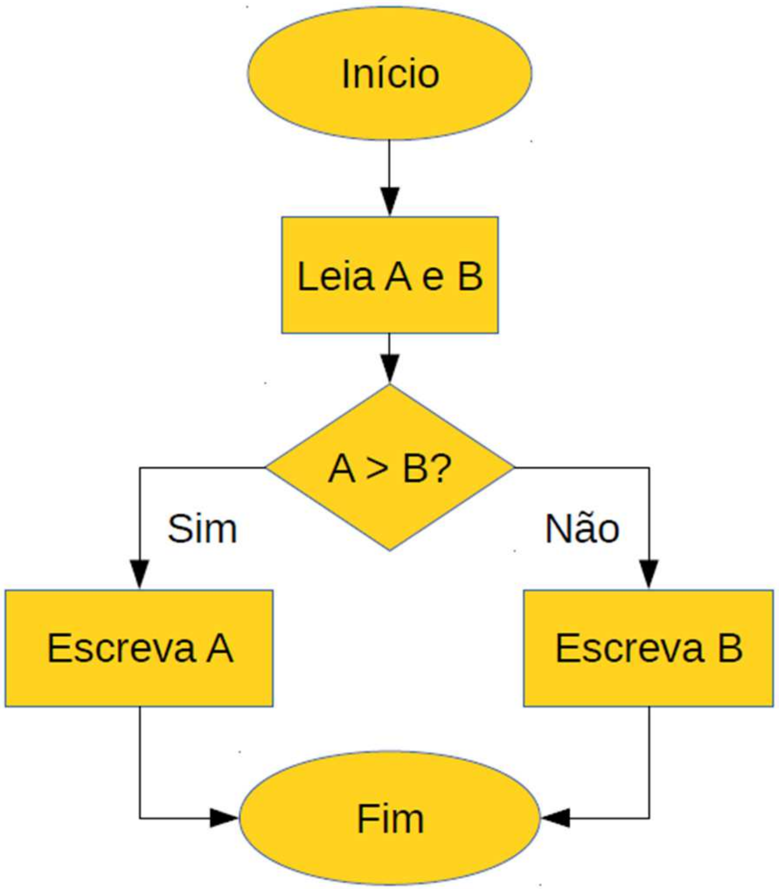
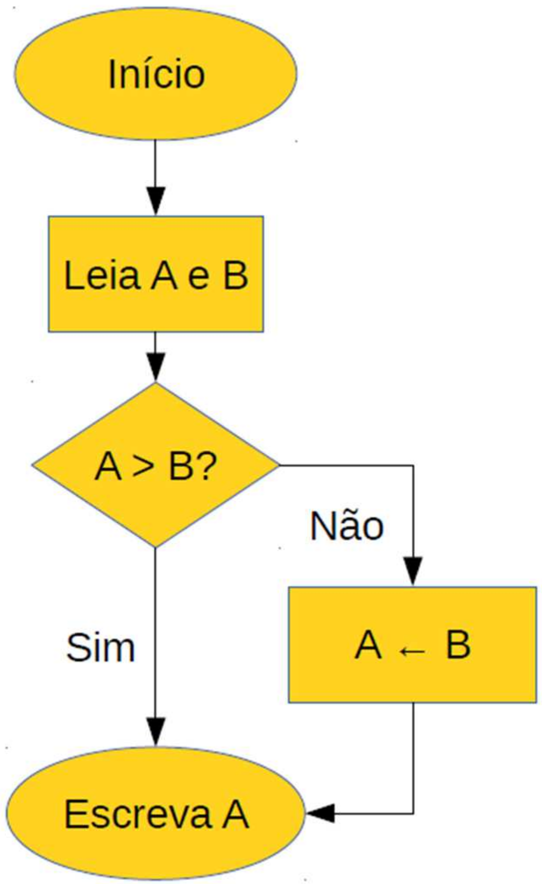
ICC – Complexidade

PROFª. SARA MELO SLIDES ADAPTADOS DO PROFº JEAN PONCIANO

Como escolher um algoritmo?





Algoritmo

- Algoritmo é um procedimento computacional bem especificado que torna um valor (ou conjunto) como entrada e produz um valor (ou conjunto) como saída.
  - Moleza!!!! ☺

- Eficiência de Algoritmos????

- Dois principais recursos:
  - Tempo de execução.
  - Espaço ocupado.

Como saber o tempo de processamento?

- **Opção 1**: Medindo!
  - Não é uma boa opção.
  - Depende da implementação, do compilador e do hardware.

Resultados jamais devem ser generalizados!!!!

- **Opção 2:** Estudando o número de vezes que as operações são executadas.

