
```

(programação **dinâmica**)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
  for (int i = 2; i <= n; i++) {
     fib = f1 + f2;
```

(programação dinâmica)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
  for (int i = 2; i <= n; i++) {
      fib = f1 + f2;
      f2 = f1;
```

(programação dinâmica)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
  for (int i = 2; i <= n; i++) {
     fib = f1 + f2;
     f2 = f1;
     f1 = fib;
```

(programação dinâmica)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
  for (int i = 2; i <= n; i++) {
      fib = f1 + f2;
      f2 = f1;
      f1 = fib;
   }
```

(programação dinâmica)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
  for (int i = 2; i <= n; i++) {
     fib = f1 + f2;
     f2 = f1;
     f1 = fib;
  }
  return fib;
```

(programação dinâmica)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
  for (int i = 2; i <= n; i++) {
     fib = f1 + f2;
     f2 = f1;
     f1 = fib;
  }
  return fib;
}
```

(programação dinâmica)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
  for (int i = 2; i <= n; i++) {
     fib = f1 + f2;
     f2 = f1;
     f1 = fib;
  }
  return fib;
}
```

fib(0) fib(1)

(programação dinâmica)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
  for (int i = 2; i <= n; i++) {
     fib = f1 + f2;
     f2 = f1;
     f1 = fib;
  }
  return fib;
}
```

fib(0) fib(1) fib(2)

(programação dinâmica)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
  for (int i = 2; i <= n; i++) {
     fib = f1 + f2;
     f2 = f1;
     f1 = fib;
  }
  return fib;
}
```

fib(0) fib(1) fib(2) fib(3)

(programação dinâmica)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
  for (int i = 2; i <= n; i++) {
     fib = f1 + f2;
     f2 = f1;
     f1 = fib;
  }
  return fib;
}
```

fib(0) fib(1) fib(2) fib(3) fib(4)

(programação **dinâmica**)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
  for (int i = 2; i <= n; i++) {
     fib = f1 + f2;
     f2 = f1;
     f1 = fib;
  }
  return fib;
}
```

fib(0) fib(1) fib(2) fib(3) fib(4) fib(5)

(programação **dinâmica**)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
  for (int i = 2; i <= n; i++) {
     fib = f1 + f2;
     f2 = f1;
     f1 = fib;
  }
  return fib;
}
```

fib(0) fib(1) fib(2) fib(3) fib(4) fib(5) fib(6)

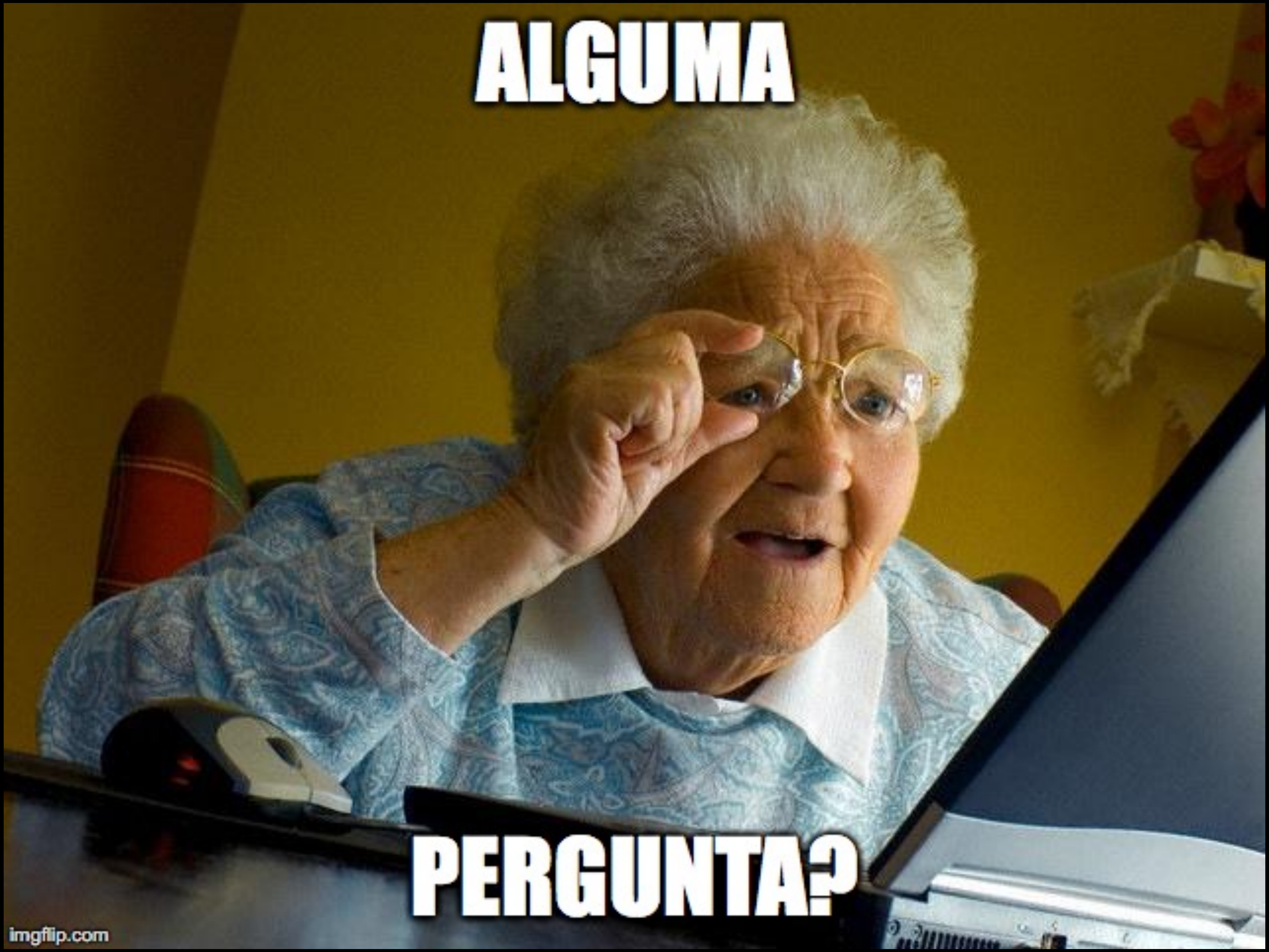
(programação **dinâmica**)

