

Ordenação

Algoritmos e Estruturas de Dados II

Introdução

- Ordenar
 - Processo de rearranjar um conjunto de objetos em uma certa ordem (crescente/decrescente)
- Objetivo
 - Facilitar a recuperação posterior dos itens
 - Ex: Lista telefônica, dicionário, ...
- Por que estudar métodos de ordenação?
 - Diferentes formas de resolver um problema
 - Olhar crítico: quais critérios utilizar na escolha de um?
Como analisar/comparar diferentes métodos?

Conceitos básicos

- Chave de ordenação
 - O campo(s) em que a ordenação é baseada
- Algoritmos de ordenação
 - Métodos utilizados para ordenar um conjunto de registros de acordo com uma chave

```
typedef long TipoChave;  
typedef struct TipoItem {  
    TipoChave Chave;  
    /* outros componentes */  
} TipoItem;
```

Critérios de avaliação

- Diferentes critérios
 - Estabilidade
 - Localização dos dados
 - Utilização de memória
 - Tipo de ordenação

Estabilidade

- Método é dito estável se a ordem relativa dos registros com chaves iguais não se altera durante a ordenação

João	80
Maria	65
José	70
Carlos	65
Ana	70
Joana	75

Maria	65
Carlos	65
José	70
Ana	70
Joana	75
João	80



Carlos	65
Maria	65
Ana	70
José	70
Joana	75
João	80



Localização dos dados

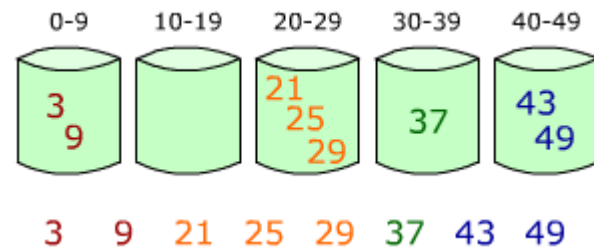
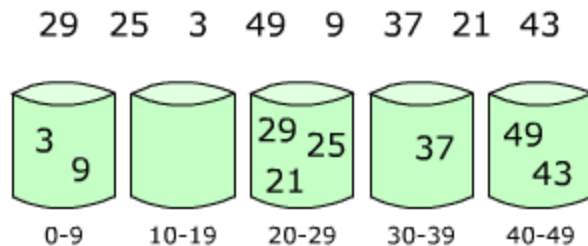
- Ordenação **interna**:
 - Todos os dados na memória principal (RAM).
 - Representação vetorial, acesso em $O(1)$
- Ordenação **externa**:
 - Capacidade limitada da memória principal
 - Dados em memória secundária (disco)

Utilização de memória

- Sem memória extra
 - Não utiliza memória adicional, ou utiliza uma quantidade constante de memória adicional
- Com memória extra
 - Alguns métodos precisam duplicar os registros

Tipo de ordenação

- Comparação
 - São realizadas comparações entre as chaves
 - Maioria dos algoritmos de ordenação
- Distribuição
 - Não são realizadas comparações das chaves
 - Distribuição (agrupamento) → Coleta
 - Exemplo: **bucket sort**



Critério de Avaliação

- Sendo n o número de registros no arquivo, as medidas de complexidade de tempo relevantes são:
 - Número de **comparações** $C(n)$ entre chaves.
 - Número de **movimentações** $M(n)$ de itens
- Complexidade de memória

Outras Considerações

- O uso **econômico da memória** disponível é um requisito primordial na ordenação interna.
- Métodos de ordenação que não usam espaço adicional são os preferidos.
- Métodos que utilizam listas encadeadas **não** são muito utilizados.
- Métodos que fazem cópias dos itens a serem ordenados possuem menor importância.

