Método InsertionSort

Estrutura de Dados II

Jairo Francisco de Souza

Método InsertionSort (Inserção)

- Técnica básica
- Ordenação de cartas de baralho.
- Compara-se dois elementos. Se o primeiro for maior que o segundo, compara-se o elemento com os seus anteriores até encontrar um elemento anterior menor.
- É feita uma única passagem, mas com vários retornos.
- O procedimento é encerrado quando ordena-se o elemento da última posição do vetor.

Método InsertionSort (Inserção)

- Vantagens
  - Simplicidade do algoritmo
  - Duas vezes mais rápido que BubbleSort e normalmente mais rápido que SelectionSort
- Desvantagens
  - Ainda é considerado um algoritmo lento
- Indicações
  - Tabelas muito pequenas
  - Demonstrações didáticas

Método InsertionSort (Inserção)

```
(1) procedimento InsertionSort(A : lista, n: inteiro)
(2)    para i ← 1 até n faça
(3)        pivo ← A[i];
(4)        j ← i – 1;
(5)        enquanto (j >= 0 AND A[j] > pivo) faça
(6)            A[j+1] = A[j];
(7)            j ← j – 1;
(8)       fim-enquanto
(9)     A[j + 1] = pivo;
(10)   fim-para
```

Método InsertionSort (Inserção)

44	55	12	42	94	18	06	67

1ª Iteração:

pivo = 55
j = 0

Não há troca

44	55	12	42	94	18	06	67

Método InsertionSort (Inserção)

44	55	12	42	94	18	06	67

2ª Iteração:

pivo = 12
j = 1

Anda com 55 uma casa

44	55	55	42	94	18	06	67

Método InsertionSort (Inserção)

44	55	12	42	94	18	06	67

2ª Iteração:

pivo = 12
j = 0

Anda com 44 uma casa.

44	44	55	42	94	18	06	67

Método InsertionSort (Inserção)

44	55	12	42	94	18	06	67

2ª Iteração:

pivo = 12
j = -1

A[j+1] = pivo

12	44	55	42	94	18	06	67

Método InsertionSort (Inserção)

12	44	55	42	94	18	06	67

3ª Iteração:

pivo = 42
j = 2

Anda com o 55 uma casa

12	44	55	55	94	18	06	67

Método InsertionSort (Inserção)

12	44	55	42	94	18	06	67

3ª Iteração:

pivo = 42
j = 1

Anda com o 42 uma casa

12	44	44	55	94	18	06	67

Método InsertionSort (Inserção)

12	44	55	42	94	18	06	67

3ª Iteração:

pivo = 42
j = 0

12 > 42. Critério de parada. A[j+1] = pivo

12	42	44	55	94	18	06	67

Análise da Complexidade

Se a tabela estiver ordenada (melhor caso), então o algoritmo a percorre somente 1 vez. Ou seja, O(n).

Por outro lado, elementos podem ser mudados de posição repetidas vezes, o que torna o algoritmo lento. Ou seja, no pior caso, temos $O(n^2)$

O caso médio é $O(n^2)$, contudo, como os elementos anteriores a A[i] já estão corretamente ordenados, o método é mais rápido que o BubbleSort e um pouco mais rápido que o SelectionSort.

Estudo da estabilidade

O algoritmo é considerado estável, pois não há a possibilidade de elementos iguais mudar de posição no processo de ordenação

Exemplo: [4^1 6 7 2 9 8 4^2 1 3 4^3 9 0]

1ª Iteração: [**4^1 6** | 7 2 9 8 4^2 1 3 4^3 9 0]

5ª Iteração: [**2 4^1 6 7 8 9** | 4^2 6 7 4^3 9 0]

6ª Iteração: [**2 4^1 4^2 6 7 8 9** | 6 7 4^3 9 0]

9ª Iteração: [**2 4^1 4^2 4^3 6 6 7 7 8 9** | 9 0]

Análise de Desempenho

n	C(n)	M(n)	TEMPO (s)
100	4950	297	0,000
500	124750	1497	0,010
1000	499500	2997	0,020
2000	1999000	5997	0,060
3000	4498500	8997	0,140
4000	7998000	11997	0,310
5000	12497500	14997	0,381
6000	17997000	17997	0,531
7000	24496500	20997	0,731
8000	31996000	23997	0,941
9000	40495500	26997	1,181
10000	49995000	29997	1,452
20000	199990000	59997	5,859
30000	449985000	89997	13,059
40000	799980000	119997	23,244