Recursão

Recursão

Recursão

Recursão

Calcule os números de fibonacci

Fibonacci

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

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

sendo fib(0) = 1 e fib(1) = 1

fib(2) = fib(1) + fib(0) = 1 + 1 = 2

fib(3) = fib(2) + fib(1) = 2 + 1 = 3

fib(4) = fib(3) + fib(2) = 3 + 2 = 5

Função fib(n)

```
int fib(int n) {



}
```

Função fib(n)

```
int fib(int n) {


}
```

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

sendo fib(0) = 1 e fib(1) = 1
```

Função fib(n)

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

sendo $fib(0) = 1$ e $fib(1) = 1$

```
int fib(int n) {
   return fib(n - 1) + fib(n - 2);
}
```

Função fib(n)

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

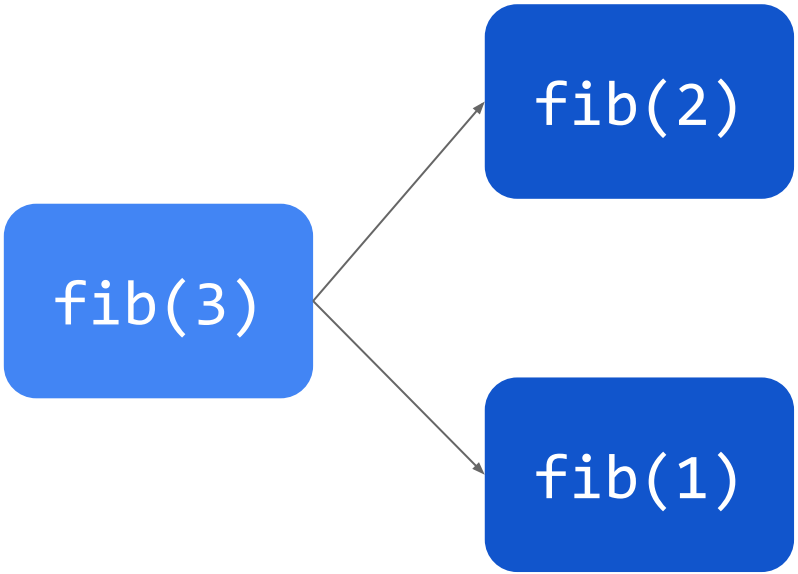
sendo $fib(0) = 1$ e $fib(1) = 1$

```
int fib(int n) {

   return fib(n - 1) + fib(n - 2);

}
```

O que acontece quando se chama **fib(3)** ?

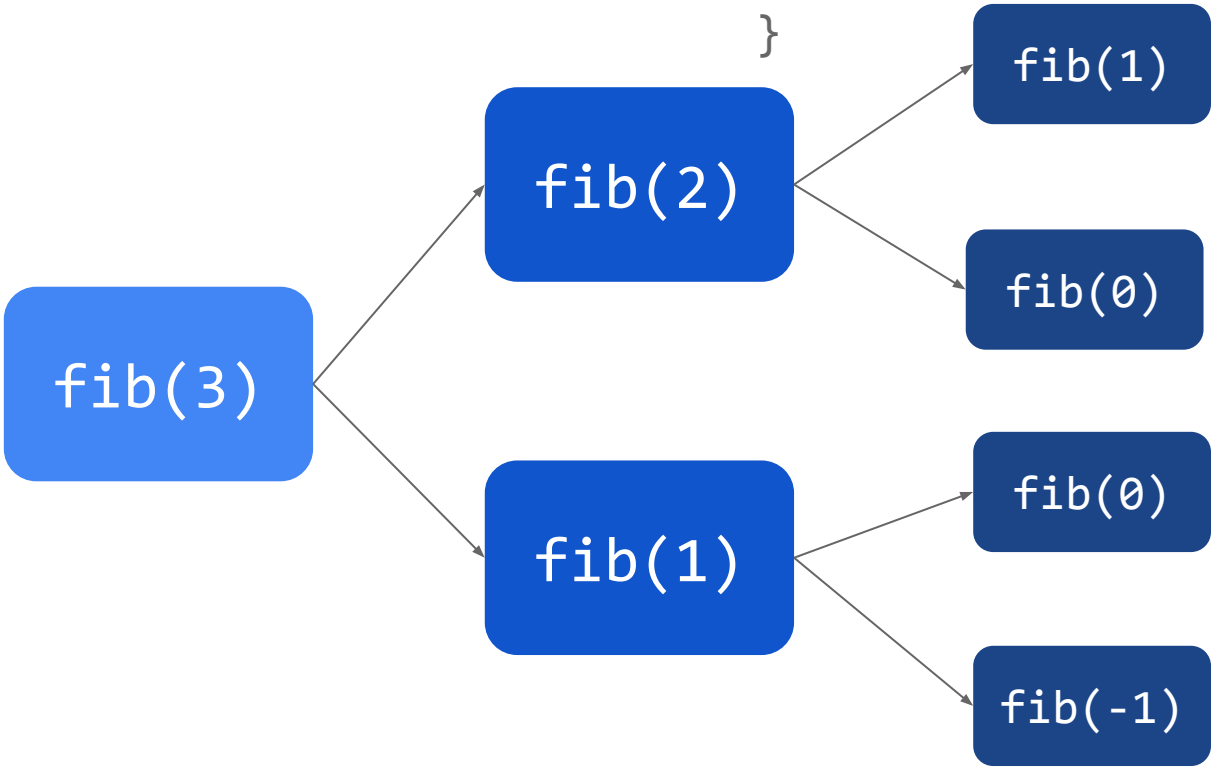
Função fib(n)

