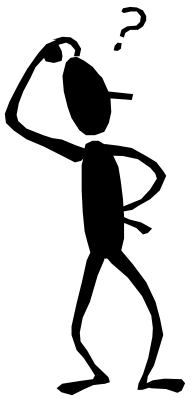
Métodos de Ordenação

Danielle B. Colturato

Introdução



- Como encontrar um nome na lista telefônica se não estiverem ordenados segundo um determinado critério?
- O mesmo acontece com dicionários, índices de livros, folhas de pagamento, listas de estudantes e outros materiais organizados alfabeticamente.
- Escolher um critério de ordenação
  - Ordem natural
    - Ex: números – ordem natural: ascendente ou descendente

Introdução - continuação

- Existem uma variedade de modos mas somente alguns podem ser considerados significativos e eficientes;
- Escolha do melhor método:
  - Estabelecer critérios de eficiência;
  - Selecionar um método para comparar quantitativamente diferentes algoritmos;

Introdução - continuação

- Comparação independente da máquina
  - Definir propriedades críticas dos algoritmos de ordenação:
    - Número de comparações;
    - Número de movimentos de dados.
  - A eficiência dessas duas operações depende do tamanho do conjunto de dados.

Eficiência

- Três casos:
  - Melhor caso: dados em ordem

    1 | 2 | 5 | 8 | 20
  - Pior caso: dados em ordem inversa

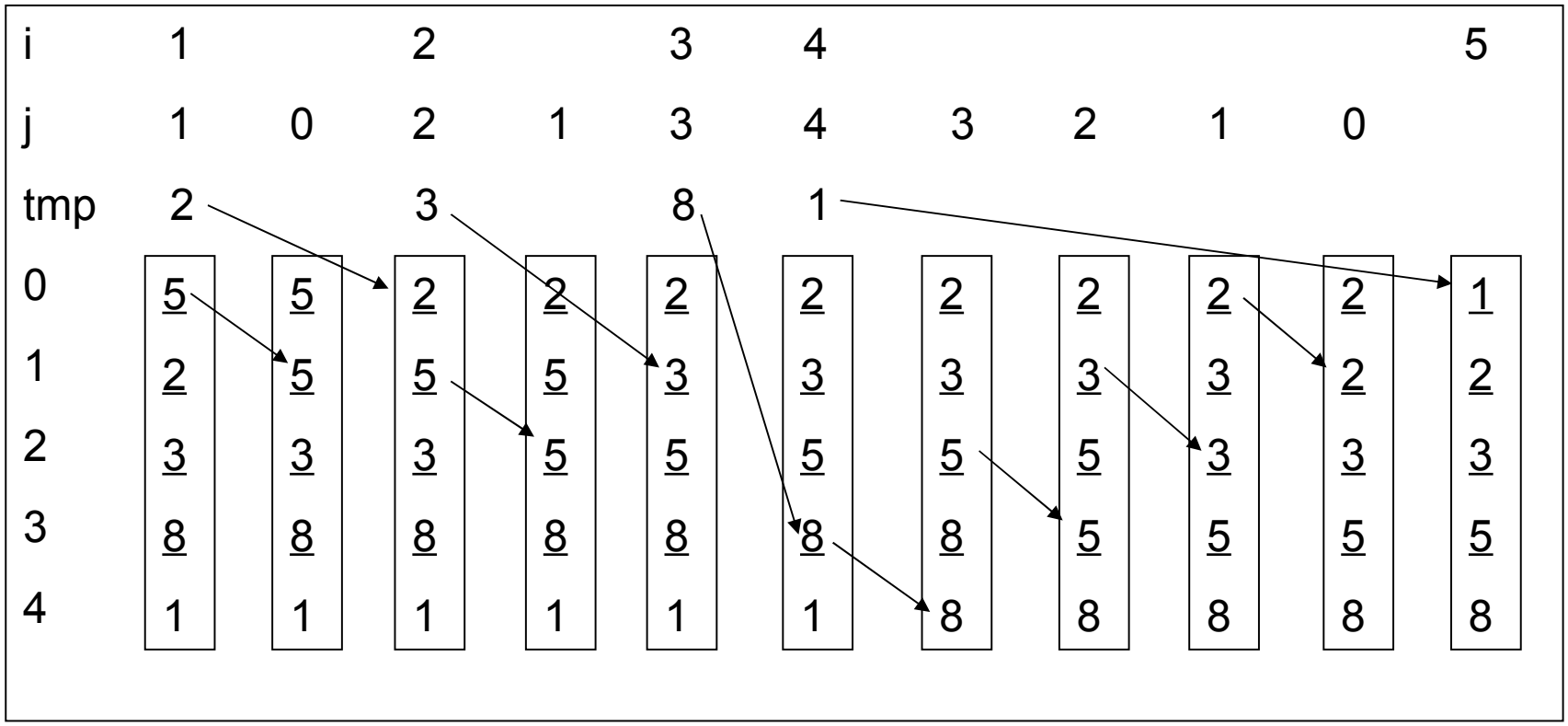
    20 | 8 | 5 | 2 | 1
  - Caso médio: dados em ordem aleatória

