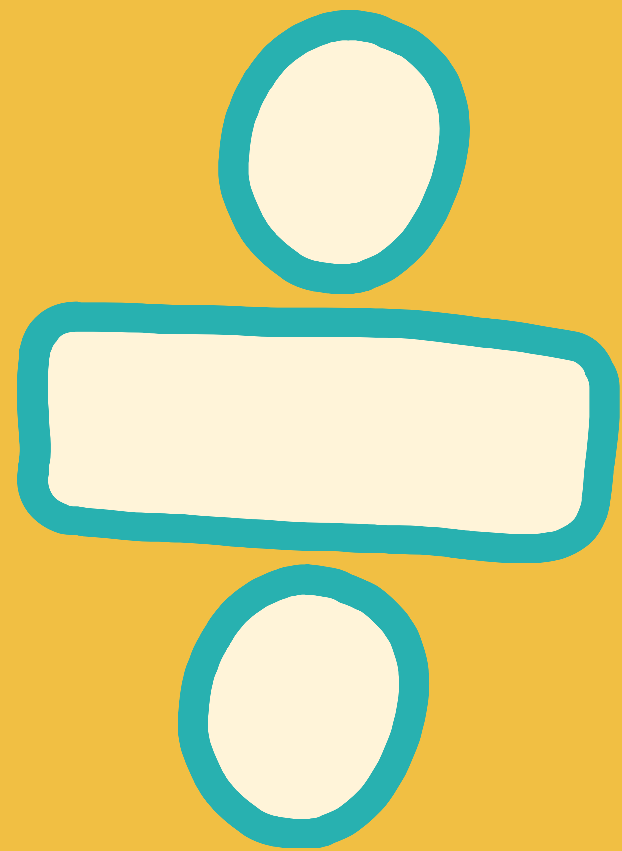


DIVISÃO & CONQUISTA

PAE André Luís Mendes Fakhoury
Prof. João do Espírito Santo Batista Neto

SCC0218 - Algoritmos Avançados e Aplicações

O que é?

Uma técnica, e não um algoritmo em si

Vocês provavelmente já viram vários exemplos de algoritmos divisão e conquista:

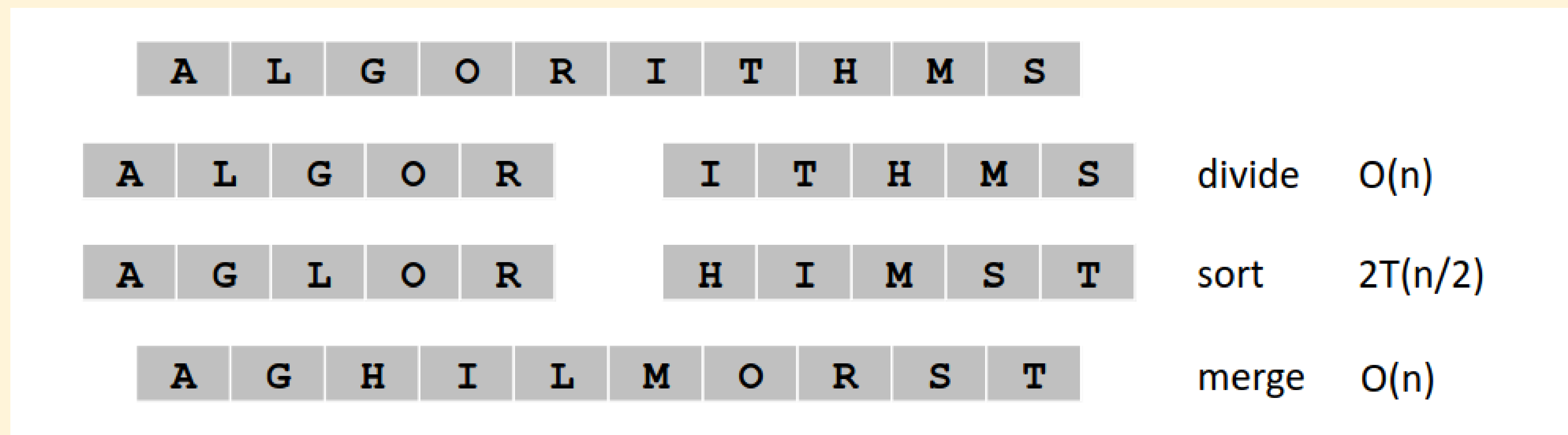
- Merge sort
- Quick sort
- Busca binária
- ...

Relembrando...

Merge sort

Para ordenar um vetor **v**:

- Divide v em duas metades
- Ordena, recursivamente, cada metade
- Mescla as duas metades para ficar ordenado



Ideia geral

Um algoritmo **divisão e conquista** normalmente segue os seguintes itens:

- Divide o problema original em subproblemas menores;
- Resolve cada subproblema (normalmente de forma recursiva);
- Se necessário, as soluções dos subproblemas são combinadas.

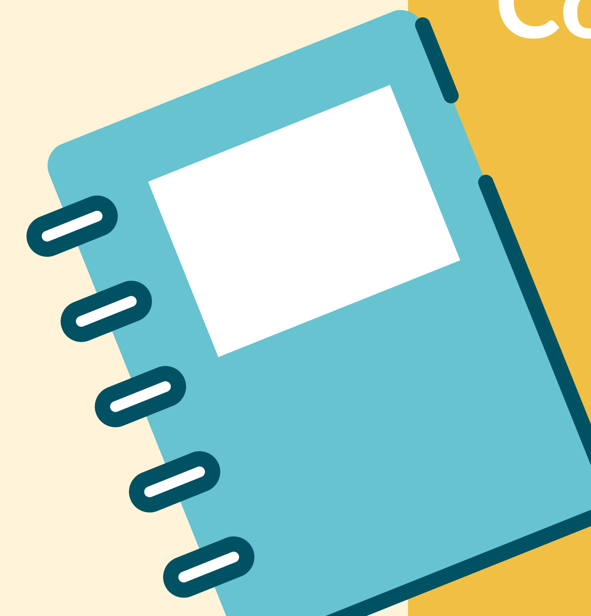
Ideia geral

Um algoritmo **divisão e conquista** normalmente segue os seguintes itens:

- Divide o problema original em subproblemas menores;
- Resolve cada subproblema (normalmente de forma recursiva);
- Se necessário, as soluções dos subproblemas são combinadas.

Divisão

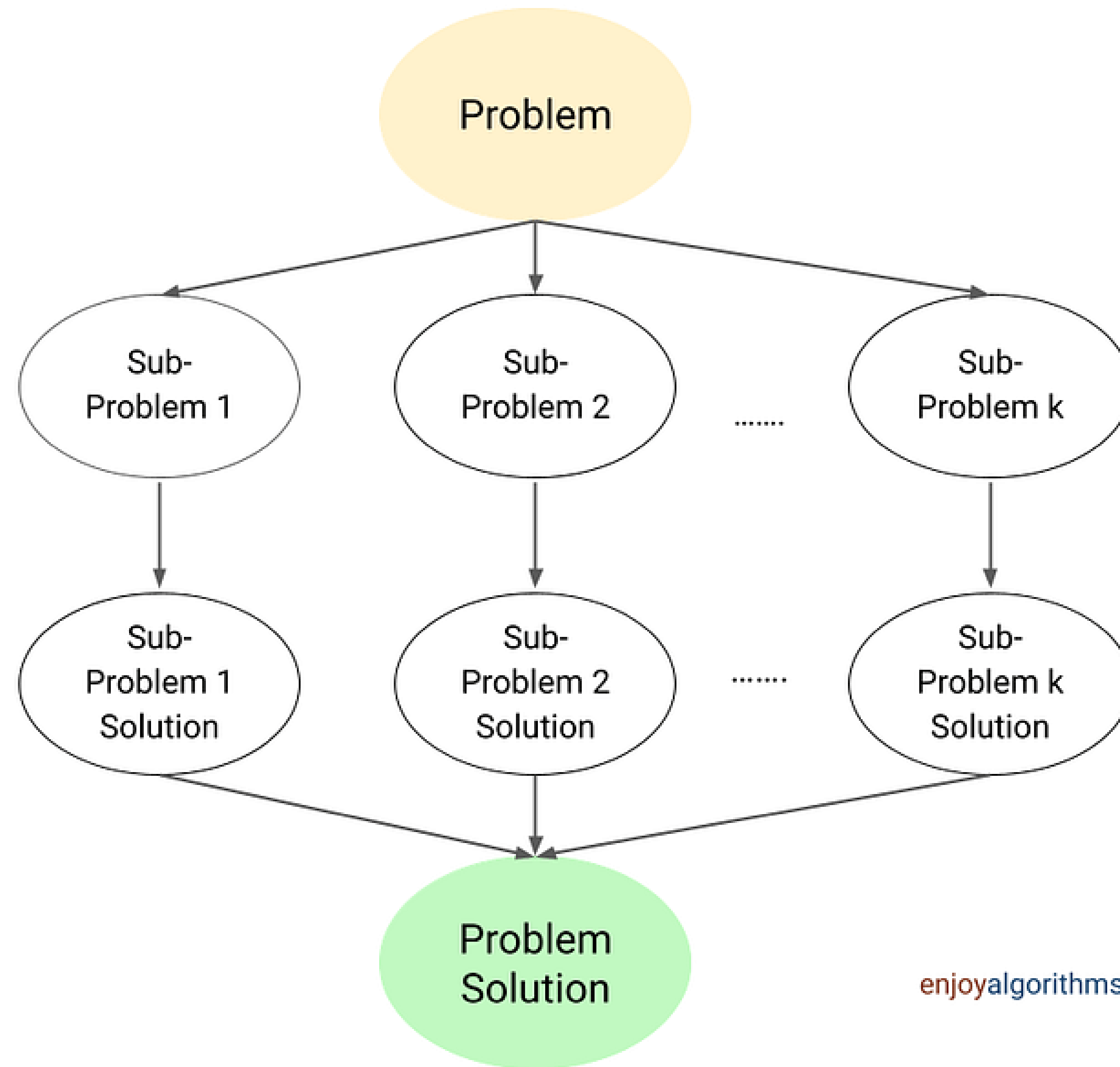
“Conquista”



