Ordenação

Ordenação

- Ordenar corresponde ao processo de rearranjar um conjunto de objetos em uma ordem ascendente ou descendente.
- O objetivo principal da ordenação é facilitar a recuperação posterior de itens do conjunto ordenado.

Ordenação

- Os algoritmos de ordenação trabalham sobre arquivos. Um arquivo de tamanho n é uma seqüência de n itens r[0], r[l],...,r[n-1]. Cada item no arquivo é chamado registro.
- Apenas uma parte do registro, chamada chave, é utilizada para controlar a ordenação. Cada chave, k[i], é associada a um registro r[i]. Diz-se que o arquivo está classificado pela chave se i < j implicar que k[i] precede k[j] em alguma classificação nas chaves (no caso de ordenação ascendente).

Ordenação

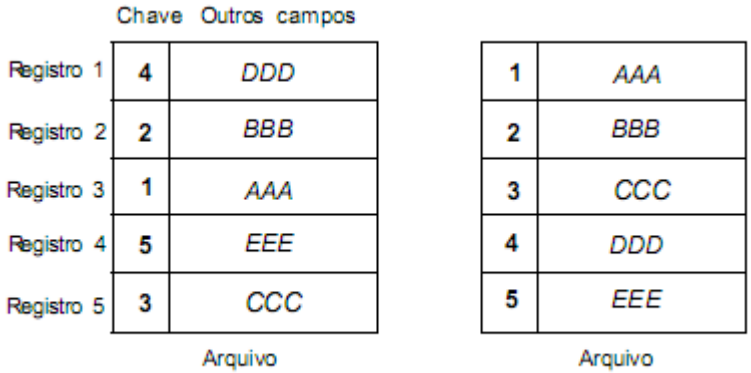
- Uma classificação é considerada **interna** se os registros que ela classificar estiverem na memória principal, e **externa** se alguns dos registros que ela classificar estiverem no armazenamento auxiliar.
- Na ordenação interna, qualquer registro pode ser imediatamente acessado. Os dados são organizados na forma de vetores.
- Na ordenação externa, os registros são acessados sequencialmente ou em grandes blocos. Os dados são organizados em arquivos, onde em cada arquivo apenas o dado de cima é visível.

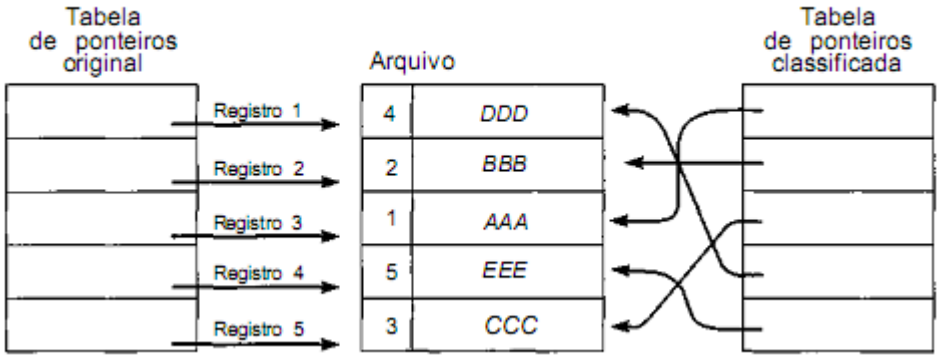
Ordenação

- É possível que dois registros num arquivo tenham a mesma chave. Uma técnica de classificação é chamada **estável** se, para todos os registros i ej, k[i] seja igual a k[j]; se r[i] precede r[j] no arquivo original, r[i] precederá r[j] no arquivo classificado. Ou seja, uma **classificação estável** mantém os registros com a mesma chave na mesma ordem relativa em que estavam antes da classificação.

Ordenação

Uma classificação ocorre sobre os próprios registros ou sobre uma tabela auxiliar de ponteiros.





