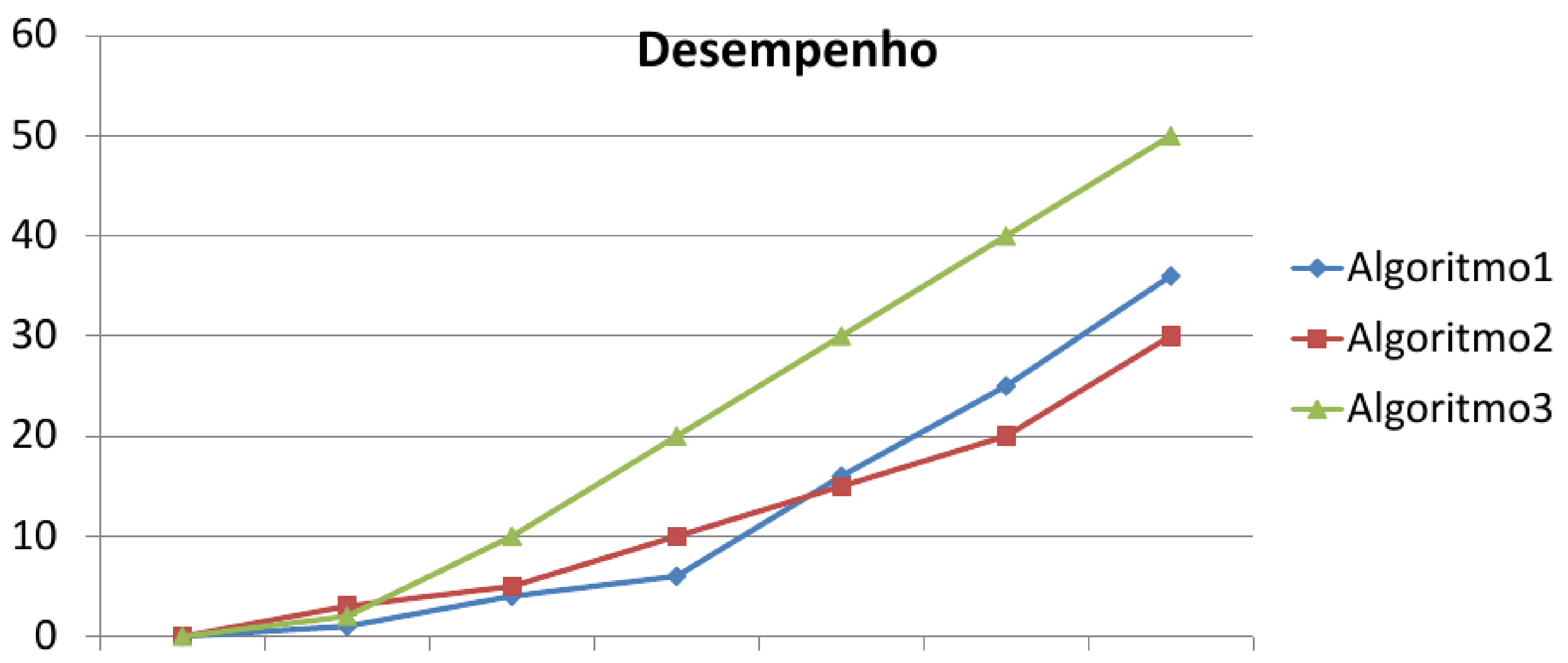
Teoria da Complexidade

Prof. Marcelo Dib Cruz

Teoria da complexidade



Complexidade de Algoritmos

Teoria da complexidade

- Algoritmo: Uma seqüência de instruções bem definidas e não ambígüas. Implementado para resolver um problema.
- Determinísticos: O resultado de cada operação é determinado de forma única (Alg. Caminho mínimo, Ordenação, Árvore Geradora Mínima, etc.)
- Não Determinísticos: Capaz de escolher uma dentre várias alternativas possíveis a cada passo

Teoria da complexidade

- Em geral, os algoritmos podem ser avaliados utilizando-se o tempo de execução e a quantidade de memória utilizada.
- O tempo de execução é o parâmetro mais utilizado para avaliação da eficiência de algoritmos.
- O tempo de execução depende dos seguintes fatores: do programador, dos dados de entrada, da qualidade do código, do hardware utilizado e da complexidade do algoritmo implementado.

