Aula 2

Estrutura de Dados

Prof. Vinicius Pozzobon Borin

Conversa Inicial

- O objetivo desta aula é apresentar os conceitos que envolvem algoritmos de buscas e de ordenação de estruturas de dados
- Diferentes algoritmos serão apresentados
- Cada algoritmo apresentará um raciocínio lógico diferente e poderá ter um desempenho diferente, pois sua complexidade assintótica será diferente

- Algoritmos de ordenação
  - Ordenação por troca (*bubble sort*)
  - Ordenação por intercalação (*merge sort*)
  - Ordenação rápida (*quick sort*)
- Algoritmos de busca
  - Busca sequencial
  - Busca binária

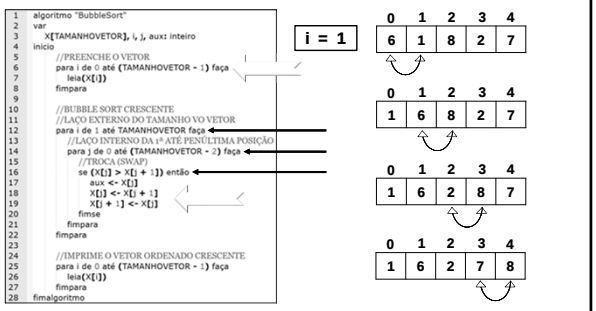
Algoritmos de ordenação

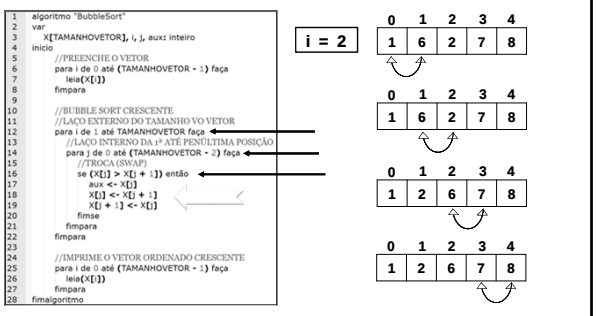
- Por que ordenar dados? Algumas aplicações de ordenação estão listadas a seguir
  - Visualizar dados listados, todos os nomes de pessoas em uma agenda de e-mails por ordem alfabética
  - Programas de renderização gráfica utilizam algoritmos de ordenação para desenhar objetos gráficos em uma ordem predefinida

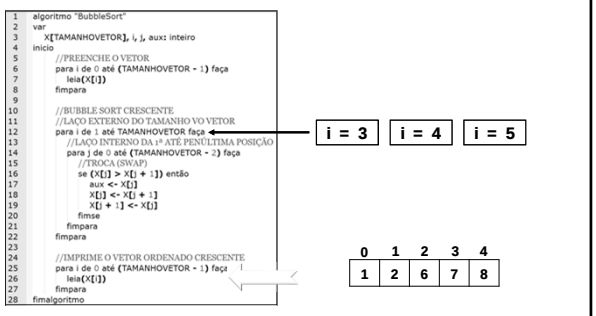
- **Um algoritmo de ordenação é um método que descreve como os dados serão ordenados. Esses algoritmos são independentes dos tipos de dados, do volume de dados e da linguagem de programação**

- **As ordenações serão realizadas em estruturas de dados homogêneas, numéricas e unidimensionais (vetores) ao longo desta aula**
- **De forma análoga, seria possível implementar o mesmo algoritmo para quaisquer outros tipos de dados, como caracteres alfanuméricos ou para estruturas heterogêneas e/ou bidimensionais**

Algoritmo de ordenação por troca (*bubble sort*)







Bubble sort crescente x decrescente

```
10    //BUBBLE SORT CRESCENTE
11    //LAÇO EXTERNO DO TAMANHO VO VETOR
12    para i de 1 até TAMANHOVETOR faça
13      //LAÇO INTERNO DA 1ª ATÉ PENÚLTIMA POSIÇÃO
14      para j de 0 até (TAMANHOVETOR - 2) faça
15        //TROCA (SWAP)
16        se (X[j] > X[j + 1]) então
17          aux <- X[j]
18          X[j] <- X[j + 1]
19          X[j + 1] <- X[j]
20        fimse
21      fimpara
22    fimpara
```