```
int fib_iter(int n)
{
  int fib, f1 = 1, f2 = 0;
  for (int i = 2; i <= n; i++) {
     fib = f1 + f2;
     f2 = f1;
     f1 = fib;
  }
  return fib;
}
```

```
int fib_rec(int n)
{
   if (n <= 1) return n;
   return fib(n-1) + fib(n-2);
}
```

fib(0) fib(1) fib(2) fib(3) fib(4) fib(5) fib(6)

(programação **dinâmica**)


ALGUMA
PERGUNTA?
imgflip.com

dividir para conquistar	intersecções entre subproblemas
recursão de cauda	demais casos

recursão de cauda

recursão de cauda

```
int fat_rec(int n)
```

recursão de cauda

```
int fat_rec(int n)
{
```

recursão de cauda

```
int fat_rec(int n)
{
  if (n <= 1) return 1;
```

recursão de cauda

```
int fat_rec(int n)
{
  if (n <= 1) return 1;
  return n * fat_rec(n-1);
```

recursão de cauda

```
int fat_rec(int n)
{
  if (n <= 1) return 1;
  return n * fat_rec(n-1);
}
```

recursão de cauda

recursão de cauda

```
int fat_cauda(int n, int f)
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n* f);
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n* f);
}
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n* f);
}
```

```
int fat_rec(int n)
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n* f);
}
```

```
int fat_rec(int n)
{
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n * f);
}
```

```
int fat_rec(int n)
{
    return fat_cauda(n, 1);
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n * f);
}
```

```
int fat_rec(int n)
{
    return fat_cauda(n, 1);
}
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n * f);
}

int fat_rec(int n)
{
    return fat_cauda(n, 1);
}
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n * f);
}

int fat_rec(int n)
{
    return fat_cauda(n, 1);
}
```

```
int fat(int n)
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n * f);
}

int fat_rec(int n)
{
    return fat_cauda(n, 1);
}
```

```
int fat(int n)
{
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n * f);
}

int fat_rec(int n)
{
    return fat_cauda(n, 1);
}
```

```
int fat(int n)
{
   int f = 1;
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n * f);
}

int fat_rec(int n)
{
    return fat_cauda(n, 1);
}
```

```
int fat(int n)
{
   int f = 1;
   while (n > 1) {
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n * f);
}

int fat_rec(int n)
{
    return fat_cauda(n, 1);
}
```

```
int fat(int n)
{
   int f = 1;
   while (n > 1) {
      f = n * f;
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n * f);
}

int fat_rec(int n)
{
    return fat_cauda(n, 1);
}
```

```
int fat(int n)
{
   int f = 1;
   while (n > 1) {
      f = n * f;
      n = n - 1;
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n * f);
}

int fat_rec(int n)
{
    return fat_cauda(n, 1);
}
```

```
int fat(int n)
{
   int f = 1;
   while (n > 1) {
      f = n * f;
      n = n - 1;
   }
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n * f);
}

int fat_rec(int n)
{
    return fat_cauda(n, 1);
}
```

```
int fat(int n)
{
   int f = 1;
   while (n > 1) {
      f = n * f;
      n = n - 1;
   }
   return f;
```

recursão de cauda

```
int fat_cauda(int n, int f)
{
  if (n <= 1) return f;
  return fat_cauda(n-1, n * f);
}

