Universidade Federal de Mato Grosso do Sul - UFMS

Lucas Gonzalez de Queiroz

Weber Veloso Mourão

Campo Grande , 2014

Lucas Gonzalez de Queiroz

Weber Veloso Mourão

Relatório técnico apresentado como requisito
parcial para obtenção de aprovação na
disciplina Algoritmos e Programação II,
do Curso de Ciência da Computação,
na Universidade Federal de Mato Grosso do Sul.

Prof. Me. Liana Duenha

Campo Grande , 2014

RESUMO

Este relatório experimental tem o objetivo de apresentar a maneira como que diversos algoritmos de ordenação , sendo eles recursivos ou iterativos , comportam-se diante de uma grande quantidade de entradas realizadas por um gerador de entradas , diversas comparações entre eles.

Palavras-Chave : relatório experimental,algoritmos, ordenação, comparações;

SUMÁRIO

INTRODUÇÃO

Imagine que temos um problema computacional onde você precisa ordenar uma quantidade de elementos , utilizando algoritmos de ordenação.Qual deles é o mais lento ? Qual é o mais rápido ? Qual utilizar para um quantidade determinada de entradas?. Estas perguntas serão respondidas ao longo deste relatório.

Neste relatório,apresentamos os conceitos de algoritmos,algoritmos para ordenação, realizamos testes com alguns dos algoritmos ,geramos diversas entradas, e, a partir delas, realizamos comparações utilizando gráficos, tabelas, entre outros. Por fim, determinamos a eficiência de cada um dos algoritmos testados.

DESENVOLVIMENTO

ALGORITMOS

Um algoritmo nada mais é do que uma receita que mostra passo a passo os procedimentos necessários para a resolução de uma tarefa. Ele não responde a pergunta “o que fazer?”, mas sim “como fazer”. Em termos mais técnicos, um algoritmo é uma sequência lógica, finita e definida de instruções que devem ser seguidas para resolver um problema ou executar uma tarefa.

Os algoritmos são muito utilizados na área de programação, descrevendo as etapas que precisam ser efetuadas para que um programa execute as tarefas que lhe são designadas.

Podemos ilustrar um algoritmo pelo exemplo de uma receita culinária, embora muitos algoritmos sejam mais complexos. Um Algoritmo mostra passo a passo os procedimentos necessários para resolução de um problema.

ALGORITMOS DE ORDENAÇÃO

Ordenação é o ato de se colocar os elementos de uma sequência de informações, ou dados, em uma relação de ordem predefinida. O termo técnico em inglês para ordenação é *sorting*, cuja tradução literal é "classificação".

Dado uma sequência de n dados:

$$< a_1, a_2, ..., a_n >$$

O problema de ordenação é uma permutação dessa sequencia:

$$< a'_1, a'_2, ..., a'_n >$$

tal que:

$$a'_1 \leq a'_2 \leq ... \leq a'_n$$

para alguma relação de ordem.

Algumas ordens são facilmente definidas. Por exemplo, a ordem numérica, ou a ordem alfabética ,crescentes ou decrescentes. Contudo, existem ordens, especialmente de dados compostos, que podem ser não triviais de se estabelecer.

Um algoritmo que ordena uma conjunto, geralmente representada num vetor, é chamado de algoritmo de ordenação. Algoritmo de ordenação em ciência da computação é um algoritmo que coloca os elementos de uma dada sequência em uma certa ordem, em outras palavras, efetua sua ordenação completa ou parcial. As ordens mais usadas são a numérica e a lexicográfica.Existem várias razões para se ordenar uma sequência. Uma delas é a possibilidade se acessar seus dados de modo mais eficiente.

Entre os mais importantes, podemos citar *bubble sort* (ou ordenação por flutuação), *heap sort* (ou ordenação por *heap*), *insertion sort* (ou ordenação por inserção), *merge sort*(ou ordenação por mistura) e o *quicksort*.

ALGORITMOS UTILIZADOS

Existem diversos algoritmos para ordenar dados , no entanto nos concentraremos nos 5 principais : Selection Sort, Bubble Sort, Quicksort , Mergesort e Insertion Sort.

BUBBLE SORT

Por ser simples e de entendimento e implementação fáceis, o Bubble Sort (bolha) está entre os mais conhecidos e difundidos métodos de ordenação de arranjos. Mas não se trata de um algoritmo eficiente, é estudado para fins de desenvolvimento de raciocínio.