Divide
Dividing the problem into smaller sub-problems

Conquer
Solving each sub-problems recursively

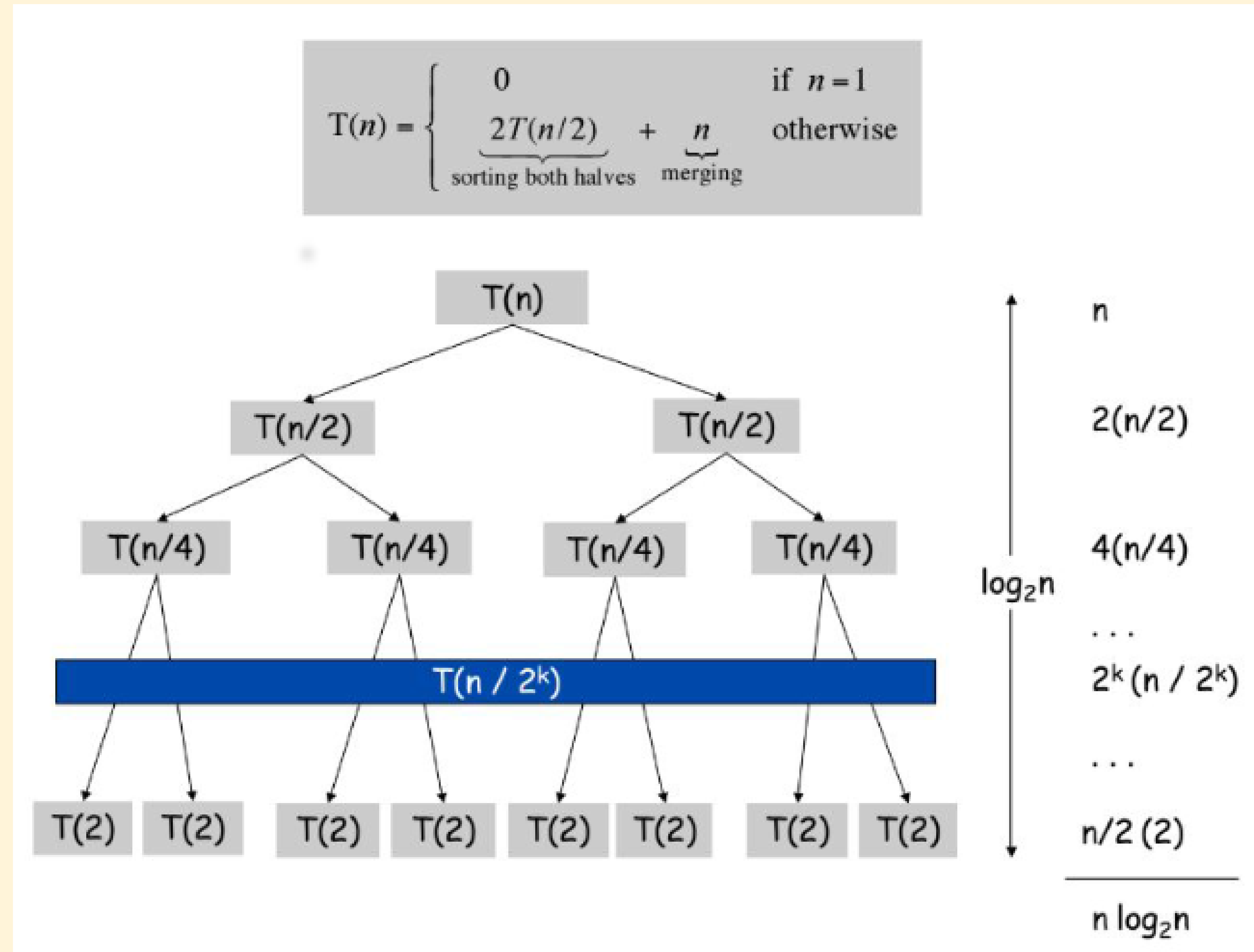
Combine
Combining sub-problem solutions to build the original problem solution



enjoyalgorithms.com

E a complexidade?

Árvore de recursão



Teorema Mestre

Para a seguinte recorrência T(n):

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

Temos as seguintes complexidades:

If $f(n) \in \Theta(n^d)$ where $d \geq 0$ in recurrence (5.1), then

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d, \\ \Theta(n^d \log n) & \text{if } a = b^d, \\ \Theta(n^{\log_b a}) & \text{if } a > b^d. \end{cases}$$

Vamos mudar (um pouco) o problema: **contando inversões**

Contexto: análise de ranking.

Vários sites utilizam *filtragem colaborativa* para “rankear” (dar prioridade) suas preferências.

Vamos mudar (um pouco) o problema: **contando inversões**

Contexto: análise de ranking.

Vários sites utilizam *filtragem colaborativa* para “rankear” (dar prioridade) suas preferências.

Vamos supor um sistema de ranking para catálogo de filmes. Vamos comparar 2 *rankings*:

- O site faz algum *ranking* tentando ordenar priorizando os filmes que eu prefiro;
- Eu faço meu próprio *ranking* dos filmes que eu prefiro

Vamos mudar (um pouco) o problema: **contando inversões**

Contexto: análise de ranking.

Vários sites utilizam *filtragem colaborativa* para "rankear" (dar prioridade) suas preferências.

Vamos supor um sistema de ranking para catálogo de filmes. Vamos comparar 2 *rankings*:

- O site faz algum *ranking* tentando ordenar priorizando os filmes que eu prefiro;
- Eu faço meu próprio *ranking* dos filmes que eu prefiro

Agora, precisamos de alguma métrica para ver quantos filmes ele ordenou errado.

Vamos mudar (um pouco) o problema: **contando inversões**

A **métrica de similaridade** pode ser a quantidade de inversões entre os dois rankings.

Minhas preferências: 1, 2, ..., n
Outras preferências: $a_1, a_2, \ldots, a_n$

Filmes estão **invertidos** se $i < j$, mas $a_i > a_j$

Vamos mudar (um pouco) o problema: **contando inversões**

A **métrica de similaridade** pode ser a quantidade de inversões entre os dois rankings.

Minhas preferências: 1, 2, ..., n
Outras preferências: $a_1, a_2, ..., a_n$

Filmes estão **invertidos** se $i < j$, mas $a_i > a_j$

Contando inversões

Dado um array **a**, uma inversão é formada por dois índices **i** e **j** tal que $a_i > a_j$.

O número total de inversões de um array indica quão longe um array está de ser totalmente ordenado.

Exemplo: [3, 1, 4, 2]

Contando inversões

Dado um array **a**, uma inversão é formada por dois índices **i** e **j** tal que $a_i > a_j$.

O número total de inversões de um array indica quão longe um array está de ser totalmente ordenado.

Exemplo: [3, 1, 4, 2]

Inversões (pelos índices): (1, 2), (1, 4), (3, 4)

Como resolver?

Força bruta

Testa todos os pares e confere!

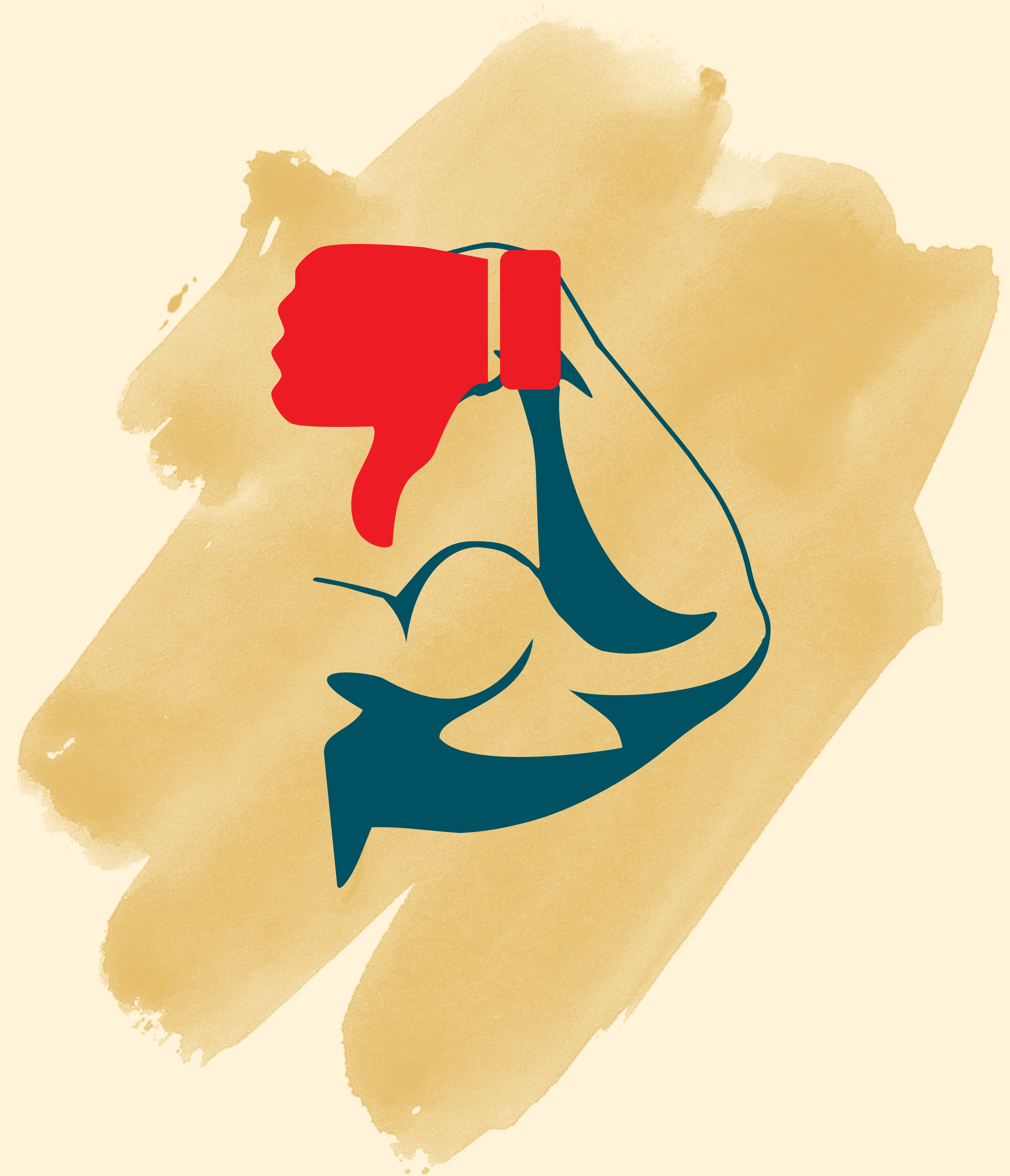
```
inversions = 0
for i = 1..n
    for j in i..n
        if a[i] > a[j]
            inversions += 1
```

