COM112 - ALGORITMO E ESTRUTURA DE DADOS II[1]

Pedro Henrique Del Bianco Hokama
UNIFEI

[1]Baseado nos slides elaborados por Charles Ornelas Almeida, Israel Guerra e Nivio Ziviani

Ordenação

Ordenação

- Ordenar: processo de rearranjar um conjunto de objetos em uma ordem ascendente ou descendente.

Ordenação

- Ordenar: processo de rearranjar um conjunto de objetos em uma ordem ascendente ou descendente.
- A ordenação visa facilitar a recuperação posterior de itens do conjunto ordenado.

Ordenação

- Ordenar: processo de rearranjar um conjunto de objetos em uma ordem ascendente ou descendente.
- A ordenação visa facilitar a recuperação posterior de itens do conjunto ordenado.
  - Dificuldade de se utilizar um catálogo telefônico se os nomes das pessoas não estivessem listados em ordem alfabética.

Ordenação

- Ordenar: processo de rearranjar um conjunto de objetos em uma ordem ascendente ou descendente.
- A ordenação visa facilitar a recuperação posterior de itens do conjunto ordenado.
  - Dificuldade de se utilizar um catálogo telefônico se os nomes das pessoas não estivessem listados em ordem alfabética.
- Notação utilizada nos algoritmos:

Ordenação

- Ordenar: processo de rearranjar um conjunto de objetos em uma ordem ascendente ou descendente.
- A ordenação visa facilitar a recuperação posterior de itens do conjunto ordenado.
  - Dificuldade de se utilizar um catálogo telefônico se os nomes das pessoas não estivessem listados em ordem alfabética.
- Notação utilizada nos algoritmos:
  - Os algoritmos trabalham sobre os registros de um arquivo ou dados na memória.

Ordenação

- Ordenar: processo de rearranjar um conjunto de objetos em uma ordem ascendente ou descendente.
- A ordenação visa facilitar a recuperação posterior de itens do conjunto ordenado.
  - Dificuldade de se utilizar um catálogo telefônico se os nomes das pessoas não estivessem listados em ordem alfabética.
- Notação utilizada nos algoritmos:
  - Os algoritmos trabalham sobre os registros de um arquivo ou dados na memória.
  - Cada registro possui uma chave utilizada para controlar a ordenação.

Ordenação

- Ordenar: processo de rearranjar um conjunto de objetos em uma ordem ascendente ou descendente.
- A ordenação visa facilitar a recuperação posterior de itens do conjunto ordenado.
  - Dificuldade de se utilizar um catálogo telefônico se os nomes das pessoas não estivessem listados em ordem alfabética.
- Notação utilizada nos algoritmos:
  - Os algoritmos trabalham sobre os registros de um arquivo ou dados na memória.
  - Cada registro possui uma chave utilizada para controlar a ordenação.
  - Podem existir outros componentes em um registro.

Ordenação

- Estrutura de um registro:

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

Ordenação

- Estrutura de um registro:

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

- Qualquer tipo de chave sobre o qual exista uma regra de ordenação bem-definida pode ser utilizado.

Ordenação

- Estrutura de um registro:

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

- Qualquer tipo de chave sobre o qual exista uma regra de ordenação bem-definida pode ser utilizado.
- Um método de ordenação é estável se a ordem relativa dos itens com chaves iguais não se altera durante a ordenação.

Ordenação

Ordenação

- Alguns dos métodos de ordenação mais eficientes não são estáveis.

Ordenação

- Alguns dos métodos de ordenação mais eficientes não são estáveis.
- A estabilidade pode ser forçada quando o método é não-estável.

Ordenação

- Alguns dos métodos de ordenação mais eficientes não são estáveis.
- A estabilidade pode ser forçada quando o método é não-estável.
- Sedgewick (1988) sugere agregar um pequeno índice a cada chave antes de ordenar, ou então aumentar a chave de alguma outra forma

Ordenação

Ordenação

- Classificação dos métodos de ordenação:

Ordenação

- Classificação dos métodos de ordenação:
  - Interna: arquivo a ser ordenado cabe todo na memória principal.

Ordenação

- Classificação dos métodos de ordenação:
  - Interna: arquivo a ser ordenado cabe todo na memória principal.
  - Externa: arquivo a ser ordenado não cabe na memória principal.

