UFSC-CTC-INE

INE5384 - Estruturas de Dados

Ordenação de Dados (II)

Prof. Ronaldo S. Mello

2002/2

Algoritmos de Ordenação Simples

- Alta complexidade
  - indicado apenas para a ordenação de vetores com “poucos” elementos (“n” pequeno)
- Necessita-se de algoritmos com melhor desempenho para tratar vetores grandes
- Candidatos:
  - *QuickSort*
  - *HeapSort*
  - *MergeSort*

QuickSort

- *QuickSort* também é um método de troca
  - ordena através de sucessivas trocas entre pares de elementos do vetor
- Aplica um método "dividir para conquistar"
  - divide um problema em 2 ou mais sub-problemas
  - resolve recursivamente cada sub-problema (dividindo novamente)
  - combina as soluções menores (sub-problemas) para obter a solução do problema geral

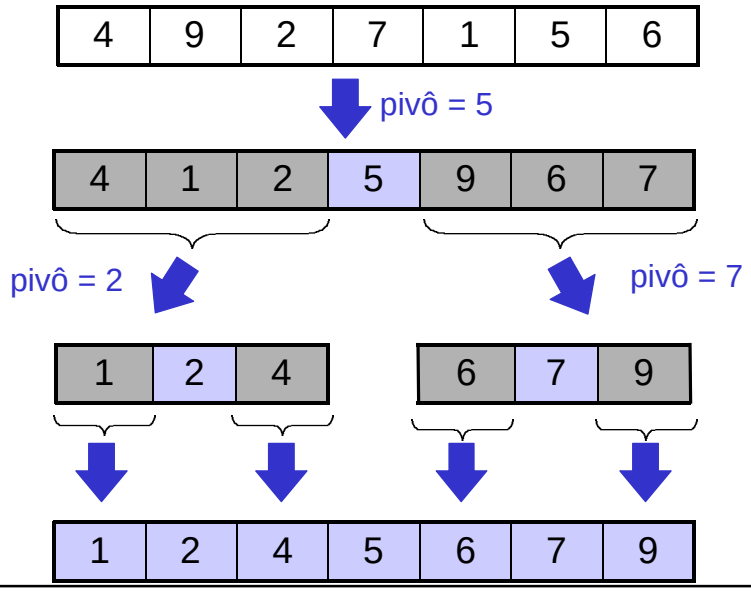
QuickSort - Funcionamento

1. Seleciona um elemento do vetor (pivô - e_p)
   - utilizando algum critério de seleção
2. Rearranja o vetor da seguinte forma:
   - segmento *s1*: elementos menores ou iguais ao pivô
   - segmento *s2*: elementos maiores ou iguais ao pivô

(e_p está na sua posição correta!)

4. Executa *QuickSort* recursivamente para *s1* e *s2*

QuickSort - Exemplo



QuickSort - Seleção do Pivô

- A escolha do pivô é crítica para o desempenho do algoritmo
  - *pior escolha*: gera um segmento com tamanho 0 e outro com tamanho n-1

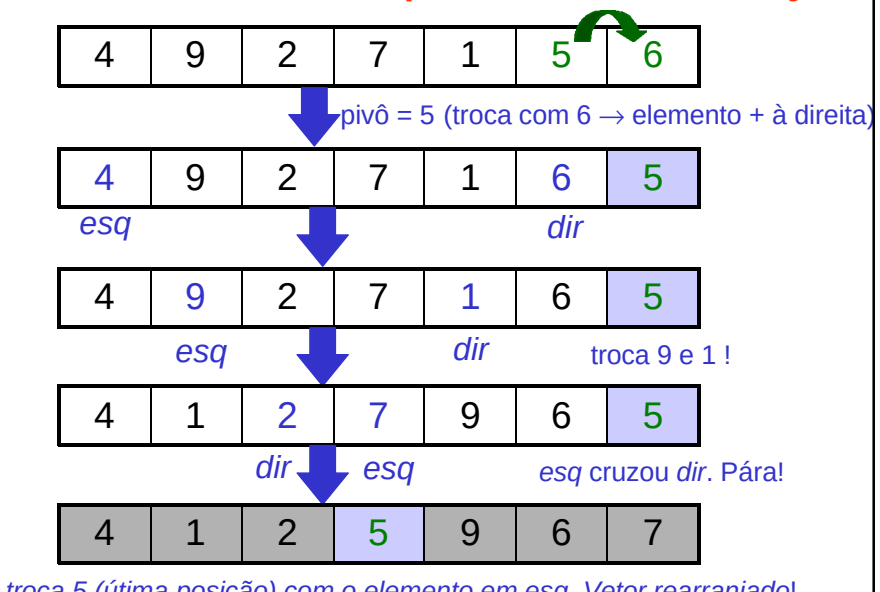
  - *melhor escolha*: gera segmentos balanceados

- Melhor pivô: elemento mais próximo da média de valores dos elementos
  - *complexidade para determiná-lo:* ***O(n)***

QuickSort - Rearranjo do Vetor

- Transfere o pivô para uma das extremidades do vetor (direita, p.ex.)
- Utiliza apontadores (*esq* e *dir*) que partem das extremidades do vetor e se deslocam para o centro do vetor, trocando elementos de segmento quando necessário
- O deslocamento termina quando um apontador cruza o outro
- Transfere-se o pivô (*troca*) para a posição apontada por *esq*

QuickSort - Exemplo de Rearranjo



troca 5 (útima posição) com o elemento em esq. Vetor rearranjado!