Força bruta

Testa todos os pares e confere!

```
inversions = 0
for i in 1..n
    for j in i..n
        if a[i] > a[j]
            inversions += 1
```

$O(n^2)$



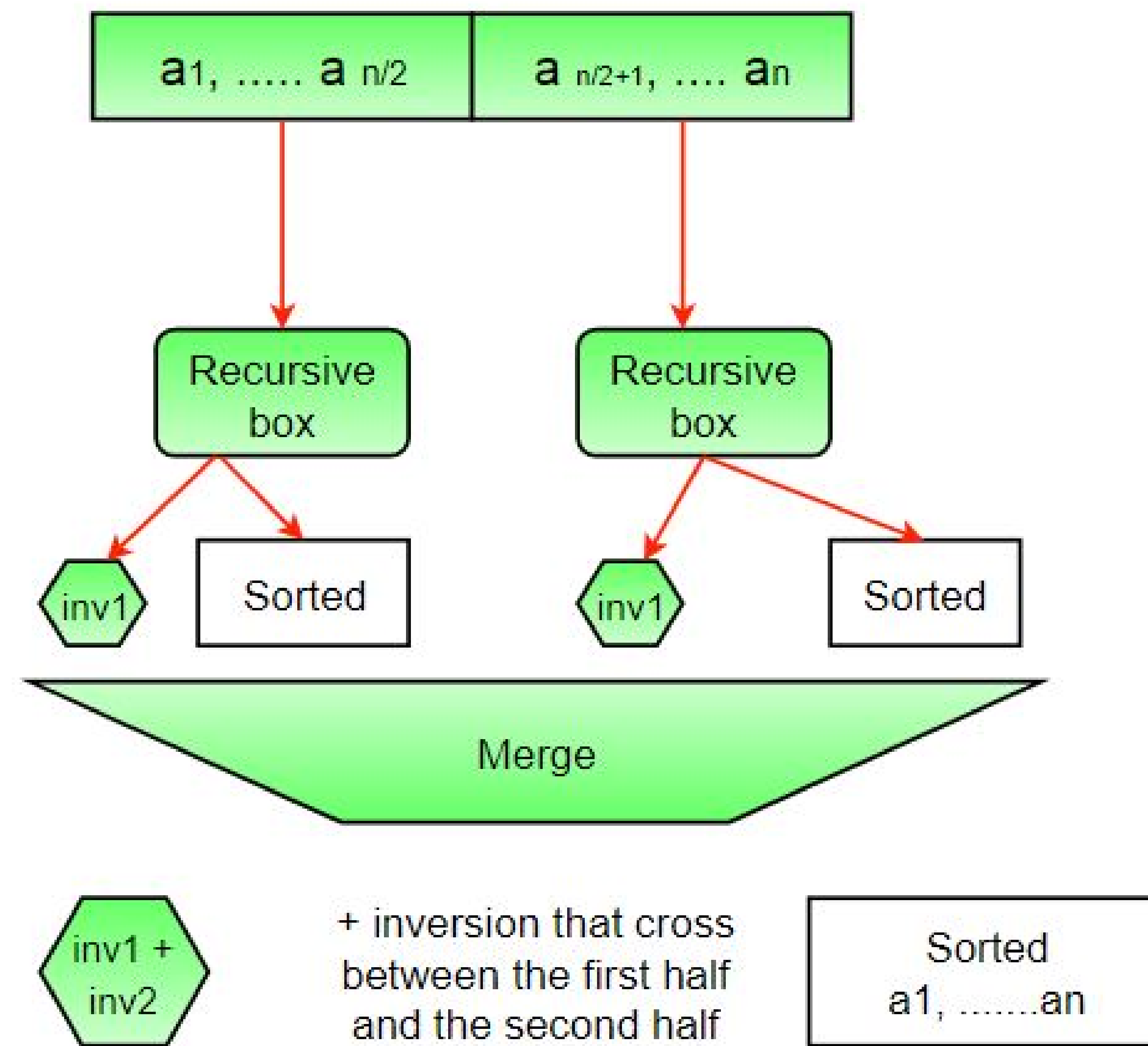
Contando inversões: divisão e conquista

Estratégia parecida com o **merge sort**:

Para o seguinte vetor:

- **Divisão:** divide o vetor em 2
- **Conquista:** calcular as inversões para cada metade
- **Combinar:** contar quantas recursões existem para os índices **i** que estão na primeira metade e **j** que estão na segunda metade (e ordenar o vetor)

Contando inversões: divisão e conquista



https://www.geeksforgeeks.org/inversion-count-in-array-using-merge-sort/

Contando Inversões: Implementação

Pré-condição. [Merge-and-Count] A e B ordenados.

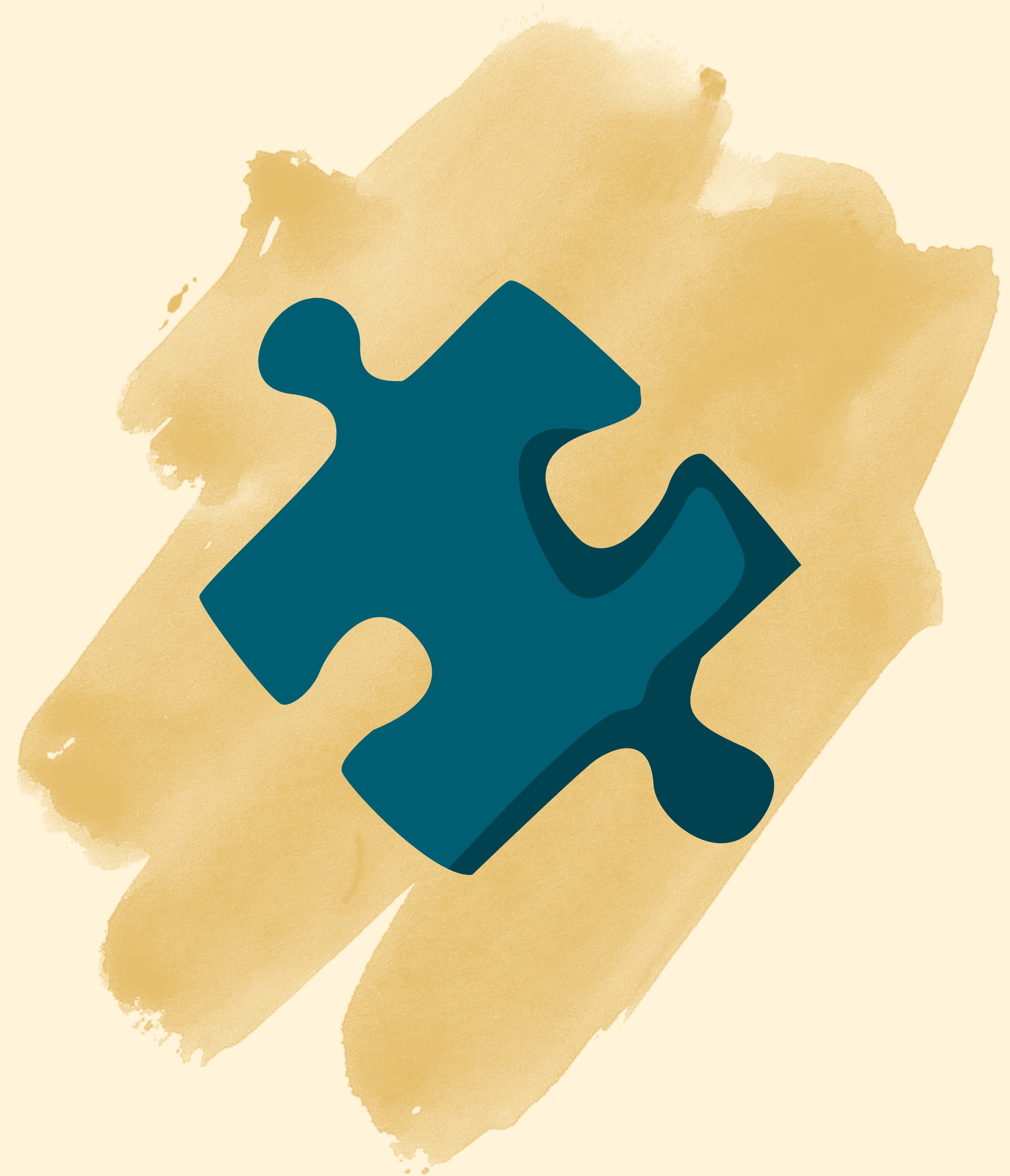
Pós-condição. [Sort-and-Count] L ordenado.

```
Sort-and-Count(L) {
   if list L has one element
      return 0 and the list L

   Divide the list into two halves A and B
   (rA, A) ← Sort-and-Count(A)
   (rB, B) ← Sort-and-Count(B)
   (r , L) ← Merge-and-Count(A, B)

   return r = rA + rB + r and the sorted list L
}
```

Outros exemplos

Alguns outros exemplos de problemas com soluções ***divisão e conquista***:

- Closest Pair of Points: encontrar o par de pontos mais próximos entre si em O(nlogn)
- Strassen's Algorithm: multiplicação de duas matrizes em O(n^2.8974) em vez de O(n^3)
- Cooley–Tukey Fast Fourier Transform (FFT) algorithm: calcular o FFT em O(nlogn)
- Karatsuba algorithm: multiplicação de BigIntegers (numeros com muitos dígitos)



Tem um problema muito mais conhecido que esses...