Teoria da complexidade

- **Tempo de Execução** de um algoritmo varia com o input e normalmente aumenta com o tamanho do input.
  - caso médio é difícil de determinar.
  - normalmente, olha-se para o pior caso possível de tempo de execução:
    - mais simples de analizar
    - crucial nas aplicações mais exigentes, jogos, etc.

Teoria da complexidade

- Análise empírica – executando o programa com inputs de tamanho e composição variados, mas
  - nem sempre é simples concretizar o algoritmo, ou
  - os resultados podem não ser conclusivos,
  - comparação de algoritmos obriga a usar igual software e hardware.
- Análise teórica
  - usa uma descrição de mais alto nível do algoritmo em vez da implementação,
  - caracteriza o tempo de execução como uma função do tamanho do input, n,
  - tem em conta todos os possíveis inputs,
  - permite avaliar eficiência de forma independente do ambiente de hardware/software.

Teoria da complexidade

Como calcular o tempo de execução do algoritmo seguinte:

```
int findMax(int A[], int n) {
    int max= A[0];
    int i= 1;
    while (i <= n-1) {
        if (A[i]>max)
          max= A[i];
        i= i+1;
    }
    return max;
}
```

2 operações
1 operação
n operações
n-1 vezes
2 ops
2 ops
2 ops
1 operação

Teoria da complexidade

- Neste Caso :
  - Acessar ao conteúdo de um endereco custa 1 unidade de tempo
  - max = A[0] ; 1 leitura de A[0] + 1 atribuição a max

Teoria da complexidade

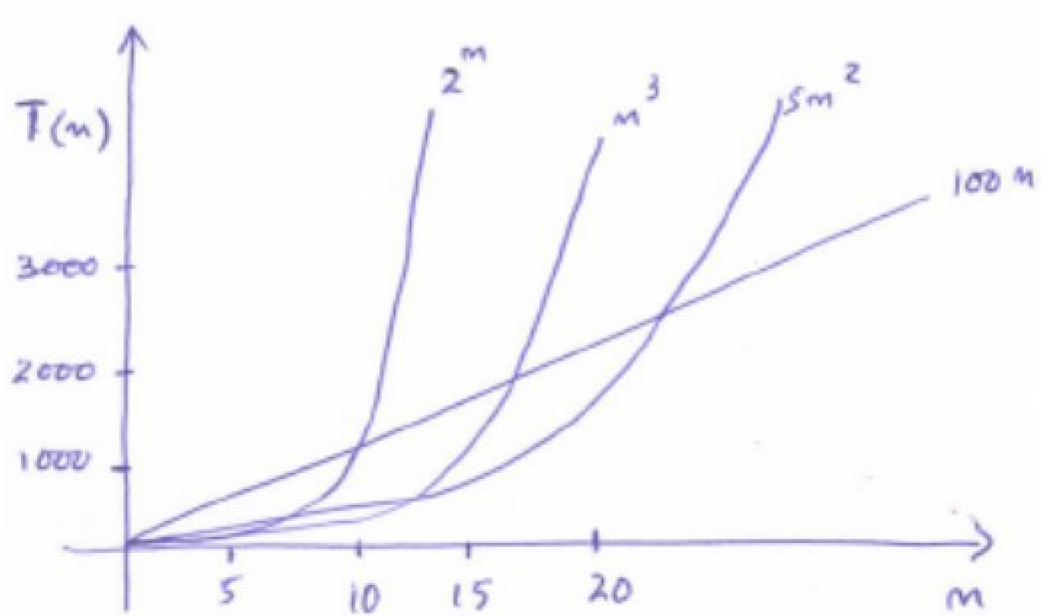
Casos possíveis:

- caso mais favorável (`A[0]` é o maior elemento):
  $t(n) = 2 + 1 + n + 4(n-1) + 1 = 5n$ operações primitivas
- pior caso:
  $t(n) = 2 + 1 + n + 6(n-1) + 1 = 7n - 2$
- caso médio? difícil de calcular, depende da distribuição do input; usar teoria de probabilidades.

