Organização e Recuperação da Informação

Métodos de ordenação

DC – UFSCar
Jander Moreira

Conteúdo

- Agenda
  - Introdução
  - Métodos clássicos para ordenação
    - Métodos básicos
      - Bubble sort
      - Inserção
      - Seleção
    - Métodos eficientes
      - Shell sort
      - Heap sort

Introdução

- Ordenação em memória principal
  - Considerações
    - Complexidade de tempo
      - Desempenho dos algoritmos em função do número de itens
    - Complexidade de espaço
      - Uso de espaço adicional de memória em função do número de itens

Métodos básicos

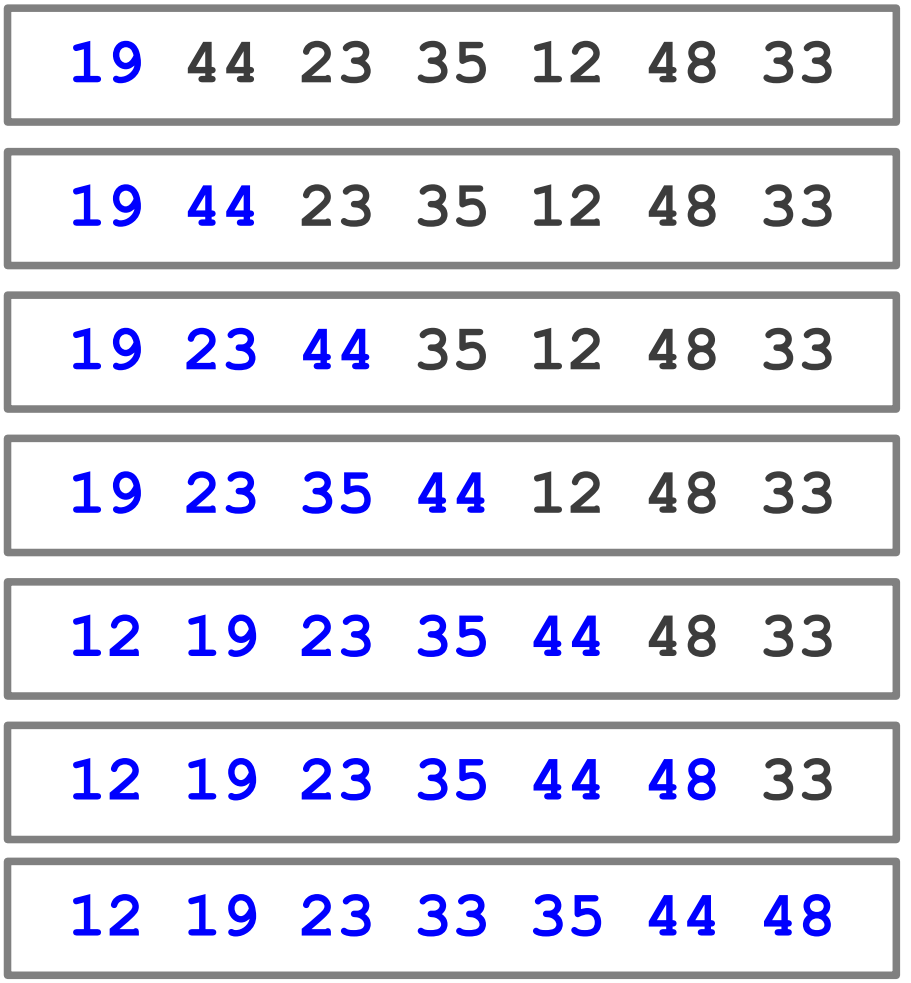
- Métodos
  - Inserção / inserção direta / insertion
  - Seleção / seleção direta / selection
  - Bubblesort / borbulhamento