Binary search
1 3 4 6 7 8 10 13 14 18 19 21 24 37 40 45 71

Binary search
3 4 6 7 8 10 13
18 19 21 24 37 40 45 71

Busca binária (padrão)

Verificar se um elemento existe em um array **ordenado**

Binary Search

	0	1	2	3	4	5	6	7	8	9
Search 23	2	5	8	12	16	23	38	56	72	91
	L=0	1	2	3	M=4	5	6	7	8	H=9
23 > 16 take 2nd half	2	5	8	12	16	23	38	56	72	91
	0	1	2	3	4	L=5	6	M=7	8	H=9
23 < 56 take 1st half	2	5	8	12	16	23	38	56	72	91
	0	1	2	3	4	L=5, M=5	H=6	7	8	9
Found 23, Return 5	2	5	8	12	16	23	38	56	72	91

Busca Binária **na Resposta**

A mesma ideia da busca binária pode ser usada para resolver outros problemas...

A busca binária pode ser vista da seguinte forma:

- Começamos com um espaço de tamanho **n**;
- Sabemos que a resposta existe nesse intervalo (se existir);
- Cada iteração da busca binária corta o espaço de busca pela metade;
- O algoritmo testa O(logn) posições.

Vamos agora **mudar** o que estamos buscando.

Problema: **Fábrica**

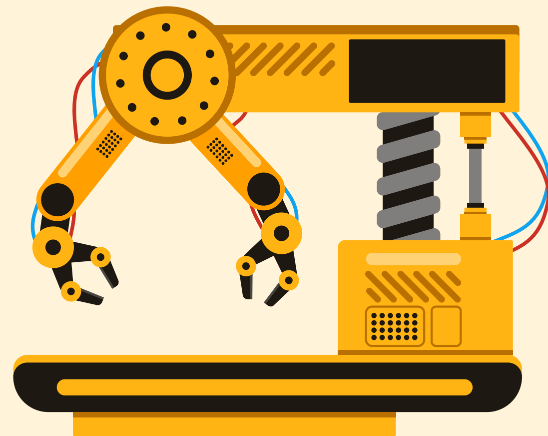
Uma fábrica possui **n** máquinas que podem ser utilizadas para fazer produtos. Seu objetivo é fabricar um total de **t** produtos.

Para cada máquina, você sabe o número de segundos que ela leva para fabricar um único produto. As máquinas podem trabalhar simultaneamente, e você pode decidir o agendamento que quiser.

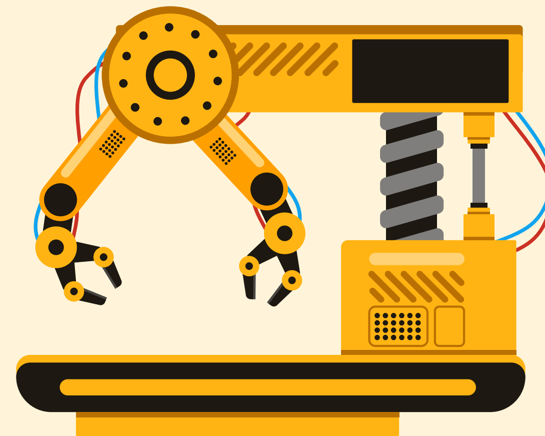
Qual é o menor tempo (inteiro) possível para fabricar **t** produtos?

Problema: Fábrica

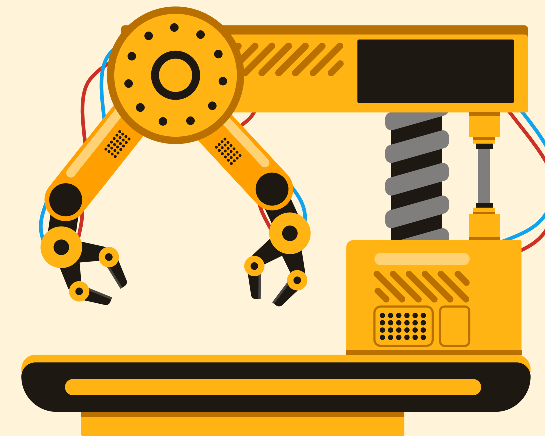
Exemplo com 3 máquinas (n=3) para fabricar 7 produtos (k=7)



3 s



2 s

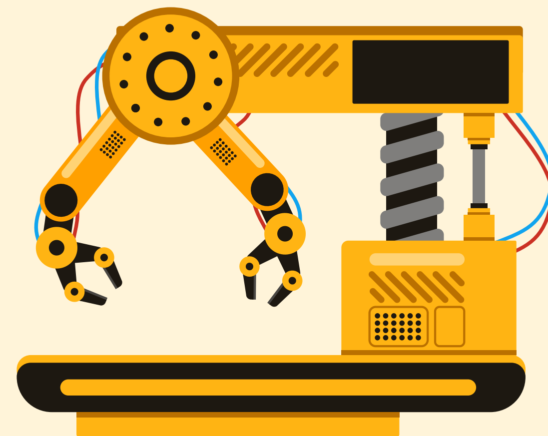


5 s

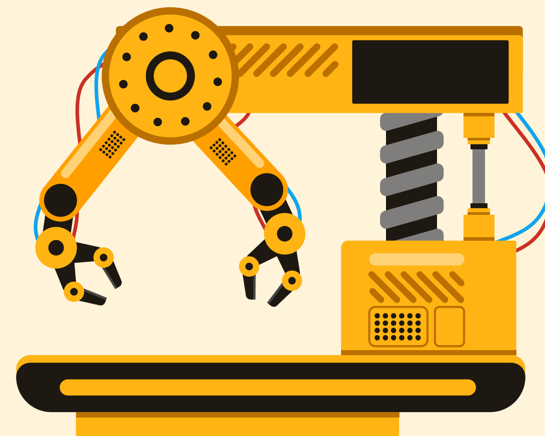
Precisamos calcular o **tempo mínimo** para fabricar 7 produtos. Como podemos pensar nisso?

Problema: <u>**Fábrica**</u>

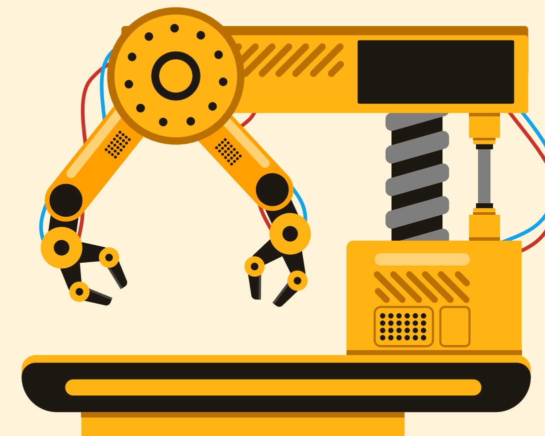
Exemplo com 3 máquinas (n=3) para fabricar 7 produtos (k=7)



3 s



2 s



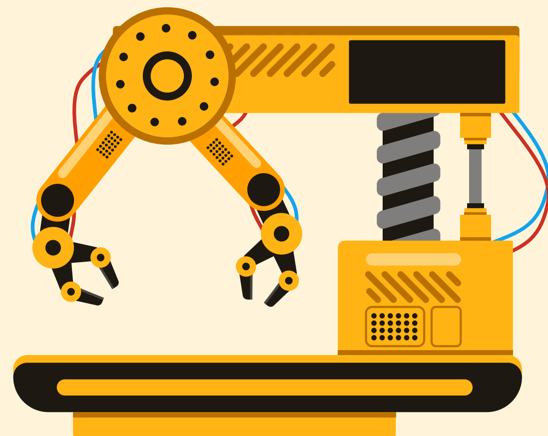
5 s

Precisamos calcular o **tempo mínimo** para fabricar 7 produtos. Como podemos pensar nisso?

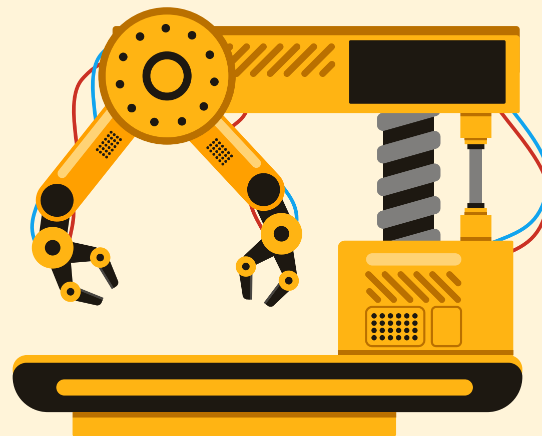
Conseguimos calcular quantos produtos são feitos dada qualquer quantia de tempo **t**.

