Ordenação: Quick Sort

Partição de Vetor

Resolver o problema da Partição de Vetor serve para projetar o Quicksort.

Partição de Vetor

Instância: (v, a, b, x) onde v é um vetor indexado por $[a..b]$ e x é um valor.

Resposta: modifica v de forma a garantir a existência de um índice $m \in [a-1..b]$ tal que

$$v[i] \leq x, \forall i \in [a..m],$$

e

$$x < v[i], \forall i \in [m+1..b].$$

Devolve este índice m.

$\leq x$	x	$> x$

- x é chamado de *pivô*.

Particiona(v, a, b, x)

```
m ← a − 1
i ← a
Para i ← a to b
    Se v[i] ≤ x
        m ← m + 1
        Troca(v, m, i)
Devolva m
```

Executar Particiona$(v, 1, 6, v[6])$

i	1	2	3	4	5	6
$v[i]$	6	10	13	5	8	3

Importante: após execução do Particiona, x fica na posição que deveria estar se o vetor estivesse ordenado.

Análise do Particiona

$C_P(n)$: número de comparações com elementos do vetor em instâncias de tamanho n.

As comparações são feitas sempre que o laço é executado. O laço é executado n vezes, portanto, $C_P(n) = n$.

Quicksort

É um algoritmo projetado usando a técnica "divisão e conquista".

```
Ordena_Q(v, a, b)
Se a ≥ b
    Devolva v
m ← Particiona(v, a, b, v[b])
Ordena_Q(v, a, m − 1)
Ordena_Q(v, m + 1, b)
```

Análise

$C(n)$: número de comparações com elementos de vetor feitas pelo Quicksort.

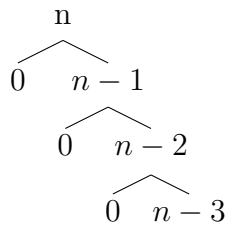
$$C(n) = \begin{cases} 0, & \text{se } n \leq 1 \\ C(k) + C(n-k-1) + C_P(n), & \text{se } n > 1 \end{cases}$$

onde k e $n-k-1$ são os tamanhos das partições obtidas. Substituindo $C_P(n) = n$,

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1 \\ C(k) + C(n-k-1) + n, & \text{se } n > 1 \end{cases}$$

Pior Caso

Um **caso ruim** é quando, por exemplo, o Particiona produz partições de tamanho 0 e $n-1$ (vetor já ordenado).



Esse caso ruim é o **pior caso**?

- Sim, mas não vamos provar isso aqui (só em Análise de Algoritmos).

Assim, a relação de recorrência fica:

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1 \\ C(0) + C(n-1) + n, & \text{se } n > 1 \end{cases}$$

ou

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1 \\ C(n-1) + n, & \text{se } n > 1 \end{cases}$$

Resolvendo essa recorrência:

$$\begin{aligned} C(n) &= C(n-1) + n \\ &= C(n-2) + (n-1) + n \\ &\vdots \\ &= C(1) + 2 + \ldots + n \\ &= n(n+1)/2 - 1 = (n^2 + n - 2)/2 = \frac{n^2}{2}(1 + 1/n + 2/n^2) \approx \frac{n^2}{2} \end{aligned}$$

Melhor Caso

Ocorre quando a Particiona produz as duas partições com tamanho "igual".

Caso n par:

$\lfloor n/2 \rfloor$	**x**	$\lceil n/2 \rceil - 1$

Caso n ímpar:

$\lfloor n/2 \rfloor$	**x**	$\lceil n/2 \rceil - 1$

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1 \\ C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil - 1) + n, & \text{se } n > 1 \end{cases}$$

Teorema. *A relação de recorrência:*

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1 \\ C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil - 1) + n, & \text{se } n > 1 \end{cases}$$

tem como solução $C(n) \approx n \log_2 n$.

Resolver essa recorrência é **difícil**, será visto em Análise de Algoritmos.

Mas assumindo simplificações, podemos ter uma ideia da demonstração do teorema.

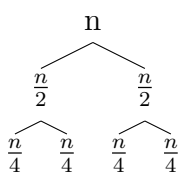
Simplificação 1: Assumimos que n é potência de 2, ou seja, $n = 2^k$.

Simplificação 2: $C(\lceil n/2 \rceil - 1)$ fazemos ser igual a $C(\lceil n/2 \rceil)$.

Ambas simplificações tornam as contas mais fáceis. Em particular, a 1ª simplificação serve para tirar pisos e tetos. Assim, a relação de recorrência simplificada é

$$C(n) = \begin{cases} 0, & \text{se } n \leq 1 \\ 2C(\frac{n}{2}) + n, & \text{se } n > 1 \end{cases}$$

$\log n$



Resolvendo a recorrência:

$$\begin{aligned} C(n) &= 2C(n/2) + n \\ &= 2^2 C(\frac{n}{2^2}) + n + n \\ &= 2^3 C(\frac{n}{2^3}) + n + n \\ &\vdots \\ &= 2^{\log_2 n} C(1) + n \log_2 n \\ &= n \log_2 n. \end{aligned}$$

Na prática: O Quicksort é um dos mais eficientes algoritmos de ordenação.

Na teoria: Seu pior caso é aproximadamente igual aos dos algoritmos mais simples de ordenação.