```
int fib(int n) {
    return fib(n - 1) + fib(n - 2);
}
```



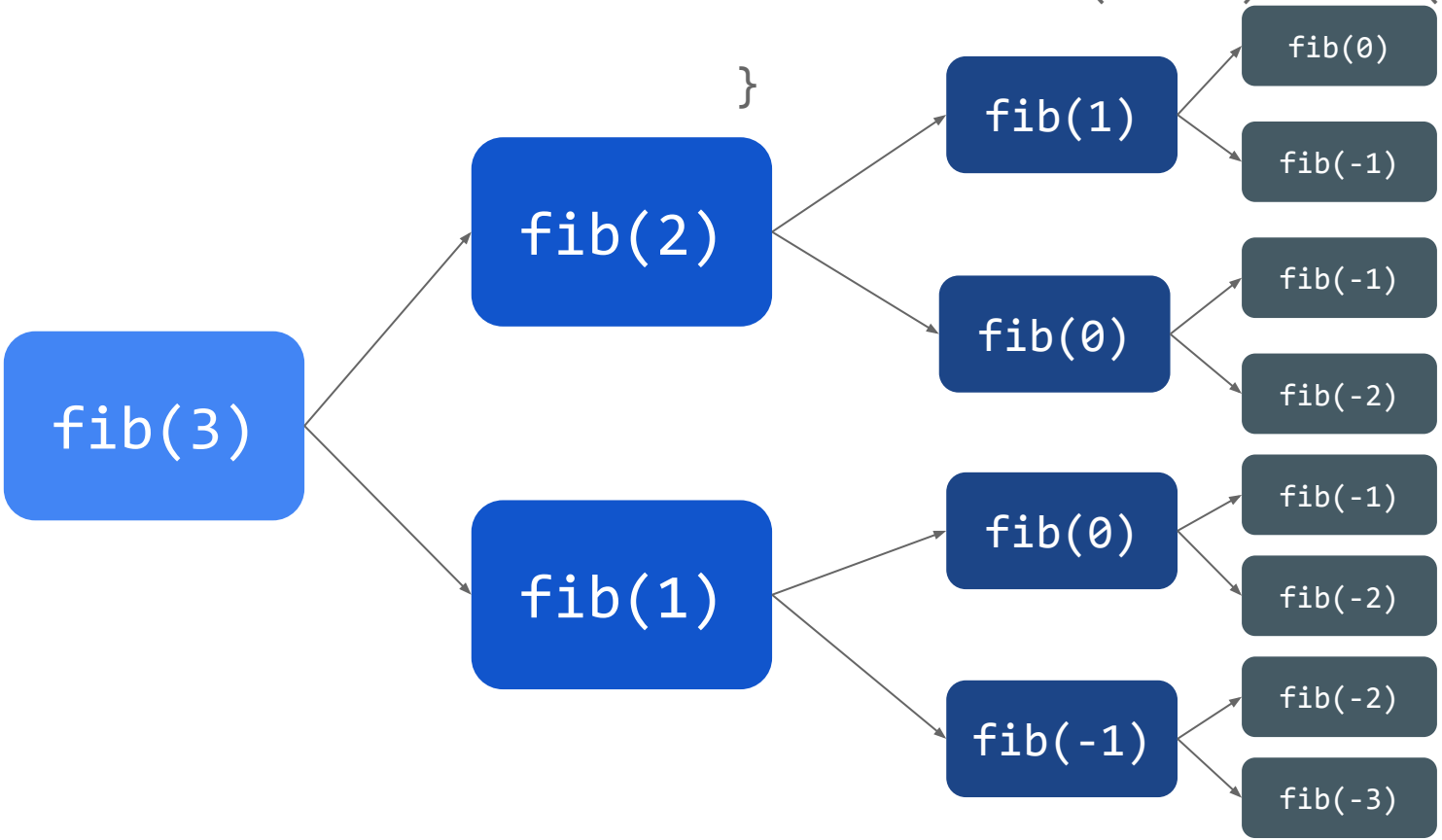
Função fib(n)