Ordenação

- Ordenação por Seleção - Um dos algoritmos mais simples de ordenação
- Princípio de funcionamento:
  - Selecione o menor (ou maior) item do vetor
  - Troque-o com o item que esta na primeira (última, no caso do maior) posição do vetor

  Repita estas duas operações com os n-1 itens restantes depois com os n-2 itens até que reste apenas um elemento

Ordenação

- Ordenação por Seleção – Exemplo

	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

- Ordenação por Seleção
  - O algoritmo de ordenação por seleção é um dos métodos de ordenação mais simples que existem.
  - O fato do arquivo já estar ordenado não ajuda em nada.
  - O algoritmo não é estável pois ele nem sempre deixa os registros com chaves iguais na mesma posição relativa

- Ordenação por Troca – Método da Bolha (Bubble Sort)

  A idéia básica por trás da classificação por bolha é percorrer o arquivo seqüencialmente várias vezes. Cada passagem consiste em comparar cada elemento no arquivo com seu sucessor (x[i] com x[i + 1]) e trocar os dois elementos se eles não estiverem na ordem correta.

- Ordenação por Troca – Método da Bolha (Bubble Sort)

Examine o seguinte arquivo: 25, 57, 48, 37, 12, 92, 86, 32.

As seguintes comparações são feitas na primeira passagem:

x[0]	com	x[1]	(25 com 57)	nenhuma troca
x[1]	com	x[2]	(57 com 48)	troca
x[2]	com	x[3]	(57 com 37)	troca
x[3]	com	x[4]	(57 com 12)	troca
x[4]	com	x[5]	(57 com 92)	nenhuma troca
x[5]	com	x[6]	(92 com 86)	troca
x[6]	com	x[7]	(92 com 33)	troca

Após a primeira passagem, o arquivo ficará na ordem

25, 48, 37, 12, 57, 86, 33, 92

- Ordenação por Troca – Método da Bolha (Bubble Sort)

  Observe que, depois dessa primeira passagem, o maior elemento (nesse caso, 92) está em sua posição correta dentro do vetor. Em geral, *x[n- i]* ficará na posição correta depois da iteração *i.* O método é chamado classificação por bolha porque cada número "borbulha" lentamente para posição correta. Depois da segunda passagem, o arquivo fica assim:

  25, 37, 12, 48, 57, 33, 86, 92

- Ordenação por Troca – Método da Bolha (Bubble Sort)

  Observe que, agora, 86 ficou posicionado na segunda posição mais alta. Como cada iteração posiciona um novo elemento em sua posição correta, um arquivo de *n* elementos não exige mais do que n - 1 iterações.

- Ordenação por Troca – Método da Bolha (Bubble Sort)

  O conjunto completo de iterações fica assim:

iteração 0 (arquivo original)	25 57 48 37 12 92 86 33
iteração 1	25 48 37 12 57 86 33 92
iteração 2	25 37 12 48 57 33 86 92
iteração 3	25 12 37 48 33 57 86 92
iteração 4	12 25 37 33 48 57 86 92
iteração 5	12 25 33 37 48 57 86 92
iteração 6	12 25 33 37 48 57 86 92
iteração 7	12 25 33 37 48 57 86 92

- Ordenação por Troca – Método da Bolha (Bubble Sort)

  Todos os elementos nas posições acima ou iguais a *n - i* já estão na ordem correta depois da iteração *i,* eles não precisam ser considerados nas iterações posteriores. Sendo assim, na primeira passagem, são feitas *n* - 1 comparações, na segunda passagem, *n* - 2 comparações e na passagem *(n* -1) somente uma comparação é feita (entre *x[0] ex[1]).* Portanto, o processo agiliza-se com as sucessivas passagens. Demonstramos que n - 1 passagens são suficientes para classificar um arquivo de tamanho n.