    5 | 8 | 1 | 2 | 20

Algoritmos de Ordenação Elementares

- Ordenação por Inserção
  - Inserção Direta
    - Simples, mais rápido entre os métodos básicos – BubleSort e Seleção Direta
    - Utilizado em pequenos conjuntos de dados
    - Consiste em ordenar um arquivo utilizando um sub-arquivo ordenado, localizado em seu início, e a cada novo passo, acrescentamos a este sub-arquivo mais um elemento na sua posição correta, até chegar ao último elemento do arquivo, gerando um arquivo final ordenado.

Inserção Direta



Vetor organizado por inserção direta

Inserção Direta

- Exemplo de algoritmo de ordenação por inserção direta

OBS: vetor iniciando do zero

```
para i = 1 até n faça
    temp = dado [i]
    para j = i até 0 e temp < dado [j-1]
        dado[j] = dado[j-1]
    dado[j] = temp
fim do para
```

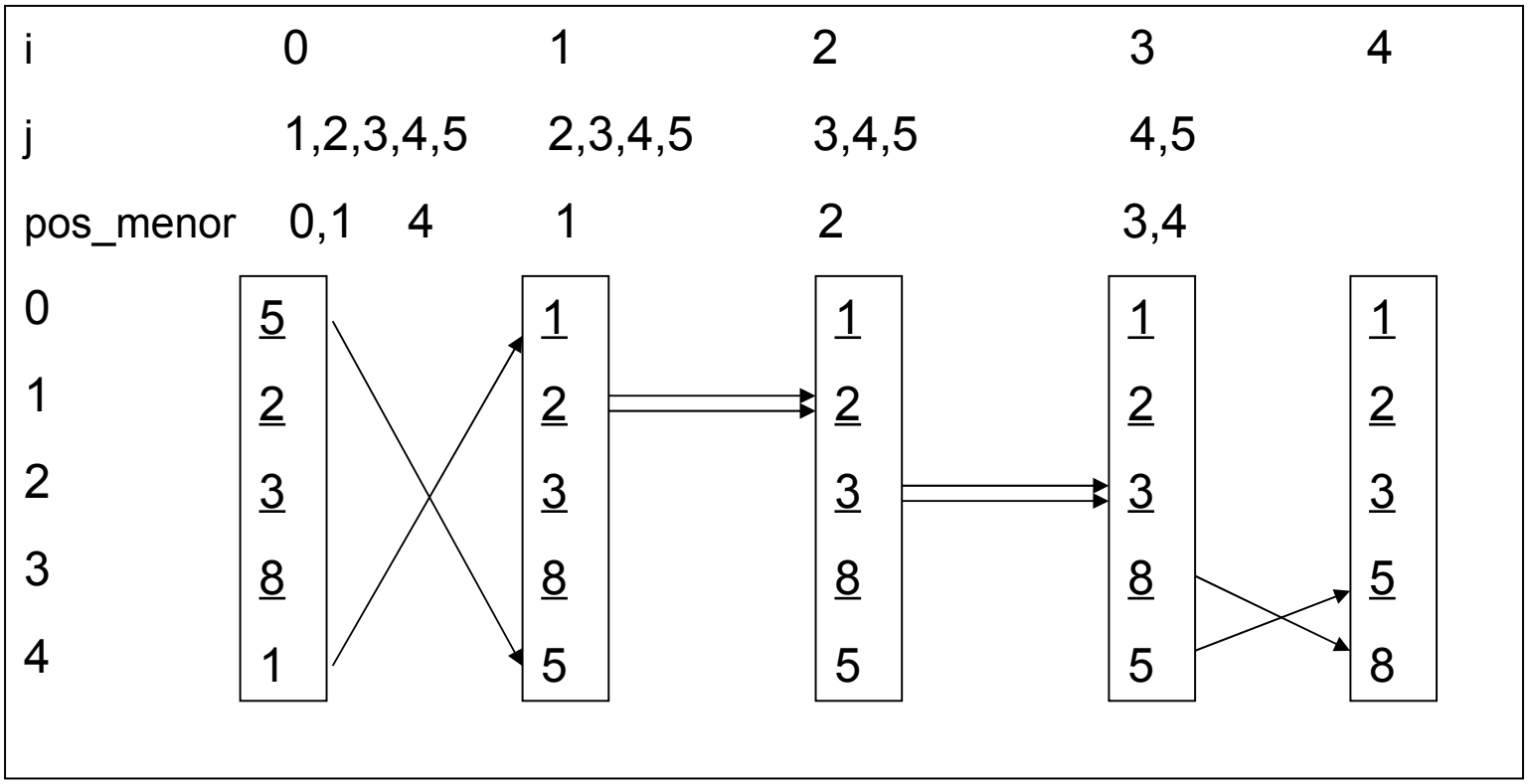
Inserção Direta

- Vantagem
  - Vetor ordenado ➡ não há movimento substancial, somente temp é inicializada e o devido valor armazenado
- Desvantagem
  - Não reconhece que elementos podem já estar em suas posições apropriadas
    - movimentação de suas posições e posterior retorno
  - Ao inserir um item todos os maiores que ele tem que ser movidos

Ordenação por seleção

- Elemento de menor valor é selecionado e trocado com o elemento da primeira posição;
- Atualiza-se o tamanho do segmento (menos um elemento);
- Repete o processo até que o segmento fique com apenas um elemento.

Ordenação por seleção



Vetor organizado por seleção

Ordenação por seleção

- Exemplo de algoritmo de ordenação por seleção

```
para i = 0 até n-1 faça
    pos_menor = i
    para j = i+1 até n faça
        se dado[j] < dado[pos_menor]
                então pos_menor = j
        fim do se
    fim do para
    temp = dado[i]
    dado[i] = dado[pos_menor]
    dado[pos_menor] = temp
fim do para
```