int fat_rec(int n)
{
    return fat_cauda(n, 1);
}
```

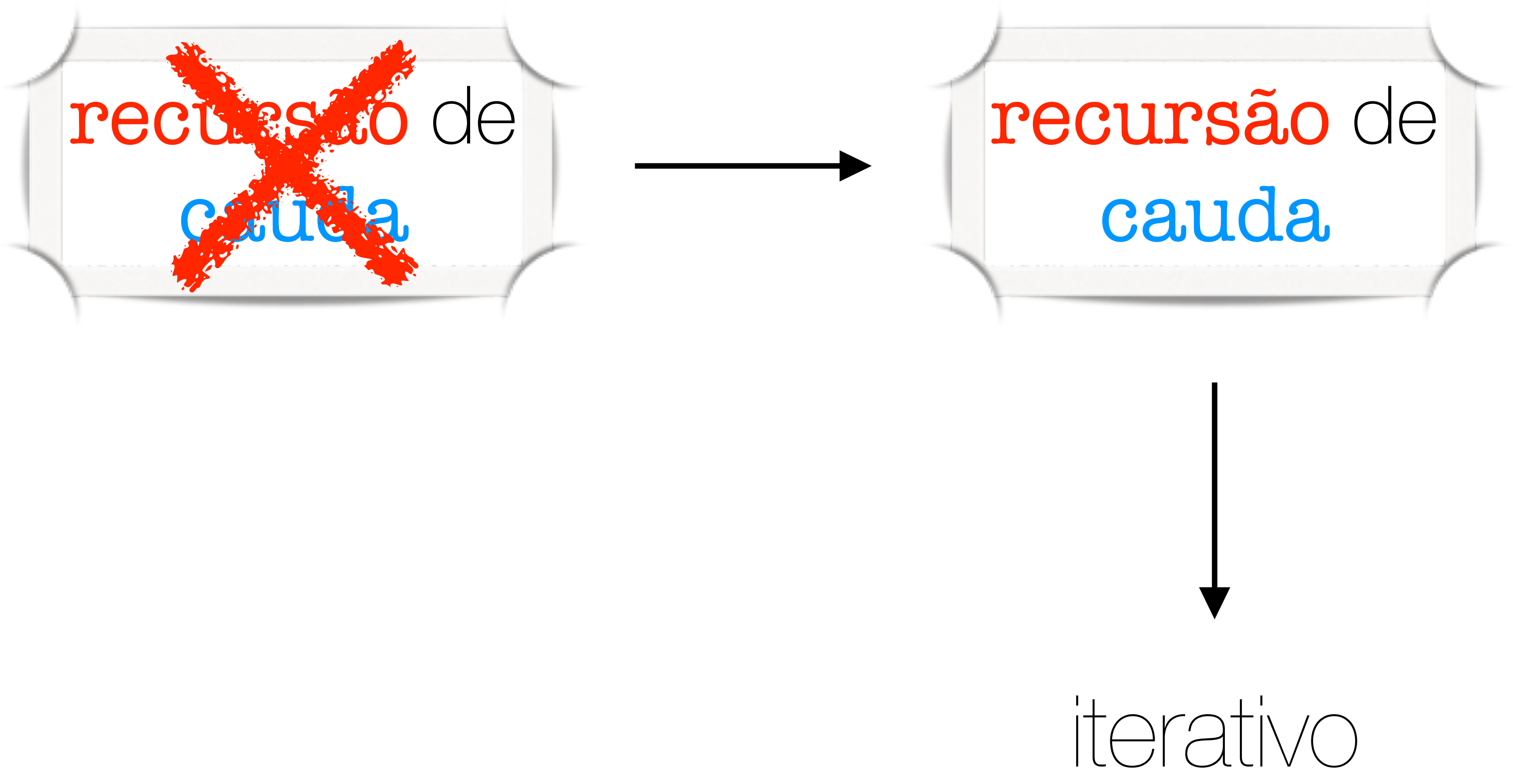
```
int fat(int n)
{
   int f = 1;
   while (n > 1) {
      f = n * f;
      n = n - 1;
   }
   return f;
}
```

🤔

~~recursão~~ de
~~cauda~~

~~recursão~~ de ~~cauda~~ → recursão de cauda

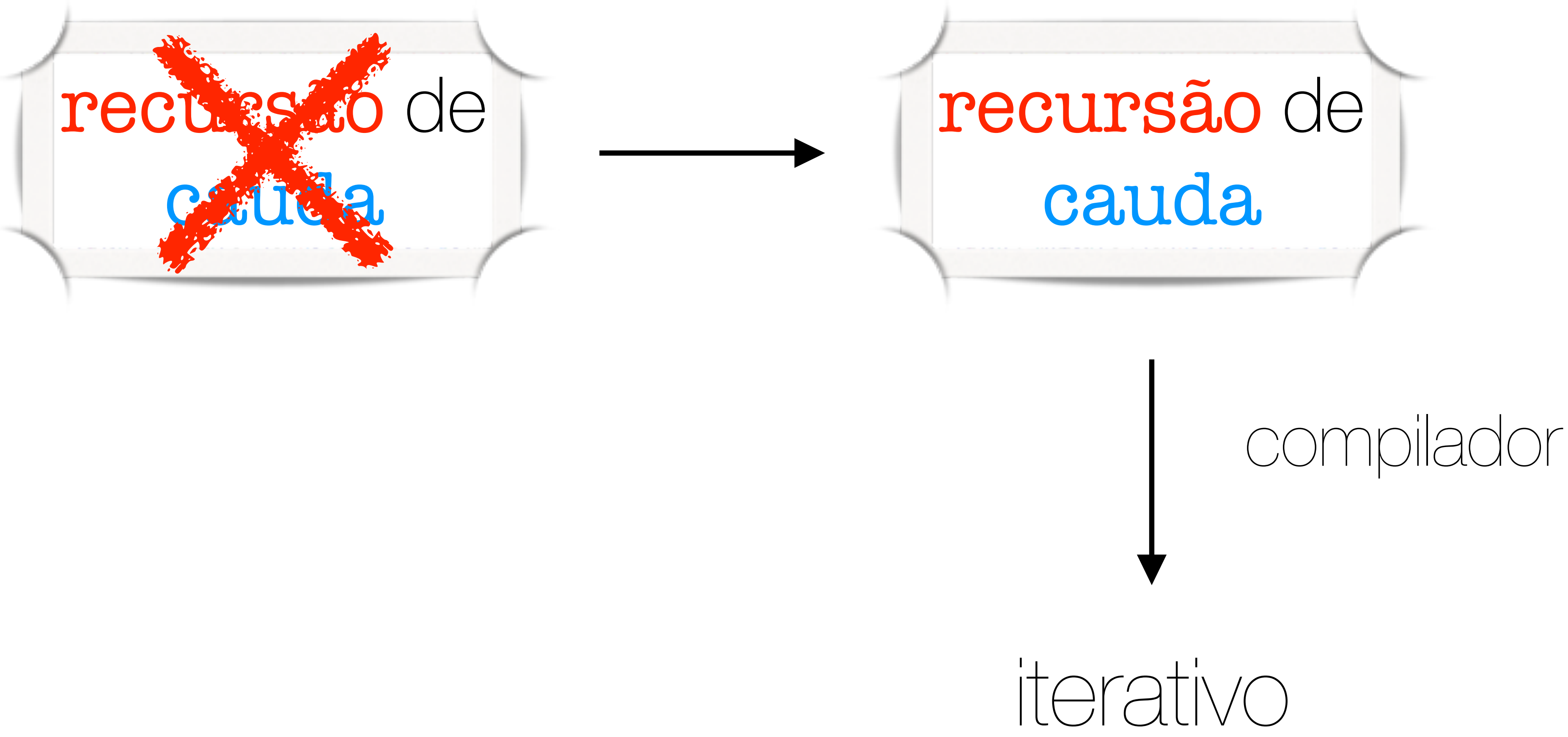
🤔