Teoria da complexidade

Então, verifica-se, que T(n) é Linear

logarítmica	$\approx \log_2 n$
linear	$\approx n$
quadrática	$\approx n^2$
cúbica	$\approx n^3$
polinomial	$\approx n^k$
exponencial	$\approx a^n (a > 1)$



Teoria da complexidade

Crescimento de algumas funções:

n	$\log_2 n$	$\sqrt{n}$	n	$n \log_2 n$	n^2	n^3	2^n
2	1	1.4	2	2	4	8	4
8	3	2.8	8	24	64	512	256
16	4	4.0	16	64	256	4096	65536
...	...	...	...	...	...	...	...
1024	10	32	1024	10240	$> 10^6$	$> 10^9$	$> 10^{308}$

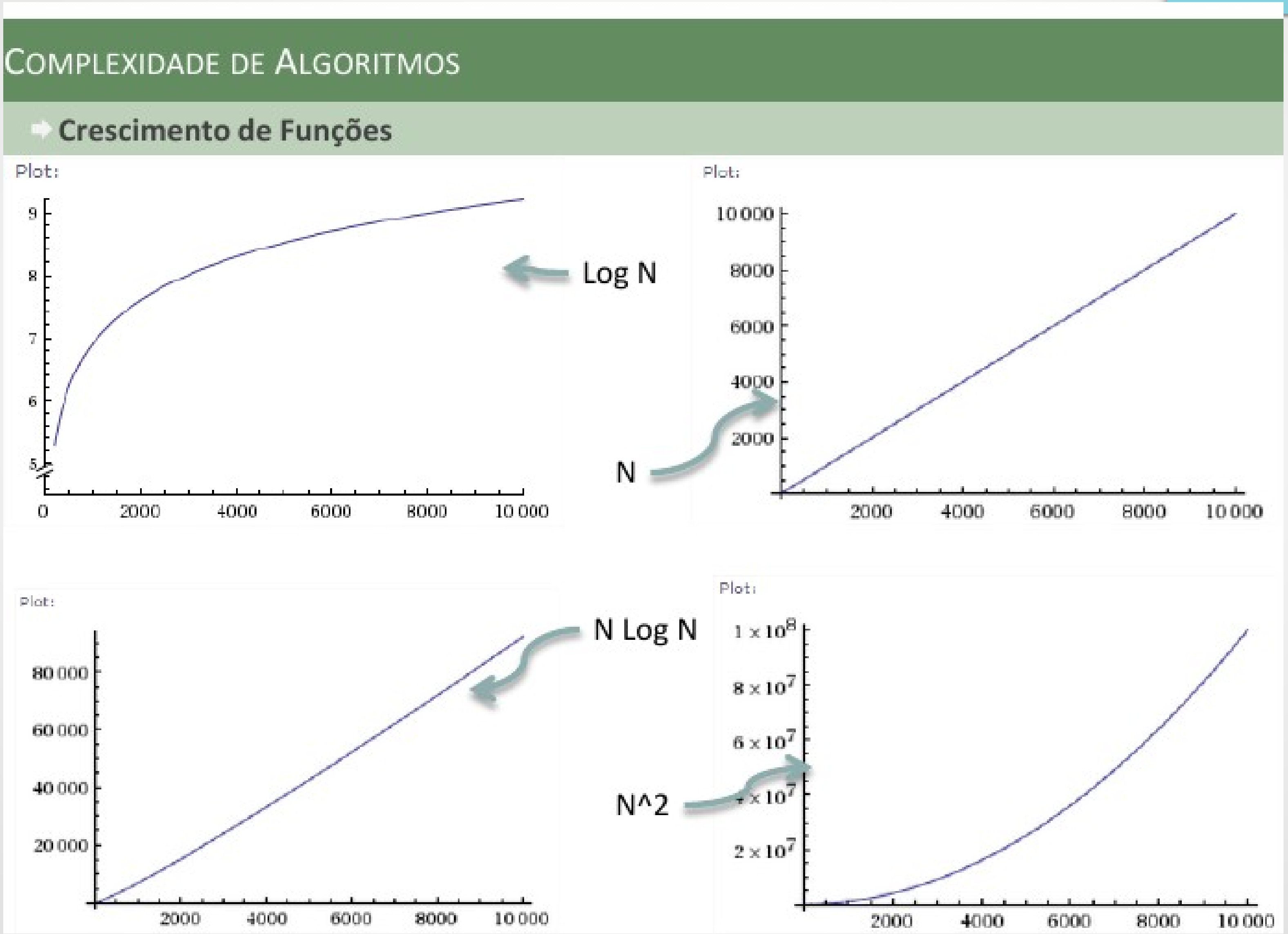
- A função que associa o tempo de execução de um algoritmo ao tamanho (n) dos dados é denominada complexidade em tempo do algoritmo. f(n)
- A complexidade de um algoritmo é medido pela quantidade de operações básicas (e.g., comparação de dois números) necessárias para resolver o problema.
- Classes de Problemas e funções de complexidade
    - f(n)=1 = C. Constante
    - f(n)= O(log n) = Logarítmica (Busca Binária, Ordenação)
    - f(n)= O(n) = Linear (Busca Linear)