Métodos básicos

- Inserção
  - Conceito
    - Divisão do arranjo em duas partes
      - Parte ordenada inicial: primeira posição do arranjo
      - Parte a ordenar: todas as demais posições
    - Ordenação
      - Transferência do primeiro item da parte a ordenar para a parte ordenada, fazendo uma inserção ordenada, até terminar

Métodos básicos

- Inserção



Métodos básicos

- Inserção
  - Algoritmo

```
inserção_direta(dados[]: inteiro; tamanho: inteiro)
   para i ← 1 até tamanho – 1 faça
      { seleciona primeiro da parte a ordenar }
      auxiliar ← dados[i]

      { abre espaço deslocando os elementos maiores que dados[i] }
      j ← i – 1
      enquanto dados[j] > auxiliar e j ≥ 0 faça
         dados[j + 1] ← dados[j]  { desloca uma posição }
         j ← j – 1
      fim-enquanto

      { coloca na posição correta }
      dados[j + 1] ← auxiliar
   fim-para
```

Métodos básicos

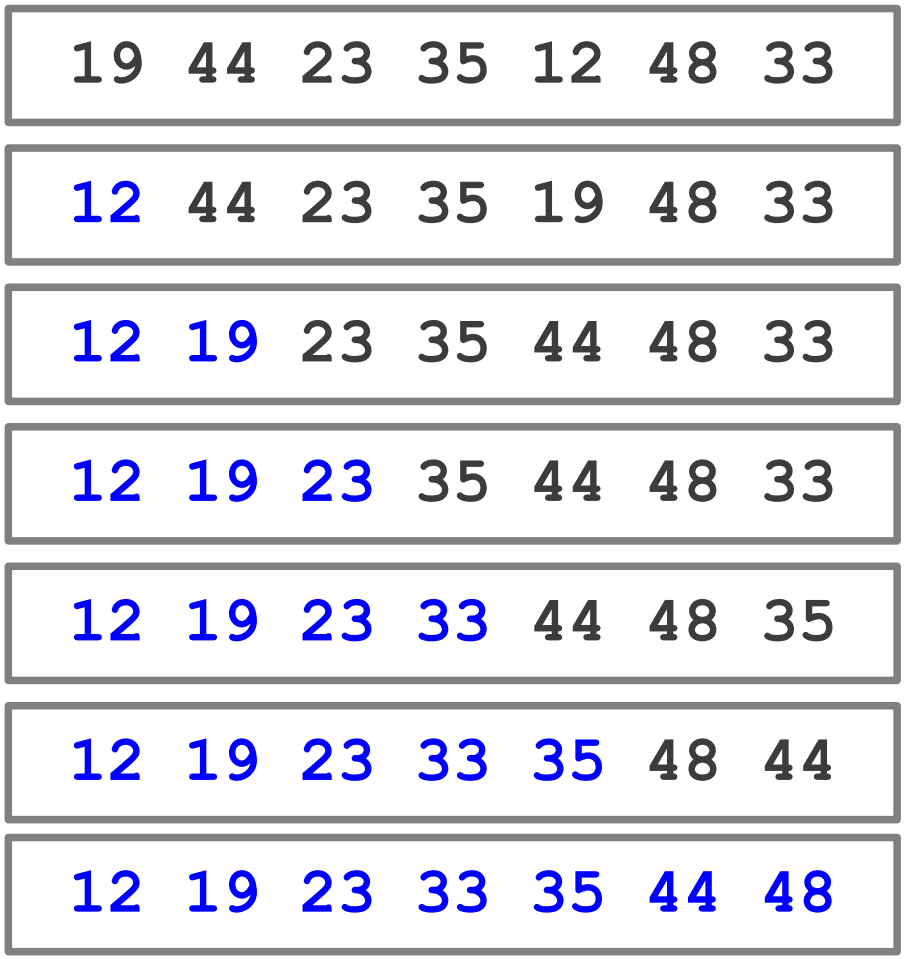
- Inserção
  - Desempenho
    - Melhor caso
      - Arranjo próximo da ordenação
      - O(n)
    - Pior caso
      - Arranjo próximo à ordenação inversa
      - $O(n^2)$
    - Caso médio
      - Arranjo aleatoriamente distribuído
      - $O(n^2)$

