GA-026: Algoritmos I

Prof. Luiz Gadelha

Programa de Pós-Graduação em Modelagem Computacional, P4/2019
Laboratório Nacional de Computação Científica

8 de outubro de 2019

Quicksort

- O **Quicksort** segue a estratégia de dividir para conquistar para ordenar um subvetor $A[p, ..., r]$:
  - **Dividir**: particionar o subvetor $A[p, ..., r]$ em dois subvetores $A[p, ..., q-1]$ e $A[q+1, ..., r]$ de forma que os elementos de $A[p, ..., q-1]$ são menores ou iguais a $A[q]$, que por sua vez é menor ou igual aos elementos de $A[q+1, ..., r]$;
  - **Conquistar**: ordenar os dois subvetores, $A[p, ..., q-1]$ e $A[q+1, ..., r]$ recursivamente com o Quicksort;
  - **Combinar**: essa etapa não é necessária pois os subvetores já estão ordenados.

Quicksort

```
QUICKSORT(A, p, r)
1  if p < r
2      q = PARTITION(A, p, r)
3      QUICKSORT(A, p, q − 1)
4      QUICKSORT(A, q + 1, r)
```

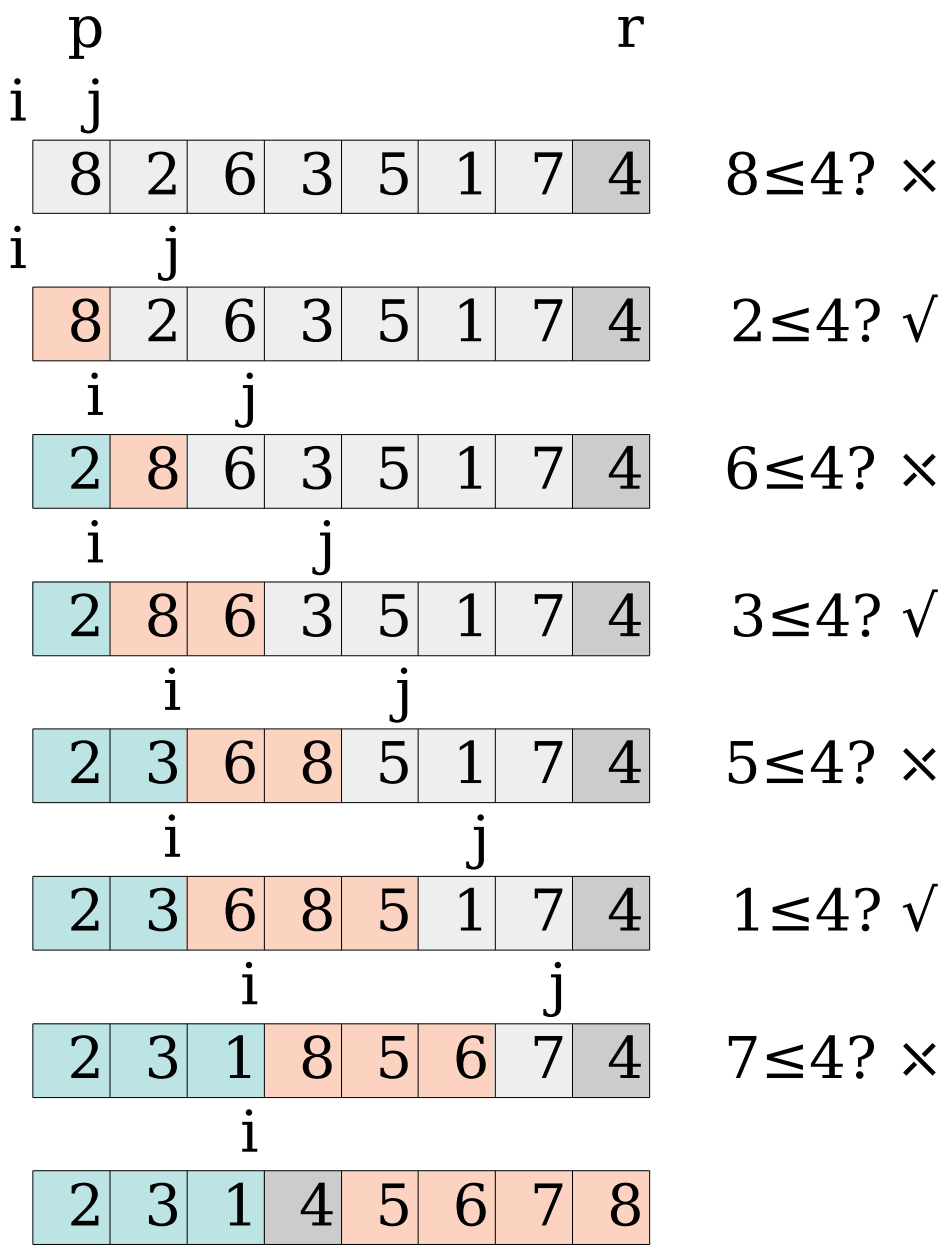
- Fonte: Cormen, T., Leiserson, C., Rivest, R., Stein, C. (2009). Introduction to Algorithms. MIT Press.