Teoria da complexidade

- f(n)= O(n^2) = Quadrática (Soma de Matrizes)
- f(n)= O(n^3) = Cúbica (Produto de Matrizes,Caminho Mínimo Floyd)
- f(n)=O(2^n) = Exponencial (Não são úteis sobo ponto de vista prático;
- f(n)=O(n!)=C. Fatorial . Bem pior que exponencial (Caixeiro Viajante).

Os tipos (6) e (7) aparecem na solução de problemas quando se usa a força bruta para resolvê-los.

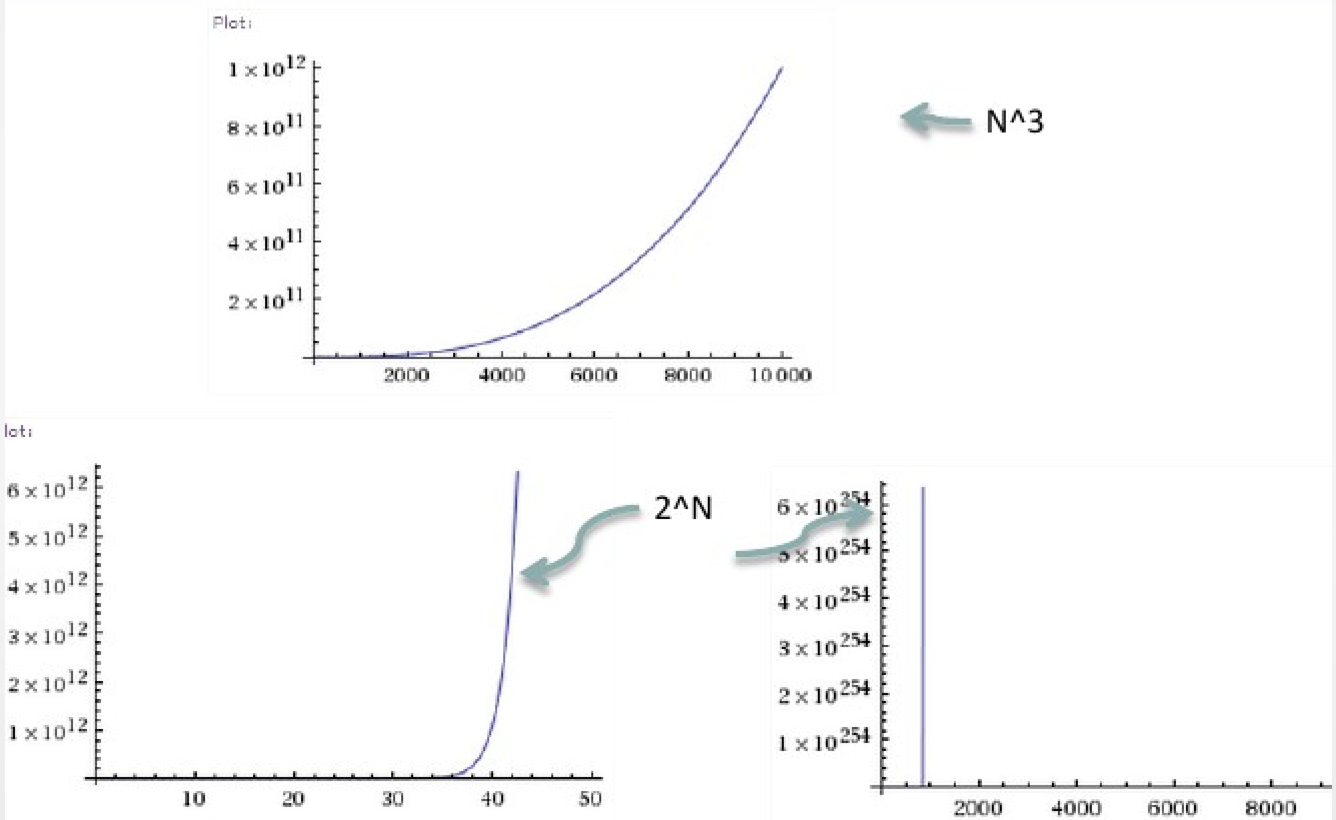
Teoria da complexidade



Teoria da complexidade

Complexidade de Algoritmos

Crescimento de Funções



Teoria da complexidade

- É possível obter uma ordem de grandeza do tempo de execução através de métodos analíticos;
- O objetivo destes métodos é determinar uma expressão matemática que traduza o comportamento de tempo de um algoritmo ;
- Ao contrário do modo empírico, o analítico visa aferir o tempo de execução de forma independente do computador utilizado, da linguagem utilizada e etc;