```
int fib(int n) {
    return fib(n - 1) + fib(n - 2);
}
```



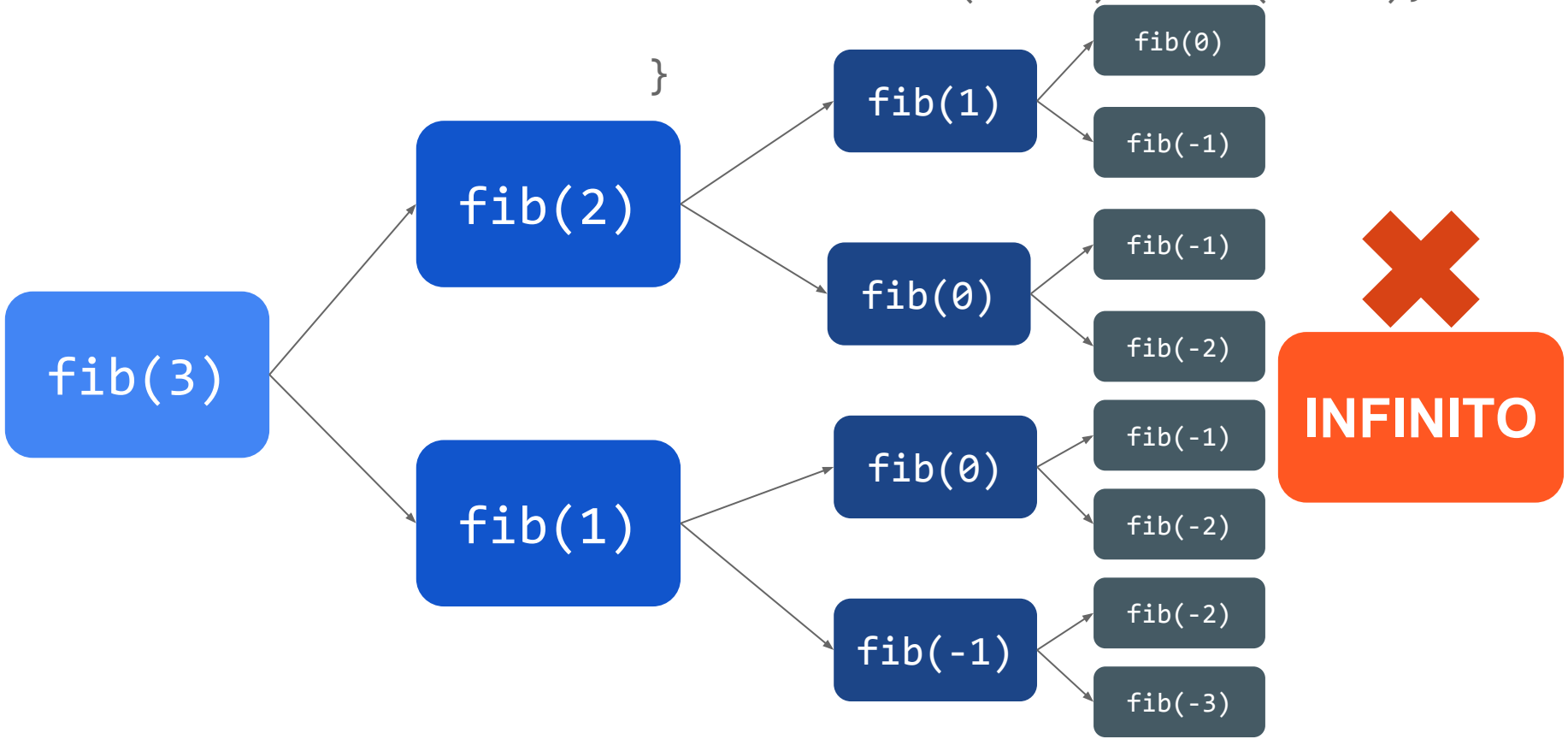
Função fib(n)