Métodos de ordenação

- Métodos simples:
 - Complexidade: $C(n) = O(n^2)$
 - Programas pequenos
 - Bons para conjuntos pequenos de itens
- Métodos eficientes
 - Complexidade: $C(n) = O(n \log n)$
 - Programas mais complexos
 - Adequado para conjuntos maiores de itens
- Visualização: <http://www.sorting-algorithms.com>

Métodos simples

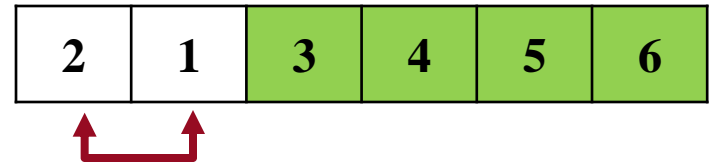
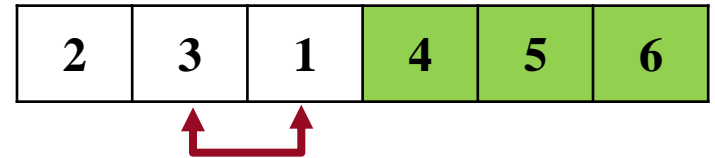
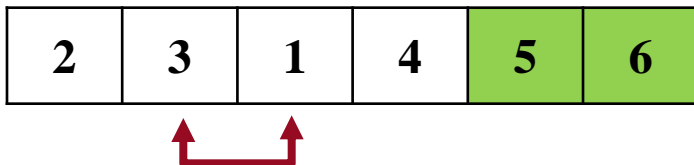
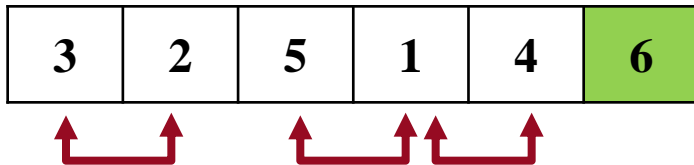
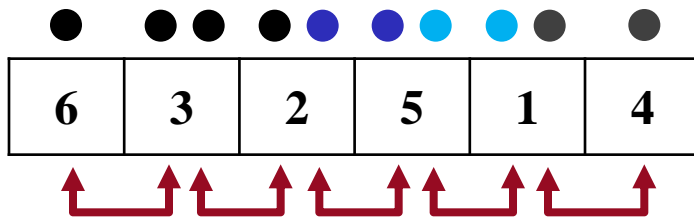
- Bolha (*BubbleSort*)
- Seleção (*SelectSort*)
- Inserção(*InsertSort*)

Método Bolha

- Os elementos vão “subindo” a cada iteração do método até a posição correta para ordenação da lista
- O método poderia parar quando nenhum elemento trocasse de posição
- Como os elementos são trocados frequentemente, há um alto custo de troca de elementos

Método Bolha

6	3	2	5	1	4
---	---	---	---	---	---



Método Bolha

```
void Bolha (Item* v, int n ) {  
    int i, j;  
    Item aux;  
  
    for( i = 0; i < n-1; i++ ) {  
        for( j = 1; j < n-i; j++ ) {  
            if ( v[j].Chave < v[j-1].Chave ) {  
                aux = v[j];  
                v[j] = v[j-1];  
                v[j-1] = aux;  
            }  
        }  
    }  
}
```

Análise de Complexidade

- Comparações – $C(n)$
- Movimentações – $M(n)$

Análise de Complexidade

■ Comparações – $C(n)$

$$\begin{aligned}C(n) &= \sum_{i=0}^{n-2} \sum_{j=1}^{n-i-1} 1 = \sum_{i=0}^{n-2} (n-i-1) = \sum_{i=0}^{n-2} n - \sum_{i=0}^{n-2} i - \sum_{i=0}^{n-2} 1 \\&= n(n-1) - \frac{(0+n-2)(n-1)}{2} - (n-1) \\&= \frac{n^2 - n}{2} = O(n^2)\end{aligned}$$

■ Movimentações – $M(n)$ (pior caso)

$$M(n) = 3C(n)$$

Ordenação por Bolha

■ Vantagens:

- ❑ Algoritmo simples
- ❑ Algoritmo **estável**

■ Desvantagens:

