ORDENAÇÃO

Prof. Marcelo Dib

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.

- Por exemplo, Dificuldade de se utilizar um catálogo telefônico se os nomes das pessoas não estivessem listados em ordem alfabética.

Ordenação

- Notação utilizada nos algoritmos:
  - Os algoritmos trabalham sobre os registros de um arquivo.
  - Cada registro possui uma chave utilizada para controlar a ordenação.
  - Podem existir outros componentes em um registro.

Classificação dos métodos de 0rdenação

Métodos eficientes:

- Adequados para arquivos maiores.
- Requerem O(n log n) comparações.
- Usam menos comparações.
- As comparações são mais complexas nos detalhes.
- Métodos simples são mais eficientes para pequenos arquivos.

Métodos simples:

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

Ordenação

Métodos Simples

- Ordenação por Troca
- Por Seleção
- Por Inserção

Métodos Eficientes

- QuickSort

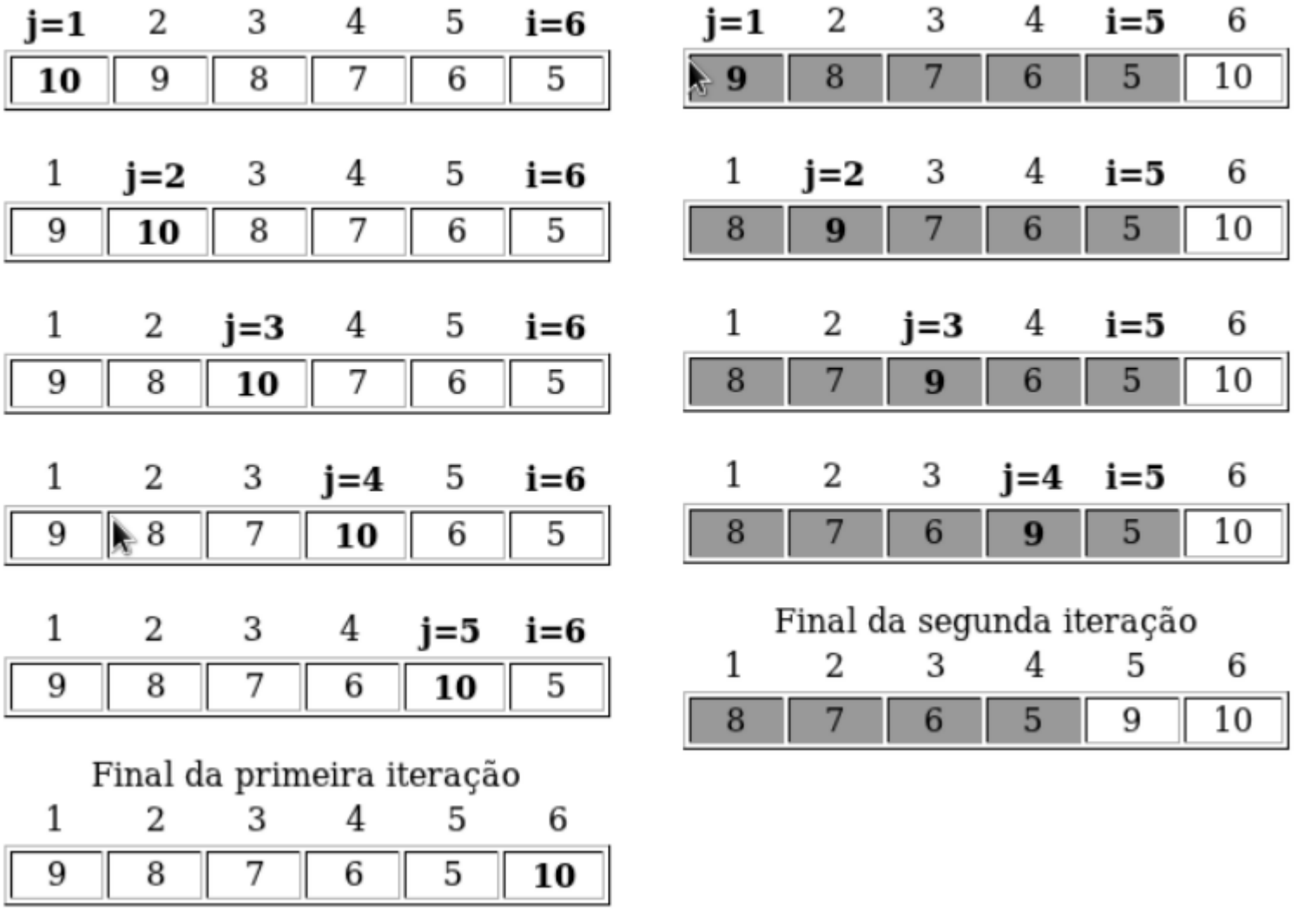
Ordenação

Ordenação por Troca (BubbleSort)

- Outro algoritmo simples, útil para ordenação de vetores pequenos (desempenho ruim).
- Idéia básica:
  - Compare o primeiro elemento com o segundo. Se estiverem desordenados, então efetue a troca de posição. Compare o segundo elemento com o terceiro e efetue a troca de posição, se necessário;
  - Repita a operação anterior até que o penúltimo elemento seja comparado com o último. Ao final desta repetição o elemento de maior valor estará em sua posição correta, a n-ésima posição do vetor;
  - Continue a ordenação posicionando o segundo maior elemento, o terceiro,..., até que todo o vetor esteja ordenado.

Ordenação

- Exemplo:



Ordenação

```
void bubbleSort(int v[200], int n)
{
    int i, j, aux;
    for(i = n-1; i > 0; i--) {
        for(j = 0; j < i; j++) {
            if(v[j] > v[j+1]) {
                aux = v[j]; v[j] = v[j+1]; v[j+1] = aux; //troca
            }
        }
    }
}
```

Ordenação

Ordenação por Seleção

- Um dos algoritmos mais simples de ordenação
- 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

	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

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