O princípio do Bubble Sort é a troca de valores entre posições consecutivas, fazendo com que os valores mais altos (ou mais baixos) "borbulhem" para o final do arranjo (daí o nome Bubblesort).

Casos de Complexidade do Algoritmo:

- Melhor: O(n).
- Médio: O(n²).
- Pior: O(n²).

Exemplo de um algoritmo Bubble Sort implementado:

```
169  void bubble_sort(int vbub[MAX], int tamanho)
170 {
171      //variaveis locais
172      int temp,i,j,compbub = 0, trocas_bubble=0;
173      int trocou=1;
174 
175      for(i = tamanho-1; i > 0 && trocou == 1; i--)
176          {
177          trocou = 0;
178          for(j = 0; j < i; j++)
179          {
180        compbub++;
181                if(vbub[j] > vbub[j+1])
182                {
183                    temp       = vbub[j];
184                    vbub[j]    = vbub[j+1];
185                    vbub[j+1] = temp;
186                    trocas_bubble++;
187                    trocou = 1;
188                }//fim if
189 
190          }//fim for
191      }//fim for
192      return;
193 }//fim funcao Bubble
```

INSERTION SORT

Insertion sort, ou *ordenação por inserção*, é um simples algoritmo de ordenação, eficiente quando aplicado a um pequeno número de elementos. Em termos gerais, ele percorre um vetor de elementos da esquerda para a direita e à medida que avança vai deixando os elementos mais à esquerda ordenados. O algoritmo de inserção funciona da mesma maneira com que muitas pessoas ordenam cartas em um jogo de baralho como o pôquer.

Casos de Complexidade do Algoritmo:

- Melhor: O(n).
- Médio: O(n²).
- Pior: O(n²).

Exemplo de um algoritmo Insertion Sort implementado:

```
135  //Diálogo: Método da inserção para ordenação de vetores
136  void insertion_sort(int vins[MAX], int tamanho)
137  {
138      //variaveis locais
139      int j,w,x,comp = 0, trocas_insertion=0;
140
141      for(j = 1; j < tamanho; j++)
142          {
143          x = vins[j];
144
145          for(w = j-1; w >= 0 && vins[w] > x; w--)
146          {
147              vins[w+1] = vins[w];
148              comp++;
149
150          }
151
152          vins[w+1] = x;
153      trocas_insertion++;
154      }//fim for
155      return;
156  }//fim insertion
```

SELECTION SORT

Selection é um algoritmo de ordenação baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o de segundo menor valor para a segunda posição, e assim é feito sucessivamente com os (n-1) elementos restantes, até os últimos dois elementos.

Casos de Complexidade do Algoritmo:

- Melhor: O(n²).
- Médio: O(n²).
- Pior: O(n²).

Exemplo de um algoritmo Selection Sort implementado:

```
98  //Diálogo: Método de seleção para ordenação de vetores
99  void selection_sort(int vselec[MAX], int tamanho)
100 {
101     //declaracao variaveis locais
102     int temp,n,i,menor,comp = 0, trocas_selection=0;
103     for(i = 0; i < tamanho-1; i++)
104         {
105             menor = i;
106         for(n = i+1; n < tamanho; n++)
107         {
108
109             if(vselec[n] < vselec[menor])
110             {
111                 menor = n;
112                 comp++;
113
114             }
115         }//fim for
116
117         temp = vselec[menor];
118         vselec[menor] = vselec[i];
119         vselec[i] = temp;
120     trocas_selection++;
121     }//fim for
122     return;
```

MERGE SORT

O **merge sort**, ou ordenação por mistura, é um exemplo de algoritmo de ordenação do tipo dividir para conquistar.

Sua ideia básica consiste em Dividir(o problema em vários sub problemas e resolver esses sub problemas através da recursividade) e Conquistar(após todos os sub problemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos sub problemas).Como o algoritmo do Merge Sort usa a recursividade em alguns problemas esta técnica não é muito eficiente devido ao alto consumo de memória e tempo de execução.

Casos de Complexidade do Algoritmo:

- Melhor: O(n log n).
- Médio: O(n log n).
- Pior: O(n log n).