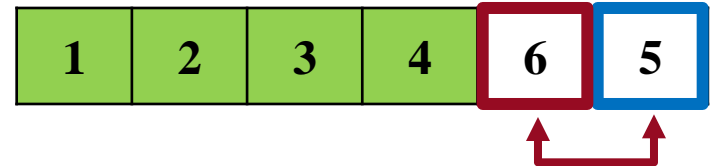
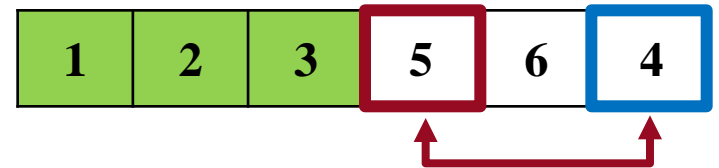
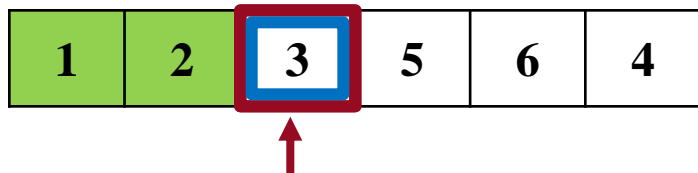
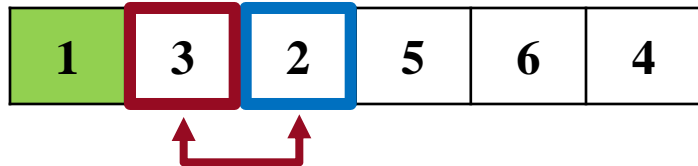
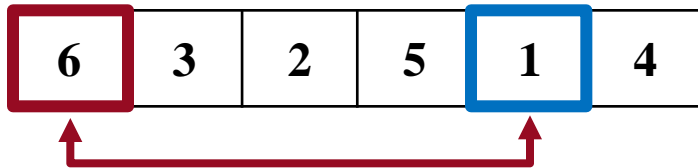
- ❑ O fato de o arquivo já estar ordenado não ajuda em nada, pois o custo de comparações continua quadrático.
- ❑ Muitas comparações redundantes

Método Seleção

- Algoritmo bastante intuitivo
 - Seleção do n -ésimo menor (ou maior) elemento da lista
 - Troca o n -ésimo menor (ou maior) elemento com o n -ésimo elemento da lista
 - Uma única troca por vez é realizada

Método Seleção

6	3	2	5	1	4
---	---	---	---	---	---



Método Seleção

```
void Selecao (Item* v, int n) {  
    int i, j, Min;  
    Item x;  
  
    for (i = 0; i < n - 1; i++) {  
        Min = i;  
        for (j = i + 1 ; j < n; j++) {  
            if ( v[j].Chave < v[Min].Chave)  
                Min = j;  
        }  
        x = v[Min];  
        v[Min] = v[i];  
        v[i] = x;  
    }  
}
```

Análise de Complexidade

- Comparações – $C(n)$
- Movimentações – $M(n)$

Análise de Complexidade

■ Comparações – $C(n)$

$$\begin{aligned}C(n) &= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} (n-i-1) = \sum_{i=0}^{n-2} n - \sum_{i=0}^{n-2} i - \sum_{i=0}^{n-2} 1 \\&= n(n-1) - \frac{(0+n-2)(n-1)}{2} - (n-1) \\&= \frac{n^2 - n}{2} = O(n^2)\end{aligned}$$

■ Movimentações – $M(n)$

$$M(n) = 3(n-1)$$

Ordenação por Seleção

■ Vantagens:

- ❑ Custo linear no tamanho da entrada para o número de movimentos de registros.
- ❑ É um bom algoritmo a ser utilizado **para arquivos com registros muito grandes.**
- ❑ É muito interessante para arquivos pequenos.

■ Desvantagens:

- ❑ O fato de o arquivo já estar ordenado não ajuda em nada, pois o custo continua quadrático.
- ❑ O algoritmo **não é estável.**

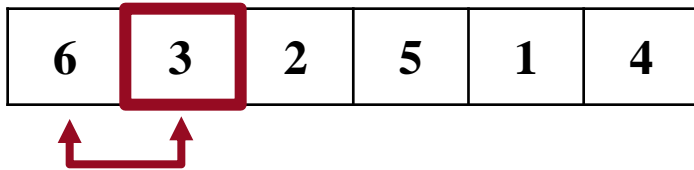
Método Inserção

- Algoritmo utilizado pelo jogador de cartas
 - As cartas são ordenadas da esquerda para direita uma por uma.
 - O jogador escolhe a segunda carta e verifica se ela deve ficar antes ou na posição que está.
 - Depois a terceira carta é classificada, deslocando-a até sua correta posição
 - O jogador realiza esse procedimento até ordenar todas as cartas
- Alto custo em remover uma carta de uma posição e colocá-la em outra quando a representação é por arranjos

