UNISINOS

Estruturas Avançadas de Dados I
(QuickSort)

Prof. Gilberto Irajá Müller

Última atualização 06/06/2018

Introdução

- Desenvolvido em 1959 e publicado em 1961 por Charles Antony Richard Hoare;
- Considerado o algoritmo mais rápido de ordenação;
- Segundo relatos, a ideia do algoritmo surgiu a partir de uma visita de Hoare a Rússia como estudante. Como parte do seu projeto, necessitava classificar as palavras em russo com uma equivalência às palavras em inglês;
- Hoare propôs o método em função do desempenho ruim dos demais;
- Diversas variações do algoritmo original, sendo que muitas linguagens o utilizam. O Java, por exemplo, utiliza o Quicksort Dual Pivot proposto por Vladimir Yaroslavskiy.

Introdução (cont.)

- Como o Merge Sort, o Quicksoft usa o paradigma dividir para conquistar;
- **Dividir**: O array A[p...r] é particionado em dois subarrays A[p...q – 1] e A[q + 1...r], de forma que cada elemento de A[p...q – 1] seja menor ou igual a A[q] que, por sua vez, é menor ou igual a cada elemento de A[q + 1...r]. O índice q (pivot) é calculado como parte desse procedimento de particionamento;
- **Conquistar**: Os dois subarrays são ordenados por chamadas recursivas;
- **Combinar**: Como os subarrays são ordenados localmente, não é necessário nenhum trabalho para combiná-los: o array A[p...r] já está ordenado.

Introdução (cont.)

- A escolha do pivot (q) possui várias estratégias:
  - **Primeiro (ou último) elemento do array**: funciona bem quando os elementos do array estão dispostos de forma aleatória. Quando o array está ordenado ou quase ordenado, a maioria dos elementos ficará entre os subarrays gerando um desempenho ruim;
  - **Elemento aleatório**: funciona bem na maioria dos casos, mesmo com o array quase ordenado. Observar que a escolha de um número aleatório demanda tempo computacional; chamado de *randomized quicksort*;
  - **Mediana**: é o elemento do meio (50%) do array, pois, assim, divide-se o array em duas partes iguais. Infelizmente, calcular a mediana é difícil, pois o array precisa estar ordenado. Uma aproximação seria considerar o pivot como o centro do array usando os índices das extremidades. Ex.:
    - S = {5, 4, 1, 9, 3, 10, 8};
    - left = 0, right = 6 (length – 1);
    - center = (left + right) / 2 = 3;
    - S[center] = **9**.
- Há vários algoritmos de particionamento. Ver bibliografia no slide final.

Implementação do Quicksort

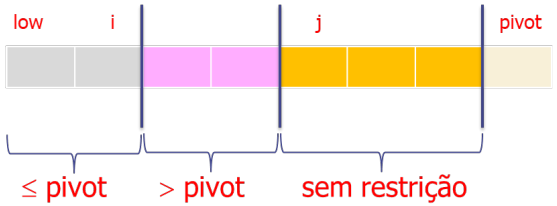
```
public static <T extends Comparable<? super T>> void quickSort(T[] a) {
   sort(a, 0, a.length - 1);
}
```

```
private static <T extends Comparable<? super T>> void sort(T[] a, int low,
int high) {
   if (low >= high) return;
   int p = partition(a, low, high);
   sort(a, low, p - 1);
   sort(a, p + 1, high);
}
```

```
private static <T extends Comparable<? super T>> int partition(T[] a, int low, int
high) {
   T pivot = a[high];
   int i = low - 1;
   for (int j = low; j < high; j++) {
       if (a[j].compareTo(pivot) <= 0) {
           exchange(a, ++i, j);
       }
   }
   exchange(a, i + 1, high);
   return i + 1;
}
```