Métodos básicos

- Seleção
  - Conceito
    - Divisão do arranjo em duas partes
      - Parte ordenada inicial: vazia
      - Parte a ordenar: todos os itens do arranjo
    - Ordenação
      - Seleção do menor item da parte a ordenar para transferi-lo para a parte ordenada, até restar apenas um item

Métodos básicos

- Seleção



Métodos básicos

- Seleção
  - Algoritmo

```
seleção_direta(dados[]: inteiro; tamanho: inteiro)
   para i ← 0 até tamanho – 2 faça
      { localiza o menor elemento da partição }
      menor ← dados[i]
      posição ← i
      para j ← i + 1 até tamanho – 1 faça
         se dados[j] < menor então
            menor ← dados[j]
            posição ← j   { armazena a posição }
         fim-se
      fim-para

      { coloca o menor elemento no início da partição }
      dados[posição] ← dados [i]
      dados[i] ← menor
   fim-para
```

Métodos básicos

- Seleção
  - Desempenho
    - $O(n^2)$

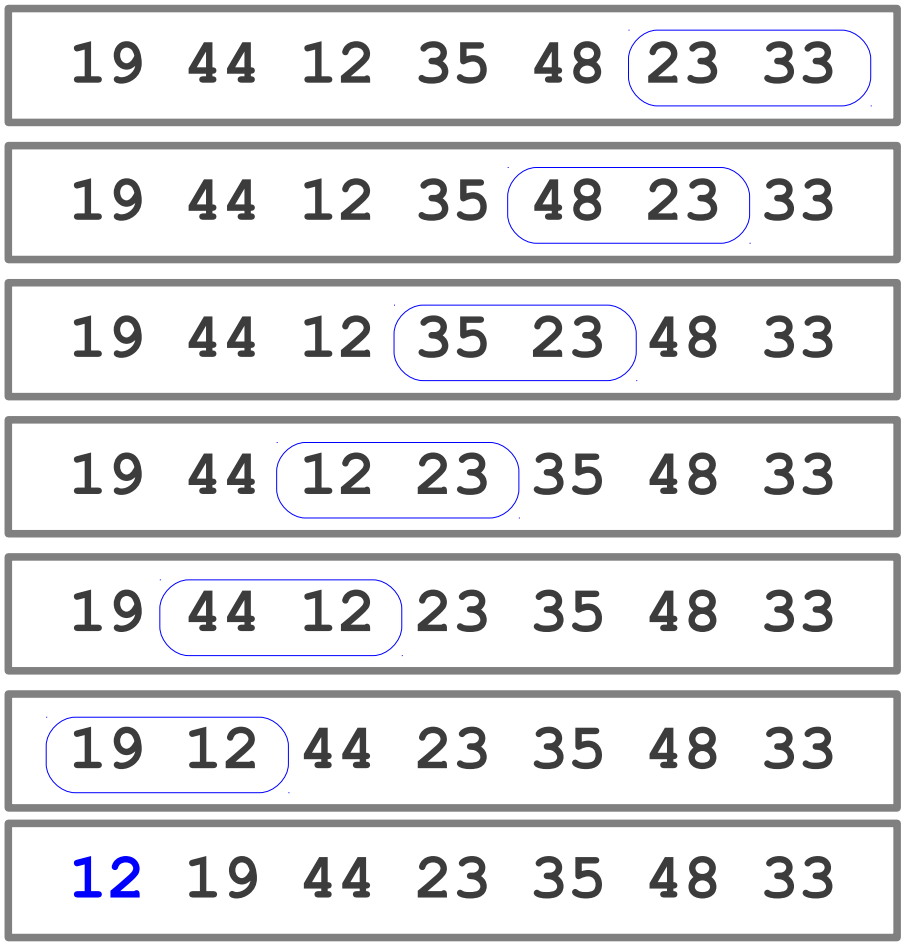
Métodos básicos

- Bubblesort
  - Conceito
    - Divisão do arranjo em duas partes
      - Parte ordenada inicial: vazia
      - Parte a ordenar: todos os itens do arranjo
    - Ordenação
      - Por verificações de ordenação local, trocar vizinhos fora de ordem, "arrastando" os menores para o início do arranjo, aumentando a parte ordenada item a item

