

Introdução à Ciência da Computação II

Quick-Sort Pt. II: Partição In-Place & Quick-Select

Prof. Ricardo J. G. B. Campello

1

Aula de Hoje

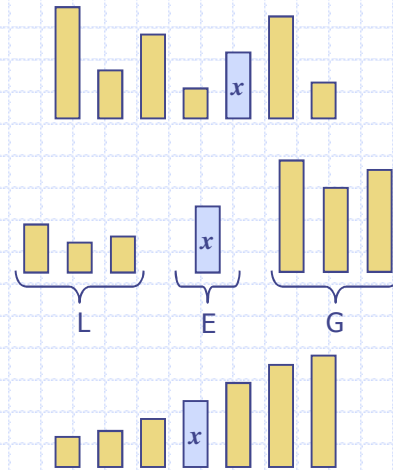
- ◆ Ordenação Quick-Sort com Partição In-Place
 - Partição In-Place Conceitual
 - Implementação em C
- ◆ Problema de Seleção
 - Algoritmo Quick-Select

2

Algoritmo Quick-Sort (Revisão)

◆ **Quick-sort** é um algoritmo de ordenação baseado no paradigma de **divisão e conquista**:

- **Divisão**: toma aleatoriamente um elemento x (**pivô**) e particiona a seqüência em:
 - ◆ L : elementos menores que x
 - ◆ E : elementos iguais a x
 - ◆ G : elementos maiores que x
- **Recursão**: ordena L e G
- **Conquista**: junta L , E e G



3

Partição In-Place

- Note que se for possível executar a fase de divisão operando por trocas sobre a própria seqüência de valores original, o algoritmo se aproxima de uma versão in-place
- Existe mais de uma maneira de executar tal operação de partição de forma in-place
- Discutiremos uma delas a seguir...

4

Partição In-Place

- Toma-se o último valor da sequência (mais à direita) como pivô e utilizam-se dois índices:
 - f_h : controla a posição na qual todos os valores à sua esquerda (não inclusa) são menores que o pivô
 - i : percorre todos valores da sequência, da esquerda para a direita, comparando-os com o pivô
 - Cada elemento menor que o pivô encontrado em i é trocado com aquele em f_h e incrementa-se f_h
 - Ao final, troca-se o elemento em f_h com o pivô

5

Exemplo de Execução

$V =$

| | |
|-------|-----|
| f_h | p |
| 85 | 50 |
| i | |

24 63 45 17 31 96

6

Exemplo de Execução

$V =$

| | | | | | | | | | |
|-------|----|----|----|----|----|----|----|-----|----|
| f_h | 85 | 24 | 63 | 45 | 17 | 31 | 96 | p | 50 |
|-------|----|----|----|----|----|----|----|-----|----|

 i

$V[i] < V[p] ?$

$85 > 50$

7

Exemplo de Execução

$V =$

| | | | | | | | | | |
|-------|----|----|----|----|----|----|----|-----|----|
| f_h | 85 | 24 | 63 | 45 | 17 | 31 | 96 | p | 50 |
|-------|----|----|----|----|----|----|----|-----|----|

 i

8

Exemplo de Execução

$V =$

| | | | | | | | | | |
|-------|----|----|----|----|----|----|----|-----|----|
| f_h | 85 | 24 | 63 | 45 | 17 | 31 | 96 | p | 50 |
|-------|----|----|----|----|----|----|----|-----|----|

i

$V[i] < V[p] ?$
 $24 < 50$


9

Exemplo de Execução

$V =$

| | | | | | | | | | |
|-------|----|----|----|----|----|----|----|-----|----|
| f_h | 85 | 24 | 63 | 45 | 17 | 31 | 96 | p | 50 |
|-------|----|----|----|----|----|----|----|-----|----|

i



10

Exemplo de Execução

$V =$

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 24 | 85 | 63 | 45 | 17 | 31 | 96 | 50 |
|----|----|----|----|----|----|----|----|

f_h p
 i

11

Exemplo de Execução

$V =$

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 24 | 85 | 63 | 45 | 17 | 31 | 96 | 50 |
|----|----|----|----|----|----|----|----|