Ordenação

- Classificação dos métodos de ordenação:
  - Interna: arquivo a ser ordenado cabe todo na memória principal.
  - Externa: arquivo a ser ordenado não cabe na memória principal.
- Diferenças entre os métodos:

Ordenação

- Classificação dos métodos de ordenação:
  - Interna: arquivo a ser ordenado cabe todo na memória principal.
  - Externa: arquivo a ser ordenado não cabe na memória principal.
- Diferenças entre os métodos:
  - Em um método de ordenação interna, qualquer registro pode ser imediatamente acessado.

Ordenação

- Classificação dos métodos de ordenação:
  - Interna: arquivo a ser ordenado cabe todo na memória principal.
  - Externa: arquivo a ser ordenado não cabe na memória principal.
- Diferenças entre os métodos:
  - Em um método de ordenação interna, qualquer registro pode ser imediatamente acessado.
  - Em um método de ordenação externa, os registros são acessados sequencialmente ou em grandes blocos.

Ordenação

- Classificação dos métodos de ordenação:
  - Interna: arquivo a ser ordenado cabe todo na memória principal.
  - Externa: arquivo a ser ordenado não cabe na memória principal.
- Diferenças entre os métodos:
  - Em um método de ordenação interna, qualquer registro pode ser imediatamente acessado.
  - Em um método de ordenação externa, os registros são acessados sequencialmente ou em grandes blocos.
- A maioria dos métodos de ordenação é baseada em comparações das chaves.

Ordenação

- Classificação dos métodos de ordenação:
  - Interna: arquivo a ser ordenado cabe todo na memória principal.
  - Externa: arquivo a ser ordenado não cabe na memória principal.
- Diferenças entre os métodos:
  - Em um método de ordenação interna, qualquer registro pode ser imediatamente acessado.
  - Em um método de ordenação externa, os registros são acessados sequencialmente ou em grandes blocos.
- A maioria dos métodos de ordenação é baseada em comparações das chaves.
- Existem métodos de ordenação que utilizam o princípio da distribuição.

Ordenação por distribuição

Ordenação por distribuição

- Exemplo de ordenação por distribuição: considere o problema de ordenar um baralho com 52 cartas na ordem:

$$\clubsuit < \diamondsuit < \heartsuit < \spadesuit$$

desempatando por

$$A < 2 < 3 < \ldots < 10 < J < Q < K$$

Ordenação por distribuição

- Exemplo de ordenação por distribuição: considere o problema de ordenar um baralho com 52 cartas na ordem:

$$\clubsuit < \diamondsuit < \heartsuit < \spadesuit$$

desempatando por

$$A < 2 < 3 < \ldots < 10 < J < Q < K$$

- Algoritmo:
  1. Distribuir as cartas em treze montes: ases, dois, três, . . ., reis.
  2. Colete os montes na ordem contraria, de forma que o K fique em cima.
  3. Distribua novamente as cartas em quatro montes: paus, ouros, copas e espadas.
  4. Colete os montes na ordem especificada.

Ordenação por distribuição

Ordenação por distribuição

- Métodos como o ilustrado são também conhecidos como **ordenação digital**, **radixsort** ou **bucketsort**.

Ordenação por distribuição

- Métodos como o ilustrado são também conhecidos como **ordenação digital**, **radixsort** ou **bucketsort**.
- O método não utiliza comparação entre chaves.

Ordenação por distribuição

- Métodos como o ilustrado são também conhecidos como **ordenação digital**, **radixsort** ou **bucketsort**.
- O método não utiliza comparação entre chaves.
- Uma das dificuldades de implementar este método está relacionada com o problema de lidar com cada monte.

Ordenação por distribuição

- Métodos como o ilustrado são também conhecidos como **ordenação digital**, **radixsort** ou **bucketsort**.
- O método não utiliza comparação entre chaves.
- Uma das dificuldades de implementar este método está relacionada com o problema de lidar com cada monte.
- Se para cada monte nós reservarmos uma área, então a demanda por memória extra pode tornar-se proibitiva.

Ordenação por distribuição

- Métodos como o ilustrado são também conhecidos como **ordenação digital**, **radixsort** ou **bucketsort**.
- O método não utiliza comparação entre chaves.
- Uma das dificuldades de implementar este método está relacionada com o problema de lidar com cada monte.
- Se para cada monte nós reservarmos uma área, então a demanda por memória extra pode tornar-se proibitiva.
- Sabendo a priori a distribuição das cartas o custo para ordenar um arquivo com n elementos é da ordem de $O(n)$.

Ordenação Interna

Ordenação Interna

- Na escolha de um algoritmo de ordenação interna deve ser considerado o tempo gasto pela ordenação.