Estimar o uso dos recursos antes de executá-lo depende da **Análise do Algoritmo**!

Como saber o tempo de processamento?

Para medir o custo de execução de uma algoritmo é comum definir uma função de custo ou **função de complexidade** *f* , em que *f(n)* é a medida do tempo necessário para executar um algoritmo para um problema de tamanho *n*.

Como saber o tempo de processamento?

Para determinar o tempo de execução deve definir um modelo de computação, o qual consiste de:

- Um processador genérico.
- Operações executadas de modo sequencial.
  - Operações aritméticas, atribuições e manipulações de índices têm tempo constante;
  - O custo das operações relevantes para a solução do problema (Comparação).

Exemplo de tempo de processamento

- Achar o maior número inteiro em um vetor

```
int vmax(int *vec, int n) {                        -
        int i;                                     -
        int max = vec[0];                          1
        for(i = 1; i < n; i++) {                   n-1
                if(vec[i] > max) {                 n-1
                        max = vec[i];              A < n-1
                }                                  -
        }                                          -
        return max;                                1
}
```

- Complexidade: **f(n) = n-1** (o que é um tempo muito bom).

Análise do tempo de processamento

- Análise feita em função de n.
  - n indica o tamanho da entrada
    - Número de elementos do vetor
    - Número de vértices num grafo
    - Número de linhas de uma matriz
    - ...

- Diferentes entradas podem ter custos diferentes
  - Melhor caso, caso médio, **pior caso**.
  - Em geral, o pior caso é o mais interessante, já que define o maior tempo de processamento possível.

Análise do tempo de processamento

- Exemplo 1:
  - Considere dois algoritmos que resolvem o mesmo problema e que possuem os números de operações a seguir:

1. Algoritmo 1: $f_1(n) = 2n^2 + 5n$ operações
2. Algoritmo 2: $f_2(n) = 500n + 4000$ operações

  - Qual algoritmo é melhor?

Análise do tempo de processamento

- Exemplo 1:
  - Considere dois algoritmos que resolvem o mesmo problema e que possuem os númer de operações a seguir:

1. Algoritmo 1: $f_1(n) = 2n^2 + 5n$ operações
2. Algoritmo 2: $f_2(n) = 500n + 4000$ operações

  - Qual algoritmo é melhor?
    - **Depende do valor de n (tamanho da entrada). Compare as duas funções para n = 10 e r 10000.**

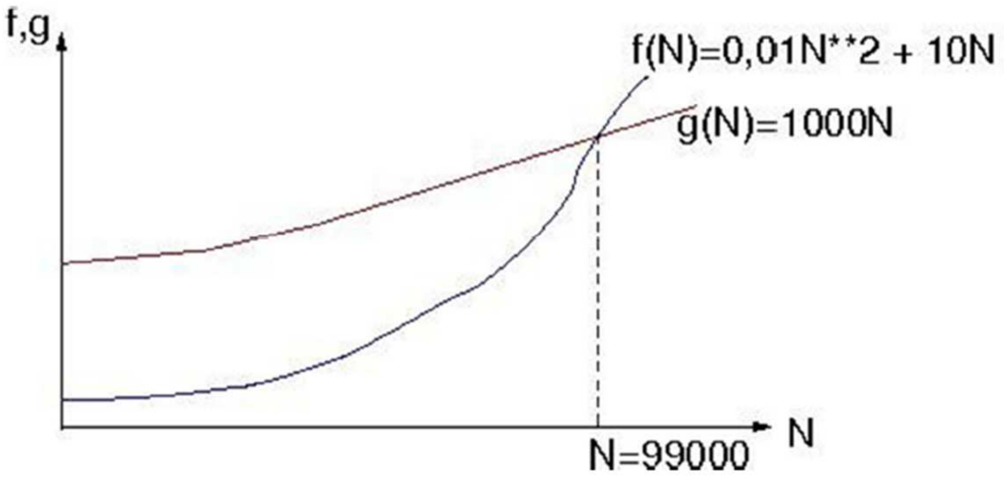
Análise do tempo de processamento

- Exemplo 2:

Qual dos algoritmos é o mais rápido?

– f(N) = 0,01N2 + 10N

– g(N) = 1000N + 10



Análise do tempo de processamento

- Para valores suficientemente pequenos de n, qualquer algoritmo custa pouco (até os ineficientes).

- Por isso, a análise é feita para grandes valores de n.
  - Estudamos o **comportamento assintótico** das funções de complexidade (comportamento para grandes valores de n).

eterminar se um
número é par.