~~recursão~~ de ~~cauda~~

recursão de cauda

iterativo



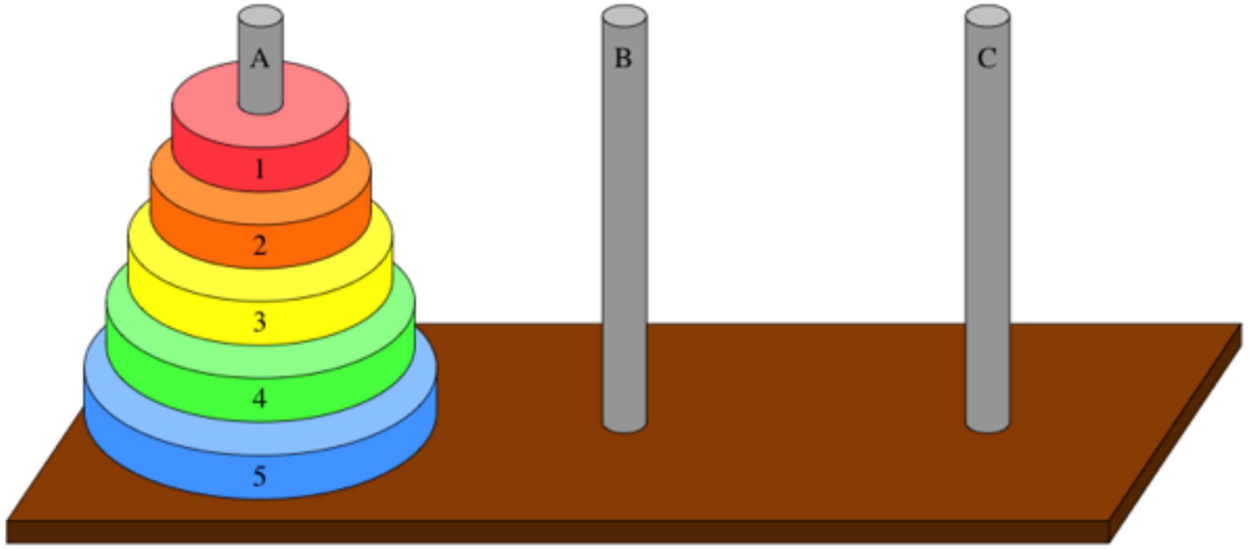
~~recursão de cauda~~

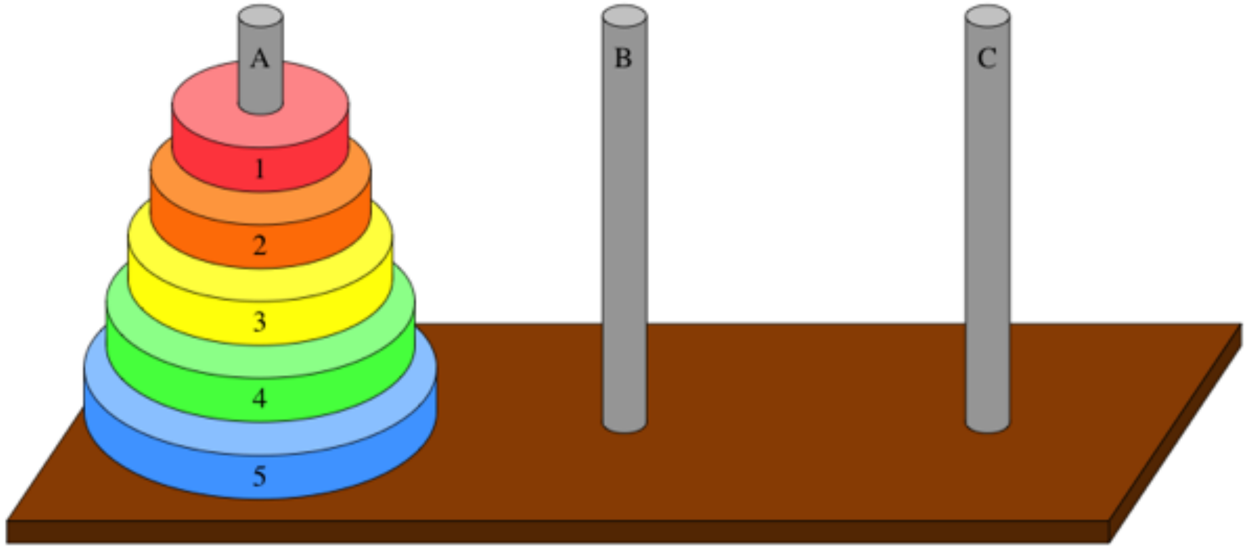
recursão de cauda

compilador

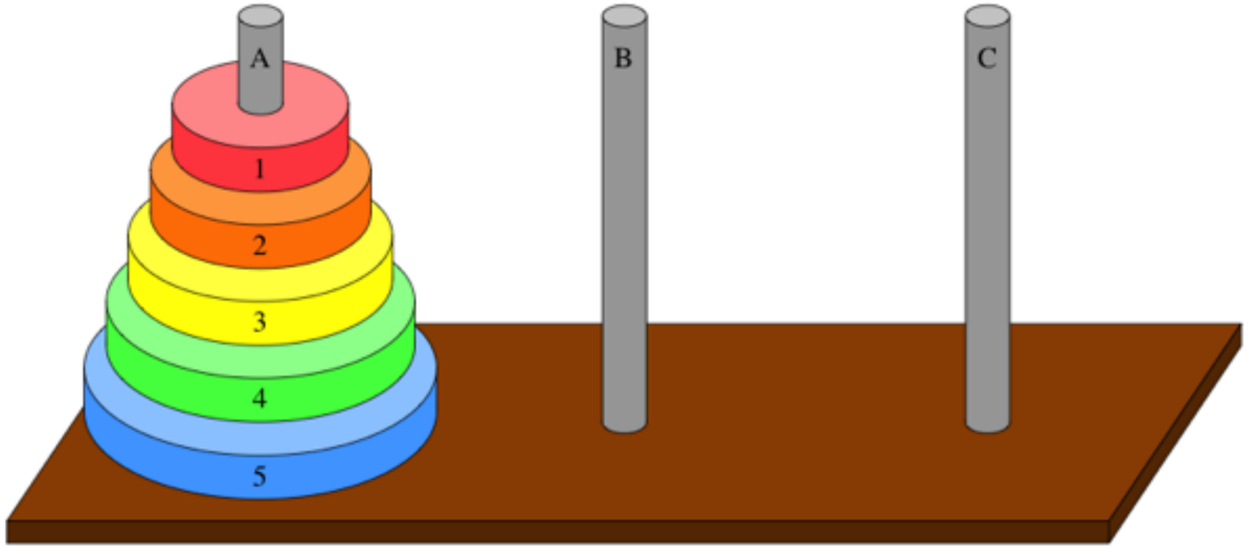
iterativo

dividir para conquistar	intersecções entre subproblemas
recursão de cauda	demais casos