Exemplo do Algoritmo Merge Sort Implementado:

```
197  void merge_sort(int p, int tam, int vmerge[MAX])
198 {
199   //variaveis locais
200   int q;
201  int comp;
202  int trocamerg;
203  //int numtrocamerg;
204   //int *comp;
205
206   //Se 0 < tamanho do vetor - 1
207     if (p < tam - 1)
208     {
209     //q = meio
210     q = (p + tam) / 2;
211     merge_sort(p , q , vmerge, compar, numtrocamerg);
212     merge_sort(q, tam, vmerge, compar, numtrocamerg);
213     intercala(p, q, tam , vmerge);
214     }//fim if
215 }//fim da funcao MergeSort
```

Exemplo da Função Intercala (Usada para auxiliar o algoritmo Merge Sort)

```
222  void intercala(int p, int q , int tam , int vmerge[MAX])
223  {
224      int i = p, j = q, k = 0, w[MAX] , t = 0;
225      while (i < q && j < tam){
226   if (vmerge[i] < vmerge[j]){
227     w[k] = vmerge[i]; i++;
228   }//fim if
229   else{
230     w[k] = vmerge[j];   j++;
231   }//fim else
232      k++;
233      t++;
234   }//fim while1
235   while (i < q){
236     w[k] = vmerge[i];  i++;  k++;
237    }//fim while
238   while (j < tam){
239       w[k] = vmerge[j];
240         j++;
241         k++;
242   }//fim while2
243  for (i = p; i < tam; i++){
244  vmerge[i] = w[i-p];
245  }//fim for
246  }//Fim da funcao intercala
```

QUICK SORT

O algoritmo Quicksort é um método de ordenação muito rápido e eficiente, inventado por C.A.R. Hoare em 19601 , quando visitou a Universidade de Moscou como estudante. Naquela época, Hoare trabalhou em um projeto de tradução de máquina para o National Physical Laboratory. Ele criou o Quicksort ao tentar traduzir um dicionário de inglês para russo, ordenando as palavras, tendo como objetivo reduzir o problema original em subproblemas que possam ser resolvidos mais fácil e rapidamente. Foi publicado em 1962 após uma série de refinamentos.2

O Quicksort é um algoritmo de ordenação por comparação não-estável.

Casos de Complexidade do Algoritmo:

- Melhor: O(n log n).
- Médio: O(n log n).
- Pior: O(n²).

Exemplo do Algoritmo QuickSort Implementado:

```
255  void quicksort(int p, int r, int vquick[MAX])
256  {
257     //Declaracao das variaveis locais
258      int j;
259      //int comparaquick;
260     while (p < r) {
261        j = separa( p, r, vquick);
262        // printf("%d ", trocaquick);
263        if (j - p < r - j) {
264           quicksort( p, j-1 , vquick);
265           p = j + 1;
266        }//fim if
267         else {
268           quicksort( j+1, r, vquick);
269           r = j - 1;
270        }//fim else
271     }//fim while
272  }//Fim da Funcao Quick_Sort
```

Exemplo da Função Separa (Usada para auxiliar o algoritmo Quick Sort)

```
278  int separa(int p, int r, int vquick[MAX])
279  {
280      //variaveis locais
281      int c = vquick[p], i = p+1, j = r, t;
282     while (/*A*/ i <= j)
283          {
284        if (vquick[i] <= c){
285          ++i;
286        }
287        else if (c < vquick[j]){
288         --j;
289        }
290        else
291          {
292           t = vquick[i], vquick[i] = vquick[j], vquick[j] = t;
293           ++i; --j;
294          }//fim else
295      }//fim while
296     // agora i == j+1
297     vquick[p] = vquick[j], vquick[j] = c;
298     return j;
299  }//Fim Funcao separa
```

PROCEDIMENTOS EXPERIMENTAIS

Para realização prática deste experimento, foram feito testes com os algoritmos estudados, os testes foram os seguintes:

Verificar o comportamento dos algoritmos em relação as movimentações de trocas e comparações.

Para isso , unimos todos os algoritmos de ordenação em apenas um arquivo.