Ordenação Interna

- Na escolha de um algoritmo de ordenação interna deve ser considerado o tempo gasto pela ordenação.
- Sendo n o número registros no arquivo, as medidas de complexidade relevantes são:

Ordenação Interna

- Na escolha de um algoritmo de ordenação interna deve ser considerado o tempo gasto pela ordenação.
- Sendo n o número registros no arquivo, as medidas de complexidade relevantes são:
  - Número de comparações $C(n)$ entre chaves.

Ordenação Interna

- Na escolha de um algoritmo de ordenação interna deve ser considerado o tempo gasto pela ordenação.
- Sendo n o número registros no arquivo, as medidas de complexidade relevantes são:
  - Número de comparações $C(n)$ entre chaves.
  - Número de movimentações $M(n)$ de itens do arquivo.

Ordenação Interna

- Na escolha de um algoritmo de ordenação interna deve ser considerado o tempo gasto pela ordenação.
- Sendo n o número registros no arquivo, as medidas de complexidade relevantes são:
  - Número de comparações $C(n)$ entre chaves.
  - Número de movimentações $M(n)$ de itens do arquivo.
- O uso econômico da memória disponível é um requisito primordial na ordenação interna.

Ordenação Interna

- Na escolha de um algoritmo de ordenação interna deve ser considerado o tempo gasto pela ordenação.
- Sendo n o número registros no arquivo, as medidas de complexidade relevantes são:
  - Número de comparações $C(n)$ entre chaves.
  - Número de movimentações $M(n)$ de itens do arquivo.
- O uso econômico da memória disponível é um requisito primordial na ordenação interna.
- Métodos de ordenação **in situ** são os preferidos.

Ordenação Interna

- Na escolha de um algoritmo de ordenação interna deve ser considerado o tempo gasto pela ordenação.
- Sendo n o número registros no arquivo, as medidas de complexidade relevantes são:
  - Número de comparações $C(n)$ entre chaves.
  - Número de movimentações $M(n)$ de itens do arquivo.
- O uso econômico da memória disponível é um requisito primordial na ordenação interna.
- Métodos de ordenação **in situ** são os preferidos.
- Métodos que utilizam listas encadeadas não são muito utilizados.

Ordenação Interna

- Na escolha de um algoritmo de ordenação interna deve ser considerado o tempo gasto pela ordenação.
- Sendo n o número registros no arquivo, as medidas de complexidade relevantes são:
  - Número de comparações $C(n)$ entre chaves.
  - Número de movimentações $M(n)$ de itens do arquivo.
- O uso econômico da memória disponível é um requisito primordial na ordenação interna.
- Métodos de ordenação **in situ** são os preferidos.
- Métodos que utilizam listas encadeadas não são muito utilizados.
- Métodos que fazem cópias dos itens a serem ordenados possuem menor importância.

Ordenação Interna

Ziviani classifica os métodos de ordenação interna:

Ordenação Interna

Ziviani classifica os métodos de ordenação interna:

- Métodos simples:

Ordenação Interna

Ziviani classifica os métodos de ordenação interna:

- Métodos simples:
  - Adequados para pequenos arquivos.

Ordenação Interna

Ziviani classifica os métodos de ordenação interna:

- Métodos simples:
  - Adequados para pequenos arquivos.
  - Requerem $O(n^2)$ comparações.

Ordenação Interna

Ziviani classifica os métodos de ordenação interna:

- Métodos simples:
  - Adequados para pequenos arquivos.
  - Requerem $O(n^2)$ comparações.
  - Produzem programas pequenos.

Ordenação Interna

Ziviani classifica os métodos de ordenação interna:

- Métodos simples:
  - Adequados para pequenos arquivos.
  - Requerem $O(n^2)$ comparações.
  - Produzem programas pequenos.
- Métodos eficientes:

Ordenação Interna

Ziviani classifica os métodos de ordenação interna:

- Métodos simples:
  - Adequados para pequenos arquivos.
  - Requerem $O(n^2)$ comparações.
  - Produzem programas pequenos.
- Métodos eficientes:
  - Adequados para arquivos maiores.

Ordenação Interna

Ziviani classifica os métodos de ordenação interna:

- Métodos simples:
  - Adequados para pequenos arquivos.
  - Requerem $O(n^2)$ comparações.
  - Produzem programas pequenos.
- Métodos eficientes:
  - Adequados para arquivos maiores.
  - Requerem $O(nlogn)$ comparações.