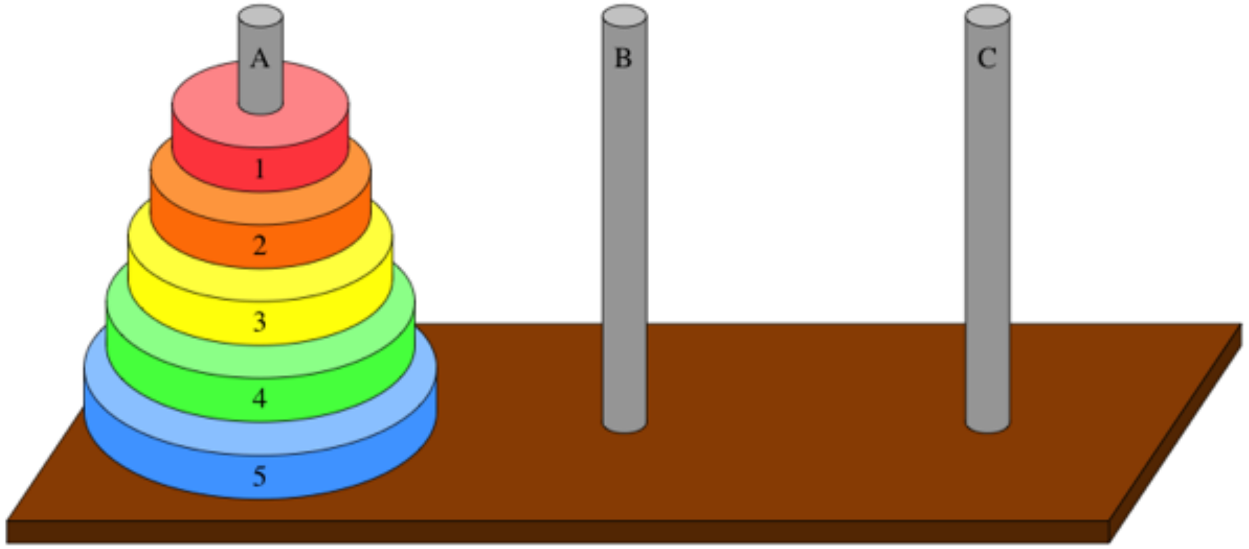

A
B
C
1
2
3
4
5



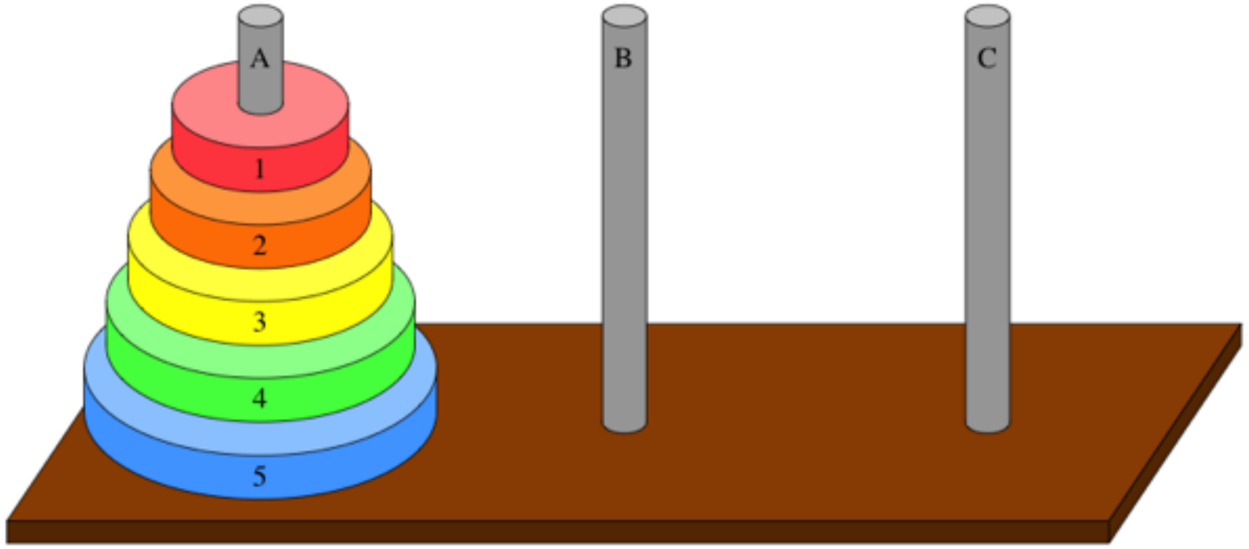
hanoi (altura, origem, destino, aux)



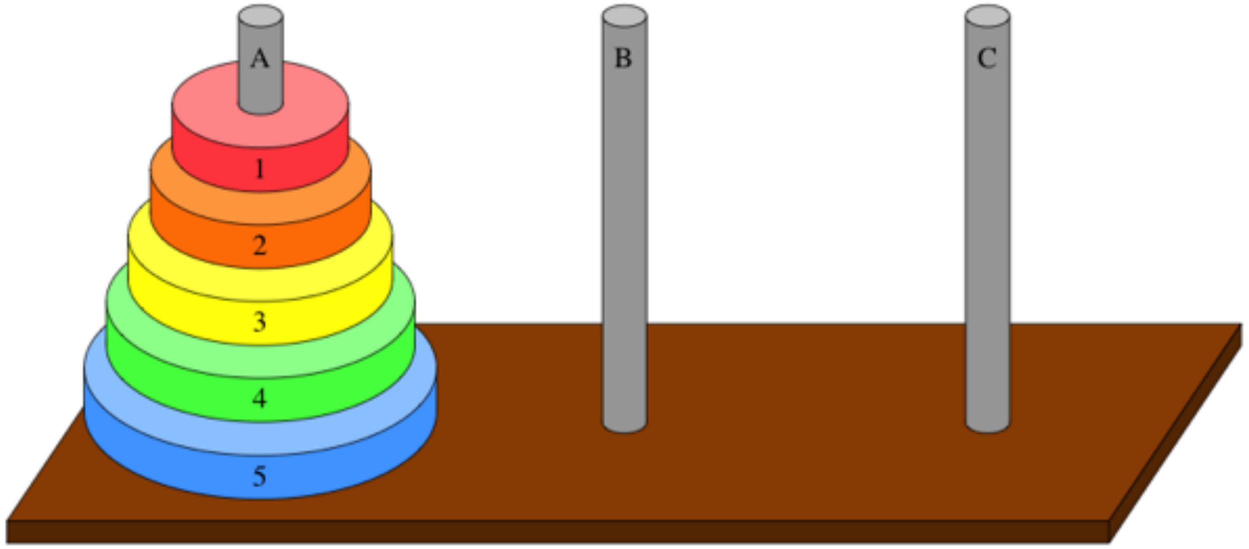
```
hanoi (altura, origem, destino, aux)
    if (altura == 1)
```



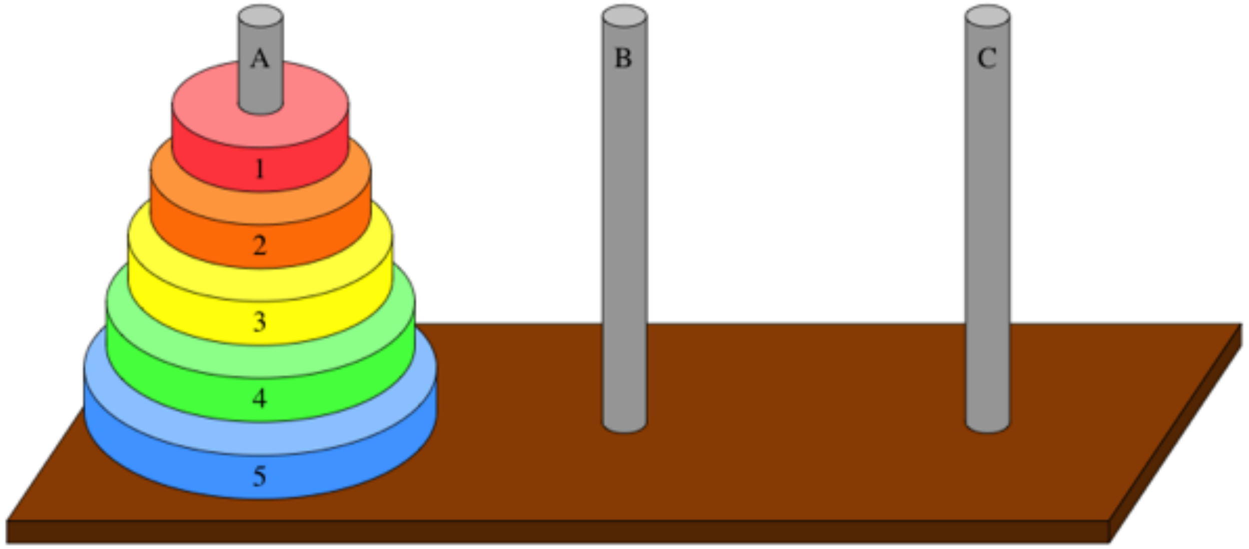
```
hanoi (altura, origem, destino, aux)
    if (altura == 1)