Algoritmo Gerador de Entradas:

```
289  //GERADOR
290  int main (int argc, char **argv)
291  {
292
293      //printf("argc %d  argv %s\n", argc, argv[1]);
294
295      int m=atoi(argv[1]); // quantidade de casos de teste
296      int n=1;  // quantidade de elementos do vetor que sera gerado
297      int i;
298
299      printf("%d",m);
300
301      while (n<=m)
302      {
303          printf("\n");
304          printf("%d\n",n);
305          for(i=1;i<=n;i++)
306              printf("%d ",rand()%200);
307          n++;
308      }
309
310  }
311
```

Utilizando o algoritmo acima , geramos diversos casos de teste para nosso programa , assim , podemos analisar o comportamento dos cinco algoritmos de ordenação utilizando gráficos. Para criar os gráficos , utilizamos o programa:

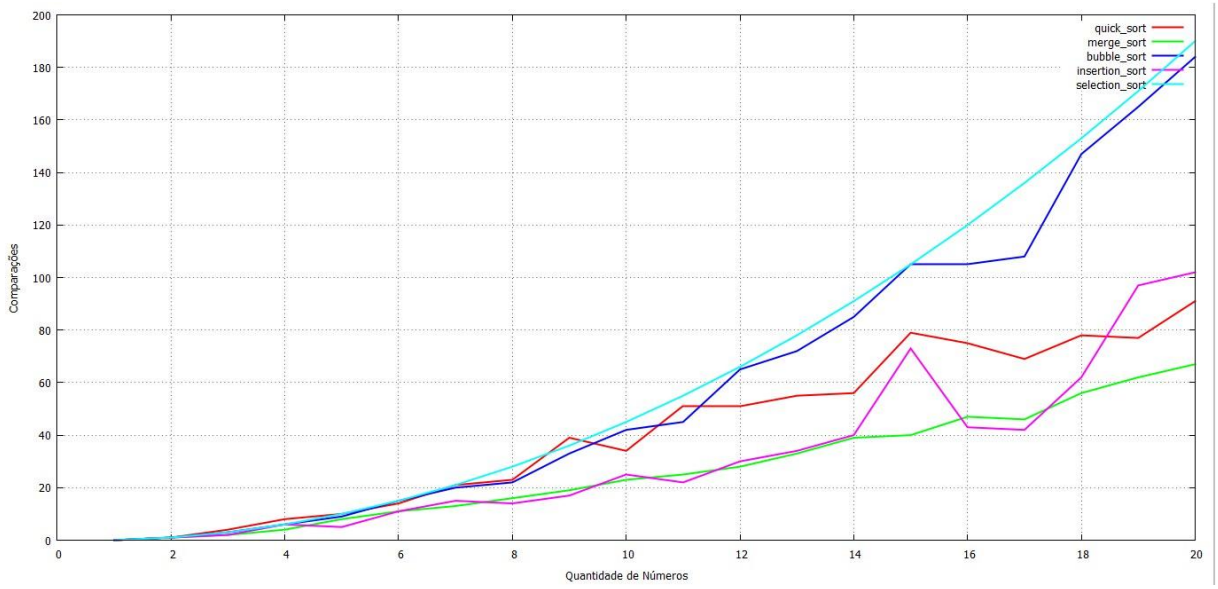
GNUPLOT

RESULTADOS

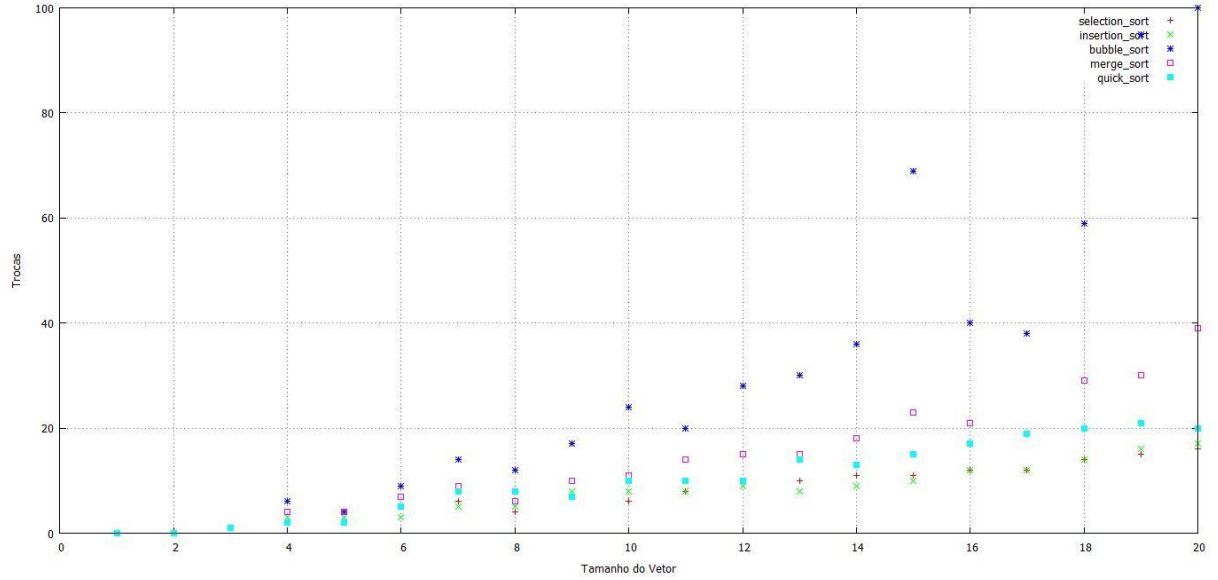
No nosso experimento , geramos diversas entradas aleatórias, e com o resultado , construímos os gráficos a seguir:

Testando o algoritmo para 20 entradas,obtivemos:

Número de Comparações

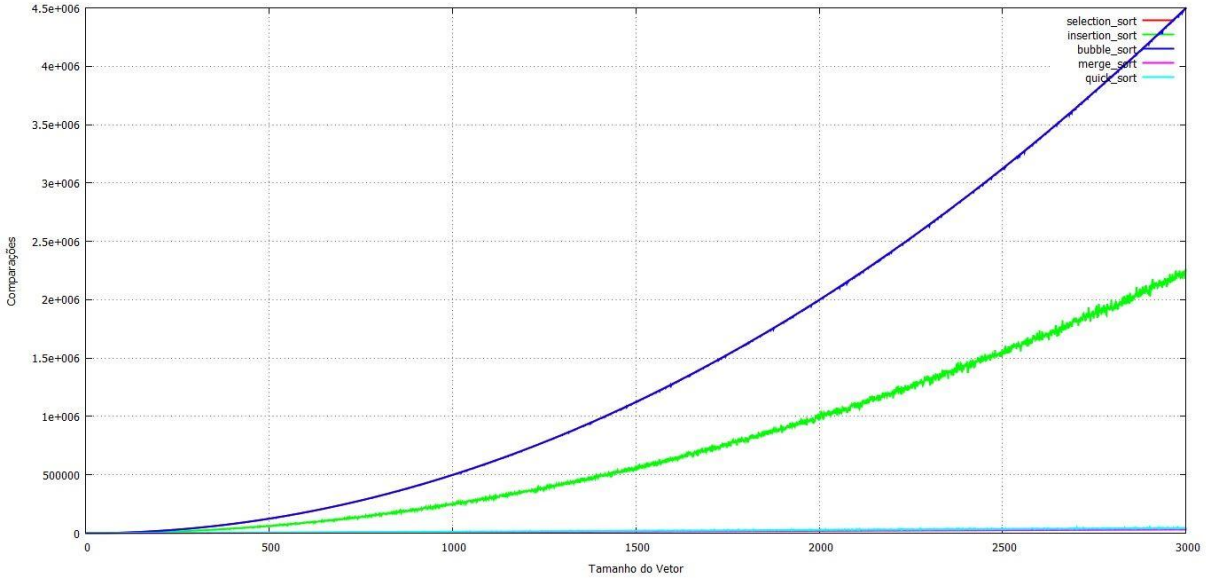


Número de Trocas para 20 entradas:



Outros gráficos,mas para entradas maiores:

Numero de comparações para 3000 entradas



Número de trocas para 3000 entradas

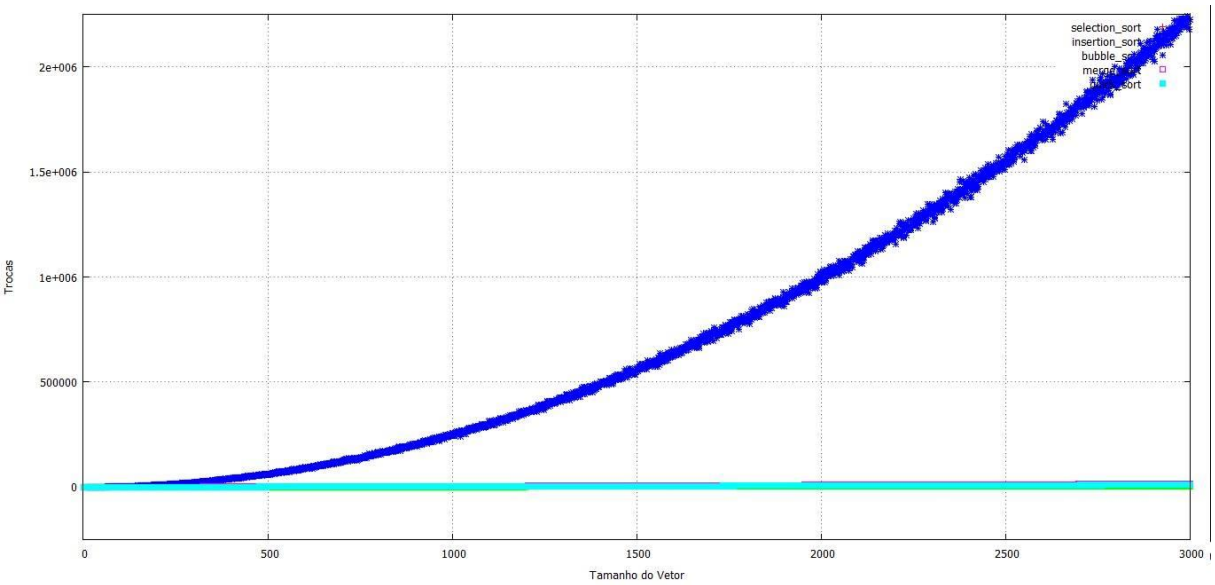


Tabela para 3000 entradas aleatórias:

Algoritmo	Número de Trocas	Número de Comparacões
Selection Sort	**2981**	**4498500**
Insertion Sort	**2986**	**2257096**
Bubble Sort	**2257083**	**4495799**
Merge Sort	**15706**	**30901**
Quick Sort	**44877**	**10741**

Tabela para 20 entradas aleatórias:

Algoritmo	Número de Trocas	Número de Comparacões
Selection Sort	**17**	**190**
Insertion Sort	**18**	**111**
Bubble Sort	**110**	**175**
Merge Sort	**13**	**60**
Quick Sort	**27**	**98**

CONCLUSÕES E RECOMENDAÇÕES

Com base nos testes realizados foram obtidas as seguintes conclusões:

Bubble Sort

É o que executa em maior número de trocas e maior número de comparações. Até em listas já ordenadas,e também em todos os casos, o bubble sort se mostrou um algoritmo ineficiente.

Selection Sort

Foi o segundo pior algoritmo , no entanto , ele realiza menos trocas do que o Insertion tanto com grandes e pequenas entradas.

Insertion Sort

É o melhor algoritmo para entradas pequenas .Realiza menos comparações do que o Selection , porém , como foi mostrado no gráfico para 3000 entradas , ao aumentar o número de entradas , o número de comparações cresce absurdamente,tornando-se o segundo pior no quesito de comparações em grandes entradas.

Merge Sort

É um algoritmo eficiente para entradas pequenas , superando o Quick Sort , todavia , ao ser executado para entradas maiores , o número de comparações cresce bastante.

Quick Sort

Como o próprio nome já diz , é um algoritmo rápido. Foi , durante os testes, o segundo melhor algoritmo para entradas pequenas , e , ao realizar grandes trocas e comparações , obteve-se um resultado muito satisfatório.

REFERÊNCIAS

Websites:

http://www.ft.unicamp.br/liag/siteEd/implementacao/selection-sort.php
http://www.ft.unicamp.br/liag/siteEd/implementacao/insertion-sort.php
http://www.ft.unicamp.br/liag/siteEd/implementacao/bubble-sort.php

http://pt.wikibooks.org/wiki/Algoritmos_e_Estruturas_de_Dados/Algoritmos_de_Ordena%C3%A7%C3%A3o

http://www.ime.usp.br/~song/mac5710/slides/01complex.pdf

Obra Literárias:

C - Como Programar - 6ª Ed. 2011,

Paul Deitel; Paul Deitel; Deitel, Harvey; Deitel, Harvey / Pearson Education - Br