Ordenação Interna

Ziviani classifica os métodos de ordenação interna:

- Métodos simples:
  - Adequados para pequenos arquivos.
  - Requerem $O(n^2)$ comparações.
  - Produzem programas pequenos.
- Métodos eficientes:
  - Adequados para arquivos maiores.
  - Requerem $O(nlogn)$ comparações.
  - Usam menos comparações.

Ordenação Interna

Ziviani classifica os métodos de ordenação interna:

- Métodos simples:
  - Adequados para pequenos arquivos.
  - Requerem $O(n^2)$ comparações.
  - Produzem programas pequenos.
- Métodos eficientes:
  - Adequados para arquivos maiores.
  - Requerem $O(nlogn)$ comparações.
  - Usam menos comparações.
  - As comparações são mais complexas nos detalhes.

Ordenação Interna

Ziviani classifica os métodos de ordenação interna:

- Métodos simples:
  - Adequados para pequenos arquivos.
  - Requerem $O(n^2)$ comparações.
  - Produzem programas pequenos.
- Métodos eficientes:
  - Adequados para arquivos maiores.
  - Requerem $O(nlogn)$ comparações.
  - Usam menos comparações.
  - As comparações são mais complexas nos detalhes.
  - Métodos simples são mais eficientes para pequenos arquivos.

Ordenação Interna

Ordenação Interna

- Tipos de dados e variáveis utilizados nos algoritmos de ordenação interna:

```
typedef int TipoIndice;
typedef TipoItem TipoVetor[MAXTAM + 1];
/* MAXTAM + 1 por causa da sentinela */
TipoVetor A;
```

Ordenação Interna

- Tipos de dados e variáveis utilizados nos algoritmos de ordenação interna:

```
typedef int TipoIndice;
typedef TipoItem TipoVetor[MAXTAM + 1];
/* MAXTAM + 1 por causa da sentinela */
TipoVetor A;
```

- O índice do vetor vai de 0 até *MaxTam*, devido às chaves sentinelas.

Ordenação Interna

- Tipos de dados e variáveis utilizados nos algoritmos de ordenação interna:

```
typedef int TipoIndice;
typedef TipoItem TipoVetor[MAXTAM + 1];
/* MAXTAM + 1 por causa da sentinela */
TipoVetor A;
```

- O índice do vetor vai de 0 até *MaxTam*, devido às chaves sentinelas.
- O vetor a ser ordenado contém chaves nas posições de 1 até n.

Ordenação por Seleção

Ordenação por Seleção

- Um dos algoritmos mais simples de ordenação.

Ordenação por Seleção

- Um dos algoritmos mais simples de ordenação.
- Algoritmo:

Ordenação por Seleção

- Um dos algoritmos mais simples de ordenação.
- Algoritmo:
  - Selecione o menor item do vetor.

Ordenação por Seleção

- Um dos algoritmos mais simples de ordenação.
- Algoritmo:
  - Selecione o menor item do vetor.
  - Troque-o com o item da primeira posição do vetor.

Ordenação por Seleção

- Um dos algoritmos mais simples de ordenação.
- Algoritmo:
  - Selecione o menor item do vetor.
  - Troque-o com o item da primeira posição do vetor.
  - Repita essas duas operações com os $n - 1$ itens restantes, depois com os $n - 2$ itens, até que reste apenas um elemento.

Ordenação por Seleção

Ordenação por Seleção

- O método é ilustrado abaixo:

	1	2	3	4	5	6
Chaves iniciais:	*O*	*R*	*D*	*E*	*N*	*A*
i = 1	***A***	*R*	*D*	*E*	*N*	***O***
i = 2	*A*	***D***	***R***	*E*	*N*	*O*
i = 3	*A*	*D*	***E***	***R***	*N*	*O*
i = 4	*A*	*D*	*E*	***N***	***R***	*O*
i = 5	*A*	*D*	*E*	*N*	***O***	***R***

Ordenação por Seleção

- O método é ilustrado abaixo:

	1	2	3	4	5	6
Chaves iniciais:	*O*	*R*	*D*	*E*	*N*	*A*
i = 1	***A***	*R*	*D*	*E*	*N*	***O***
i = 2	*A*	***D***	***R***	*E*	*N*	*O*
i = 3	*A*	*D*	***E***	***R***	*N*	*O*
i = 4	*A*	*D*	*E*	***N***	***R***	*O*
i = 5	*A*	*D*	*E*	*N*	***O***	***R***

- As chaves em negrito sofreram uma troca entre si.

