Análise e Técnicas de Algoritmos

Jorge Figueiredo

Análise de Algoritmos de Ordenação

Agenda

- Ordenação baseada em comparação
  – Insertion Sort
  – Mergesort
  – Quicksort
- Ordenação em tempo linear

Problema da Ordenação

Formalmente pode assim ser definido:

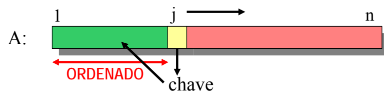
Ordenação
Entrada: Uma seqüência de n números $\langle a_1, a_2, ..., a_n \rangle$.
Saída: Uma reordenação da seqüência de entrada $\langle a'_1, a'_2, ..., a'_n \rangle$, onde $a'_1 \leq a'_2 \leq ... \leq a'_n$.

Em geral, consideramos a seqüência de entrada como um array de n elementos.

Estratégia de Ordenação

- Alguns algoritmos clássicos de ordenação utilizam divisão-e-conquista:
  – Quebra a entrada original em duas partes.
  – Recursivamente ordena cada uma das partes.
  – Combina as duas partes ordenadas.
- Duas categorias de soluções:
  – Quebra simples, combinação difícil.
  – Quebra difícil, combinação simples.

Insertion Sort



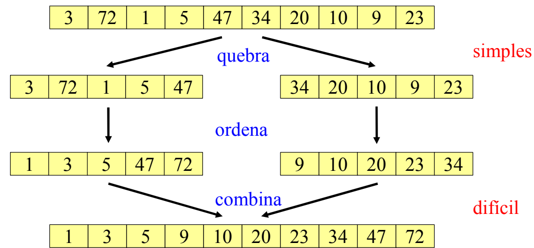

```
InsertionSort(A, n)
for j ← 2 to n do
    chave ← A[j]
    ► insere A[j] na parte ordenada A[1..j-1]
    i ← j – 1
    while i > 0 e A[i] > chave do
        A[i + 1] ← A[i]
        i ← i – 1
    A[i + 1] ← chave
```

Análise do Insertion Sort

- Pior caso:
  – Entrada em ordem reversa.
  – $O(n^2)$
- *Caso médio:*
  – $O(n^2)$
- *Melhor caso:*
  – *Entrada ordenada.*
  – $O(n)$

MergeSort

- Partição simples.
- Combinação mais trabalhosa.



Análise do MergeSort

- Requer resolução de recorrência.
- Melhor caso = Caso médio = Pior caso.
  - *O(n.logn)*

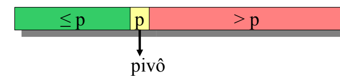
```
MergeSort(A, inicio, fim)
if inicio < fim then
      meio ← (inicio + fim) div 2
      MergeSort(A, inicio, meio)
      MergeSort(A, meio + 1, fim)
      Intercala(A, inicio, meio, fim)
```

QuickSort

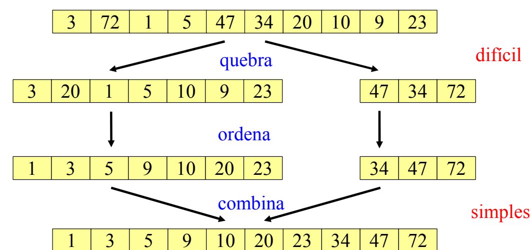
- Proposto por C.A.R. Hoare em 1962.
- Como o MergeSort, utiliza uma estratégia de divisão-e-conquista.
- A parte mais complicada é a quebra.
- A combinação é simples.
- Ao contrário do MergeSort, ordena *in place*.
- Ponto chave é encontrar uma estratégia de particionamento eficiente.

QuickSort

- Divisão: escolher um pivô. Dividir o array em duas partes em torno do pivô.
- Conquista: Recursivamente ordenar os dois sub-arrays.
- Combinação: Trivial.