- # Ordenação por Troca – Método da Bolha (Bubble Sort)

  Entretanto, no exemplo do arquivo anterior, de oito elementos, o arquivo foi classificado depois de cinco iterações, tornando as duas últimas iterações desnecessárias.

iteração 0 (arquivo original)	25 57 48 37 12 92 86 33
iteração 1	25 48 37 12 57 86 33 92
iteração 2	25 37 12 48 57 33 86 92
iteração 3	25 12 37 48 33 57 86 92
iteração 4	12 25 37 33 48 57 86 92
iteração 5	12 25 33 37 48 57 86 92
iteração 6	12 25 33 37 48 57 86 92
iteração 7	12 25 33 37 48 57 86 92

- Ordenação por Troca – Método da Bolha (Bubble Sort)

  Para eliminar as passagens desnecessárias, precisamos detectar o fato de que o arquivo já está classificado. Mas essa é uma tarefa simples, uma vez que num arquivo classificado não ocorre nenhuma troca em uma passagem. Se mantivermos um registro da ocorrência ou não de trocas em determinada passagem, poderemos determinar se serão necessárias passagens adicionais. Sob esse método, se o arquivo puder ser classificado em menos de *n* - 1 passagens, a última passagem não fará nenhuma troca.

- Ordenação por Troca – Método da Bolha (Bubble Sort)

  – O melhor caso ocorre quando o arquivo está completamente ordenado.
  – O pior caso ocorre quando o arquivo esta em ordem reversa.

  Isto quer dizer que quanto mais ordenado estiver o arquivo melhor é a atuação do método um arquivo completamente ordenado

Ordenação por ***por troca de partição*** (ou ***quicksort)***

- Quicksort é o algoritmo de ordenação interna mais rápido que se conhece para uma ampla variedade de situações sendo provavelmente mais utilizado do que qualquer outro algoritmo

- Idéia Básica:
  - Partir o problema de ordenar um conjunto com n itens em dois problemas menores
  - Ordenar independentemente os problemas menores
  - Combinar os resultados para produzir a solução do problema maior

QuickSort

- Seja *x* um vetor e *n* o número de elementos no vetor a ser classificado.
- Escolha um elemento *a* numa posição específica dentro do vetor (por exemplo, *a* pode ser escolhido como o primeiro elemento de modo que *a* = x[0]).
- Suponha que os elementos de *x* sejam particionados de modo que *a* seja colocado na posição *j* e as seguintes condições sejam observadas:

1. Cada elemento nas posições 0 até *j* - 1 seja menor ou igual a *a*
2. Cada elemento nas posições *j +* 1 até *n* -1 seja maior ou igual a a.

QuickSort

- Se as duas condições anteriores forem mantidas para determinado *a e j, a* será o i-ésimo menor elemento de *x,* de forma que *a* permanecerá na posição *j* quando o vetor estiver totalmente classificado.

- Se o processo anterior for repetido com os subvetores x[0] até x[j - 1] e x[j + 1] até x[n - 1] e com quaisquer vetores criados pelo processo em sucessivas iterações, o resultado final será um vetor classificado

QuickSort

Exemplo:

- Se um vetor inicial for dado como:

  25, 57, 48, 37, 12, 92, 86, 33

  e o primeiro elemento (25) for colocado na posição correta, o vetor resultante será:

  12, **25**, 57, 48, 37, 92, 86, 33

  Nesse ponto, 25 está em sua posição correta dentro do vetor (x[1]), cada elemento abaixo dessa posição (12) é menor ou igual a 25 e cada elemento acima dessa posição (57, 48, 37, 92, 86 e 33) é maior ou igual a 25.

  Como 25 está em sua posição final, o problema inicial foi decomposto na classificação dos dois subvetores: (12) e (57 48 37 92 86 33)

QuickSort

Exemplo:

Não é necessário fazer nada para classificar o primeiro desses subvetores; um arquivo de um elemento já está classificado. Para classificar o segundo subvetor, o processo é repetido e, em seguida, o subvetor é subdividido. Agora, o vetor inteiro pode ser visualizado como:

12 25 (57 48 37 92 86 33)

onde os parênteses encerram os subvetores que ainda serão classificados. Repetir o processo sobre o subvetor *x[2]* até *x[7]* resulta em:

12 25 (48 37 33) 57 (92 86)

QuickSort

E as repetições posteriores resultam em:

12 25 (37 33) 48 57 (92 86)

12 25 (33) 37 48 57 (92 86)

12 25 33 37 48 57 (92 86)

12 25 33 37 48 57 (86) 92

12 25 33 37 48 57 86 92

O vetor final está classificado. Por aquilo que foi apresentado, o quicksort pode ser definido como um procedimento recursivo