Exemplo

```
T pivot = a[high];
int i = low - 1;
for (int j = low; j < high; j++) {
    if (a[j].compareTo(pivot) <= 0){
        exchange(a, ++i, j);
    }
}
exchange(a, i + 1, high);
return i + 1;
```



low = 0, high = 7

i	j						pivot
8	1	6	4	0	3	9	5

8 > 5 (não troca)

i		j					pivot
8	1	6	4	0	3	9	5

1 < 5 (troca)

i		j					pivot
1	8	6	4	0	3	9	5

6 > 5 (não troca)

i			j				pivot
1	8	6	4	0	3	9	5

4 < 5 (troca)

	i			j			pivot
1	4	6	8	0	3	9	5

0 < 5 (troca)

		i			j		pivot
1	4	0	8	6	3	9	5

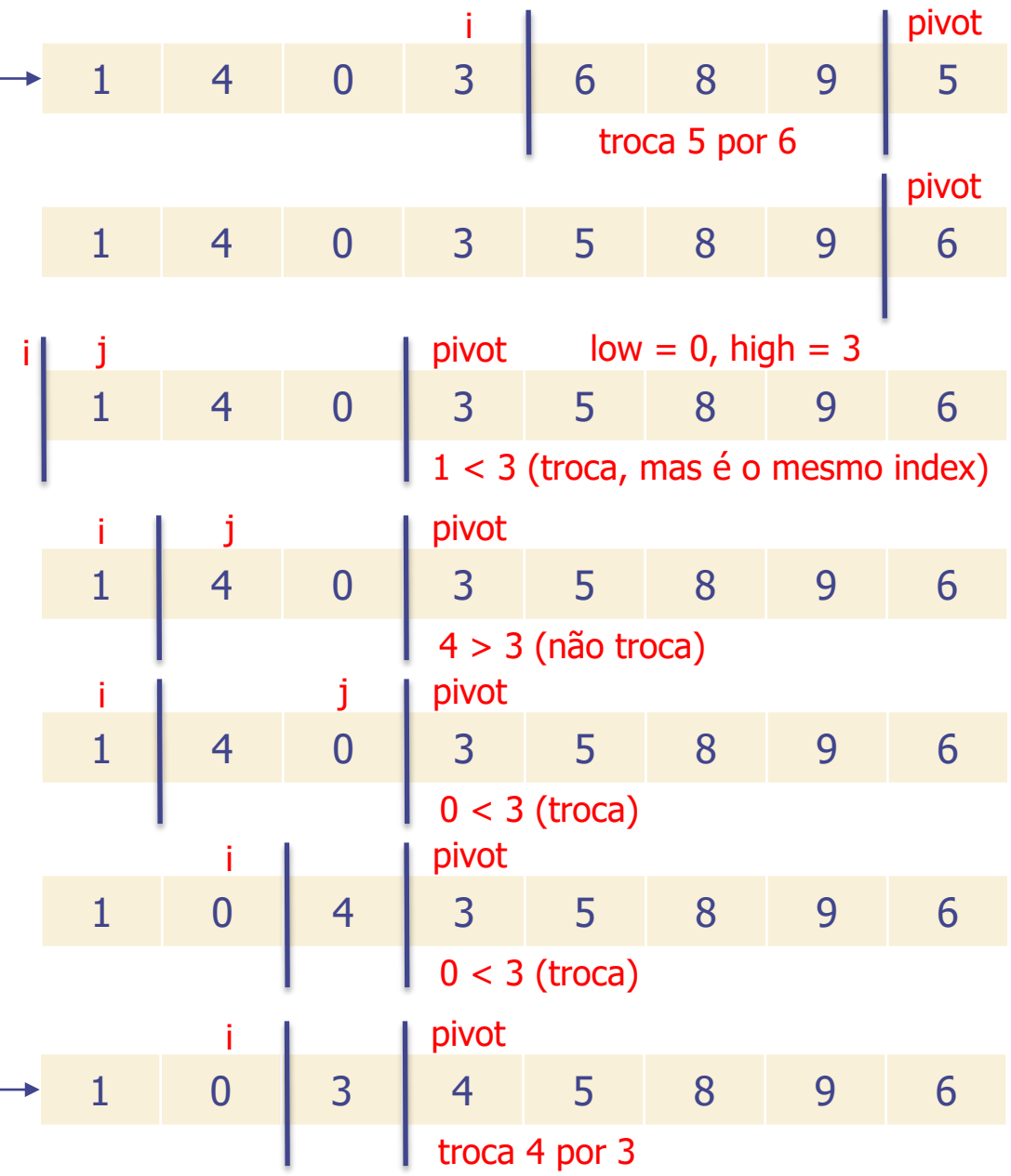
3 < 5 (troca)

			i			j	pivot
1	4	0	3	6	8	9	5

9 > 5 (não troca)

