O algoritmo DC

Depois que a gente escreve um algoritmo de divisão e conquista, a gente pode obter o seu tempo de execução por meio da anotação do código.

Para isso, a gente faz a anotação do código assim

```
T(n) =   func Mergesort-DC ( L, inicio, fim )

              Se ( inicio = fim )
   O(1)       |    Retorna
              meio ← ( inicio + fim ) / 2
T(n/2) ←      Mergesort ( L, inicio, meio )
T(n/2) ←      Mergesort ( L, meio+1, fim )
   O(n)       Intercala ( L, inicio, meio, fim )
```

Quer dizer, a ideia é que $T(n)$ é a função que especifica o tempo de execução do algoritmo.

E as anotações estão dizendo que

$$T(n) \quad = \quad 2 \cdot T\,(n/2) \quad + \quad O(n)$$

Sim! uma equação de recorrência — (*ou um rocambole ...*)

E para obter a sua solução, basta desenrolar o rocambole

$$\begin{aligned}
T(n) \quad &= \quad 2 \cdot T\left(\tfrac{n}{2}\right) \quad + \quad n \\
&= \quad 2 \cdot \left[\, 2 \cdot T\left(\tfrac{n}{2^2}\right) \quad + \quad \tfrac{n}{2} \,\right] \quad + \quad n \\
&= \quad 2^2 \cdot T\left(\tfrac{n}{2^2}\right) \quad + \quad n \quad + \quad n \\
&= \quad 2^3 \cdot T\left(\tfrac{n}{2^3}\right) \quad + \quad n \quad + \quad n \quad + \quad n \\
&= \quad \ldots \\
&= \quad 2^{\log n} \cdot T\left(\tfrac{n}{2^{\log n}}\right) \quad + \quad \underbrace{n \quad + \quad \ldots \quad + \quad n}_{\log\, n} \\
&= \quad n \cdot \cancel{T(1)} \quad + \quad n \quad + \quad \ldots \quad + \quad n \\
&= \quad O(n \log n)
\end{aligned}$$

Vejamos mais exemplos.

Exemplos

a. Busca binária

Abaixo nós temos o código da busca binária já anotado

```
T(n) =   func BB ( k, L, inicio, fim )

              Se ( inicio > fim )          Retorna ( 0 )
   O(1)       |
              meio ← ( inicio + fim ) / 2
              Se ( L[meio] = k )           Retorna ( 1 )
T(n/2) ←      SSe ( L[meio] > k )          Retorna ( BB ( k, L, inicio, meio-1 ) )
T(n/2) ←      Senão                          Retorna ( BB ( k, L, meio+1, fim ) )
```

E aqui, a esperteza é notar que no máximo uma chamada recursiva é realizada.

Logo, a equação de recorrência fica assim

$$T(n) \quad = \quad T\left(\frac{n}{2}\right) \quad + \quad O(1)$$

E agora, é só desenrolar o rocambole

$$\begin{aligned}
T(n) \quad &= \quad T\left(\tfrac{n}{2}\right) \quad + \quad 1 \\
&= \quad T\left(\tfrac{n}{2^2}\right) \quad + \quad 1 \quad + \quad 1 \\
&= \quad T\left(\tfrac{n}{2^3}\right) \quad + \quad 1 \quad + \quad 1 \quad + \quad 1 \\
&= \quad \ldots \\
&= \quad T\left(\tfrac{n}{2^{\log n}}\right) \quad + \quad \underbrace{1 \quad + \quad \ldots \quad + \quad 1}_{\log\, n} \\
&= \quad O(\log n)
\end{aligned}$$

◇

b. Ordenação 2/3

Imagine que L é uma lista de tamanho n que contém números inteiros

E imagine que o problema é

→ *Colocar a lista* L *em ordem crescente*

Daí, uma ideia é fazer o seguinte

1. Ordenar os primeiros $2n/3$ elementos

2. Depois ordenar os últimos $2n/3$ elementos

3. E depois ordenar os primeiros $2n/3$ elementos outra vez

— (*onde cada ordenação é feita recursivamente pelo mesmo método*)

Ao final do processo, a lista está completamente ordenada — (*porque?*)

E isso nos dá mais um algoritmo de divisão e conquista.