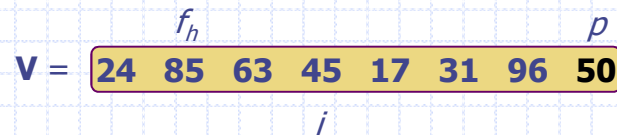
f_h p
 i

$V[i] < V[p] ?$

$63 > 50$

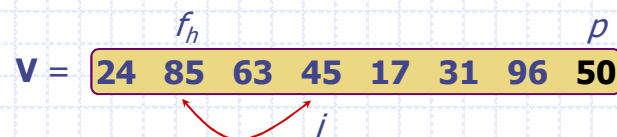
12

Exemplo de Execução



13

Exemplo de Execução

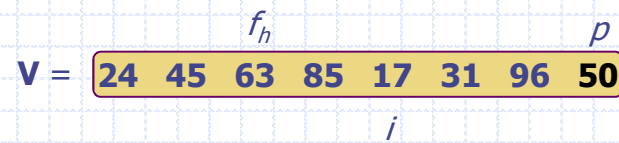


$V[i] < V[p] ?$

$45 < 50$

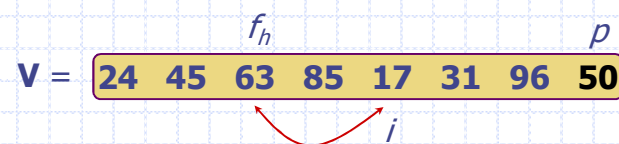
14

Exemplo de Execução



15

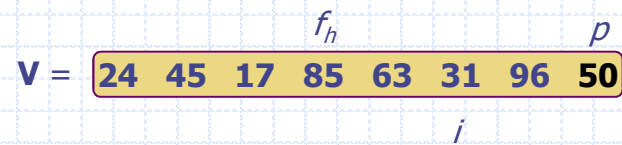
Exemplo de Execução



$V[i] < V[p] ?$
 $17 < 50$

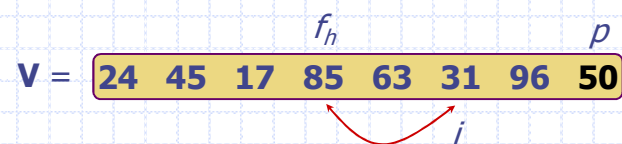
16

Exemplo de Execução



17

Exemplo de Execução



$V[i] < V[p] ?$

$31 < 50$

18

Exemplo de Execução

$V =$

| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 24 | 45 | 17 | 31 | 63 | 85 | 96 | 50 |
|----|----|----|----|----|----|----|----|

f_h p
 i

19

Exemplo de Execução

$V =$

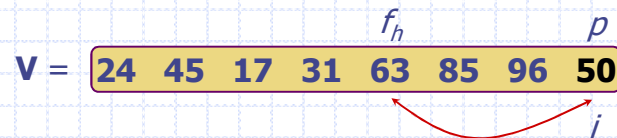
| | | | | | | | |
|----|----|----|----|----|----|----|----|
| 24 | 45 | 17 | 31 | 63 | 85 | 96 | 50 |
|----|----|----|----|----|----|----|----|

f_h p
 i

$V[i] < V[p] ?$
 $96 > 50$

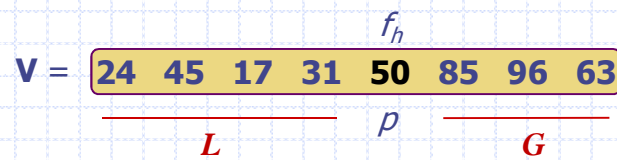
20

Exemplo de Execução



21

Exemplo de Execução



22

Quick-Sort com Partição In-Place

- ◆ Com o procedimento anterior, podemos re-implementar o algoritmo Quick-Sort visto na aula anterior
 - de forma mais eficiente
- ◆ Implementação em C:
 - no quadro...

23

Observações

- ◆ Note que a política de troca se o elemento for menor que o pivô implica que eventuais elementos iguais ao pivô ficarão na subsequência G
 - Ficarão em L se a troca for feita para elementos menores ou iguais ao pivô
 - De qualquer forma, esses elementos acabarão em suas posições corretas após a ordenação recursiva das subsequências L e G

24

Observações

- ◆ Note também que a ordem relativa entre quaisquer elementos com valores repetidos (em geral, não apenas no que refere ao pivô) pode ser invertida
- ◆ Em outras palavras, o algoritmo Quick-Sort com Partição In-Place **Não é Estável**

25

Observações

- ◆ É importante observar que, tecnicamente, o algoritmo quick-sort como um todo não é rigorosamente in-place
 - de fato, as chamadas recursivas podem requerer memória auxiliar proporcional ao número de elementos no vetor (um pivô por chamada)
- ◆ Porém...

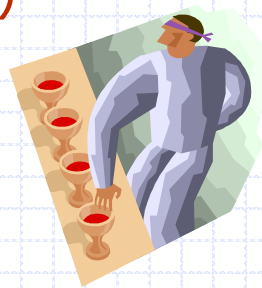
26

Observações

- ◆ É possível substituir as chamadas recursivas de Quick-Sort pelo uso de uma pilha explícita...
 - associada a uma estratégia de inserção e retirada desta pilha que garante que seu tamanho seja $O(\log n)$
 - como $O(\log n)$ é muito menor que n , embora não seja constante, muitos autores chamam esta versão "in-place"
 - para saber mais, veja, por exemplo:
 - (Tenenbaum et al., 1990)
 - (Goodrich & Tamassia, 2004)

27

Seleção (Ordenação Parcial)



O Problema de Seleção

- ◆ Dado um inteiro k positivo e uma seqüência de n elementos x_1, x_2, \dots, x_n , encontre o k -ésimo menor elemento desta seqüência de acordo com uma determinada relação de ordem total
- ◆ Evidentemente, podemos ordenar os elementos em tempo $O(n \log n)$ e então indexar o k -ésimo elemento