Ordenação por Seleção

```
 1  void Selecao(TipoItem *A, TipoIndice n){
 2     TipoIndice i, j, Min;
 3     TipoItem x;
 4     for (i = 0; i < n - 1; i++){
 5        Min = i;
 6        for (j = i + 1; j < n; j++){
 7           if (A[j].Chave < A[Min].Chave){
 8              Min = j;
 9           }
10        }
11        x = A[Min];
12        A[Min] = A[i];
13        A[i] = x;
14     }
15  }
```

Ordenação por Seleção

Ordenação por Seleção

- $C(n) = \frac{n^2}{2} - \frac{n}{2}$

Ordenação por Seleção

- $C(n) = \frac{n^2}{2} - \frac{n}{2}$
- $M(n) = 3(n-1)$

Ordenação por Seleção

- $C(n) = \frac{n^2}{2} - \frac{n}{2}$
- $M(n) = 3(n-1)$
- A atribuição $Min = j$ da linha 8 é executada em média $n \log n$ vezes, Knuth (1973)

Ordenação por Seleção

- $C(n) = \frac{n^2}{2} - \frac{n}{2}$
- $M(n) = 3(n-1)$
- A atribuição $Min = j$ da linha 8 é executada em média $n \log n$ vezes, Knuth (1973)
- A complexidade do algoritmo de ordenação por seleção é portanto $O(n^2)$

Ordenação por Seleção

Vantagens:

Ordenação por Seleção

Vantagens:

- Custo linar para o número de movimentos de registros.

Ordenação por Seleção

Vantagens:

- Custo linar para o número de movimentos de registros.
- É o algoritmo a ser utilizado para arquivos com registros muito grandes.

Ordenação por Seleção

Vantagens:

- Custo linar para o número de movimentos de registros.
- É o algoritmo a ser utilizado para arquivos com registros muito grandes.
- É muito interessante para arquivos pequenos.

Desvantagens:

Ordenação por Seleção

Vantagens:

- Custo linar para o número de movimentos de registros.
- É o 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.

Ordenação por Seleção

Vantagens:

- Custo linar para o número de movimentos de registros.
- É o 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**.

Ordenação por Inserção

Ordenação por Inserção

- Método preferido dos jogadores de cartas.

Ordenação por Inserção

- Método preferido dos jogadores de cartas.
- Algoritmo:

Ordenação por Inserção

- Método preferido dos jogadores de cartas.
- Algoritmo:
  - Em cada passo a partir de i=2 faça:

Ordenação por Inserção

- Método preferido dos jogadores de cartas.
- Algoritmo:
  - Em cada passo a partir de i=2 faça:
    - Selecione o i-ésimo item da sequência fonte.

Ordenação por Inserção

- Método preferido dos jogadores de cartas.
- Algoritmo:
  - Em cada passo a partir de i=2 faça:
    - Selecione o i-ésimo item da sequência fonte.
    - Coloque-o no lugar apropriado na sequência destino de acordo com o critério de ordenação.

Ordenação por Inserção

Ordenação por Inserção

- O método é ilustrado abaixo:

	1	2	3	4	5	6
Chaves iniciais:	**O**	*R*	*D*	*E*	*N*	*A*
i = 2	**O**	**R**	*D*	*E*	*N*	*A*
i = 3	**D**	**O**	**R**	*E*	*N*	*A*
i = 4	**D**	**E**	**O**	**R**	*N*	*A*
i = 5	**D**	**E**	**N**	**O**	**R**	*A*
i = 6	**A**	**D**	**E**	**N**	**O**	**R**

- As chaves em negrito representam a sequência destino.

Ordenação por Inserção

```
 1  void Insercao(TipoItem *A, TipoIndice n){
 2     TipoIndice i, j;
 3     TipoItem x;
 4     for ( i = 2; i <= n ; i++){
 5        x = A[i];
 6        j = i - 1;
 7        A[0] = x; /* sentinela */
 8        while ( x.Chave < A[j].Chave){
 9           A[j+1] = A[j];
10           j--;
11        }
12        A[j+1] = x;
13     }
14  }
```

Ordenação por Inserção

Considerações sobre o algoritmo:

Ordenação por Inserção

Considerações sobre o algoritmo:

- O processo de ordenação pode ser terminado pelas condições:

Ordenação por Inserção

Considerações sobre o algoritmo:

- O processo de ordenação pode ser terminado pelas condições:
  - Um item com chave menor que o item em consideração é encontrado.