```
int fib(int n) {
    return fib(n - 1) + fib(n - 2);
}
```



Função fib(n)

```
int fib(int n) {
    return fib(n - 1) + fib(n - 2);
}
```



Função fib(n)

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

sendo fib(0) = 1 e fib(1) = 1

```
int fib(int n) {

   return fib(n - 1) + fib(n - 2);
}
```

Função fib(n)

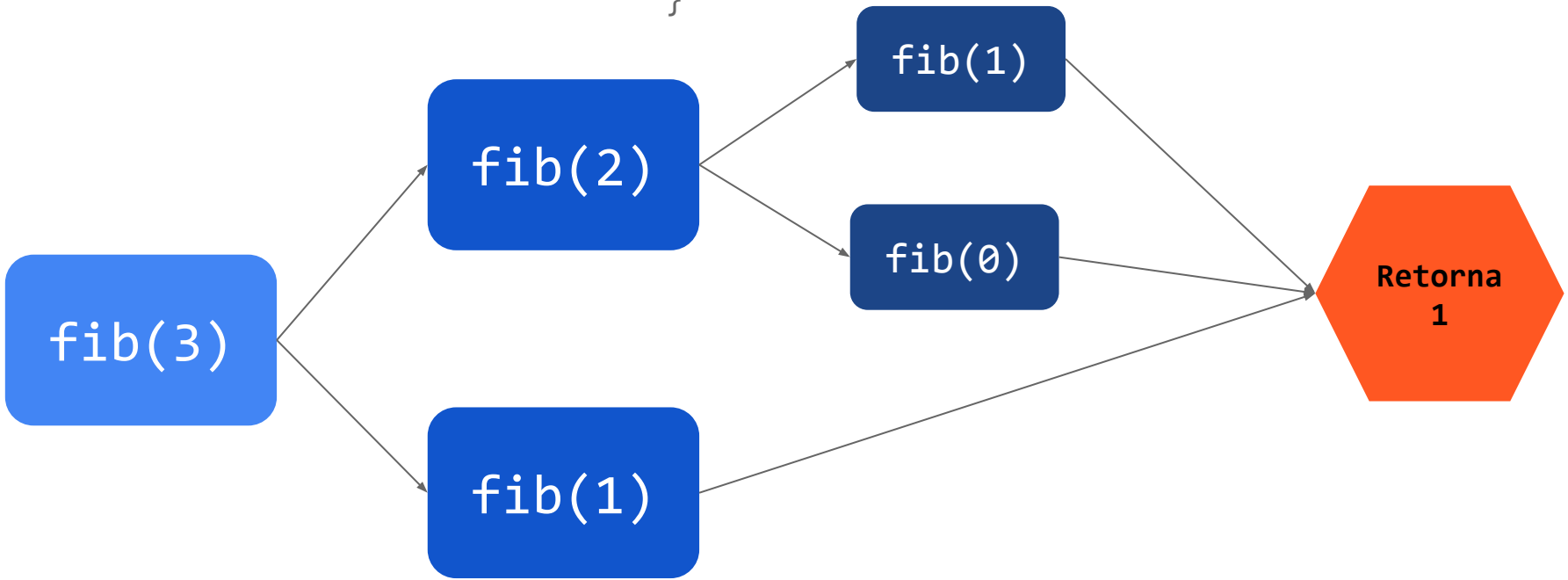
$$fib(n) = fib(n - 1) + fib(n - 2)$$

sendo fib(0) = 1 e fib(1) = 1