Teoria da complexidade

- A tarefa de obter uma expressão matemática para avaliar o tempo de um algoritmo em geral não é simples, mesmo considerando-se uma expressão aproximada
- Algumas simplificações são necessárias:
    - Suponha que a quantidade de dados a serem manipulados pelo algoritmo sejam suficientemente grande;
    - Somente o comporttamento assintótico será considerado;
    - Não serão consideradas constantantes aditivas ou multiplicativas na expressão matemática obtida;

Teoria da complexidade

- É necessário definir a variável em relação a qual a expressão matemática avaliará o tempo de execução.
  - Um algoritmo opera a partir de uma entrada para produzir uma saída.
  - A função matemática obtida será expressa em função da entrada.

Teoria da complexidade

- O processo de execução de um algoritmo pode ser dividido em etapas elementares, denominadas **passos.**
- Cada **passo** consiste na execução de um numero fixo de operações básicas cujos tempos de execução são considerados constantes.
- A operação básica de maior frequencia na execução do algoritmo é denominada **operação dominante**.
- Como a expressão do tempo de execução do algoritmo será a menos de constantes aditivas e multiplicativas, **o número de passos de um algoritmo** pode ser interpretado como sendo o **número de execuções da operação dominante**.

Teoria da complexidade

- Na realidade, o número de passos de um algoritmo constitui a informação de que se necessita para avaliar o seu comportamento de tempo.
- O algoritmo de um único passo possui tempo de execução constante
- Pelo exposto, é natural definir a expressão matemática de avaliação do tempo de execução de um algoritmo como sendo uma função que fornece o numero de passos efetuados pelo algorittmo a partir de uma certa entrada.

Teoria da complexidade

Exemplo1 : Soma de matrizes

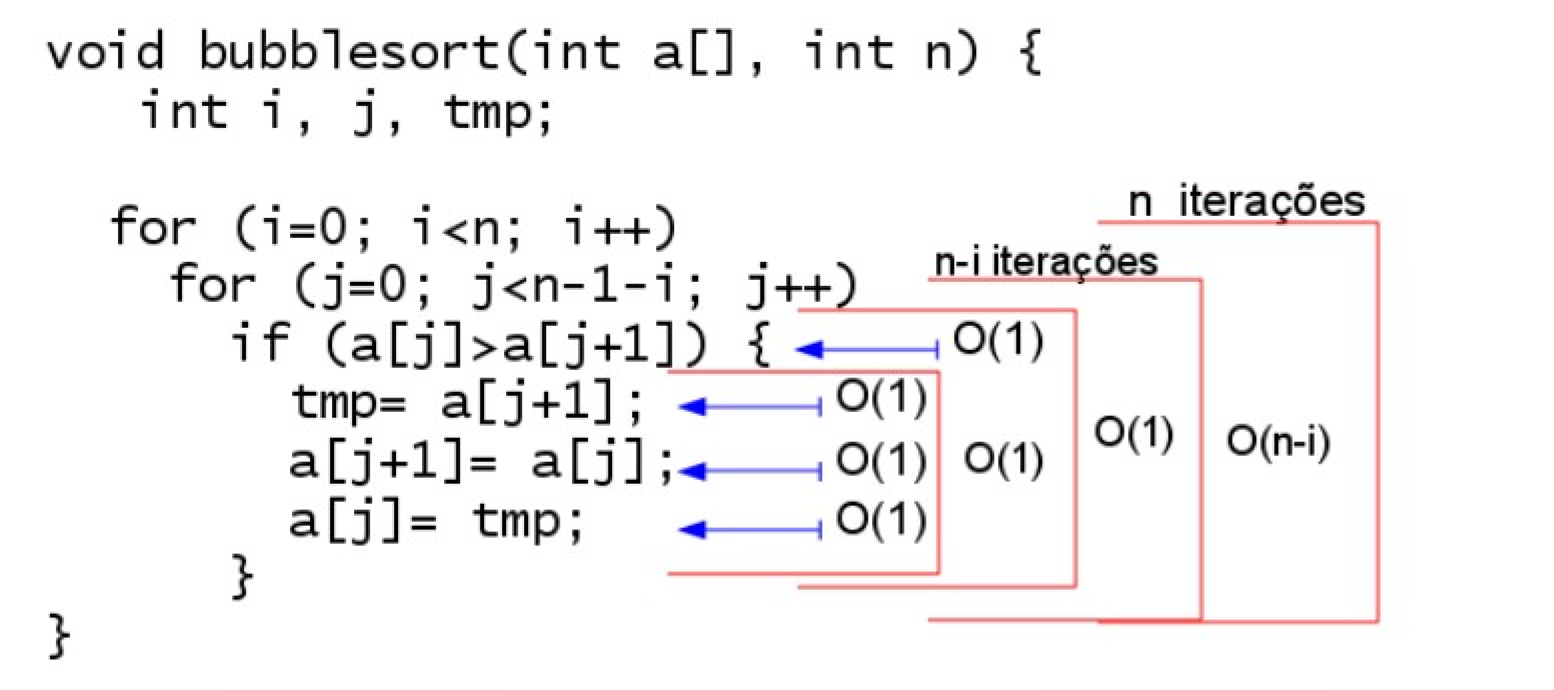
```
void somarMatrizes(int a[][L], int b[][L],  int c[][L],  int n)   {
                int i, j;
          for (i = 0; i < n; i ++)
          for (j = 0; j < n; j++)
              c[i][j] = a[i][j] + b[i][j];
   }
```