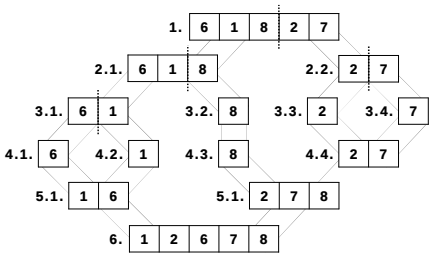
```
10    //BUBBLE SORT DECRESCENTE
11    //LAÇO EXTERNO DO TAMANHO VO VETOR
12    para i de 1 até TAMANHOVETOR faça
13      //LAÇO INTERNO DA 1ª ATÉ PENÚLTIMA POSIÇÃO
14      para j de 0 até (TAMANHOVETOR - 2) faça
15        //TROCA (SWAP)
16        se (X[j] < X[j + 1]) então
17          aux <- X[j]
18          X[j] <- X[j + 1]
19          X[j + 1] <- X[j]
20        fimse
21      fimpara
22    fimpara
```

Complexidade assintótica para o pior caso:

$\mathbf{0}_{BubbleSort}(n^2)$

Algoritmo de ordenação por intercalação (*merge sort*)

Merge sort: funcionamento



Merge sort: algoritmo

```
18  função merge(X, inicio, fim)
19  var
20      meio: inteiro
21  início
22          se (inicio < fim) então
23              meio <- parteinteira((inicio + fim) / 2) //DIVIDE AO MEIO
24              merge(X, inicio, meio) //CONTINUA DIVIDINDO A PARTIR DA 1ª METADE
25              merge(X, meio + 1, fim) //CONTINUA DIVIDINDO A PARTIR DA 2ª METADE
26              intercala(X, inicio, fim, meio) //AGREGA DE VOLTA 2 VETORES ORDENADOS
27          fimse
28  fimfunção
```

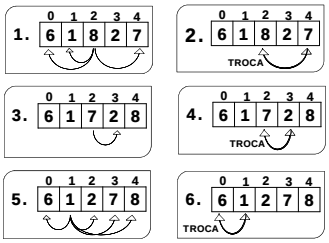
Merge sort: algoritmo

- Complexidade da função *merge*
  - $O(log(n))$
- Complexidade da função intercala
  - $O(n)$
- Complexidade resultante
  - $\mathbf{0}_{MergeSort}(n.\log(n))$

Algoritmo de ordenação rápida (*quick sort*)

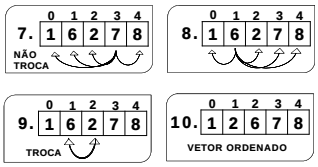
Quick sort: funcionamento

Pivô Troca Compara à esquerda Compara à direita



Quick sort: funcionamento

Pivô Troca Compara à esquerda Compara à direita



Quick sort: algoritmo

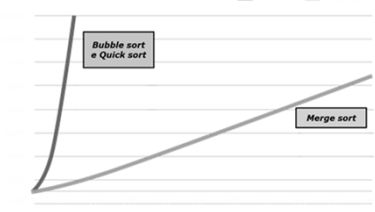
```
19  função quicksort(X, inicio, fim)
20  var
21      div: inteiro
22  inicio
23      se (inicio < fim) entao
24          div = particao(X, inicio, fim) //RETORNA O PONTO DE PARADA
25          quicksort(X, inicio, div) //CONTINUA DIVIDINDO A PARTIR DA 1ª PARTE
26          quicksort(X, div+1, fim) //CONTINUA DIVIDINDO A PARTIR DA 2ª PARTE
27      fimse
28  fimfunção
```