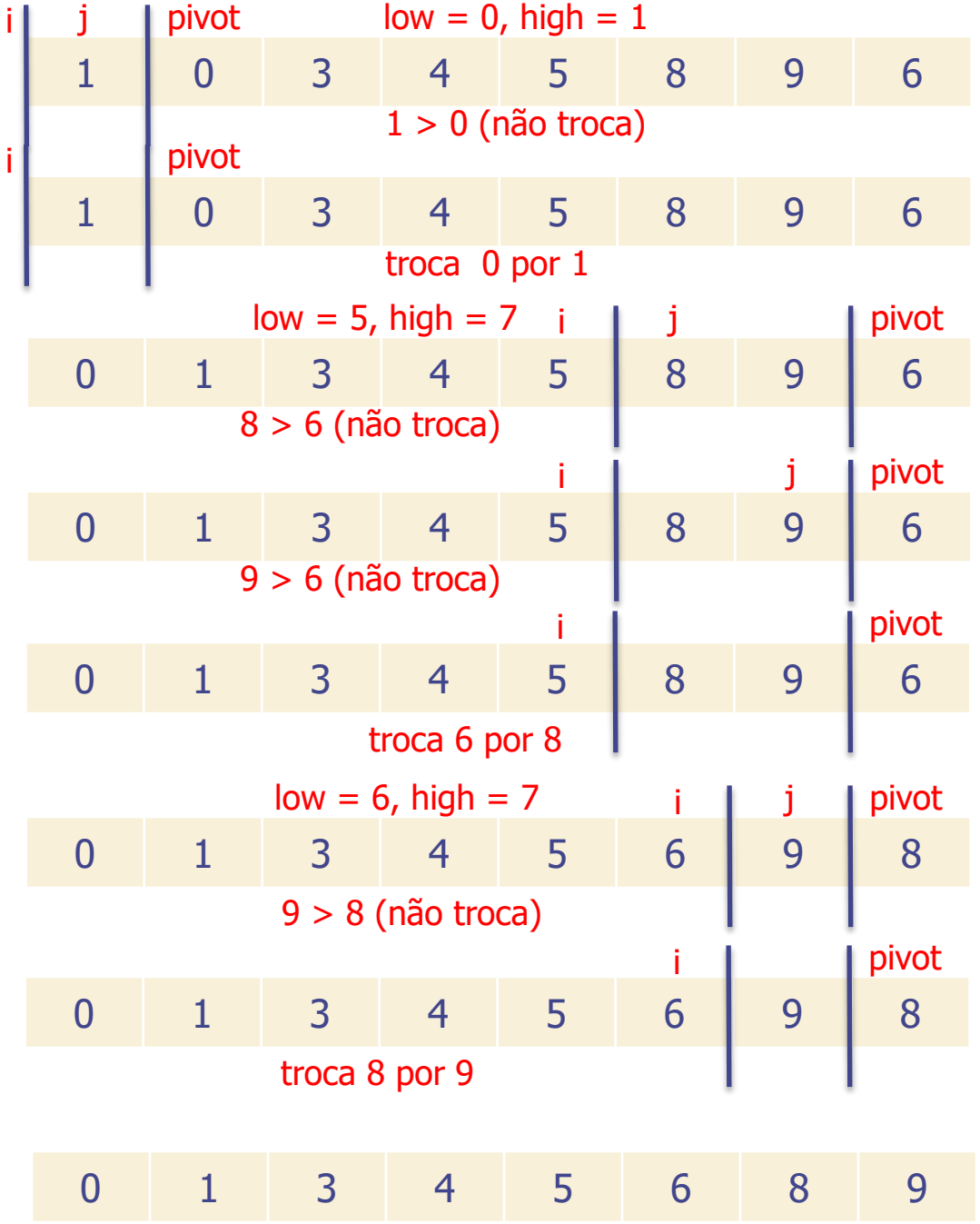
Exemplo

```
T pivot = a[high];
int i = low - 1;
for (int j = low; j < high; j++) {
    if (a[j].compareTo(pivot) <= 0){
        exchange(a, ++i, j);
    }
}
exchange(a, i + 1, high);
return i + 1;
```



Exemplo

```
T pivot = a[high];
int i = low - 1;
for (int j = low; j < high; j++) {
    if (a[j].compareTo(pivot) <= 0){
        exchange(a, ++i, j);
    }
}
exchange(a, i + 1, high);
return i + 1;
```