Teoria da complexidade

Exemplo2 : produto de matrizes

```
void somarMatrizes(int A[][L], int B[][L],  int C[][L],  int n)
{  int i, j;
for(i=0; i<n; i++)
        for(j=0; j<n; j++)
                for(k=0; k<n; k++)
                     C[i][j] += A[i][k] * B[k][j];
```

Teoria da complexidade



Teoria da complexidade

Efectivamente, o número de iterações total é a soma das iterações que o ciclo `j` faz para cada valor de `i`, i.e.

$$\begin{aligned}\sum_{i=1}^{n-1}(n-i) &= (n-1)+(n-2)+\ldots+2+1= \\ &= \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2} \Rightarrow \mathcal{O}(n^2)\end{aligned}$$

Teoria da complexidade

```
int pesqBinNaoRec(int x, int v[], int e, int d) {
  while (e<=d) {
     int meio= (e+d)/2;
     if (x==v[meio])
       return meio;
     if (x<v[meio]) d= meio-1;
     else e= meio+1;
  }
  return -1; // sinaliza que não encontrou
}
```

tempo constante i.e. $O(1)$

Quantas iterações tem este ciclo?

Teoria da complexidade

- Inicialmente o intervalo de valores é n ($e = 0$ e $d = n - 1$). Em cada passo do ciclo, o intervalo reduz-se a metade.
- Portanto, a questão que se coloca é:
  - dado um inteiro n, quantas divisões inteiras por 2 são necessárias para que chegue a 1?
  - i.e. qual dos valores seguintes será o primeiro a ser < 1?
    $n/2, n/4, n/8, \ldots, n/2^k, \ldots$

Teoria da complexidade

- É necessário resolver a equação: $n/2^k < 1$

  se aplicarmos logaritmos, temos $k > \log_2 n$
- Podemos então dizer que:
  - com $k = \lceil \log_2 n \rceil$ sabemos que ao fim de um máximo de k iterações, encontramos o valor ou podemos concluir que o valor que procuramos não existe.
- Em resumo, dizemos que a função `pesqbin()` tem complexidade logarítmica, $\mathcal{O}(\log_2 n)$, pois o número de iterações necessárias não excede $\log_2 n$, sendo n a dimensão dos dados.