Quick sort: algoritmo

```
30  função particao(X, inicio, fim)
31  var
32      posicao_pivo, pivo, i, j: inteiro
33  inicio
34      posicao_pivo = parteinteira((inicio + fim)/2);
35      pivo = X[posicao_pivo] //ENCONTRA O PIVÔ
36      i = inicio - 1
37      j = fim + 1
38      enquanto (i < j) faça
39          repita //PROCURA POR VALOR INCOERENTE À DIREITA DO PIVÔ
40              j = j - 1
41          até (X[j] <= pivo)
42          repita //PROCURA POR VALOR INCOERENTE À ESQUERDA DO PIVÔ
43              i = i + 1
44          até (X[i] >= pivo)
45          //TROCA UM VALOR INCOERENTE À ESQUERDA
46          //COM UM VALOR À DIREITA DO PIVÔ
47          se (i < j) então
48              troca(X, i, j);
49          fimse
50      fimenquanto
51      //RETORNA ONDE O LADO DIREITO PAROU
52      //ESTE VAOR SERÁ USADO COMO NOVO MEIO
53      retorne j
54  fimfunção
```

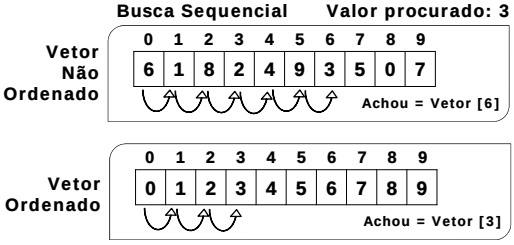
Complexidade: O(n²)

Desempenho comparativo dos algoritmos de ordenação



Algoritmos de busca

Busca sequencial: funcionamento



Busca sequencial: algoritmo

```
1   algoritmo "BuscaSequencial_NaoOrdenado"
2   var
3       X[TAMANHOVETOR], i, achou, buscado: inteiro
4   inicio
5       //PREENCHE O VETOR COM DADOS ALEATORIOS
6       para i de 0 até (TAMANHOVETOR - 1) faça
7           leia(X[i])
8       fimpara
9       //LÊ VALOR A SER BUSCADO
10      leia(buscado)
11
12      //BUSCA SEQUENCIAL
13      achou = 0
14      i = 0
15      enquanto ((i <= TAMANHOVETOR) E (achou == 0))
16          se (X[i] == buscado) então
17              achou = 1
18          senão
19              i = i + 1
20          fimse
21      fimenquanto
22      se (achou == 0) então
23          escreva("Valor não encontrado.")
24      senão
25          escreva("Valor encontrado na posição.", i + 1)
26      fimse
27  fimalgoritmo
```

Busca binária: funcionamento

Busca binária: algoritmo

```
14      //BUSCA BINARIA
15      achou = 0
16      inicio = 0
17      fim = TAMANHOVETOR
18      meio = parteinteira((inicio+fim) / 2)
19      enquanto ((inicio <= fim) E (achou == 0))
20          se (X[meio] == buscado) então
21              achou = 1
22          senão
23              se (meio < X[meio]) então
24                  fim = meio - 1
25              senão
26                  inicio = meio + 1
27                  meio = parteinteira((inicio+fim) / 2)
28              fimse
29          fimse
30      fimenquanto
31      se (achou == 0) então
32          escreva("Valor não encontrado.")
33      senão
34          escreva("Valor encontrado na posição.", meio)
35      fimse
36  fimalgoritmo
```

Desempenho comparativo dos algoritmos de busca

- **Complexidade da busca sequencial**
  - $O(log(n))$
- **Complexidade da busca binária**
  - $O(n)$

Referências

- ASCENCIO, A. F. G. Estrutura de dados: algoritmos, análise da complexidade e implementações em Java e C/C++. São Paulo: Pearson, 2011.
- ASCENCIO, A. F. G. Fundamentos da programação de computadores: Algoritmos, Pascal, C/C++ (padrão ANSI) JAVA. 3. ed. São Paulo: Pearson, 2012.
- CORMEN, T. H. Algoritmos: teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012.
- PUGA, S.; RISSETI, G. Lógica de programação e estrutura de dados. 3. ed. São Paulo: Pearson, 2016.
- MIZRAHI, V. V. Treinamento em linguagem C. 2. ed. São Paulo: Pearson, 2008.
- LAUREANO, M. Estrutura de dados com algoritmos e C. São Paulo: Brasport, 2008.