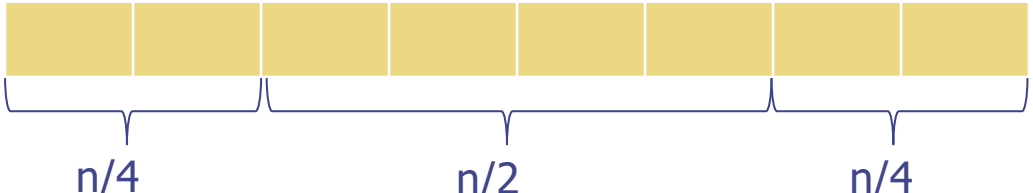


Complexidade

Método	Caso médio	Melhor caso	Pior caso	Complexidade de Espaço	Estável	Interno	Recursivo	Comparação
Bubble Sort	$O(n^2)$	O(n)	$O(n^2)$	In-place = O(1)	Sim	Sim	Não	Sim
Insertion Sort	$O(n^2)$	O(n)	$O(n^2)$	In-place = O(1)	Sim	Sim	Não	Sim
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	In-place = O(1)	Sim	Sim	Não	Sim
Shell Sort	$O(n^{7/6})$ – depende do gap	O(nlog(n))	O(nlog(n)) a $O(n^{3/2})$	In-place = O(1)	Não	Sim	Não	Sim
Heap Sort	O(nlog(n))	O(nlog(n))	O(nlog(n))	In-place = O(1)	Não	Sim	Sim	Sim
Merge Sort	O(nlog(n))	O(nlog(n))	O(nlog(n))	Out-place = O(n)	Sim	Sim (há implementações para Externo)	Sim	Sim
Quick Sort	**O(nlog(n))**	**O(nlog(n))**	**$O(n^2)$**	**In-place = O(1) (Há implementações Out-place = O(n))**	**Não (há implementações com estabilidade)**	**Sim**	**Sim**	**Sim**

Complexidade (cont.)

- Para pequenos problemas (N < 20), o método Insertion Sort é mais rápido;
- Como o Quicksort é recursivo, pode levar mais tempo em relação a outros algoritmos;
- O pior caso é raro de acontecer, pois a partição precisará estar desbalanceada, ou seja, quando o pivot é o menor ou maior elemento da sublista. Isso acontece quando o array está ordenado de forma crescente ou decrescente;
- Estudos mostram que uma boa escolha de pivot está no intervalo n/2.



Implementação Dual Pivot

```
public static <T extends Comparable<? super T>> void sort(T[] a) {
   sort(a, 0, a.length - 1);
}
```

```
private static <T extends Comparable<? super T>> void sort(T[] a, int low, int high) {
   if (high <= low)
      return;

   if (a[high].compareTo(a[low]) < 0)
      exchange(a, low, high);

   int less = low + 1, greater = high - 1;
   int i = low + 1;
   while (i <= greater) {
      if (a[i].compareTo(a[low]) < 0)
         exchange(a, less++, i++);
      else if (a[high].compareTo(a[i]) < 0)
         exchange(a, i, greater--);
      else i++;
   }
   exchange(a, low, --less);
   exchange(a, high, ++greater);

   sort(a, low, less - 1);
   if (a[less].compareTo(a[greater]) < 0)
      sort(a, less + 1, greater - 1);
   sort(a, greater + 1, high);
}
```

Ordena três subarrays

Exercícios Teóricos

- Exercício 1. Considerando o seguinte array:

a) Aplique o Quicksort (passo-a-passo).

Referências Bibliográficas

- CORMEN, Thomas H. et al. **Introduction to algorithms.** 3. ed. Cambridge: MIT, 2009. xix. 1292 p.

- https://algs4.cs.princeton.edu/lectures/23DemoPartitioning.pdf. Acessado em 24/10/2017.

- https://algs4.cs.princeton.edu/23quicksort/. Acessado em 25/10/2017.

- Sedgewick, R. (1978). "Implementing Quicksort programs". 21 (10): 847–857.

- https://web.archive.org/web/20151002230717/http://iaroslavski.narod.ru/quicksort/DualPivotQuicksort.pdf. Acessado em 25/10/2017.