QuickSort



Escolha do Pivô

- Particionamento pode ser feito de diferentes formas.
- A principal decisão é escolher o pivô.
  - Primeiro elemento do array.
  - Último elemento do array.
  - Elemento médio do array.
  - Elemento que mais ocorre no array.
  - Elemento mais próximo da média aritmética dos elementos do array.

Rotina de Particionamento

- Em nossa rotina, o pivô é o último elemento.

```
Particiona(A, L, R)
p ← A[R]
i ← R
for j ← R – 1 downto L do
      if A[j] > p then
            i ← i – 1
            swap A[i] ↔ A[j]
swap A[R] ↔ A[i]
return i
```

Exemplo de Particionamento

3	72	1	5	47	34	20	10	9	23

Exemplo de Particionamento

3	72	1	5	47	34	20	10	9	23
								j	i

Exemplo de Particionamento

3	72	1	5	47	34	20	10	9	23
							j	←	i

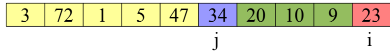
Exemplo de Particionamento

3	72	1	5	47	34	20	10	9	23
						j	←		i

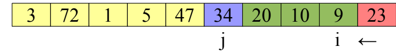
Exemplo de Particionamento

3	72	1	5	47	34	20	10	9	23
					j	←			i

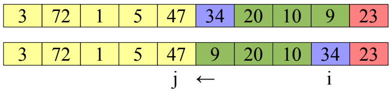
Exemplo de Particionamento



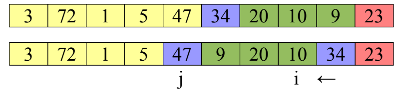
Exemplo de Particionamento



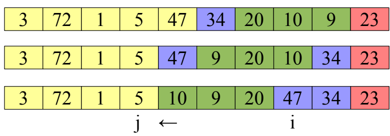
Exemplo de Particionamento



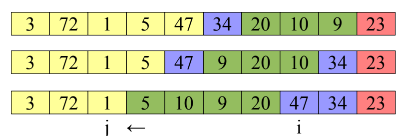
Exemplo de Particionamento



Exemplo de Particionamento



Exemplo de Particionamento



Exemplo de Particionamento

3	72	1	5	47	34	20	10	9	23
3	72	1	5	47	9	20	10	34	23
3	72	1	5	10	9	20	47	34	23
	j ←						i		

© Jorge Figueiredo, DSC/UFCG

Exemplo de Particionamento

3	72	1	5	47	34	20	10	9	23
3	72	1	5	47	9	20	10	34	23
3	72	1	5	10	9	20	47	34	23
	j					i ←			

© Jorge Figueiredo, DSC/UFCG

Exemplo de Particionamento

3	72	1	5	47	34	20	10	9	23
3	72	1	5	47	9	20	10	34	23
3	72	1	5	10	9	20	47	34	23
	j					i ←			

© Jorge Figueiredo, DSC/UFCG

Exemplo de Particionamento

3	72	1	5	47	34	20	10	9	23
3	72	1	5	47	9	20	10	34	23
3	72	1	5	10	9	20	47	34	23
3	20	1	5	10	9	72	47	34	23
j ←						i			

© Jorge Figueiredo, DSC/UFCG

Exemplo de Particionamento

3	72	1	5	47	34	20	10	9	23
3	72	1	5	47	9	20	10	34	23
3	72	1	5	10	9	20	47	34	23
3	20	1	5	10	9	72	47	34	23
j						i			

© Jorge Figueiredo, DSC/UFCG

Exemplo de Particionamento

3	72	1	5	47	34	20	10	9	23
3	72	1	5	47	9	20	10	34	23
3	72	1	5	10	9	20	47	34	23
3	20	1	5	10	9	72	47	34	23
3	20	1	5	10	9	23	47	34	72

© Jorge Figueiredo, DSC/UFCG

QuickSort - Algoritmo

- A chamada inicial é QuickSort(A, 1, n)

```
QuickSort(A, inicio, fim)
if inicio < fim then
      meio ← particiona(A, inicio, fim)
      QuickSort(A, inicio, meio - 1)
      QuickSort(A, meio + 1, fim)
```

Análise do QuickSort

- Requer a resolução de uma relação de recorrência.
- Como nem sempre os dados são divididos em duas metades de mesmo tamanho:
  - Melhor caso, pior caso e caso médio podem variar.
- Vamos assumir:
  - Tempo de partição é $O(n)$.
  - A posição final do pivô é i.