Ordenação

```
void selection_sort(int num[], int tam)
  {
    int i, j, min, aux;
    for (i = 0; i < (tam-1); i++)    {
      min = i;
      for (j = (i+1); j < tam; j++) {
        if(num[j] < num[min])
          min = j;       }
      if (i != min) {
        aux = num[i];
         num[i] = num[min];
         num[min] = aux;     }
     }
}
```

Ordenação

Vantagens:

- Custo linear no t tamanho da entrada 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

Ordenação por Inserção

- Em cada passo a partir de i=2 faça:
- Selecione o i-ésimo item da seqüência fonte.
- Coloque-o no lugar apropriado na seqüência destino de acordo com o critério de ordenação.

Ordenação

	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**

Ordenação

```
void insertionSort(int numeros[], int tam) {
   int i, j, eleito;
   for (i = 1; i < tam; i++){
      eleito = numeros[i];
      j = i - 1;
      while ((j>=0) && (eleito < numeros[j])) {
         numeros[j+1] = numeros[j];
         j--;   }
      numeros[j+1] = eleito;  }
}
```

Ordenação

- 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

Quicksort

- Proposto por Hoare em 1960 e publicado em 1962.É o algoritmo de ordenação interna mais rápido que se conhece para uma ampla variedade de situações.
- Provavelmente é o mais utilizado.
- A idéia básica é dividir o problema de ordenar um conjunto com n itens em dois problemas menores.
- Os problemas menores são ordenados independentemente.
- Os resultados são combinados para produzir a solução final.
- A parte mais delicada do método é o processo de partição

Ordenação

- O vetor A[Esq..Dir] é rearranjado por meio da escolha arbitrária de um pivô x.
- O vetor A é particionado em duas partes:
  - A parte esquerda com chaves menores ou iguais a x.
  - A parte direita com chaves maiores ou iguais a x.

Ordenação

Algoritmo para o particionamento:

1. Escolha arbitrariamente um pivô x.
2. Percorra o vetor a partir da esquerda até que A[i] ≥ x.
3. Percorra o vetor a partir da direita até que A[j] ≤ x.
4. Troque A[i] com A[j].
5. Continue este processo até os apontadores i e j se cruzarem.

