Fontes principais

1. Cormem T. H.; Leiserson C. E.; Rivest R.: Stein C.. *Introduction to Algorithms*, 3^a edição, MIT Press, 2009

2. Análise de algoritmo - IME/USP (prof. Paulo Feofiloff)
   `http://www.ime.usp.br/~pf/analise_de_algoritmos`

Problema de ordenação

Problema de ordenação

Algoritmo	pior caso	melhor caso	caso médio	complexidade de espaço
Insertion-sort	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n)$
Merge-sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$
Selection-sort	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$
Heap-sort	?	?	?	$\Theta(n)$
Quick-sort				$\Theta(n)$
Counting-sort	?	?	?	?
Radix-sort				
Bucket-sort				

Insertion-sort recursivo

Insertion-sort recursivo

```
Insertion-Sort(A, n)
1  se n ≥ 2
2     então Insertion-Sort(A, n − 1)
3           chave = A[i]
4           j = n − 1
5           enquanto (j > 0) e (A[j] > chave)
6              faça A[j + 1] = A[j]
7                   j = j − 1
8           A[j + 1] = chave
```

Análise do Insertion-sort recursivo

Quantas comparações e quantas trocas o algoritmo executa no pior caso?

Análise do Insertion-sort recursivo

Tanto o número de comparações quanto o de trocas é dado pela recorrência:

$$T(n) = \begin{cases} 0 & \text{, se } n = 1 \\ T(n-1) + n & \text{, se } n > 1 \end{cases}$$

$\Theta(n^2)$ comparações e trocas são executadas no pior caso.

Algoritmo Selection-Sort

Iterativo

Algoritmo Selection-Sort

```
Selection-Sort(A, n)                                   custo
1   para i = 1 até n − 1 faça
2       min = i
3       para j = i + 1 até n faça
4           se A[j] < A[min] então
5               min = j
6       A[i] ↔ A[min]
```

Algoritmo Selection-Sort

Selection-Sort(A, n)	custo
1 **para** $i = 1$ **até** $n - 1$ **faça**	$\Theta(n)$
2 $min = i$	$\Theta(n)$
3 **para** $j = i + 1$ **até** n **faça**	$\Theta(n^2)$
4 **se** $A[j] < A[min]$ **então**	$\Theta(n^2)$
5 $min = j$	$\Theta(n^2)$
6 $A[i] \leftrightarrow A[min]$	$\Theta(n^2)$
	Total: $\Theta(n^2)$

Portanto, a complexidade do algoritmo é $\Theta(n^2)$

Análise do algoritmo Selection-Sort (iterativo)

▷ Complexidade de pior caso (elementos em ordem decrescente): $\Theta(n^2)$, com $\Theta(n^2)$ comparações e $\Theta(n)$ trocas.

▷ Complexidade de melhor caso (elementos em ordem crescente): $\Theta(n^2)$

▷ Complexidade de caso médio (vetor aleatório): $\Theta(n^2)$

Análise do algoritmo Selection-Sort (iterativo)

▷ Complexidade de espaço: $\Theta(n)$, pois comparações e trocas são efetuadas diretamente no vetor.

Note que o Selection-Sort realiza mais comparações que o Insertion-Sort, mas menos trocas.

Análise do algoritmo Selection-Sort (iterativo)

Invariante:

O vetor $A[1 \cdots i-1]$ está ordenado e $A[1 \cdots i-1] < A[i \cdots n]$

A partir desta informação valiosa demonstre a corretude do algoritmo Selection-Sort

Análise do algoritmo Selection-Sort (recursivo)

Análise do algoritmo Selection-Sort (recursivo)

```
Selection-Sort(A, i, n)
1   se i < n então
2       min = i
3       para j = i + 1 até n faça
4           se A[j] < A[min] então min = j
5       t = A[min]
6       A[min] = A[i]
7       A[i] = t
8       Selection-Sort(A, i + 1, n)
```

Análise do algoritmo Selection-Sort (recursivo)

Quantas comparações e quantas trocas o algoritmo executa no pior caso?

O número de comparações é dado pela recorrência:

$$T(n) = \begin{cases} 0 & \text{, se } n = 1 \\ T(n-1) + \Theta(n) & \text{, se } n > 1 \end{cases}$$

$\Theta(n^2)$ comparações são executadas no pior caso.

Análise do algoritmo Selection-Sort (recursivo)

O número de trocas é dado pela recorrência:

$$T(n) = \begin{cases} 0 & , \text{ se } n = 1 \\ T(n-1) + \Theta(1) & , \text{ se } n > 1 \end{cases}$$

$\Theta(n)$ trocas são executadas no pior caso.

Insertion-Sort e Selection-Sort têm a mesma complexidade assintótica, porém em situações em que a operação de troca é muito custosa, é preferível utilizar Selection-Sort.

Ordenação por intercalação

Ordenação por intercalação

```
Merge-Sort(A, p, r)
1   se p < r
2       então q ← ⌊(p+r)/2⌋
3             Merge-Sort(A, p, q)
4             Merge-Sort(A, q + 1, r)
5             Merge(A, p, q, r)
```

Já analisamos a complexidade deste algoritmo, lembra?

Algoritmo Tim-Sort

Algoritmo Tim-Sort

Desenvolvido por Tim Peters em 2002 para a linguagem Python

É um algoritmo de ordenação híbrido de Merge Sort e Insertion-Sort

- Complexidade de pior caso: $\Theta(n \log n)$
- Complexidade de melhor caso: $\Theta(n)$
- Complexidade de caso médio: $\Theta(n \log n)$

Algoritmo Bogo-Sort

Algoritmo Bogo-Sort

É um algoritmo de ordenação perversamente ineficiente

É um algoritmo probabilístico por natureza

Para ordenar os n elementos distintos a complexidade esperada é $O(n \cdot n!)$

Não é utilizada na prática, mas útil em disciplinas de análise de algoritmos

Obrigado