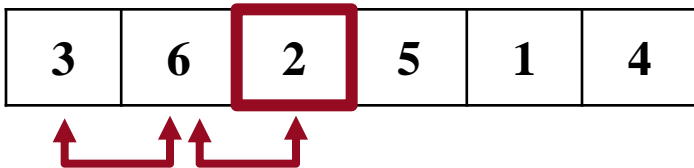
Método Inserção

6	3	2	5	1	4
---	---	---	---	---	---

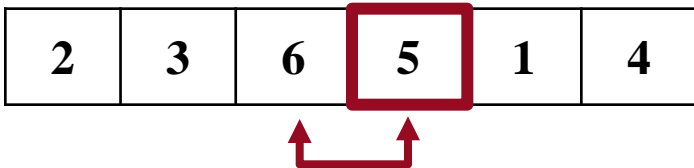
6	3	2	5	1	4
---	---	---	---	---	---



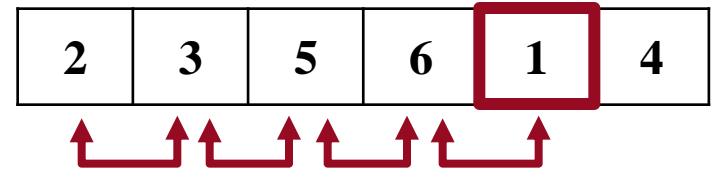
3	6	2	5	1	4
---	---	---	---	---	---



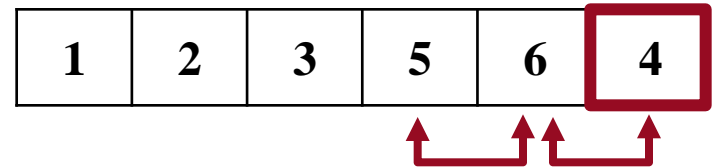
2	3	6	5	1	4
---	---	---	---	---	---



2	3	5	6	1	4
---	---	---	---	---	---



1	2	3	5	6	4
---	---	---	---	---	---



1	2	3	4	5	6
---	---	---	---	---	---



Método Inserção

```
void Insercao (Item* v, int n ) {  
    int i,j;  
    Item aux;  
  
    for (i = 1; i < n; i++) {  
        aux = A[i];  
        j = i - 1;  
        while ( ( j >= 0 ) && ( aux.Chave < v[j].Chave ) ) {  
            v[j + 1] = v[j];  
            j--;  
        }  
        v[j + 1] = aux;  
    }  
}
```

Análise de Complexidade

- Comparações – $C(n)$
- Movimentações – $M(n)$

Análise de Complexidade

■ Comparações – $C(n)$

- No anel mais interno, na i -ésima iteração, o valor de C_i é:
 - melhor caso : $C_i(n) = 1$
 - pior caso : $C_i(n) = i$
 - caso médio : $C_i(n) = 1/i (1 + 2 + \dots + i) = (i+1)/2$
- Assumindo que todas as permutações de n são igualmente prováveis no caso médio, temos:
 - melhor caso: $C(n) = (1 + 1 + \dots + 1) = n - 1$
 - pior caso : $C(n) = (1 + 2 + \dots + n-1) = n^2/2 - n/2$
 - caso médio : $C(n) = \frac{1}{2} (2 + 3 + \dots + n) = n^2/4 + n/4 - 1/2$

Análise de Complexidade

■ Movimentações – $M(n)$

$$M_i(n) = C_i(n) + 2 \text{ (iteração } i\text{)}$$

□ Logo, o número de movimentos é:

□ melhor caso : $M(n) = (3 + 3 + \dots + 3) = 3(n-1)$

□ pior caso : $M(n) = (3 + 4 + \dots + n + 1) = n^2/2 + 3n/2 - 2$

□ caso medio : $M(n) = \frac{1}{2} (4 + 5 + \dots + n + 2) = n^2/4 + 5n/4 - 1/2$

Ordenação por Inserção

- O número mínimo de comparações e movimentos ocorre quando os itens estão originalmente em ordem.
- O número máximo ocorre quando os itens estão originalmente na ordem reversa.
- É o método a ser utilizado quando o arquivo está “quase” ordenado.
- É um bom método quando se deseja adicionar uns poucos itens a um arquivo ordenado, pois o custo é linear.
- O algoritmo de ordenação por inserção é **estável**.