Análise do QuickSort

$T(n) = 1$ $n = 1$
$T(n) = T(n - i) + T(i - 1) + n$ nos demais casos

- Melhor caso:
  - Pivô sempre fica no meio.
  - $T(n) = 2T(n/2) + n$
  - $O(n.\log n)$

Análise do QuickSort

$T(n) = 1$ $n = 1$
$T(n) = T(n - i) + T(i - 1) + n$ nos demais casos

- Pior caso:
  - Pivô sempre fica na primeira ou última posição.
  - $T(n) = T(n - 1) + n$
  - $O(n^2)$

Análise do QuickSort

$T(n) = 1$ $n = 1$
$T(n) = T(n - i) + T(i - 1) + n$ nos demais casos

- Caso médio:
  - Pivô tem a mesma probabilidade $1/n$ de cair em uma das n posições.
  - $T_a(n) = n + 1/n \sum(T_a(i - 1) + T_a(n - i))$
  - $O(n.\log n)$

Ainda sobre QuickSort

- É talvez o algoritmo de ordenação mais usado.
- É fácil de implementar e muito rápido na prática.
- É tipicamente mais do que duas vezes mais rápido do que o MergeSort.

Exercício 1

- Definir um Quicksort diferente, onde o pivô é o primeiro elemento do array.
  - Escreva, em pseudo-código, um procedimento que efetua a partição do array para este novo Quicksort.
  - Ilustre a operação de partição, considerando o array A=[13, 19, 9, 5, 12, 8, 7, 4, 11, 2].

© Jorge Figueiredo, DSC/UFCG

Exercício 2

- Um problema que está relacionado com o problema de ordenação é o de encontrar o *k*-ésimo menor elemento de uma lista não ordenada.
  - Escreva um algoritmo que encontra o k-ésimo menor elemento. Esse algoritmo deve ser $\Theta(n)$ na média e no melhor caso.

Análise e Técnicas de Algoritmos – 2005.1
© Jorge Figueiredo, DSC/UFCG

Ordenação por Comparação

- Todos os algoritmos de ordenação que estudamos até agora utilizam comparação de elementos.
- Em uma ordenação por comparação, a ordem relativa de dois elementos a_i e a_j em uma seqüência é obtida utilizando testes de comparação:
  - $a_i < a_j$, $a_i \leq a_j$, $a_i = a_j$, $a_i > a_j$ e $a_i \geq a_j$.

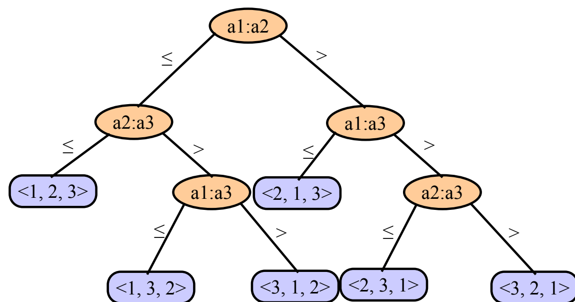
Análise e Técnicas de Algoritmos – 2005.1
© Jorge Figueiredo, DSC/UFCG

Ordenação por Comparação

- O melhor algoritmo que vimos para ordenação é O(n.log n).
- É possível encontrar uma melhor solução?
- Árvore de decisão pode nos ajudar a desvendar isso.
- É possível usar uma árvore de decisão para visualizar ordenação por comparação.