        “Movendo disco 1 da torre origem pra torre destino.”
```



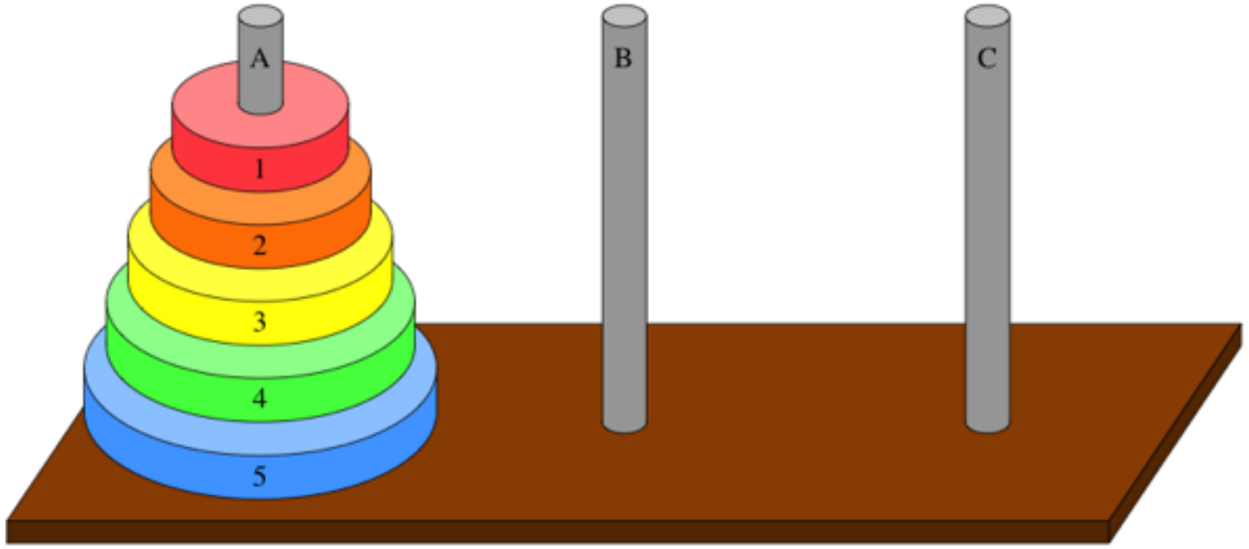
```
hanoi (altura, origem, destino, aux)
    if (altura == 1)
        “Movendo disco 1 da torre origem pra torre destino.”
        return
```



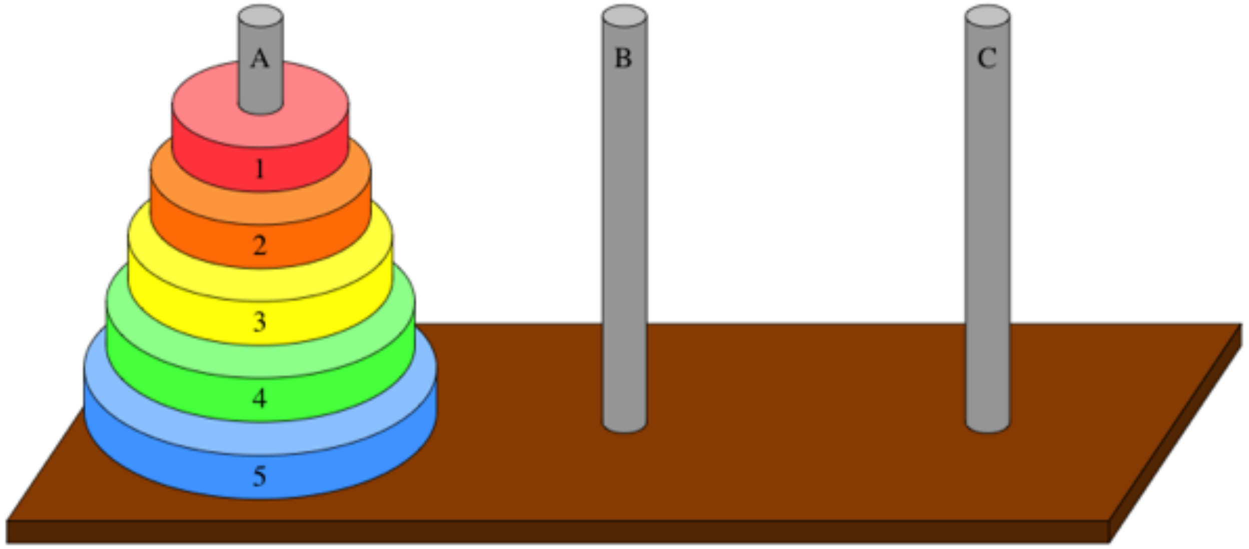
```
hanoi (altura, origem, destino, aux)