Problema: <u>**Fábrica**</u>

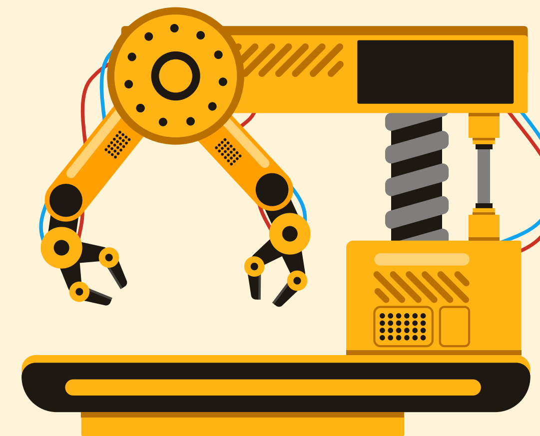
Exemplo com 3 máquinas (n=3) para fabricar 7 produtos (k=7)



3 s



2 s



5 s

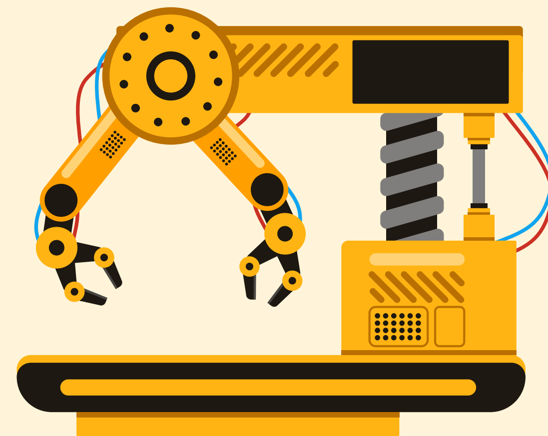
Precisamos calcular o **tempo mínimo** para fabricar 7 produtos. Como podemos pensar nisso?

Conseguimos calcular quantos produtos são feitos dada qualquer quantia de tempo **t**.

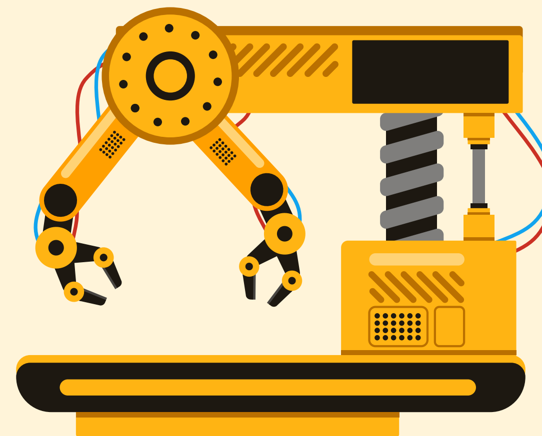
Por exemplo, em 5 segundos, conseguimos fazer 4 produtos: a primeira máquina faz 1, a segunda faz 2 e a terceira faz 1 produto (estratégia gulosa nessa parte).

Problema: **Fábrica**

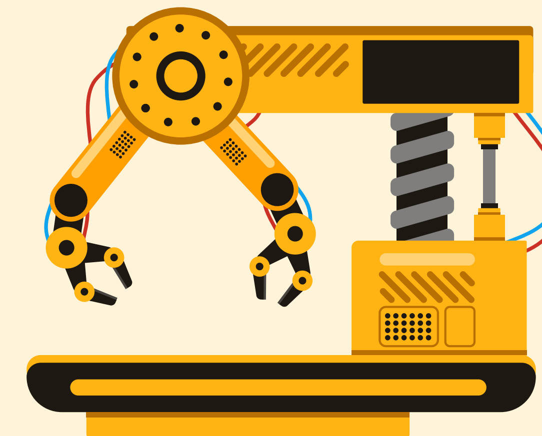
Exemplo com 3 máquinas (n=3) para fabricar 7 produtos (k=7)



3 s



2 s



5 s

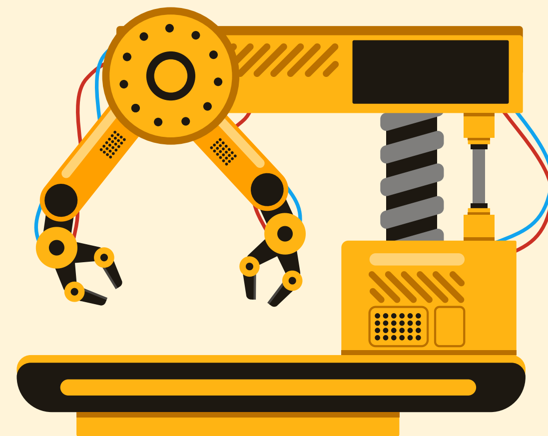
Precisamos calcular o **tempo mínimo** para fabricar 7 produtos. Como podemos pensar nisso?

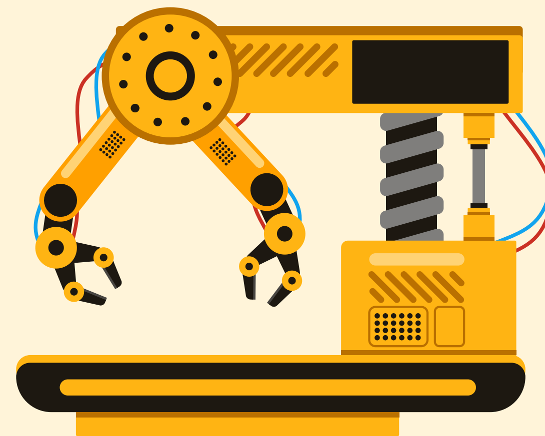
Qual a complexidade dessa função de *“teste”*?

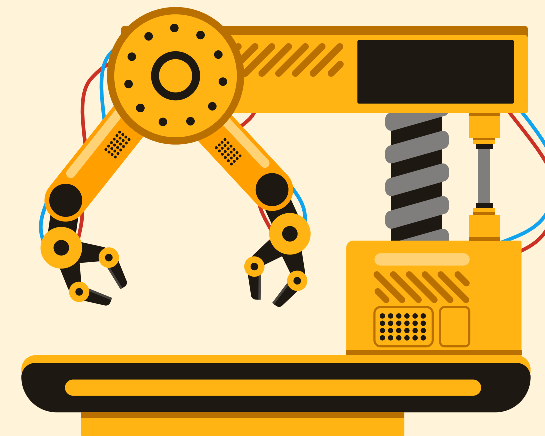
Como aplicar **busca binária** nisso?

Problema: **Fábrica**

Exemplo com 3 máquinas (n=3) para fabricar 7 produtos (k=7)


3 s


2 s

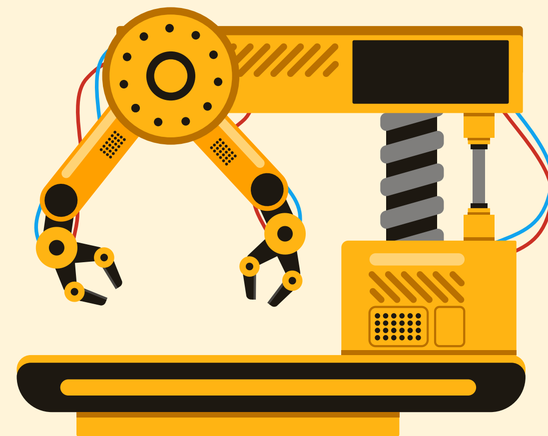

5 s

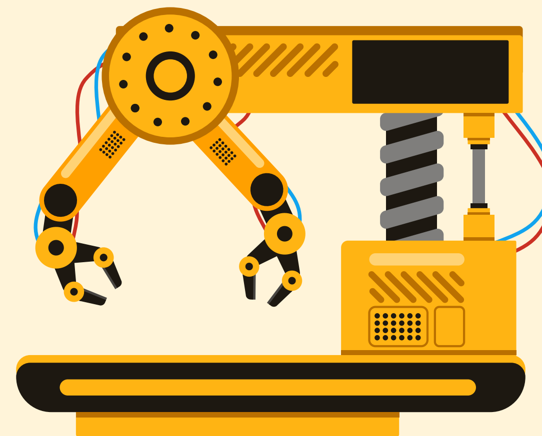
Precisamos calcular o **tempo mínimo** para fabricar 7 produtos. Como podemos pensar nisso?

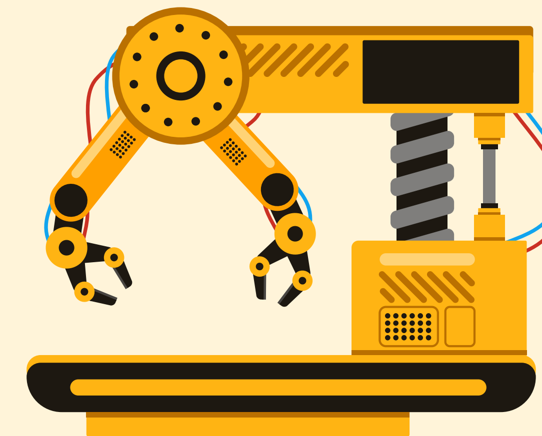
Com 1000000 segundos para a fábrica trabalhar, ela vai fazer pelo menos 7 produtos?

Problema: <u>Fábrica</u>

Exemplo com 3 máquinas (n=3) para fabricar 7 produtos (k=7)


3 s


2 s

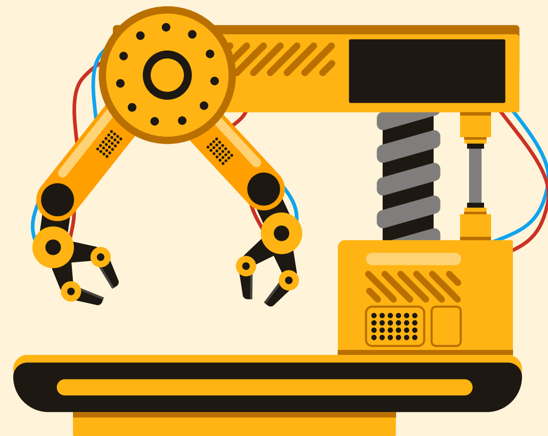

5 s

Precisamos calcular o **tempo mínimo** para fabricar 7 produtos. Como podemos pensar nisso?