Ordenação

- Função Partição:

```
void Particao(Indice Esq, Indice Dir,
              Indice *i, Indice *j, Item *A)
{ Item x, w;
  *i = Esq; *j = Dir;
  x = A[(*i + *j)/2]; /* obtem o pivo x */
  do
  { while (x.Chave > A[*i].Chave) (*i)++;
    while (x.Chave < A[*j].Chave) (*j)--;
    if ((*i)≤ (*j))
    { w = A[*i]; A[*i] = A[*j]; A[*j] = w;
      (*i)++; (*j)--;
    }
  } while (*i <= *j);
}
```

Ordenação

- Função Quicksort

```
/* Entra aqui o procedimento Particao */
void Ordena(Indice Esq, Indice Dir, Item *A)
{ Particao(Esq, Dir, &i, &j, A);
  if (Esq < j) Ordena(Esq, j, A);
  if (i < Dir) Ordena(i, Dir, A);
}

void QuickSort(Item *A, Indice *n)
{ Ordena(1, *n, A); }
```

Ordenação

- Exemplo do estado do vetor em cada chamada recursiva do procedimento Ordena:

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

Ordenação

Ao final, o vetor A[Esq..Dir] está particionado de tal forma que:

- Os itens em A[Esq],A[Esq + 1], ..., A[j] são ã menores ou iguais a x;
- Os itens em A[i], A[i + 1], ..., A[Dir] são maiores ou iguais a x.

Ordenação

Análise

Melhor caso:

- C(n) = 2C(n/2) + n = O(n log n)
- Esta situação ocorre quando cada partição divide o arquivo em duas partes iguais.

Caso médio de acordo c

C(n) ≈ 1,386n log n – 0,846n,

Análise

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

Pior caso:

- $C(n) = O(n^2)$

ordenação

Pior caso:

- $C(n) = O(n^2)$
- O pior i caso ocorre quando, sistematicamente, o pivô é escolhido como sendo um dos extremos de um arquivo já ordenado.
- Isto faz com que o procedimento Ordena seja chamado recursivamente n vezes, eliminando apenas um item em cada chamada.

Comparação

Método	Complexidade
Inserção	$O(n^2)$
Seleção	$O(n^2)$
Bolha	$O(n^2)$
Shellsort	$O(n\ lg(n)^2)$
Quicksort	$O(n\ lg(n))$

Comparação

- O método que levou menos tempo real para executar recebeu o valor 1 e os outros receberam valores relativos
- Elementos em ordem aleatória:

	5.00	5.000	10.000	30.000
Inserção	11,3	87	161	–
Seleção	16,2	124	228	–
Shellsort	1,2	1,6	1,7	2
Quicksort	1	1	1	1

Comparação

- Elementos em ordem crescente

	500	5.000	10.000	30.000
Inserção	1	1	1	1
Seleção	128	1.524	3.066	–
Shellsort	3,9	6,8	7,3	8,1
Quicksort	4,1	6,3	6,8	7,1

Comparação

- Elementos em ordem decrescente

	500	5.000	10.000	30.000
Inserção	40,3	305	575	–
Seleção	29,3	221	417	–
Shellsort	1,5	1,5	1,6	1,6
Quicksort	1	1	1	1

Ordenação por Inserção

- E o mais interessante para arquivos com menos do que 20 elementos
- O método é estável
- Possui comportamento melhor do que o método da bolha que também é estável
- Sua implementação é tão simples quanto as implementações do bolha e seleção
- Para arquivos já ordenados, o método é O(n)
- O custo é linear para adicionar alguns elementos a um arquivo já ordenado

Ordenação por Seleção

- É vantajoso quanto ao número de movimentos de registros, que é O(n)
- Deve ser usado para arquivos com elementos muito grandes, desde que o número de elementos a ordenar seja pequeno

QUICKSORT

- É o algoritmo mais eficiente que existe para uma grande variedade de situações
- O algoritmo é recursivo, o que demanda uma pequena quantidade de memória adicional
- Pior caso realiza $O(n^2)$ operações
- O principal cuidado a ser tomado é com relação à escolha do pivô