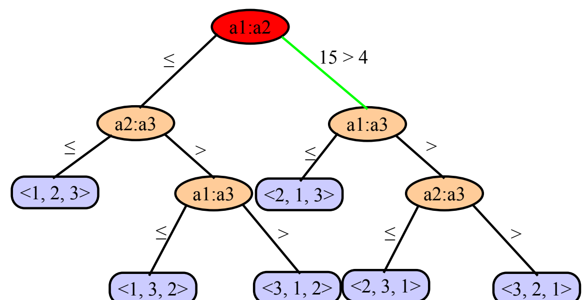
Análise e Técnicas de Algoritmos – 2005.1
© Jorge Figueiredo, DSC/UFCG

Árvore de Decisão



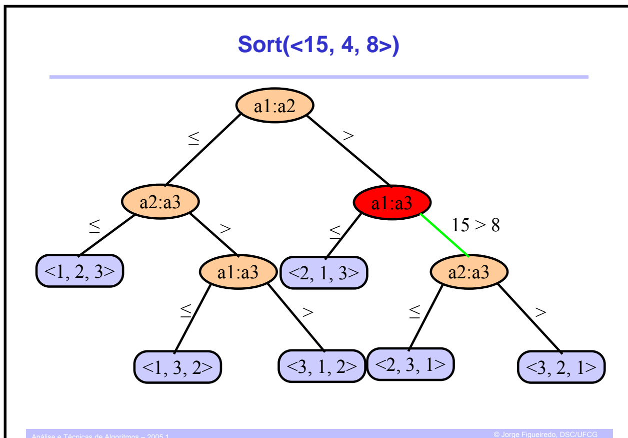
Análise e Técnicas de Algoritmos – 2005.1
© Jorge Figueiredo, DSC/UFCG

Sort(<15, 4, 8>)

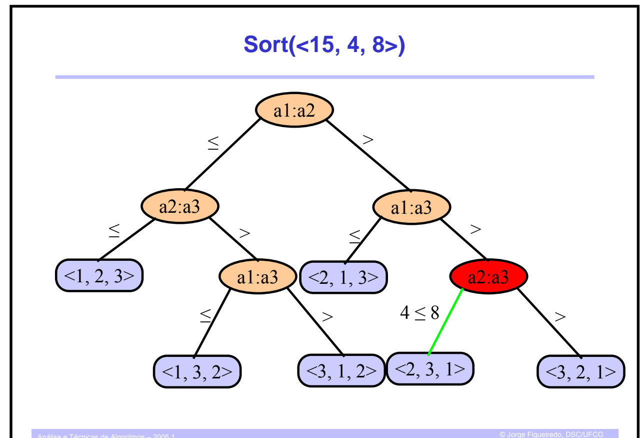


Análise e Técnicas de Algoritmos – 2005.1
© Jorge Figueiredo, DSC/UFCG

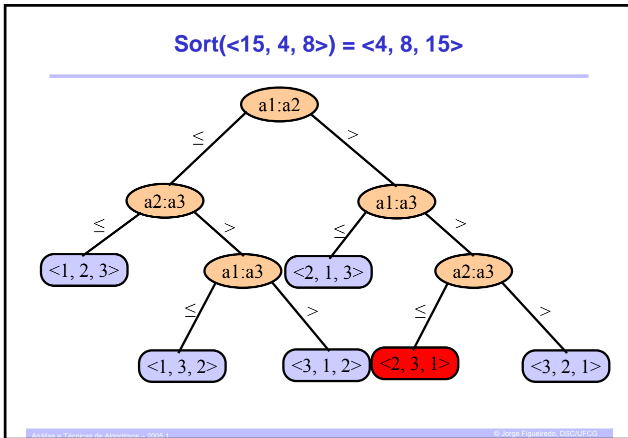
Sort(<15, 4, 8>)



Sort(<15, 4, 8>)



Sort(<15, 4, 8>) = <4, 8, 15>



Árvore de decisão

- As folhas de de uma árvore de decisão indicam uma possível ordenação para os elementos.
  - O número de permutações possível é *n!*.
- Para definir um limite inferior:
  - Uma árvore binária tem no máximo 2^d folhas, onde *d* é a sua profundidade.
  - Uma árvore binária com *L* folhas tem profundidade de pelo menos $\lceil \log L \rceil$.