Métodos básicos

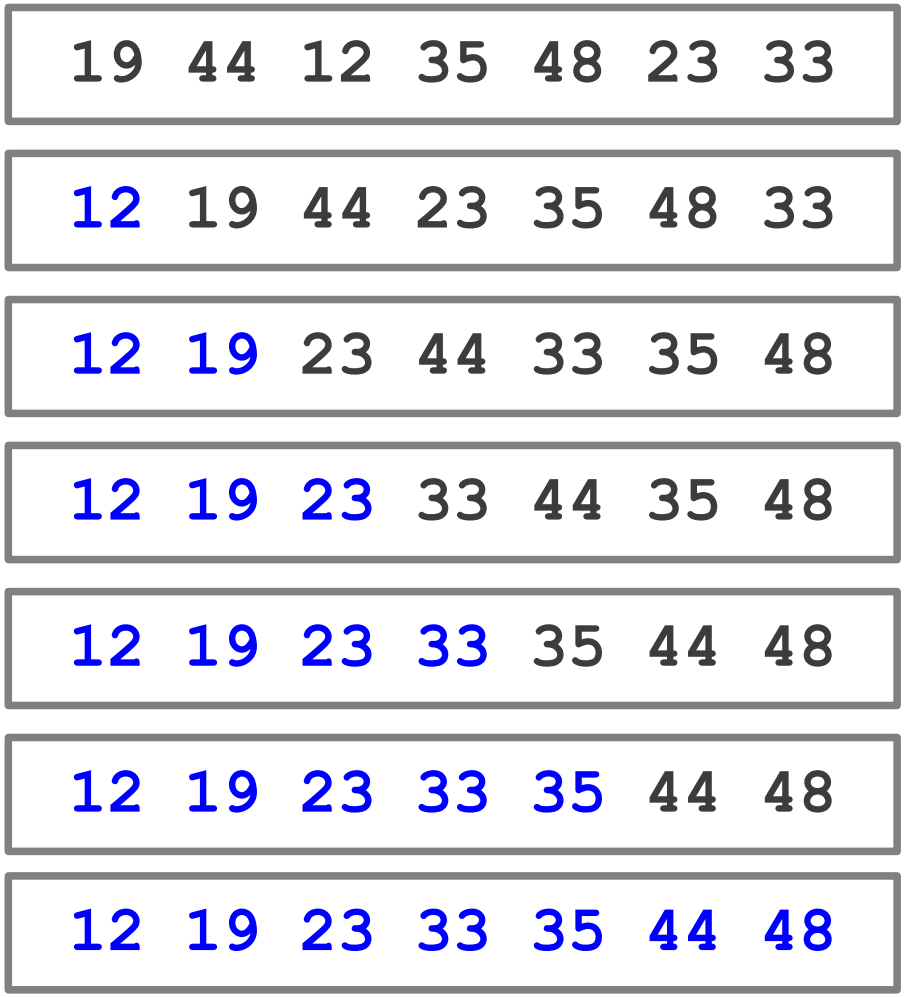
- Bubblesort

  Passo único



Métodos básicos

- Bubblesort



Métodos básicos

- Seleção
  - Algoritmo

```
bubblesort(dados[]: inteiro; tamanho: inteiro)
   para i ← 0 até tamanho – 2 faça
      { varredura }
      j ← tamanho – 1
      enquanto j > i faça
         se dados[j] < dados[j - 1] então
            { troca adjacentes }
            auxiliar ← dados[j]
            dados[j] ← dados[j – 1]
            dados[j – 1] ← auxiliar ]
         fim-se
         j ← j – 1
      fim-enquanto
   fim-para
```