```
int fib(int n) {
   if(n == 1 || n == 0) return 1;
   return fib(n - 1) + fib(n - 2);
}
```

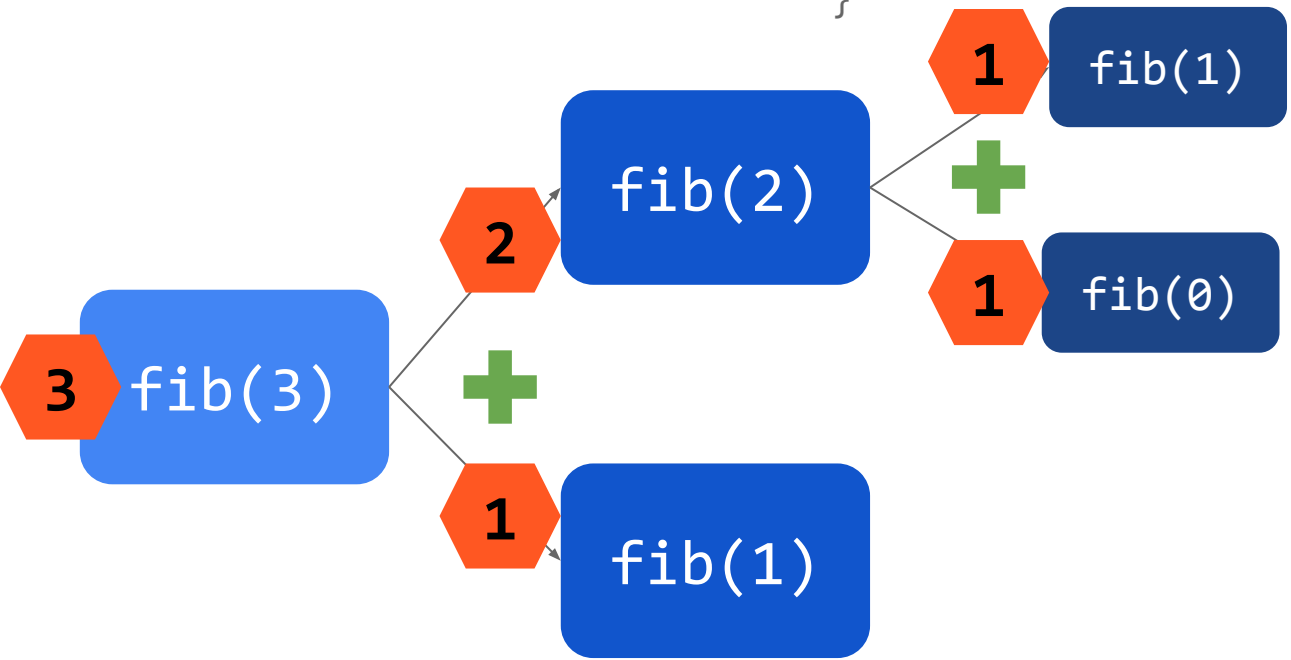
Função fib(n)

```
int fib(int n) {
    if(n == 1 || n == 0) return 1;
    return fib(n - 1) + fib(n - 2);
}
```



Função fib(n)

```
int fib(int n) {
    if(n == 1 || n == 0) return 1;
    return fib(n - 1) + fib(n - 2);
}
```



2 passos para uma recursão de sucesso!

```
int fib(int n) {
    if(n == 1 || n == 0) return 1;
    return fib(n - 1) + fib(n - 2);
}
```

1) Tenha uma condição de parada (caso base)

```
int fib(int n) {

     if(n == 1 || n == 0) return 1;

     return fib(n - 1) + fib(n - 2);

}
```

1) Tenha uma condição de parada (caso base)

2) Faça com que os parâmetros de entrada sempre mudem na chamada recursiva (recursão)

Testem: faça uma função que calcule o fatorial recursivamente

Vantagens

Reduz o código

Facilita a lógica de alguns problemas

Desvantagens

Chamar funções custa caro para o processamento

Perigo de rodar infinitamente

Dividir para Conquistar

Ordenação de Array

- Já aprendemos o select sort