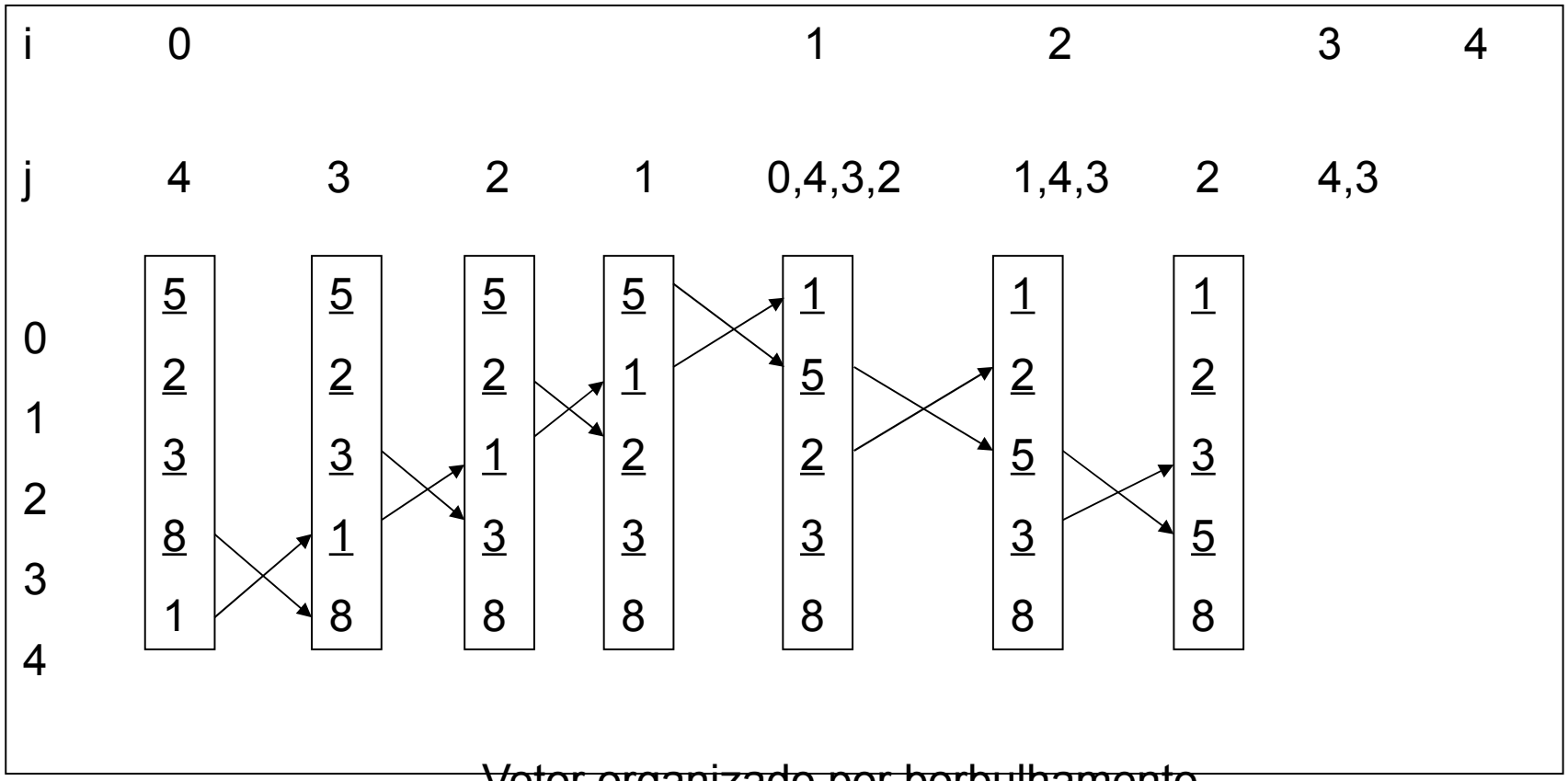
Ordenação por troca

- Durante o caminhamento do vetor, se dois elementos são encontrados fora de ordem, suas posições são trocadas;
- São realizadas comparações sucessivas de pares de elementos;
- A estratégia de escolha dos pares de elementos estabelece a diferença entre os dois métodos de ordenação por troca

Método da Bolha (Buble sort)

- Simples e lento
- A cada passo, cada elemento é comparado com o próximo;
- Se o elemento estiver fora de ordem,a troca é realizada;
- Realizam-se tantos passos quantos necessários até que não ocorram mais trocas.

Método da Bolha (Buble sort)



Vetor organizado por borbulhamento

Método da Bolha (Buble sort)

```
  public void bubbleSort(int[] arr) {
    boolean swapped = true;
    int j = 0;
    int tmp;
    while (swapped) {
        swapped = false;
        j++;
        for (int i = 0; i < arr.length - j; i++) {
            if (arr[i] > arr[i + 1]) {
                tmp = arr[i];
                arr[i] = arr[i + 1];
                arr[i + 1] = tmp;
                swapped = true;

                                }

        }
    }
}
```

Algoritmos de Ordenação Eficientes

- Método dos Incrementos Decrescentes (Shell sort)
  - Proposto por Ronald L. Shell (1959)
  - Difere do algoritmo de inserção direta, que considera apenas um segmento, por considerar vários segmentos;
  - Os segmentos são formados pegando-se os elementos que se encontram nas posições múltiplas de um determinado valor chamado incremento;

Método dos Incrementos Decrescentes (Shell sort)

- Ex: incremento igual a 4

1º segmento: vetor[0], vetor[4], vetor[8], ...
2º segmento: vetor[1], vetor[5], vetor[9], ...
3º segmento: vetor[2], vetor[6], vetor[10], ...
4º segmento: vetor[3], vetor[7], vetor[11], ...

