Algoritmos e Estrutura de Dados

Aula 6 – Algoritmos de Ordenamento: Quick Sort

Prof. Tiago A. E. Ferreira

QuickSort

- O ***Quiksort*** é um algoritmo de ordenamento que no pior caso é **$\Theta(n^2)$**
  - Contudo, na prática, geralmente é a melhor opção!
    - Devido a sua alta eficiência no caso médio: **$\Theta(n\lg n)$**
- O algoritmo ***Quicksort*** utiliza a tática de **dividir e conquistar**
  - Esta tática é dividida em três partes para a realização do ordenamento do arranjo **A[p..r]**

Dividir para Conquistar

- **Dividir:**
  - **A[p..r]** é particionado em dois subarranjos, **A[p..q-1]** e **A[q+1..r]**
    - Cada elemento de **A[p..q-1]** é **menor ou igual** que **A[q]**.
    - **A[q]** é **menor ou igual** a todos os elementos de **A[q+1..r]**
    - O índice ***q*** é calculado como parte do procedimento de particionamento

Dividir para Conquistar

- **Conquistar:**
  - Os dois subarranjos **A[p..q-1]** e **A[q+1..r]** são ordenados por chamadas recursivas ao ***quiksort***
- **Combinar:**
  - Os arranjos são ordenados localmente!
    - Não há trabalho para combiná-lo: O arranjo **A[p..r]** está inteiramente ordenado!

Quicksort – Pseudo-Código

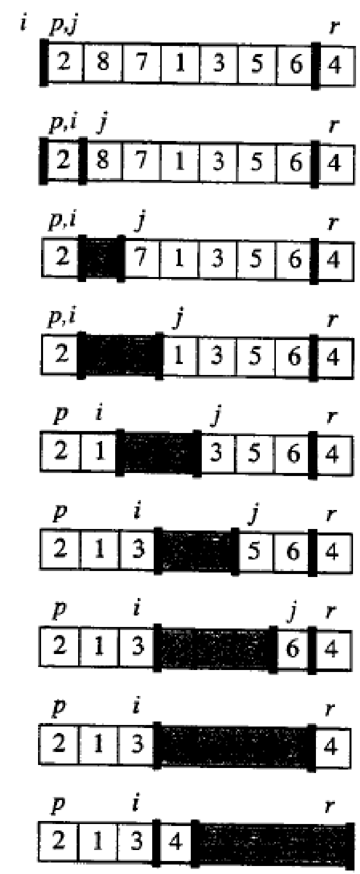
```
QUICKSORT(A, p, r)
if p < r
    then q ← PARTITION(A, p, r)
         QUICKSORT(A, p, q − 1)
         QUICKSORT(A, q + 1, r)
```

- Para ordenar um arranjo **A**, é chamado inicialmente a função **QUICKSORT(A,1,comprimento[A])**
  - **Em python, seria de 0 até comprimento[A]-1**

Particionamento do Arranjo

- O Procedimento PARTITION organiza o subarranjo **A[p..r]** localmente

```
PARTITION(A, p, r)
x ← A[r]                      ⟶ Pivô
i ← p − 1
for j ← p to r − 1
      do if A[j] ≤ x
            then i ← i + 1
                  trocar A[i] ← A[j]
trocar A[i + 1] ↔ A[r]
return i + 1
```



Analisando o Pseudo-Código

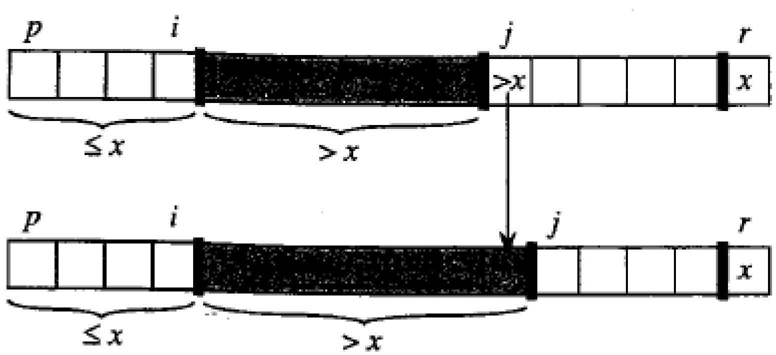
- Loop invariante:
  - É possível enunciar um loop invariante entre as linhas 3 e 6 do pseudo-código
  - Propriedades:
    - *Se* $p \leq k \leq i$*, então* $A[k] \leq x$
    - *Se* $i+1 \leq k \leq j-1$*, então* $A[k] > x$
    - *Se* $k = r$*, então* $A[k] = x$