    if (altura == 1)
        “Movendo disco 1 da torre origem pra torre destino.”
        return
    }
```



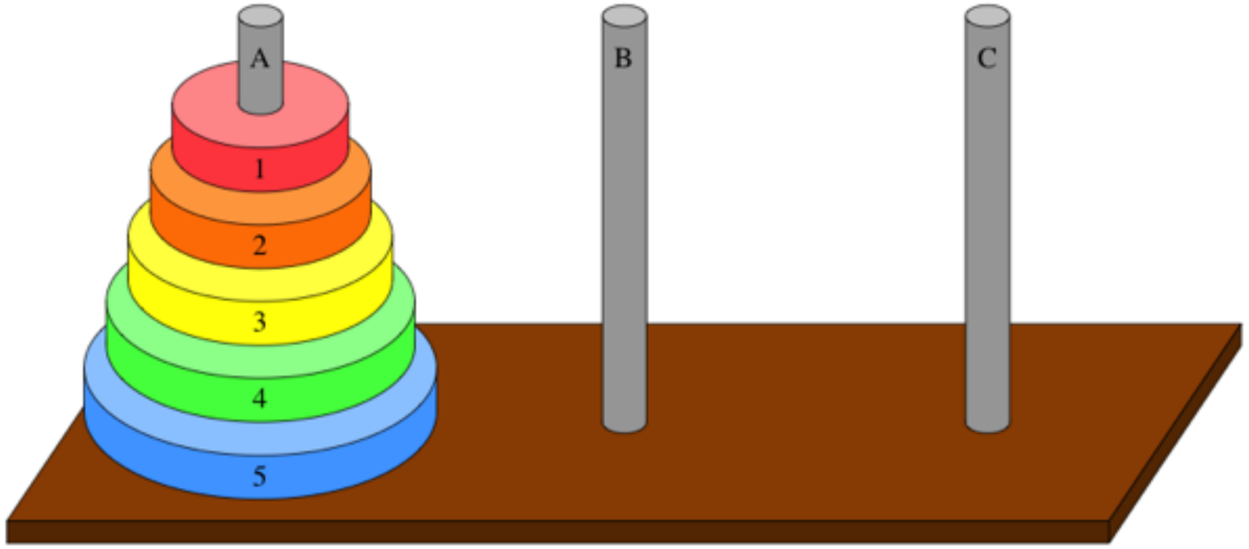
```
hanoi (altura, origem, destino, aux)
    if (altura == 1)
        "Movendo disco 1 da torre origem pra torre destino."
        return
    }
    hanoi (altura - 1, origem, aux, destino);
```