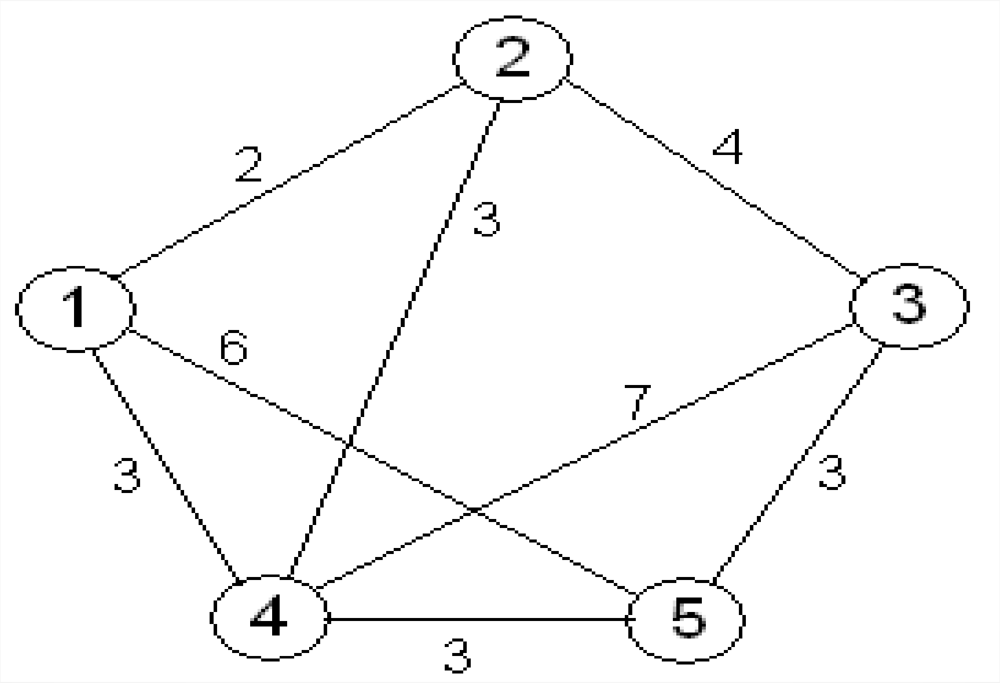
Teoria da complexidade

Teoria da complexidade

O Problema do Caixeiro Viajante (PCV) é um problema que tenta determinar a menor rota para percorrer uma série de cidades (visitando cada uma pelo menos uma vez), retornando à cidade de origem. Ele é um problema de otimização NP-difícil inspirado na necessidade dos vendedores em realizar entregas em diversos locais (as cidades) percorrendo o menor caminho possível, reduzindo o tempo necessário para a viagem e os possíveis custos com transporte e combustível.

Teoria da complexidade

Teoria da complexidade



Teoria da complexidade

Caixeiro Viajante

- Para N cidades há (N-1)! rotas.
- Para 11 cidades, há 10! = 3.628.800 rotas.
- Para 12 cidades, há 11! = 39.916.800 rotas.
- Para 26 cidades, há

  25! = 15.511.210.043.330.985.984.000.000 rotas.

Teoria da complexidade

Função tempo/ complexidade	**Quantidade de Dados: N**				
	10	20	30	40	50
N	0,00001s	0,00002s	0,00003s	0,00004s	0,00005s
N^2	0,0001s	0,0004s	0,0009s	0,0016s	0,0036s
N^3	0,001s	0,008s	0,027s	0,064s	0,125s
2^N	0,001s	1,0s	17,19 min	12,7 dias	35,7 anos
3^N	0,059s	58 min	6,5 anos	3.855 séculos	200.000.000 séculos

Teoria da complexidade

- Algoritmos polinomiais (tempo de execução) a função de complexidade é O(p(n)), onde p(n) é um polinômio.
- Algoritmos Exponenciais (tempo de execução), cuja função de complexidade é $O(c^n)$.
- De uma forma geral, os algoritmos polinimiais são considerados bons, enquanto os exponenciais são ruins.

Teoria da complexidade

- Definição : Seja A um algoritmo , $\{E_1, \ldots.. E_m\}$ o conjunto de todas as entradas possíveis de A. Denote por t_i , o número de passos efetudos por A, quando a entrada for E_i. Definem-se :
  - Complexidade do pior caso: Max $\{ t_i \}$, E_i Ɛ E
  - Complexidade do melhor caso: Min $\{ t_i \}$, E_i Ɛ E
  - Complexidade do caso medio : $\sum p_i * t_i$, $1 <= i <= m$

    onde p_i e a probabilidade de ocorrencia da entrada E_i ;