Árvore de decisão

- O caminho mais longo da raiz de uma árvore de decisão para qualquer uma de suas folhas representa o pior caso do número de comparações.
- Para *n* elementos, temos *n!* folhas:
  - $d = \lceil \log n! \rceil$
  - $d \geq \log n!$
  - $\log n! = \Theta(n.\log n)$
  - $d = \Omega(n.\log n)$

Ordenação em Tempo Linear

- Nenhuma comparação é efetuada.
- Counting Sort:

Counting Sort
Entrada: Um array de n números $A = \langle a_1, a_2, \ldots, a_n \rangle$, em que a_i assume valores $\{1, 2, \ldots, k\}$.
Saída: Um array reordenado $B = \langle b_1, b_2, \ldots, b_n \rangle$.
Array auxiliar: Um array $C = \langle c_1, c_2, \ldots, c_k \rangle$

Counting Sort

```
Counting Sort(A, B, k)
for i ← 1 to k do
        C[i] ← 0
for j ← 1 to length[A] do
        C[A[j]] ← C[A[j]] + 1
for i ←2 to k do
        C[i] ← C[i] + C[i - 1]
for j ← length[A] downto 1 do
        B[C[A[j]]] ← A[j]
        C[A[j]] ← C[A[j]] - 1
```


Counting Sort - Exemplo

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B:								

	1	2	3	4	5	6
C:						

Exemplo – Laço 1

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B:								

	1	2	3	4	5	6
C:	0	0	0	0	0	0

Exemplo – Laço 2

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B:								

	1	2	3	4	5	6
C:	0	0	1	0	0	0

Exemplo – Laço 2

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B:								

	1	2	3	4	5	6
C:	0	0	1	0	0	1

Exemplo – Laço 2

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B:								

	1	2	3	4	5	6
C:	0	0	1	1	0	1

Exemplo – Laço 2

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B:								

	1	2	3	4	5	6
C:	1	0	1	1	0	1

Exemplo – Laço 2

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B:								

	1	2	3	4	5	6
C:	1	0	2	1	0	1

Exemplo – Laço 2

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B:								

	1	2	3	4	5	6
C:	1	0	2	2	0	1

Exemplo – Laço 2

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B:								

	1	2	3	4	5	6
C:	2	0	2	2	0	1

Exemplo – Laço 2

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B:								

	1	2	3	4	5	6
C:	2	0	2	3	0	1

Exemplo – Laço 3

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4

	1	2	3	4	5	6	7	8
B:								

	1	2	3	4	5	6
C:	2	0	2	3	0	1

Exemplo – Laço 3

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4
B:								

	1	2	3	4	5	6
C:	2	2	2	3	0	1

Exemplo – Laço 3

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4
B:								

	1	2	3	4	5	6
C:	2	2	4	3	0	1

Exemplo – Laço 3

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4
B:								

	1	2	3	4	5	6
C:	2	2	4	7	0	1

Exemplo – Laço 3

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4
B:								

	1	2	3	4	5	6
C:	2	2	4	7	7	1

Exemplo – Laço 3

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4
B:								

	1	2	3	4	5	6
C:	2	2	4	7	7	8

Exemplo – Laço 4

	1	2	3	4	5	6	7	8
A:	3	6	4	1	3	4	1	4
B:	1	1	3	3	4	4	4	6

	1	2	3	4	5	6
C:	0	2	2	4	7	7

Análise do Counting Sort

- A análise é trivial:
  - O primeiro e terceiro laços correspondem a $O(k)$.
  - O segundo e quarto laços são $O(n)$.
  - O tempo total é, portanto, $O(n+k)$.
  - Na prática, $k = O(n)$.
  - Logo, o counting sort é $O(n)$.
- Counting Sort é estável.

© Jorge Figueiredo, DSC/UFCG