Mas, será que isso é uma boa ideia?

Bom, para descobrir isso nós vamos analisar o tempo de execução do algoritmo.

E para isso, nós escrevemos e anotamos o seu pseudo-código

```
T(n) =   func Ordenação-2/3 ( L, inicio, fim )

              Se ( inicio ≥ fim )          Retorna
   O(1)       |    T1 ← ( 2*inicio + fim ) / 3
              |    T2 ← ( inicio + 2*fim ) / 3
T(2n/3) ←     Ordenação-2/3 ( L, inicio, T2 )
T(2n/3) ←     Ordenação-2/3 ( L, T1+1, fim )
T(2n/3) ←     Ordenação-2/3 ( L, inicio, T2 )
```

Com base nas anotações a gente obtém a seguinte equação de recorrência

$$T(n) \quad = \quad 3 \cdot T\left(\frac{2n}{3}\right) \quad + \quad O(1)$$

E agora, é só desenrolar o rocambole

$$\begin{aligned}
T(n) \quad &= \quad 3 \cdot T\left(\tfrac{2n}{3}\right) \quad + \quad 1 \\
&= \quad 3 \cdot \left[\, 3 \cdot T\left(\tfrac{n}{(3/2)^2}\right) \quad + \quad 1 \,\right] \quad + \quad 1 \\
&= \quad 3^2 \cdot T\left(\tfrac{n}{(3/2)^2}\right) \quad + \quad 3 \quad + \quad 1 \\
&= \quad 3^3 \cdot T\left(\tfrac{n}{(3/2)^3}\right) \quad + \quad 3^2 \quad + \quad 3 \quad + \quad 1 \\
&= \quad \ldots \\
&= \quad 3^{\log_{3/2}(n)} \cdot \cancel{T(1)} \quad + \quad 3^{\log_{3/2}(n)-1} \quad + \quad \ldots \quad + \quad 3 \quad + \quad 1
\end{aligned}$$

Quer dizer, nós chegamos a

$$T(n) \quad = \quad 1 \quad + \quad 3 \quad + \quad 3^2 \quad + \quad \ldots \quad + \quad 3^{\log_{3/2}(n)}$$

— (*uma soma de PG ...*)

Então agora, a gente usa o truque da soma de PG

$$S \quad = \quad 1 \quad + \quad 3 \quad + \quad 3^2 \quad + \quad \ldots \quad + \quad 3^{\log_{3/2}(n)}$$

Quer dizer, o truque é colocar o 3 em evidência do lado direito

$$S \quad = \quad 1 \quad + \quad 3 \cdot \left[\, 1 \quad + \quad 3 \quad + \quad \ldots \quad + \quad 3^{\log_{3/2}(n)-1} \,\right]$$

E ver que o S apareceu dentro dos colchetes — (*ou quase isso ...*)

$$S \quad = \quad 1 \quad + \quad 3 \cdot \left[\, S \quad - \quad 3^{\log_{3/2}(n)} \,\right]$$

O que nos dá

$$S \quad = \quad \frac{3^{\log_{3/2}(n)+1} \quad - \quad 1}{2} \quad = \quad \frac{3}{2} \cdot 3^{\log_{3/2}(n)} \quad - \quad \frac{1}{2}$$

Nesse ponto, a gente olha para o termo

$$3^{\log_{3/2}(n)}$$

E nota que o 3 pode ser escrito assim

$$3 \quad = \quad \big(3/2\big)^{\log_{3/2}(3)}$$

Então, a gente pode reescrever o nosso termo assim

$$\Big(\big(3/2\big)^{\log_{3/2}(3)} \Big)^{\log_{3/2}(n)}$$

e depois assim

$$\big(3/2\big)^{\log_{3/2}(3) \cdot \log_{3/2}(n)}$$

e depois assim

$$\Big(\big(3/2\big)^{\log_{3/2}(n)} \Big)^{\log_{3/2}(3)}$$

e depois assim

$$n^{\log_{3/2}(3)}$$

e depois assim

$$n^{2{,}7}$$

E fazendo isso, a gente descobre que o nosso algoritmo é uma grande porcaria.

◇