Quicksort

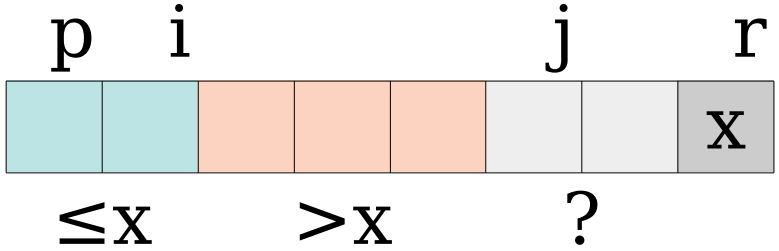
```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4       if A[j] ≤ x
5            i = i + 1
6            exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```

- Fonte: Cormen, T., Leiserson, C., Rivest, R., Stein, C. (2009). Introduction to Algorithms. MIT Press.

Quicksort

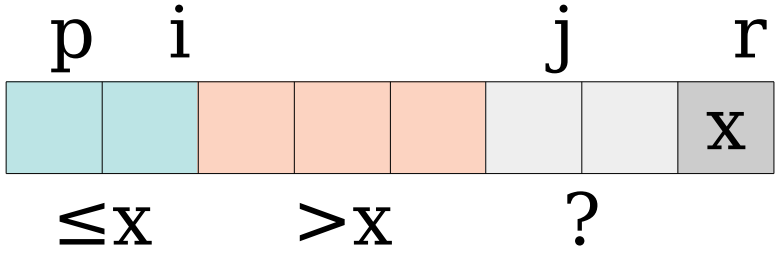

p
r
i
j
8 2 6 3 5 1 7 4
8≤4? ×
i
j
8 2 6 3 5 1 7 4
2≤4? √
i
j
2 8 6 3 5 1 7 4
6≤4? ×
i
j
2 8 6 3 5 1 7 4
3≤4? √
i
j
2 3 6 8 5 1 7 4
5≤4? ×
i
j
2 3 6 8 5 1 7 4
1≤4? √
i
j
2 3 1 8 5 6 7 4
7≤4? ×
i
2 3 1 4 5 6 7 8

Quicksort - Correção



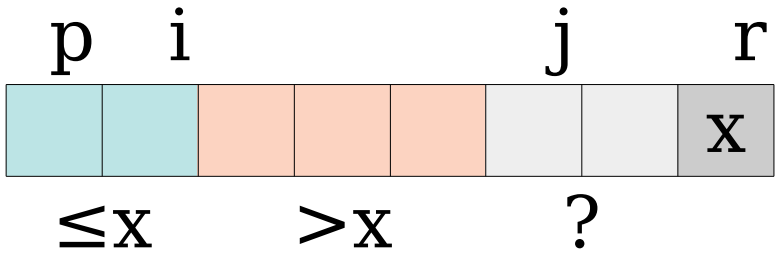
- **Invariante de laço**. No início de cada laço (l. 3–6), para um índice de vetor k:
  1. Se $p \leq k \leq i \Rightarrow A[k] \leq x$;
  2. Se $i + 1 \leq k \leq j - 1 \Rightarrow A[k] > x$;
  3. Se $k = r \Rightarrow A[k] = x$.
- Os índices entre j e $r - 1$ não são considerados pois, nesse caso, a relação com o pivô x é indeterminada.

Quicksort - Correção



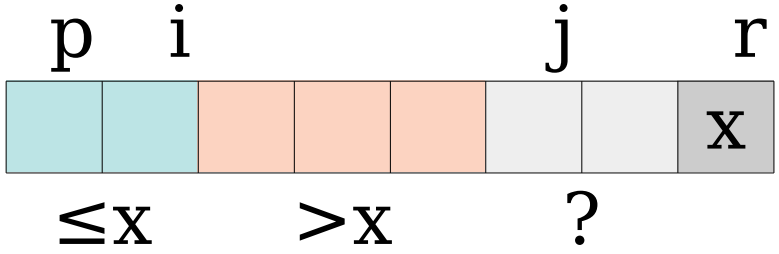
- **Inicialização**. Antes do primeiro laço, $i = p - 1$ e $j = p$:
  $\Rightarrow$ Não há valores entre p e i e entre $i + 1$ e $j - 1$ (os subvetores são vazios);
  $\Rightarrow$ As condições do IdL valem.

Quicksort - Correção



- **Manutenção**. Duas possibilidades:
  - $A[j] > x$:
    - j é incrementado
    - a condição 2 ($i + 1 \leq k \leq j - 1 \Rightarrow A[k] > x$) é válida após o laço.
    - o restante do vetor permanece igual.
  - $A[j] \leq x$:
    - i é incrementado.
    - $A[i]$ e $A[j]$ são trocados de posição.
    - j é incrementado.
    - $A[i] \leq x \Rightarrow$ a condição 1 é válida.
    - $A[j - 1] > x \Rightarrow$ a condição 2 é válida.

Quicksort - Correção



- **Terminação**. Na terminação $j = r$. Assim, todos os elementos estão em um dos três conjuntos descritos na IdL. $A[r]$ é trocado de posição com $A[i + 1]$ e passa a ocupar a sua posição correta no vetor.

Quicksort - Análise de complexidade

- A complexidade de tempo de execução do Quicksort depende de quão balanceado é o processo de particionamento.
- Pior caso: sempre divide em um vetor vazio e um vetor de tamanho $n-1$.

$$T(n) = \begin{cases} \Theta(1), & \text{se } n = 1. \\ T(n-1) + T(0) + \Theta(n). \end{cases}$$

$$T(n) = \begin{cases} \Theta(1), & \text{se } n = 1. \\ T(n-1) + \Theta(n). \end{cases}$$

$$\Rightarrow T(n) = \Theta(n^2).$$

Quicksort - Análise de complexidade

- Melhor caso: sempre divide em subvetores de tam. $\frac{n}{2}$.