Método dos Incrementos Decrescentes (Shell sort)

- A cada passo todos os elementos (segmentos) são ordenados isoladamente por inserção direta;
- No final de cada passo o processo é repetido para um novo incremento igual a metade do anterior, até que o incremento seja igual a 1;

Método dos Incrementos Decrescentes (Shell sort)

Método dos Incrementos Decrescentes (Shell sort)

- Exemplo de algoritmo de ordenação por *Shell Sort*

```
procedimento shellsort(int A,int N,int incrts,int nincrts)
    // incrts = vetor contendo os incrementos //

      int i, j, k, incr, temp

      para i=0 até i<nincrts faça
            incr = incrts[i]       // incr = tamanho do incremento //
            para j=incr até j<=N
                  temp = A[j]
                  para k=j-incr até k >=0 e temp < A[k] faça
                        k = k - incr
                        A[k+incr] = A[k]
                  fim do para
            A[k+incr] = temp
            fim do para
       fim do para
```

Quicksort (Ordenação Rápida)

- É o mais rápido entre os métodos apresentados até o momento, e também o mais utilizado;
- Idéia: “dividir para conquistar”
- Proposto por C. A. R. Hoare em 1962;
- Parte do princípio que é mais rápido classificar dois vetores com *n*/2 elementos cada um, do que um com *n* elementos (dividir um problema maior em dois menores).

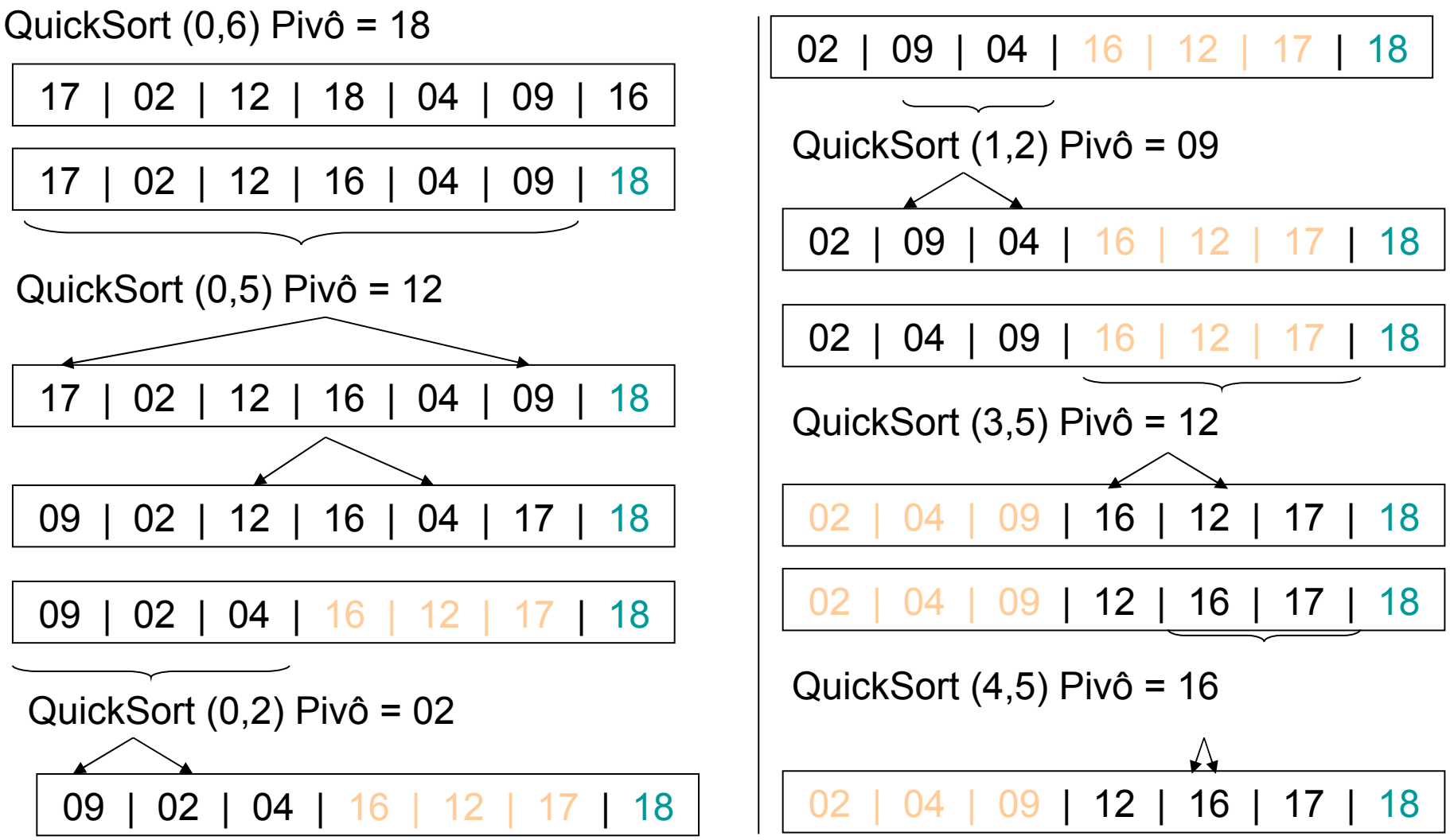
Quicksort (Ordenação Rápida)