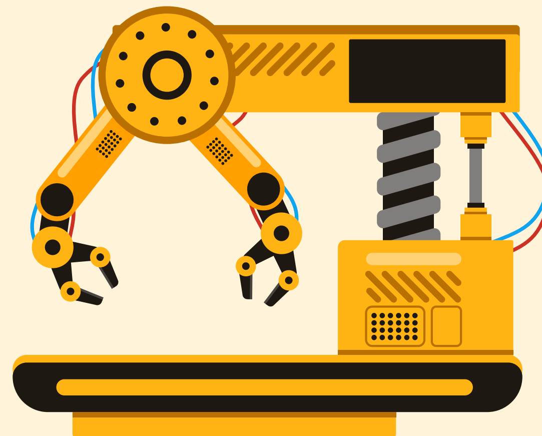
Com 1000000 segundos para a fábrica trabalhar, ela vai fazer pelo menos 7 produtos? ✓

Problema: **Fábrica**

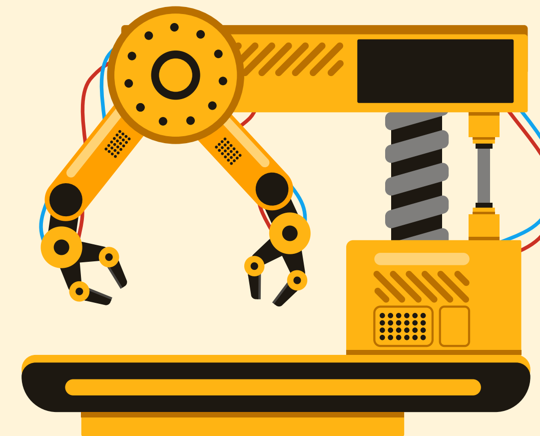
Exemplo com 3 máquinas (n=3) para fabricar 7 produtos (k=7)



3 s



2 s



5 s

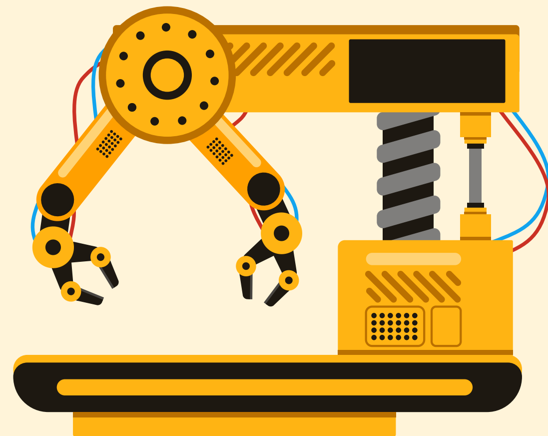
Precisamos calcular o **tempo mínimo** para fabricar 7 produtos. Como podemos pensar nisso?

Com 1000000 segundos para a fábrica trabalhar, ela vai fazer pelo menos 7 produtos? ✓

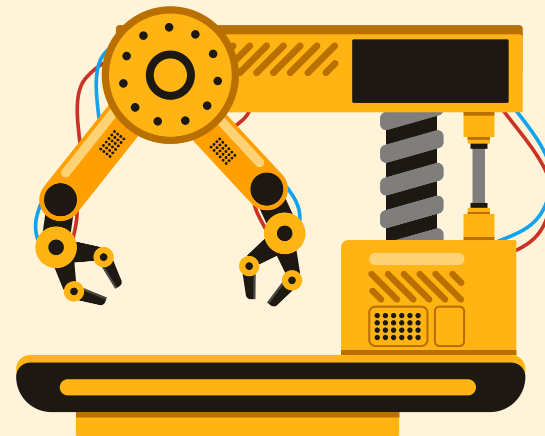
E com 1 segundo, ela consegue fazer pelo menos 7?

Problema: Fábrica

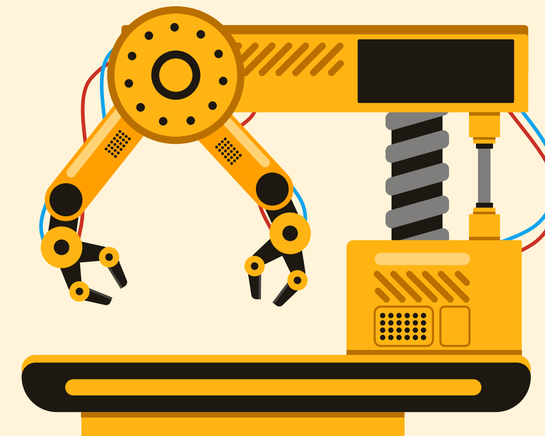
Exemplo com 3 máquinas (n=3) para fabricar 7 produtos (k=7)



3 s



2 s



5 s

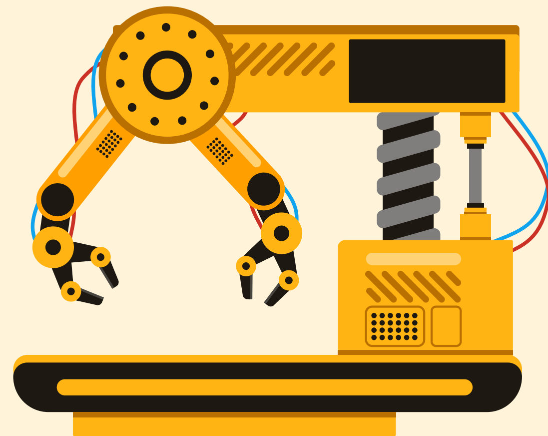
Precisamos calcular o **tempo mínimo** para fabricar 7 produtos. Como podemos pensar nisso?

Com 1000000 segundos para a fábrica trabalhar, ela vai fazer pelo menos 7 produtos? ✓

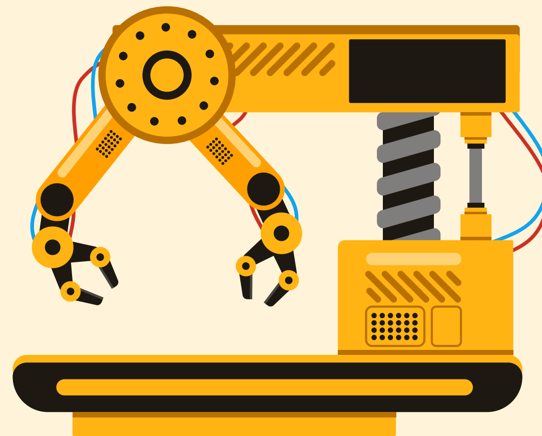
E com 1 segundo, ela consegue fazer pelo menos 7? ✗

Problema: **Fábrica**

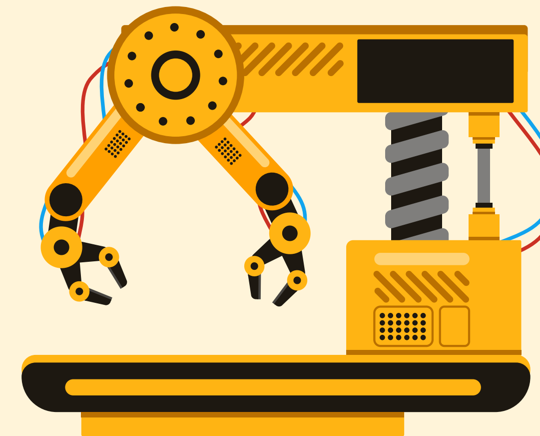
Exemplo com 3 máquinas (n=3) para fabricar 7 produtos (k=7)



3 s



2 s



5 s

Precisamos calcular o **tempo mínimo** para fabricar 7 produtos. Como podemos pensar nisso?

Com 1000000 segundos para a fábrica trabalhar, ela vai fazer pelo menos 7 produtos? ✓

E com 1 segundo, ela consegue fazer pelo menos 7? ✗

E com 500000? ...

Busca Binária
em Funções Monotônicas

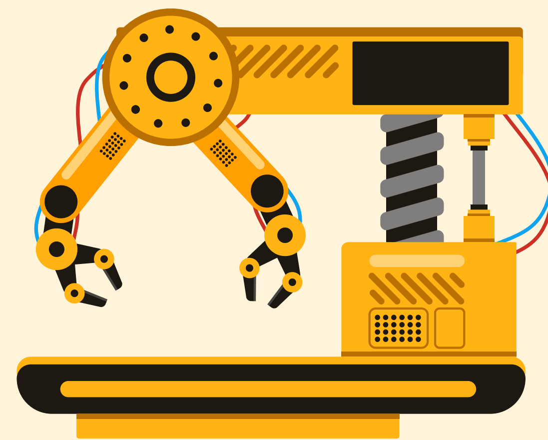
Seja **f(x)** uma função booleana monotônica. Podemos utilizar busca binária para calcular:

- O maior valor **x** tal que f(x) é verdadeiro;
- O menor valor **x** tal que f(x) é verdadeiro.

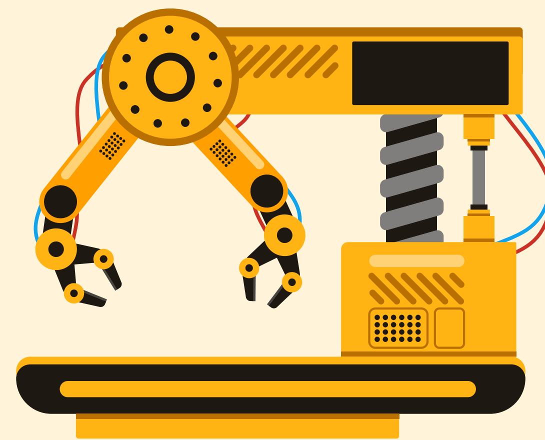
As outras possibilidades podem ser construídas a partir dessas.

Voltando ao exemplo da fábrica...