Ordenação por Inserção

Considerações sobre o algoritmo:

- O processo de ordenação pode ser terminado pelas condições:
  - Um item com chave menor que o item em consideração é encontrado.
  - O final da sequência destino é atingido à esquerda.

Ordenação por Inserção

Considerações sobre o algoritmo:

- O processo de ordenação pode ser terminado pelas condições:
  - Um item com chave menor que o item em consideração é encontrado.
  - O final da sequência destino é atingido à esquerda.
- Solução:
  - Utilizar um registro sentinela na posição zero do vetor.

Ordenação por Inserção - Complexidade

Ordenação por Inserção - Complexidade

- Seja $C(n)$ a função que conta o número de comparações.

Ordenação por Inserção - Complexidade

- Seja $C(n)$ a função que conta o número de comparações.
- No laço 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: $C_i(n) = \frac{1}{i}(1 + 2 + \ldots + i) = \frac{i+1}{2}$

Ordenação por Inserção - Complexidade

- Seja $C(n)$ a função que conta o número de comparações.
- No laço 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: $C_i(n) = \frac{1}{i}(1 + 2 + \ldots + i) = \frac{i+1}{2}$
- Assumindo que todas as permutações de n são igualmente prováveis no caso médio, temos:
  - $C(n) = (1 + 1 + \ldots + 1) = n - 1$
  - Pior caso: $C(n) = (2 + 3 + \ldots + n) = \frac{n^2}{2} + \frac{n}{2} - 1$
  - Caso médio: $\frac{1}{2}(3 + 4 + \ldots + n + 1) = \frac{n^2}{4} + \frac{3n}{4} - 1$

Ordenação por Inserção - Complexidade

Ordenação por Inserção - Complexidade

- Seja $M(n)$ a função que conta o número de movimentações de registros.

Ordenação por Inserção - Complexidade

- Seja $M(n)$ a função que conta o número de movimentações de registros.
- O número de movimentações na i-ésima iteração é:

$$M_i(n) = C_i(n) - 1 + 3 = C_i(n) + 2$$

Ordenação por Inserção - Complexidade

- Seja $M(n)$ a função que conta o número de movimentações de registros.
- O número de movimentações na i-ésima iteração é:

$$M_i(n) = C_i(n) - 1 + 3 = C_i(n) + 2$$

- Logo, o número de movimentos é:

Ordenação por Inserção - Complexidade

- Seja $M(n)$ a função que conta o número de movimentações de registros.
- O número de movimentações na i-ésima iteração é:

$$M_i(n) = C_i(n) - 1 + 3 = C_i(n) + 2$$

- Logo, o número de movimentos é:
  - Melhor caso: $M(n) = (3 + 3 + \ldots + 3) = 3(n - 1)$

Ordenação por Inserção - Complexidade

- Seja $M(n)$ a função que conta o número de movimentações de registros.
- O número de movimentações na i-ésima iteração é:

$$M_i(n) = C_i(n) - 1 + 3 = C_i(n) + 2$$

- Logo, o número de movimentos é:
  - Melhor caso: $M(n) = (3 + 3 + \ldots + 3) = 3(n - 1)$
  - Pior caso: $M(n) = (4 + 5 + \ldots + n + 2) = \frac{n^2}{2} + \frac{5n}{2} - 3$

Ordenação por Inserção - Complexidade

- Seja $M(n)$ a função que conta o número de movimentações de registros.
- O número de movimentações na i-ésima iteração é:

$$M_i(n) = C_i(n) - 1 + 3 = C_i(n) + 2$$

- Logo, o número de movimentos é:
  - Melhor caso: $M(n) = (3 + 3 + \ldots + 3) = 3(n - 1)$
  - Pior caso: $M(n) = (4 + 5 + \ldots + n + 2) = \frac{n^2}{2} + \frac{5n}{2} - 3$
  - Caso médio: $M(n) = \frac{1}{2}(5 + 6 + \ldots + n + 3) = \frac{n^2}{4} + \frac{11n}{4} - 3$

Ordenação por Inserção - Considerações

Ordenação por Inserção - Considerações

- O número mínimo de comparações e movimentos ocorre quando os itens estão originalmente em ordem.

Ordenação por Inserção - Considerações

- 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.

Ordenação por Inserção - Considerações

- 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.

Ordenação por Inserção - Considerações

- 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.

Ordenação por Inserção - Considerações

- 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.