$$T(n) = \begin{cases} \Theta(1), & \text{se } n = 1. \\ 2T(\frac{n}{2}) + \Theta(n). \end{cases}$$

  $\Rightarrow T(n) = \Theta(n \lg n)$.

- **Exercício**. Suponha que o processo de particionamento sempre divida o vetor de entrada em um subvetor de tamanho $\frac{8}{10}n$ e um subvetor de tamanho $\frac{2}{10}n$. Qual seria a complexidade de tempo de execução do Quicksort nesse caso?

Relações de Recorrência

- Algoritmos projetos com a estratégia de divisão e conquista geralmente têm complexidade de tempo expressa por uma equação de recorrência.
- Por exemplo, o melhor caso do Quicksort é expresso por:

$$T(n) = \begin{cases} \Theta(1), & \text{se } n = 1. \\ 2T(\frac{n}{2}) + \Theta(n). \end{cases}$$

  Que tem como solução $T(n) = \Theta(n \lg n)$.
- Veremos três métodos para solução de equações de recorrência:
  - Método da substituição.
  - Método de árvore de recursão.
  - Teorema mestre.

Relações de Recorrência - Método da Substituição

- O **método da substituição** segue dois passos:
  - Propor uma solução candidata;
  - Usar indução matemática para demonstrar a solução proposta.
- **Exemplo.** $T(n) = 2T(\lfloor \frac{n}{2} \rfloor) + n$.
  - Solução proposta: $T(n) = O(n \lg n)$.
  - Temos que mostrar que $T(n) \leq cn \lg n (c > 0)$.
  - Suponha que a desigualdade seja válida para todo $m < n$ (em particular para $m = \lfloor \frac{n}{2} \rfloor$).

$$\Rightarrow T(n) = 2T(\lfloor \frac{n}{2} \rfloor) + n \leq 2(c\lfloor \frac{n}{2} \rfloor \lg(\lfloor \frac{n}{2} \rfloor)) + n$$

$$\leq cn \lg(\frac{n}{2}) + n = cn \lg n - cn \lg 2 + n$$

$$= cn \lg n - cn + n \leq cn \lg n$$

Relações de Recorrência - Método da Substituição

- **Exemplo.** $T(n) = 2T(\lfloor \frac{n}{2} \rfloor) + n,\ T(1) = 1.$
  - O caso base também deve ser demonstrado, i.e. $T(n) \leq cn \lg n$ para $n \geq n_0$.
  - Tomar $n_0 = 1$ não é uma boa escolha para a base indutiva pois $c \lg 1 = 0$ e sabemos que $T(1) > 0$
  - Para $n_0 = 2$, temos $T(2) = 2T(1) + 2 = 4$.
  - Para $n_0 = 3$, temos $T(3) = 2T(\lfloor \frac{n}{2} \rfloor) + 3 = 2T(1) + 3 = 5$.
  - Para completar a base indutiva, precisamos escolher $c \geq 1$ tal que:

$$T(2) \leq c2 \lg 2 = 2c$$

$$T(3) \leq c3 \lg 3 = 3c \lg 3$$

tomando $c \geq 2$ temos que as desigualdades são válidas.

Relações de Recorrência - Método da Substituição

- Propondo boas soluções:
  - Não há regra geral (mesmo com o Teorema Mestre).
  - Árvores de recursão (p.ex. usadas no Mergesort e Quicksort).
  - Usar soluções conhecidas para equações similares.
  - Propor solução mais fraca e ir ajustando ($n^3 > n^2 > n \lg n > n > \lg n$).