Métodos básicos

- Bubblesort
  - Desempenho
    - $O(n^2)$
  - Variações
    - Shakersort
    - Término antecipado

Métodos de ordenação

- Estabilidade
  - Para itens com chave igual, a ordem relativa original é mantida após a ordenção
    - Métodos básicos estáveis:
      - Inserção e bubblesort

Métodos eficientes

- Métodos
  - Shellsort
  - Heapsort / método do monte
  - Quicksort

Métodos eficientes

- Shellsort
  - Conceito
    - Divisão do arranjo em subsequências
      - Cada subsequência, espaçada de *k* posições, é ordenada independentemente, usando algoritmo de inserção
    - Ordenação
      - A cada passo,as subsequências são definidas, usando espaçamento menor a cada passo
      - No último passo é usado espaçamento 1 (ou seja, inserção)

Métodos eficientes

- Shellsort

	19 44 23 35 12 48 33 14 25 28 49 11 12 31 23
k = 7	19 44 23 35 12 48 33 14 25 28 49 11 12 31 23
	14 25 23 35 11 12 31 19 44 28 49 12 48 33 23
k = 3	14 25 23 35 11 12 31 19 44 28 49 12 48 33 23
	14 11 12 28 19 12 31 25 23 35 33 23 48 49 44
k = 1	14 11 12 28 19 12 31 25 23 35 33 23 48 49 44
	11 12 12 14 19 23 23 25 28 31 33 35 44 48 49

Métodos eficientes

- Shellsort
  - Algo-ritmo

```
shellsort(dados[]: inteiro, tamanho:inteiro)
    { ordenação das subsequências para cada incremento }
    incremento ← primeiro_incremento
    enquanto incremento > 0 faça
        { ordena cada subsequência }
        para j ← incremento até tamanho – 1 faça
            auxiliar ← dados[j]

            { deslocamentos dentro da mesma subsequência }
            k ← j – incremento
            enquanto dados[k] > auxiliar e k ≥ 0 faça
                dados[k + incremento] ← dados[k]
                k ← k - incremento
            fim-enquanto
            dados[k + incremento] ← auxiliar
        fim-para

        { passa para o próximo incremento }
        incremento ← próximo_incremento
    fim-enquanto
```

Métodos eficientes

- Shellsort
  - Desempenho
    - Cada passo aproveita a ordenação dos passos anteriores
    - A mistura entre as subsequencias de passos consecutivos auxilia na ordenação
    - Desempenho depende dos incrementos
      - $O(n \log n)$
      - $O(n \log^2 n)$
      - $O(n^{5/3})$

Métodos eficientes

- Shellsort
  - Escolha de incrementos (Knuth, 1970)
    - Para *t* passos:
      - Passos