Teoria da complexidade

- A Complexidade de temp de pior caso, corresponde ao número de passos que o algoritmo efetua no seu pior caso de execução, isto é, para a sua entrada mais desfavorável;
- De certa forma forma, a complexidade do pior caso é a mais importante das tres;
- Ela fornece um limite supeior para o número de passos que o algoritmo pode efetuar, wm qualquer caso;
- O termo complexidade será, então, empregado com o significado de complexidade de pior caso;

Teoria da complexidade

Exemplo : Bubblesort

```
void bubblesort(int a[], int n) {
   int i, j, tmp;
                                                    n  iterações
  for (i=0; i<n; i++)
                                              n-i iterações
    for (j=0; j<n-1-i; j++)
      if (a[j]>a[j+1]) {       O(1)
        tmp= a[j+1];       O(1)
                                          O(1)      O(n-i)
        a[j+1]= a[j];      O(1)   O(1)
        a[j]= tmp;         O(1)
      }
}
```

Teoria da complexidade

Exemplo 2 : Soma e produto de matriz

Teoria da complexidade

A Notação O

Sejam f , h funções reais positivas da variável inteira n. Diz-se que f é O(h), quando existir uma constante $c > 0$ e um valor inteiro n_0 tal que :

$$n > n_0 \Rightarrow f(n) \leq c * h(n)$$

Ou seja, a função h atua como um limite superior para os valores assíntoticos da função f ;

Teoria da complexidade

Exemplos :

$f = n^2 – 1 \Rightarrow f = O(n^2)$

$f = n^4 – 5n^3 + 2n \Rightarrow f = O(n^4)$

$f = 5n + 1000 \Rightarrow f = O(n)$

A notação O será usada para exprimir complexidade;

Exemplos

Algoritmo para acessar um elemento em um vetor

```
Função acesso ( v: Vetor(N) Inteiro;  i,N: Inteiro ) : Inteiro
        Se i > N então
          erro("Acesso fora dos limites do vetor!");
        Senão
          retorne v[i];
       Fim-Se.
```

Exemplos

Considere o seguinte algoritmo para inverter um arranjo:

```
função inversão(V: Ref Vetor[n] inteiro; n: inteiro)
   var i, aux: inteiro;
   Início
   Para i := 1 até n/2 faça
      aux := V[i];
      V[i] := V[n-i+1];
      V[n-i+1] := aux;
   Fim-Para
Fim.
```

Exemplos

Algoritmo para achar o máximo elemento de um vetor

```
Função máximo (v: Vetor(N) Inteiro; N: Inteiro): Inteiro
var i, max: Inteiro;
Início
   Se N = 0 Então % c1
      erro("máximo chamado com vetor vazio!");
   Senão
      max := v[1]; % c2
      Para i := 2 Até N Faça % c3
         Se v[i] > max Então % c4
            max := v[i]; % c5
       Fim-Para
   Fim-Se
   Retorne max; % c6
Fim.
```

Exemplos

Desenvolver um algoritmo para transpor uma matriz quadrada M. Os parâmetros do algoritmo são a matriz M, de tamanho n × n. Não utilize matriz ou vetor auxiliar na solução.

Determinar a complexidade do algoritmo em função de n.

```
Função transpor(M: Ref Matriz[n,n] Inteiro; n: Inteiro)
Var aux, i, j: Inteiro;
Início
  Para i := 1 Até n-1 Faça % c1
    Para j := i+1 Até n Faça % c2
      aux := M[i][j]; % c3
      M[i][j] := M[j][i]; % c4
      M[j][i] := aux; % c5
    Fim-Para
  Fim-Para
Fim.
```

$T = (n - 1)(c1 + I)$, onde $I = (n - i)(c2 + c3 + c4 + c5)$.

Exemplos

Algoritmo para somar os elementos de uma lista

```
Função soma(L: Ref Vetor(N) Inteiro; N: Inteiro) : inteiro
Var resposta: Inteiro;
Início
   Se N = 0 Então % c1
      resposta := 0; % c2
   Senão
      resposta := (L[1] + soma(L[2..N], N-1)); % c3
   Fim-Se
   Retorne resposta; % c4
Fim.
```

Exemplos

Referencias Bibliograficas

- SZWARCFITER, J. L. ; MARKENZON, L. . Estruturas de Dados e seus Algoritmos. Rio de Janeiro: LTC, 2015. v. 1.

- Fernando Silva – DCC-FCUP -https://www.dcc.fc.up.pt/~fds/

Exemplos