Analisando o Pseudo-Código

- **Inicialização:**
  - Antes da primeira iteração
    - i =p-1
    - j = p
  - Não há nenhum número entre **p** e **i**, e entre **i+1** e **j-1**
    - **As duas primeiras propriedades são satisfeitas**
  - A atribuição na linha 1
    - **A terceira propriedade é satizfeita!**

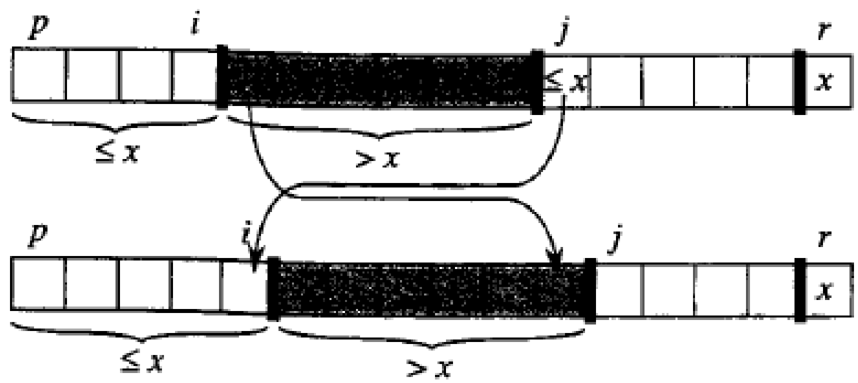
Analisando o Pseudo-Código

- **Manutenção**
  - Existem dois casos a considerar, dependendo do teste da linha 4:
    - Se A[j]>x
      - **A única ação do loop é incrementar j (as propriedades continuam válidas)**



Analisando o Pseudo-Código

- **Manutenção**
  - Existem dois casos a considerar, dependendo do teste da linha 4:
    - Se A[j] ≤ x
      - **A[i] e A[j] são permutados e então j é incrementado**
      - **Assim, fica: A[i] ≤ x (propriedade 1 satisfeita)**
      - **A[j-1] > x (propriedade 2 satisfeita)**



Analisando o Pseudo-Código

- **Término:**
  - **j=r**
    - Toda a entrada no arranjo pose ser classificada em um dos três conjuntos definidos pelo loop invariante
      - Um conjunto com todos os elementos menores que x
      - Um conjunto com todos os elementos maiores que x
      - Um conjunto unitário contende x

Desempenho do Quiksort

- O particionamento tem custo **Θ(n)**
- Particionamento no pior caso
  - São produzidos dois subproblemas: um com n-1 elementos e um com zero elementos.
  - Gera-se a recursão:

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) . \end{aligned}$$

    - O que significa **$T(n) = \Theta(n^2)$**

Desempenho do Quiksort

- Particionamento do melhor caso:
  - São produzidos dois subproblemas não maiores que n/2
    - Sendo um $\lfloor n/2 \rfloor$ e o outro $\lceil n/2 \rceil - 1$
  - Gera-se a recorrência:

$$T(n) \leq 2T(n/2) + \Theta(n)$$

    - O que significa **T(n) = O(nlgn)**

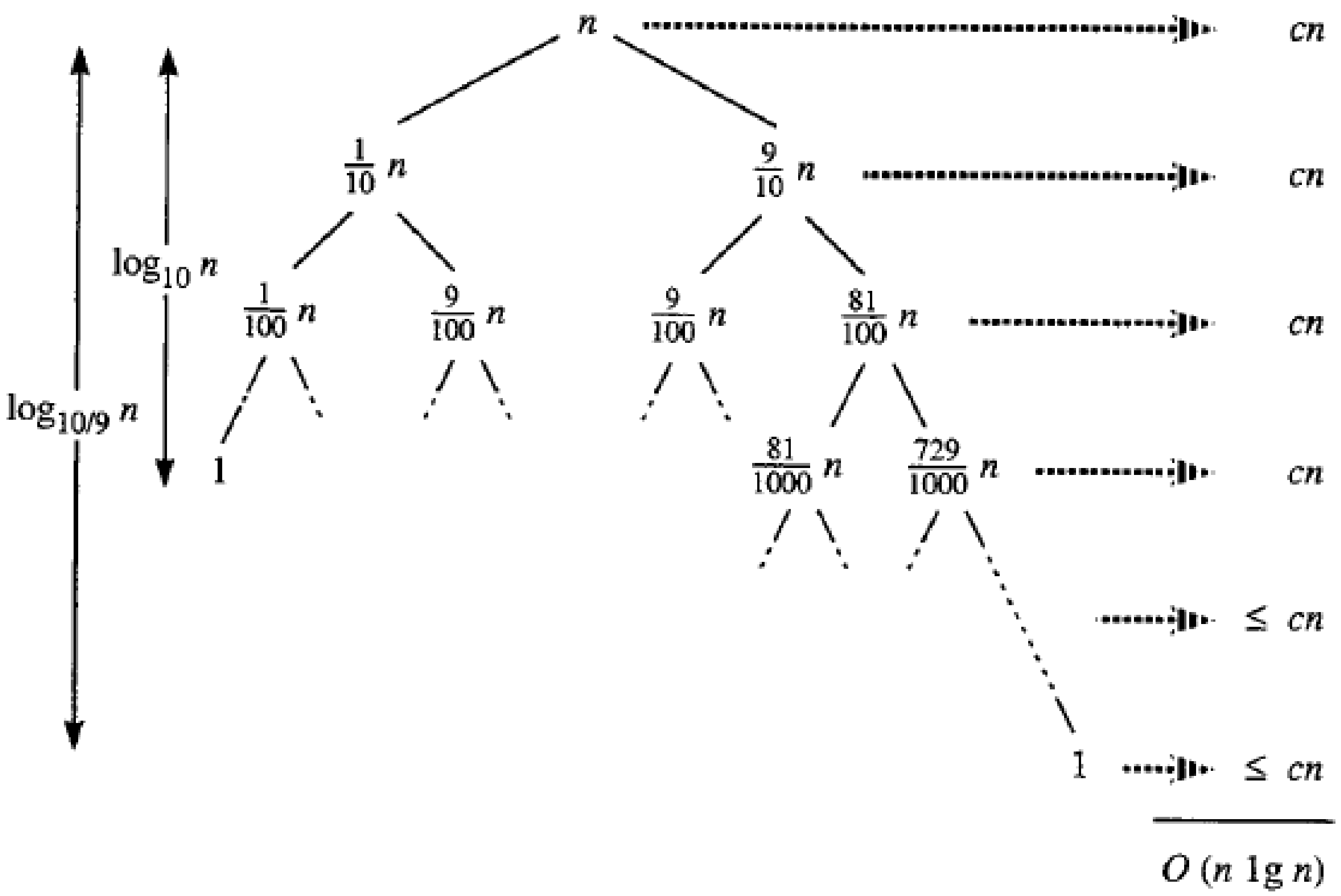
Desempenho do Quiksort

- Particionamento balanceado – Caso médio
  - No caso médio, o algoritmo quicksort se aproxima muito mais do melhor caso do que do pior caso!
  - Suponha que o particionamente sempre produza uma divisão proporcional de 9 para 1:

$$T(n) \leq T(9n/10) + T(n/10) + cn$$

    - O que significa **T(n) = O(nlgn)**

Desempenho do Quiksort



Versão Aleatória do Quicksort

- O caso médio significa um particionamento médio
  - Assim se garantirmos que o parcionamento médio esteja balanceado, também garantimos que na médio o custo do **quicksort** será **O(nlgn)**
  - Para tal aleatoriza-se a escolha no **pivô**

```
RANDOMIZED-PARTITION(A, p, r)
  i ← RANDOM(p, r)
  trocar A[p] ↔ A[i]
  return PARTITION(A, p, r)
```

```
RANDOMIZED-QUICKSORT(A, p, r)
  if p < r
     then q ← RANDOMIZED-PARTITION(A, p, r)
          RANDOMIZED-QUICKSORT(A, p, q − 1)
          RANDOMIZED-QUICKSORT(A, q + 1, r)
```

Exercício:

- Implemente o Quicksort em Python
  - Ambas as versões (normal e aleatória)
- Resolva o problema do livro Texto
  - 7-1 (pág. 129 da versão em protuguês)