        $h_t = 1$

        $h_{t-1} = 3\ h_t + 1$

      - Número de passos

        $t = \log_3 n$

      - Para 7 passos

        1093, 364, 121, 40, 13, 4 e 1

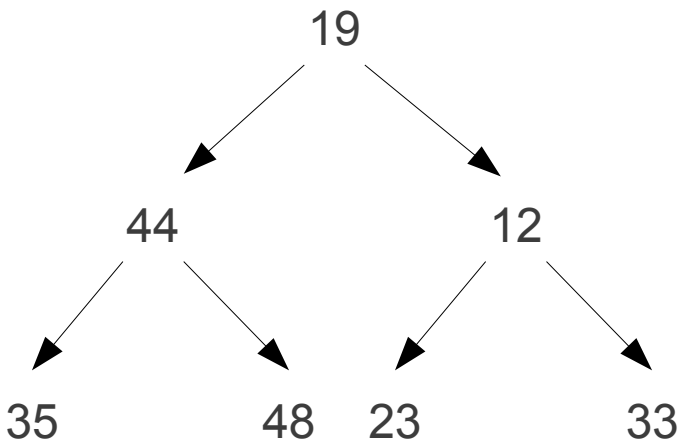
Métodos eficientes

- Heapsort
  - Conceito
    - Princípio de ordenação por seleção
      - Organizando o arranjo como uma fila de prioridade na forma de um heap (monte)
      - Divisão do arranjo em parte ordenada e a ordenar
    - Ordenação
      - Usar a fila de prioridade para selecionar o maior item da parte a ordenar e colocá-lo na parte ordendada

Métodos eficientes

- Heapsort

19	44	12	35	48	23	33



pai:	(filho – 1)/2
filho esquerdo:	2 * pai + 1
filho direito:	2 * pai + 2

Métodos eficientes

- Heapsort
  - Criação inicial do monte

```
{ criação do monte }
para i ← 1 até tamanho – 1 faça
    auxiliar ← dados[i]
    filho ← i
    pai ← (filho – 1)/2  { posição do pai }
    enquanto filho > 0 e dados[pai] < auxiliar faça
        dados[filho] ← dados[pai]
        filho ← pai
        pai ← (filho – 1)/2
    fim-enquanto
    dados[pai] ← auxiliar
fim-para
```

Métodos eficientes

- Heap sort
  - Orde-nação

```
{ ordenação }
i ← tamanho – 1
enquanto i > 1 faça
    { coloca o valor da raiz na posição definitiva }
    auxiliar ← dados[0] { raiz }
    dados[i] ← dados[0]
    dados[0] ← auxiliar

    { corrige o monte }
    auxiliar ← dados[0]  { valor incorreto, colocado na raiz }
    pai ← 0
    se dados[2*pai + 1] > dados[2*pai + 2] e 2*pai + 2 ≥ i então
        filho ← 2*pai + 1
    senão
        filho ← 2*pai + 2
    fim-se
    enquanto pai < i e dados[filho] < dados[pai] faça
        dados[pai] ← dados[filho]
        pai ← filho
        se dados[2*pai + 1] > dados[2*pai + 2] e 2*pai + 2 ≥ i então
            filho ← 2*pai + 1
        senão
            filho ← 2*pai + 2
        fim-se
    fim-enquanto
    dados[filho] ← auxiliar
fim-enquanto
```

Métodos eficientes

- Heapsort
  - Desempenho
    - O(n log n)

Métodos eficientes

- Quicksort
  - Conceito
    - Abordagem "divisão e conquista"
      - Trocas de itens dentro de uma partição, criando a partir dela duas partições independentes
    - Ordenação
      - Considerar o arranjo todo como uma partição
      - Aplicar o procedimento de partição a cada partição existente que contenha mais que um elemento, criando a partir de cada uma, duas novas partições