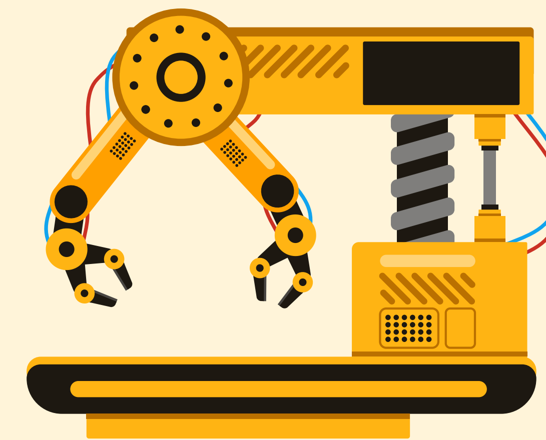
Exemplo com 3 máquinas (n=3) para fabricar 7 produtos (k=7)



3 s



2 s

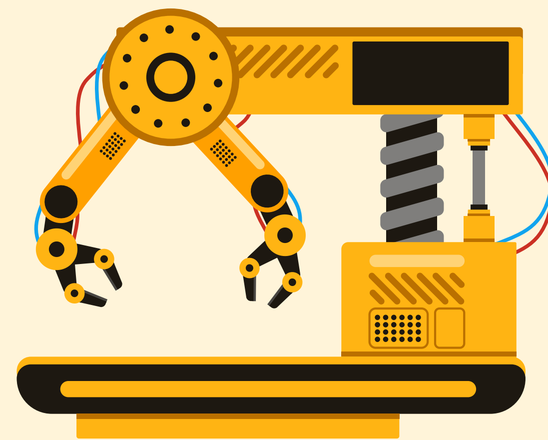


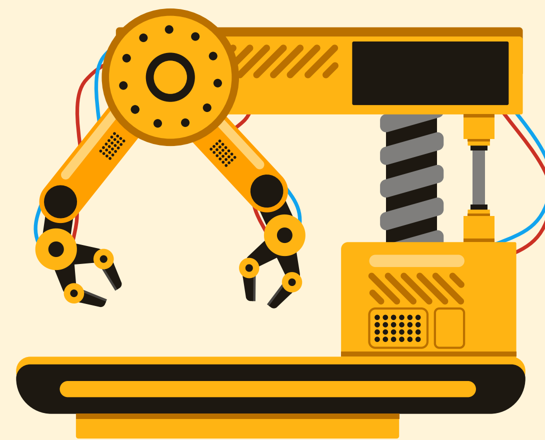
5 s

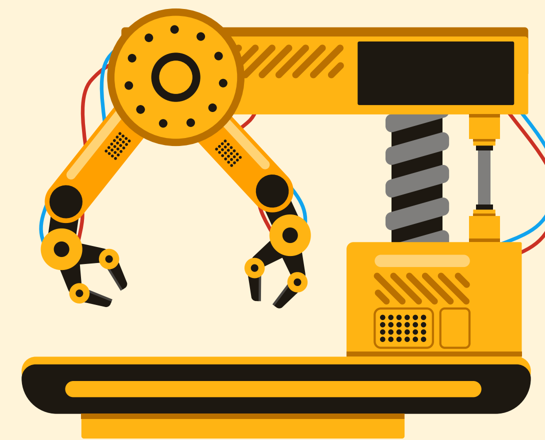
A resposta nesse caso é **8**. Em 8 segundos, a primeira máquina faz 2 produtos, a segunda faz 4 produtos e a terceira fabrica 1 produto, totalizando os 7 produtos necessários.

Voltando ao exemplo da fábrica...

Exemplo com 3 máquinas (n=3) para fabricar 7 produtos (k=7)


3 s

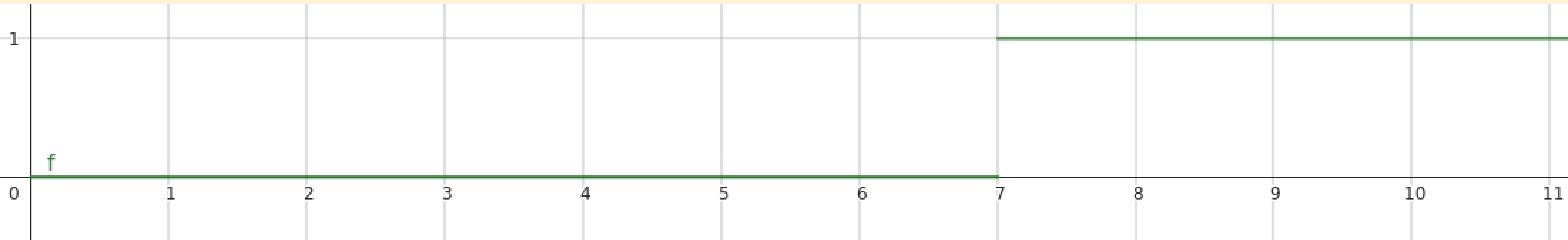

2 s


5 s

A resposta nesse caso é **8**. Em 8 segundos, a primeira máquina faz 2 produtos, a segunda faz 4 produtos e a terceira fabrica 1 produto, totalizando os 7 produtos necessários.

Note que é **impossível** fazer 7 produtos em **menos** de 8 segundos, e para todo o tempo maior que 8 segundos, conseguimos fabricar **7 ou mais** produtos.

A “função monotônica” nesse caso é essa aqui:



Para qualquer valor **menor que 7**, ela é **falsa**; para qualquer valor **maior ou igual a 7**, ela é **verdadeira**.

O que queremos achar com a busca binária é qual o **primeiro valor** que faz ela ser 1!

E a implementação?

Primeiramente vamos ver como implementar a verificação para um dado tempo **t**:

```
bool check(int tempo, vector<int>& a, int n, int k) {
    int cnt_feitos = 0;
    for (int i = 0; i < n; i++) {
        cnt_feitos += tempo / a[i];
    }
    return cnt_feitos >= k;
}
```

E a implementação?

Primeiramente vamos ver como implementar a verificação para um dado tempo **t**:

```
bool check(int tempo, vector<int>& a, int n, int k) {
    int cnt_feitos = 0;
    for (int i = 0; i < n; i++) {
        cnt_feitos += tempo / a[i];
    }
    return cnt_feitos >= k;
}
```

Complexidade: O(n)

E a implementação?

E agora a busca binária para achar o tempo certo:

```
int binary_search(vector<int>& a, int n, int k) {
    int lo = 1, hi = 1e9, mi;
    while(lo < hi) {
        mi = (lo + hi) / 2;
        if (check( tempo: mi,  &: a, n, k)) hi = mi;
        else lo = mi + 1;
    }
    return lo;
}
```

E a implementação?

E agora a busca binária para achar o tempo certo:

```
int binary_search(vector<int>& a, int n, int k) {
    int lo = 1, hi = 1e9, mi;
    while(lo < hi) {
        mi = (lo + hi) / 2;
        if (check( tempo: mi,  &: a, n, k)) hi = mi;
        else lo = mi + 1;
    }
    return lo;
}
```

Complexidade: ???

Complexidade

Suponha que a função de verificação *check* seja O(**g(x)**).

A complexidade total da busca binária, no intervalo [lo, hi], será:

$$O(g(x) * \log(hi - lo))$$

Busca Binária com Ponto Flutuante

E se **x** puder ser um número de ponto flutuante?

Por exemplo, caso a resposta no problema da Fábrica pudesse ser uma quantidade decimal de segundos. Como fazer?

Busca Binária com Ponto Flutuante

Jeito 1: **epsilon**

```
const double EPS = 1e-9; // limiar de erro

double lo = 0, hi = 1e9, mi;
while(hi - lo > EPS) {
    mi = (lo + hi) / 2;
    if (check(mi)) lo = mi;
    else hi = mi;
}

return lo;
```

Busca Binária com Ponto Flutuante

Jeito 2: **limite de iterações**

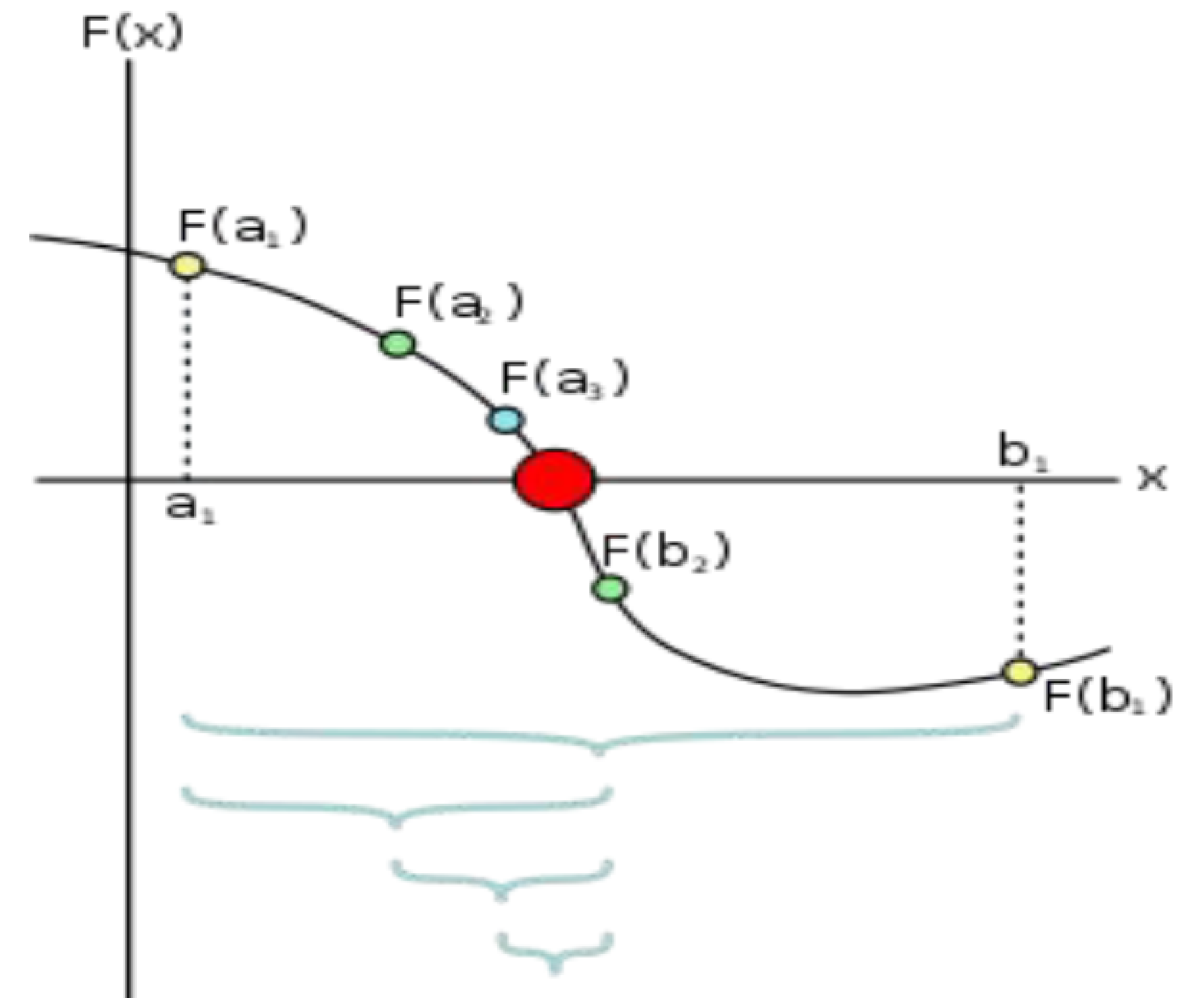
```
const int CNT = 200; // limite de iterações

double lo = 0, hi = 1e9, mi;
for (int it = 0; it < CNT; it++) {
    mi = (lo + hi) / 2;
    if (check(mi)) lo = mi;
    else hi = mi;
}

return lo;
```

Método da bisseção

Estimar a raíz de uma função f(x)

- Queremos estimar, dentro de um intervalo **[a, b]**, onde a raíz se encontra;
- f(a) e f(b) tem sinais opostos;
- f(x) deve ser contínua.