$k=3$

7 4 9 6 2 → 2 4 6 7 9

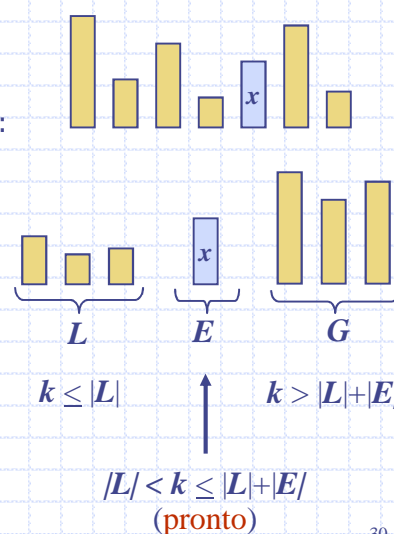
- ◆ É possível resolver o problema mais rápido?

29

Quick-Select

- ◆ **Quick-Select** é um algoritmo de seleção baseado no paradigma de **poda e busca** (*prune-and-search*):

- **Poda**: toma aleatoriamente um elemento x (**pivô**) e particiona a seqüência em:
 - ◆ L : elementos menores que x
 - ◆ E : elementos iguais a x
 - ◆ G : elementos maiores que x
- **Busca**: dependendo de k , ou a resposta está em E ou é preciso chamar recursivamente L ou G



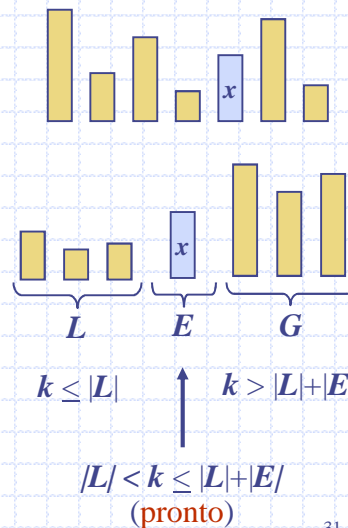
30

* OBS: $|L|$ representa a cardinalidade (no de elementos) das subseqüências

Quick-Select

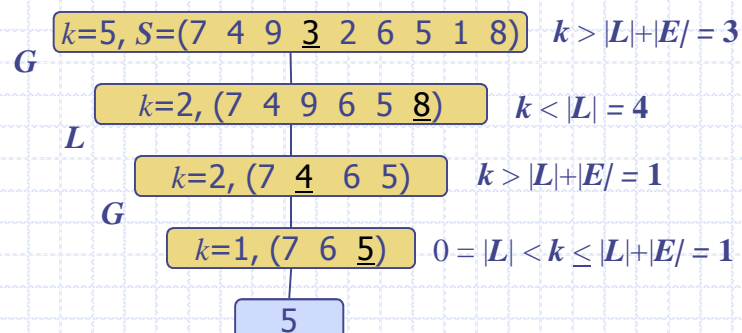
◆ **Busca:** dependendo de k , ou a resposta está em E ou é preciso chamar recursivamente L ou G

- Se o k -ésimo elemento estiver em E , seu valor é igual ao pivô
- Se o k -ésimo elemento estiver em L , busca-se pelo k -ésimo elemento recursivamente nesta subsequência
- Se o k -ésimo elemento estiver em G , busca-se pelo k' -ésimo elemento recursivamente nesta subsequência, onde $k' = k - |L| - |E|$



31

Exemplo de Execução



32

Observações

- ◆ É possível demonstrar que o tempo de execução do algoritmo quick-select é $O(n^2)$ no **pior caso**
- ◆ Porém, pode-se demonstrar que não apenas o tempo de **melhor caso** mas também o tempo esperado (**caso médio**) são $O(n)$
- ◆ Pode-se ainda demonstrar que existe uma seleção não aleatória do pivô que leva a um quick-select determinístico $O(n)$ mesmo para o pior caso (Goodrich & Tamassia, 2005)
- ◆ Outras alternativas não discutidas aqui são as versões para seleção (ordenação parcial) dos algoritmos selection-sort, insertion-sort e heap-sort
- ◆ Análises comparativas de todos esses algoritmos são apresentadas em Ziviani, N. "Projeto de Algoritmos", 2a ed., Ed. Thomson, 2004

Exercícios

- Elabore exemplos e faça testes de mesa (execução manual passo a passo) com o algoritmo Quick-Sort com Partição In-Place
 - Verifique o que acontece durante a partição de seqüências contendo valores repetidos do pivô
 - Elabore ao menos um exemplo que permita mostrar que a estabilidade não é garantida
- Implemente em C o algoritmo Quick-Select

Bibliografia

- ◆ M. T. Goodrich & R. Tamassia, *Data Structures and Algorithms in C++/Java*, John Wiley & Sons, 2002/2005
- ◆ N. Ziviani, *Projeto de Algoritmos*, Thomson, 2a. Edição, 2004
- ◆ A. M. Tenenbaum et al., *Data Structures Using C*, Prentice-Hall, 1990
- ◆ S. Skiena & M. Revilla, *Programming Challenges: The Programming Contest Training Manual*, Springer-Verlag, 2003