```
hanoi (altura, origem, destino, aux)
    if (altura == 1)
        “Movendo disco 1 da torre origem pra torre destino.”
        return
    }
    hanoi (altura - 1, origem, aux, destino);
    “Movendo disco altura da torre origem pra torre destino.”
```



```
hanoi (altura, origem, destino, aux)
    if (altura == 1)
        “Movendo disco 1 da torre origem pra torre destino.”
        return
    }
    hanoi (altura - 1, origem, aux, destino);
    “Movendo disco altura da torre origem pra torre destino.”
    hanoi (altura - 1, aux, destino, origem);
```



```
hanoi (altura, origem, destino, aux)
    if (altura == 1)
        “Movendo disco 1 da torre origem pra torre destino.”
        return
    }
    hanoi (altura - 1, origem, aux, destino);
    “Movendo disco altura da torre origem pra torre destino.”
    hanoi (altura - 1, aux, destino, origem);
}
```


dividir para conquistar	👍🍺

dividir para conquistar	👍🍺
intersecções entre subproblemas	👎

dividir para conquistar	👍🍺
intersecções entre subproblemas	👎
recursão de cauda	👍🤔

dividir para conquistar	👍🍺
intersecções entre subproblemas	👎
recursão de cauda	👍🤔
demais casos	🍺