```
for (int j = 0; j < n; j++)  {
    int iMin = j;
    for (i = j+1; i < n; i++) {
        if (a[i] < a[iMin]) {
            iMin = i;
        }
    }

    int aux = a[j];
    a[j] = a[iMin];
    a[iMin] = aux;
}
```

Encontra o **indice** do **menor** número

Troca o menor com a menor posição não ordenada

Ordenação de Array

- Já aprendemos o select sort

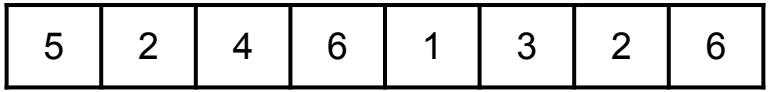
$O(N^2)$

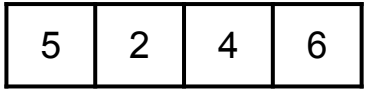
```
for (int j = 0; j < n; j++)  {
    int iMin = j;
    for (i = j+1; i < n; i++) {
        if (a[i] < a[iMin]) {
            iMin = i;
        }
    }

    int aux = a[j];
    a[j] = a[iMin];
    a[iMin] = aux;
}
```

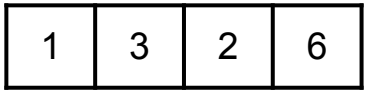
Encontra o **indice** do **menor** número

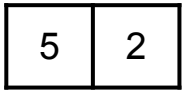
Troca o menor com a menor posição não ordenada

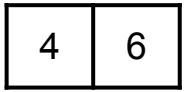
Ordenação de Array (Merge Sort)

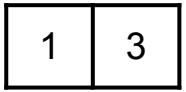


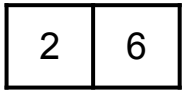


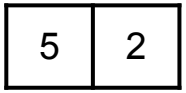


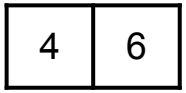


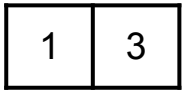


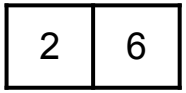






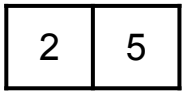


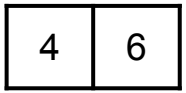


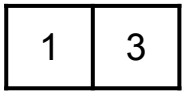


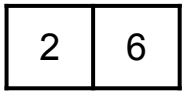
Ordene 2 números

```
void ord2(int array[2]) {
    if (array[0] > array[1]) {
        int aux = array[1];
        array[1] = array[0];
        array[0] = aux;
    }
}
```



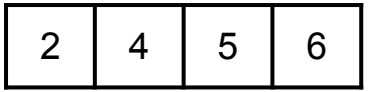


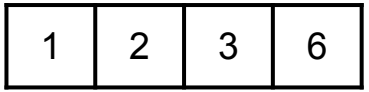


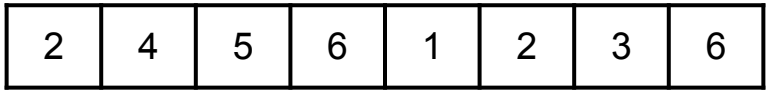


Ordene 4 números (metade esta ordenada)

```
void ord4(int array[4]) {
    ??????????
}
```

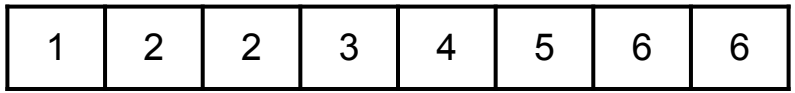


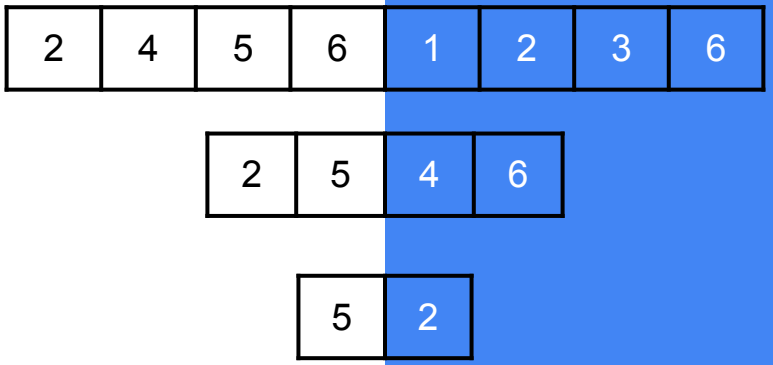




Ordene 8 números (metade esta ordenada)

```
void ord8(int array[8]) {
    ??????????
}
```





ord2, ord4, ord8

O que eles tem em comum?

Ordenam um array que é formado por **2 arrays ordenados**

Como fazer um ord genérico?

Merge Sort

O(N log N)

Fim

Fim

Fim