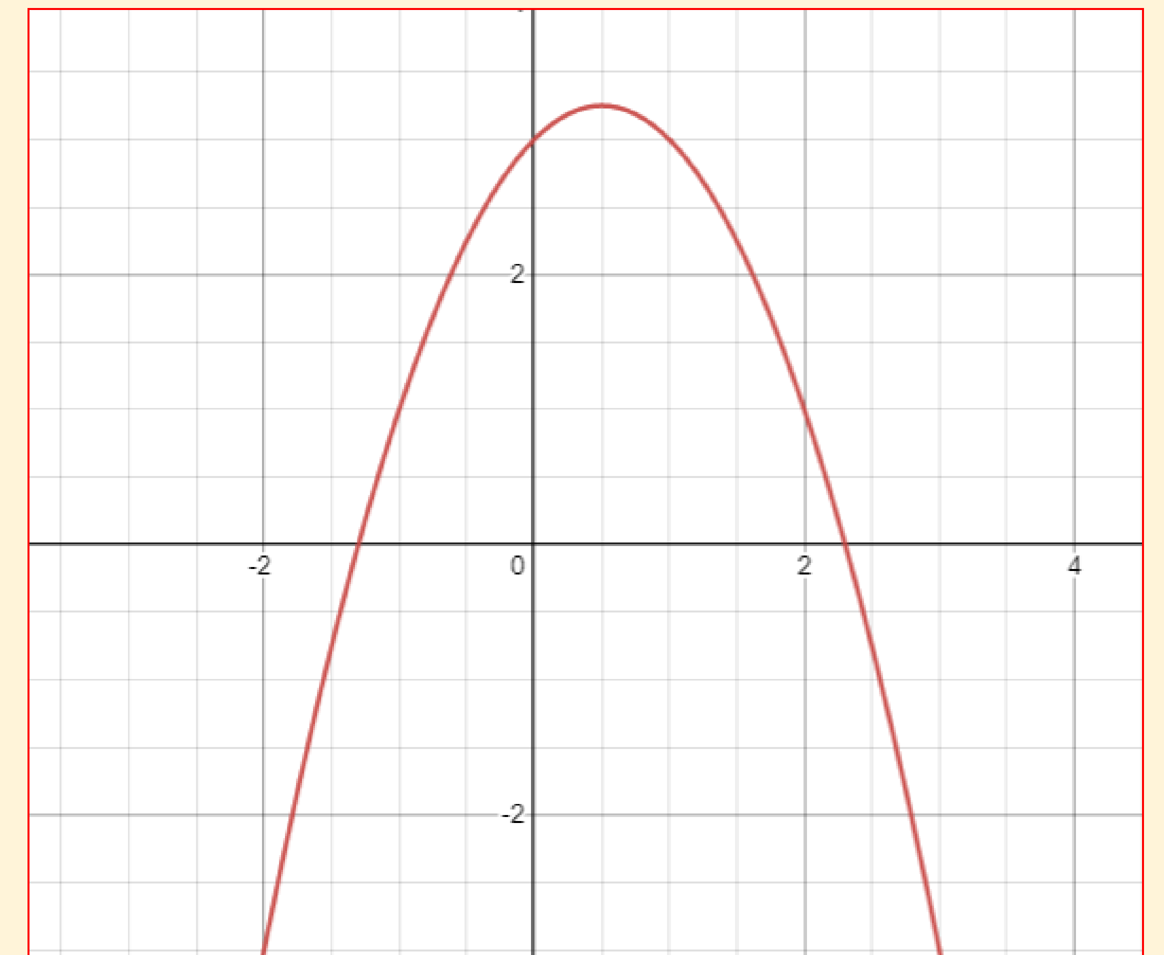
No final, é tudo a mesma coisa...



Busca Ternária

Outro algoritmo interessante é a busca ternária.

Agora, queremos achar pontos de máximo (ou mínimo) em funções “parabólicas”.



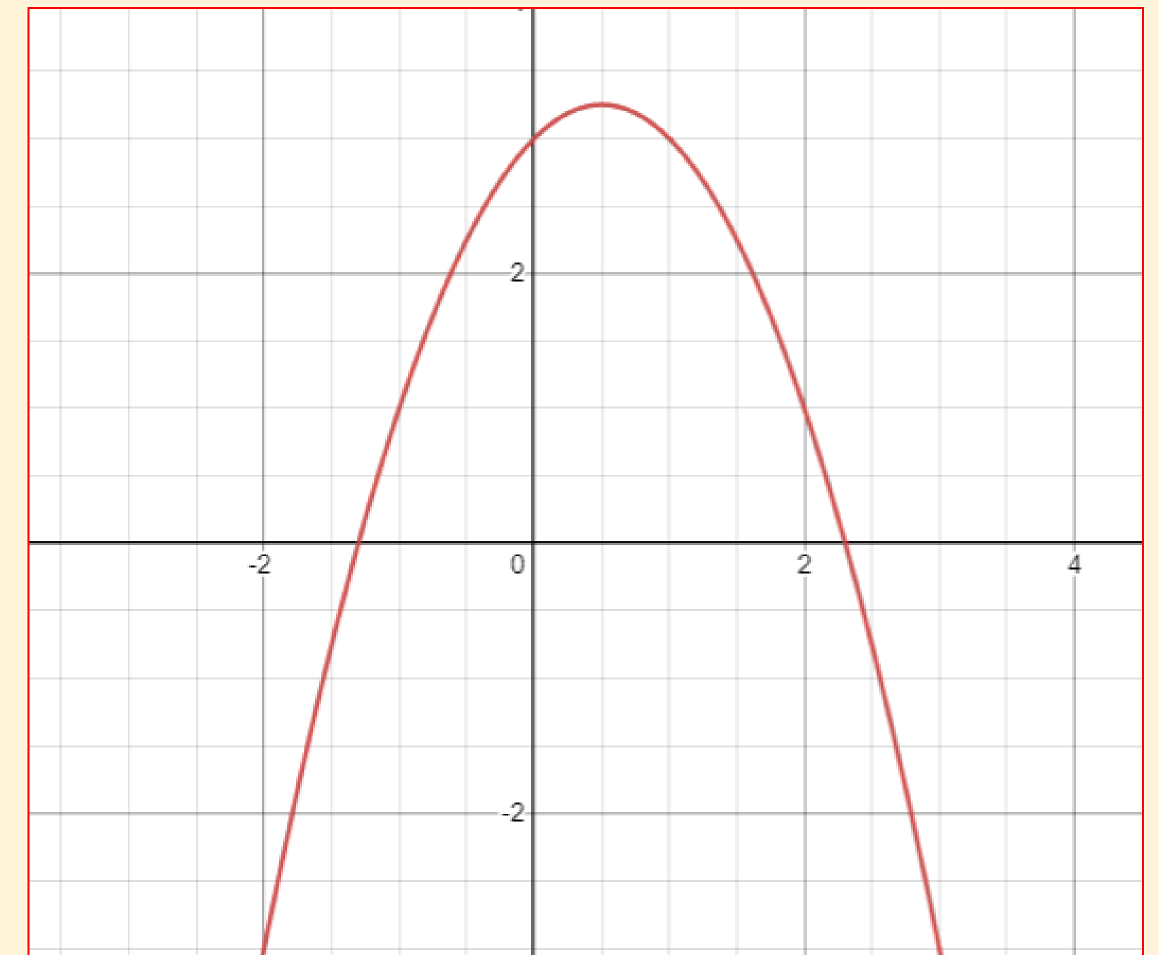
Busca Ternária

Em vez de “cortar” o array pela metade, ele é dividido em três.

Desafio: Como você resolveria o problema abaixo utilizando busca ternária?

Encontrar o valor **x** que maximize a função **f(x) = 3+x-x*x.**

Desafio 2: Como você resolveria o mesmo problema utilizando cálculo 1?



```
double f(double x) {
    return 3 + x - x * x;
}

double ternary_search() {
    const double EPS = 1e-9;

    double lo = -1e9, hi = 1e9, m1, m2;
    while(hi - lo > EPS) {
        m1 = lo + (hi - lo) / 3;
        m2 = lo + 2 * (hi - lo) / 3;
        if (f(m1) < f(m2)) lo = m1;
        else hi = m2;
    }
    return lo;
}
```

Desafio 2:

Maximizar **$f(x) = 3+x-x*x$:**

$f'(x) = 1 - 2x$

Para achar o ponto de inflexão, iguala a derivada a zero:

$f'(x) = 1 - 2x = 0$
$x = 0.5$