- A partição é realizada através da escolha arbitrária de um elemento (V[i]) chamado pivô;
- Os elementos no primeiro segmento serão menores, e os elementos no segundo segmento serão maiores do que o pivô;

Quicksort (Ordenação Rápida)

- Escolher arbitrariamente um elemento do vetor (escolher uma estratégia - normalmente o meio) e colocá-lo em uma variável auxiliar X;
- Inicializar dois ponteiros I e J (I = 1 e J = n);
- Percorrer o vetor a partir da esquerda até que se encontre um V[I] >= X (incrementando o valor de I);
- Percorrer o vetor a partir da direita até que se encontre um V[J] <= X (decrementando o valor de J);
- Trocar os elementos V[I] e V[J] (estão fora de lugar) e fazer: I = I + 1 e J = J - 1;
- Continuar esse processo até que I e J se cruzem em algum ponto do vetor;
- Após obtidos os dois segmentos do vetor através do processo de partição, cada um é ordenado recursivamente.

Quicksort (Ordenação Rápida)



Quicksort (Ordenação Rápida)

- Exemplo de algoritmo de ordenação por *QuickSort*

```
Procedimento QuickSort (int A, int esq, int dir)
Início

 int x,y;

      i = esq
      j = dir
      x = A[(esq+dir)/2]        // Elemento intermediário como “pivot” //

      faça {
         enquanto (A[i] < x e i < dir) i = i +1
         enquanto (A[j] > x e j > esq) j= j -1

         se (i<=j) então
            troca (A,i,j);
            i = i +1
            j = j -1
          fim do se
      } enquanto (i <= j)

      se (esq < j) QuickSort(A, esq, j)
      se (i<dir) QuickSort(A, i, dir)

 fim
```

Comparação entre os métodos

- tempo gasto na ordenação de vetores com 500, 5000, 10000 e 30000 elementos, organizados de forma aleatória, em ordem crescente (1, 2, 3, 4, ..., n) e em ordem decrescente (n, n-1, n - 2, ..., 1);
- Em cada tabela, o método que levou menos tempo para realizar a ordenação recebeu o valor 1 e os demais receberam valores relativos ao mais rápido.

Comparação entre os métodos

	500	5000	10000	30000
Inserção	11.3	87	161	-
Shell	1.2	1.6	1.7	2
Quick	1	1	1	1
Seleção	16.2	124	228	-
Heap	1.5	1.6	1.6	1.6

Ordem Aleatória

	500	5000	10000	30000
Inserção	1	1	1	1
Shell	3.9	6.8	7.3	8.1
Quick	4.1	6.3	6.8	7.1
Seleção	128	1524	3066	-
Heap	12.2	20.8	22.4	24.6

Ordem Ascendente

Comparação entre os métodos

	500	5000	10000	30000
Inserção	40.3	305	575	-
Shell	1.5	1.5	1.6	1.6
Quick	1	1	1	1
Seleção	29.3	221	417	-
Heap	2.5	2.7	2.7	2.9

Ordem Descendente

Comparação entre os métodos

- **Tamanho <= 50 Inserção**
- **Tamanho <= 5000 Shell Sort**

  Até 1000 elementos o Shell é mais vantajoso
- **Tamanho > 5000 Quick Sort**

  Necessita de memória adicional por ser recursivo. Evitar chamadas recursivas para pequenos intervalos. Colocar um teste antes da recursividade (se n <= 50 inserção; se n <= 1000, shell sort)

Comparação entre os métodos

Método	Complexidade
Buble Sort	$O(n^2)$
Insertion Sort	$O(n^2)$
Selection Sort	$O(n^2)$
Shell Sort	$O(n^{1.25})$
Quick Sort	$O(n \log n)$