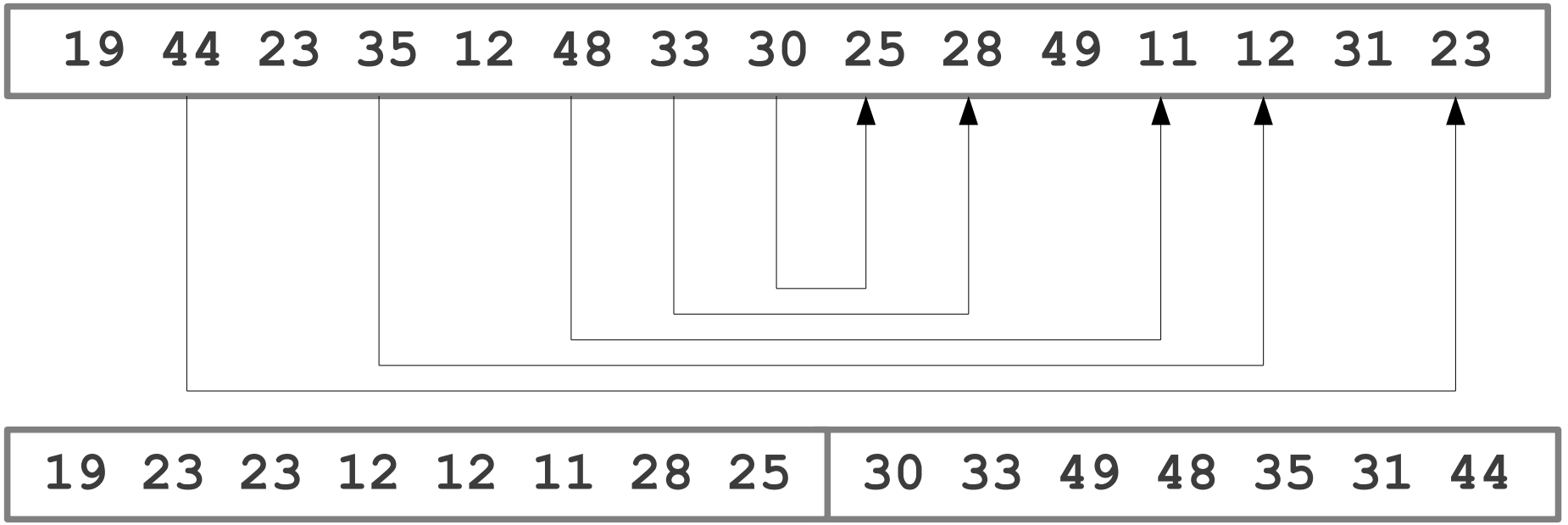
Métodos eficientes

```
partição(dados[]: inteiro, início, fim: inteiro, var esquerda, direita: inteiro)
{ particionamento dos dados entre as posições início e fim do arranjo }
    esquerda ← início
    direita ← fim
    pivô ← dados[(início + fim)/2] { elemento do meio }
    enquanto esquerda ≤ direita faça
        { procura um valor maior ou igual ao pivô }
        enquanto dados[esquerda] < pivô faça
            esquerda ← esquerda + 1
        fim-enquanto

        { procura um valor menor ou igual ao pivô }
        enquanto dados[direita] > pivô faça
            direita ← direita – 1
        fim-enquanto

        { faz a troca }
        se esquerda ≤ direita então
            auxiliar ← dados[esquerda]
            dados[esquerda] ← dados[direita]
            dados[direita] ← auxiliar
            direita ← direita – 1
            esquerda ← esquerda + 1
        fim-se
    fim-enquanto
```

Métodos eficientes

- Quicksort

pivô = 30



Métodos eficientes

```
quicksort(dados[]: inteiro, tamanho: inteiro)
   quickRecursivo(dados, 0, tamanho – 1)
```

Quicksort
Algoritmo

```
quickRecursivo(dados[]: inteiro, início, fim: inteiro)
   partição(dados, início, fim, esquerda, direita)

   { ordena primeira partição }
   se início < direita então
      quickRecursivo(dados, início, direita)
   fim-se

   { ordena segunda partição }
   se esquerda < fim então
      quickRecursivo(dados, esquerda, fim)
   fim-se
```

Quicksort
Recursão

Métodos eficientes

- Quicksort
  - Desempenho
    - Melhor caso
      - As partições criadas possuem o mesmo tamanho
      - O(n log n)
    - Pior caso
      - As partições criadas possuem tamanhos muito diferentes
      - $O(n^2)$
    - Caso médio
      - A maioria das partições criadas possuem tamanho equivalente
      - O(n log n)

Leitura

- Leituras
  - Moreira, J. Ordenação (apostila)
  - Drozdek, Capítulo 9, seções 9.1 e 9.3