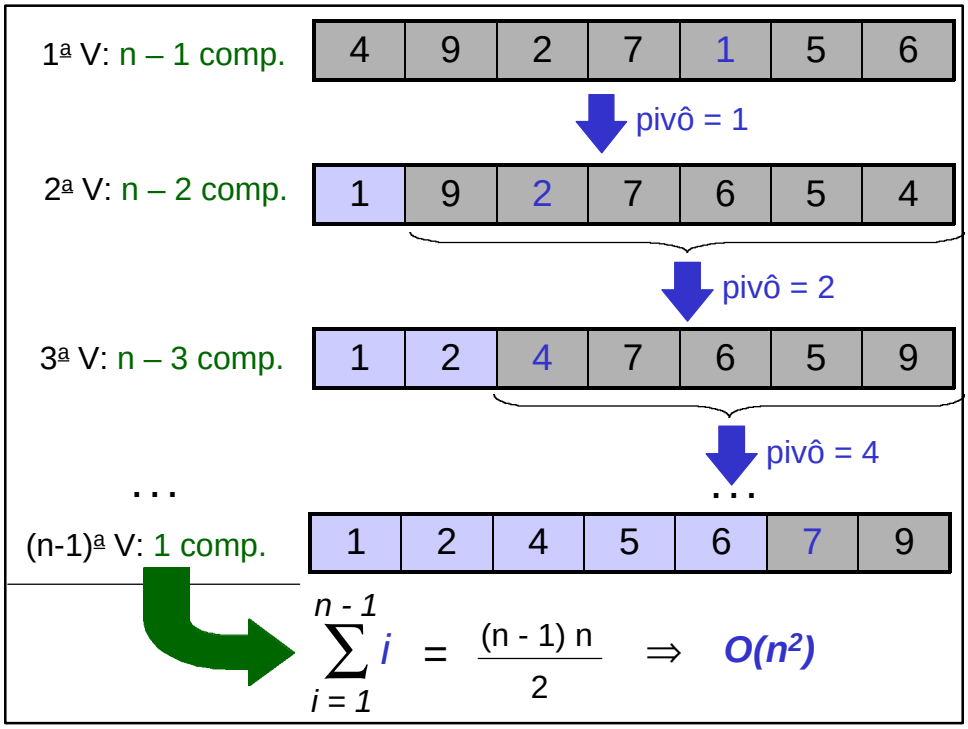
QuickSort

- Simulação de funcionamento

http://math.hws.edu/TMCM/java/xSortLab

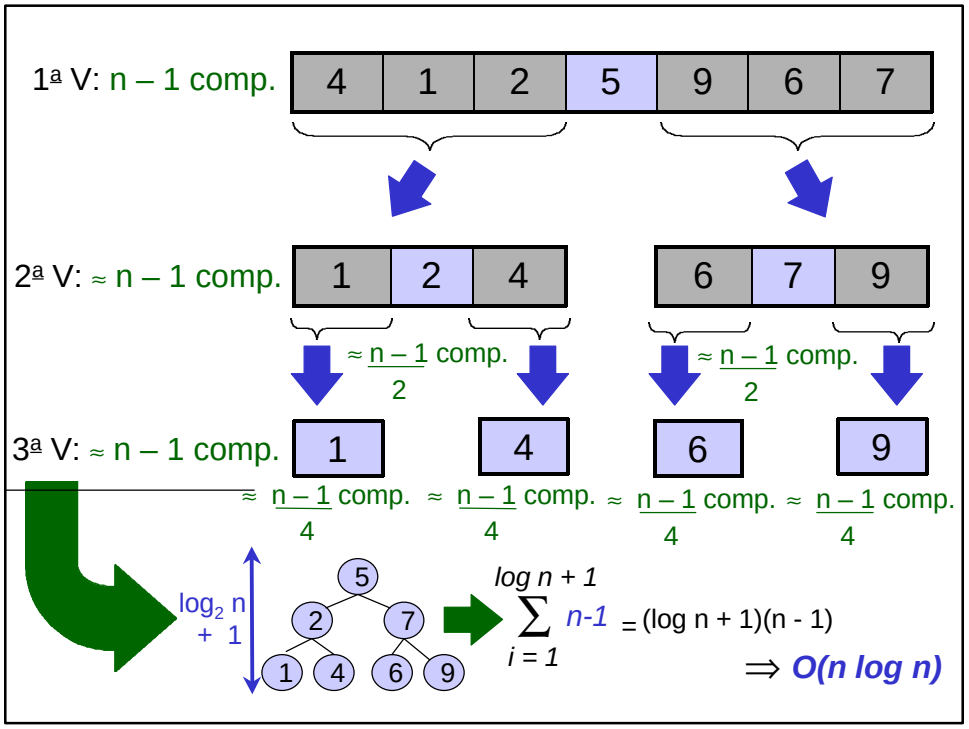
QuickSort - Complexidade

- Pior caso: segmentos totalmente desbalanceados
  - pivô é sempre o menor elemento do vetor



QuickSort - Complexidade

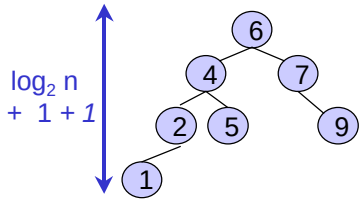
- Melhor caso: segmentos balanceados
  - pivô é sempre o elemento com valor mais próximo da média



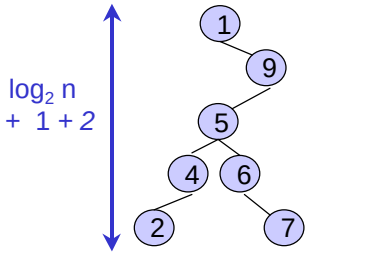
QuickSort - Complexidade

- Caso médio: segmentos “um pouco” desbalanceados

Iniciando com pivô = 6

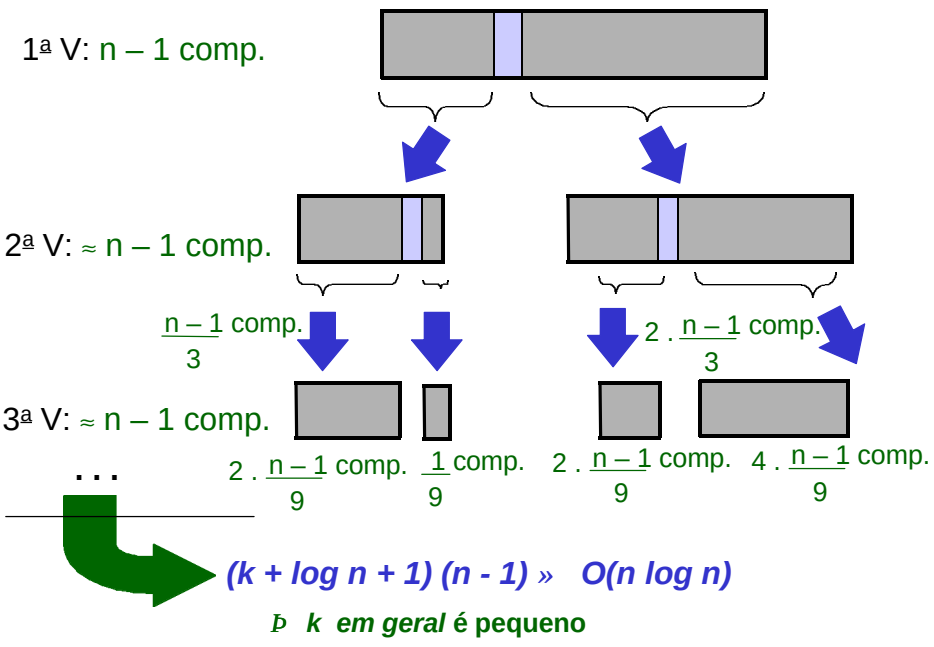


Iniciando com pivô = 1



– considera-se um fator *k* de desbalanceamento:

log n + 1 + k



Algoritmos de Ordenação por Troca

	BubbleSort	*QuickSort*
Pior caso	$O(n^2)$	$O(n^2)$
Caso médio	$O(n^2)$	$O(n \log n)$
Melhor caso	$O(n^2)$	$O(n \log n)$

- *QuickSort* não é indicado para vetores pequenos, considerando a complexidade conjunta da seleção de pivô e do rearranjo do vetor

Exercícios

- Implementar os seguintes métodos para a classe *OrdenadorQuickSort*:
  - *selecionaPivo* (seleção do elemento médio)
  - *ordena* (faz o rearranjo do vetor e chamadas recursivas do método *ordena* para os segmentos)
- Para a próxima aula: Implementar na classe *OrdenadorBinaryInsertionSort*:
  - *ordena*

  (variante do *InsertionSort* que realiza uma *busca binária* no segmento ordenado ao invés de uma busca seqüencial)