Busca binária

usca sequencial

Ordens de complexidade mais comuns

➔ Os Algoritmos têm tempo de execução proporcional a

- ***1*** : muitas instruções são executadas uma só vez ou poucas vezes isto acontecer para todo o programa diz-se que o seu tempo execução é constante)
- ***log n :*** tempo de execução é logarítmico (cresce ligeiramente medida que ***n*** cresce; quando ***n*** duplica ***log n*** aumenta mas m pouco; apenas duplica quando ***n*** aumenta para n^2)
- ***n :*** tempo de execução é linear (típico quando algum processament feito para cada dado de entrada; situação óptima quando é necess processar ***n*** dados de entrada, ou produzir ***n*** dados na saída)

nação eficiente

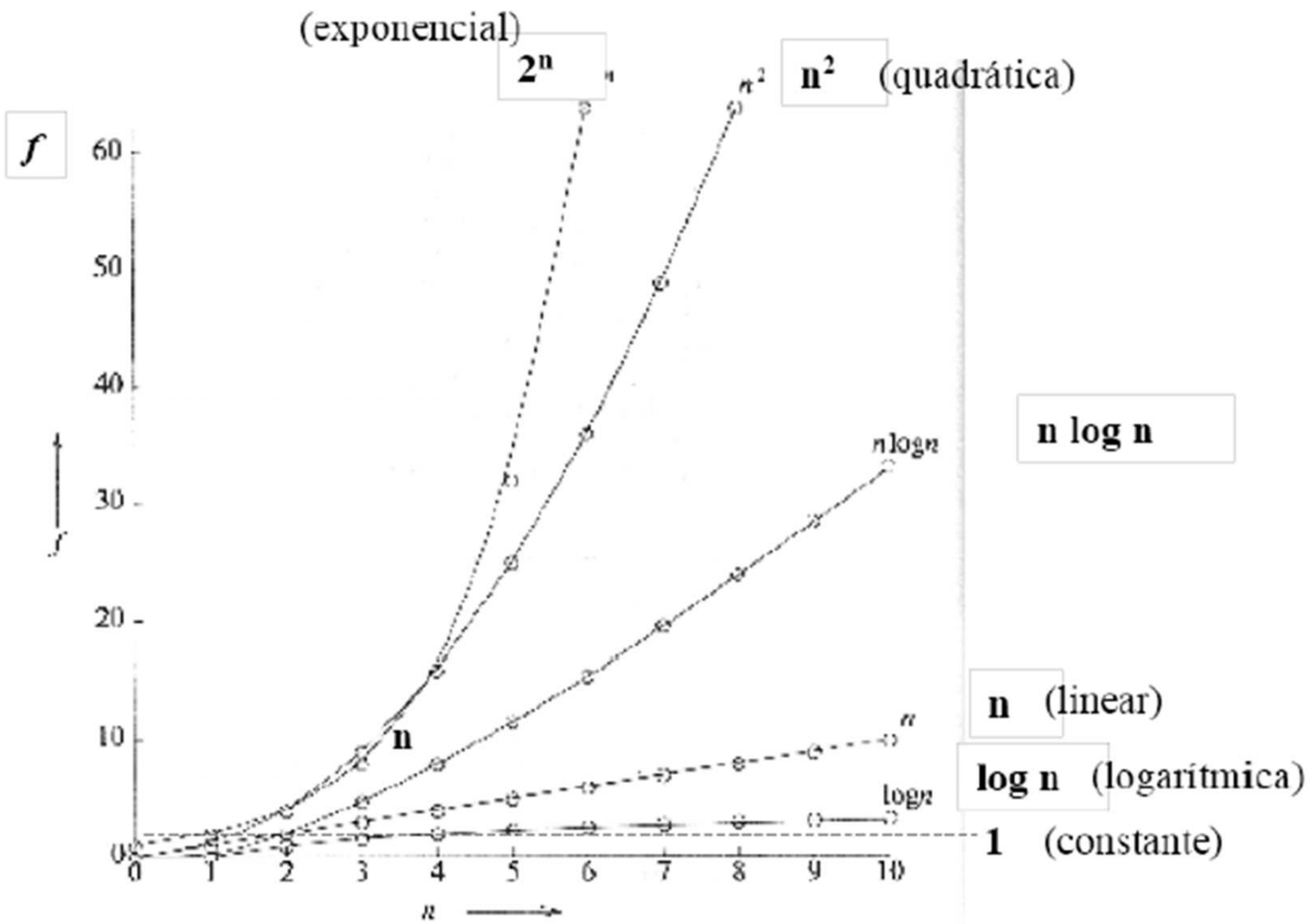
primir matriz

ultiplicação de
matrizes

Ordens de complexidade mais comuns

- ***n log n*** : típico quando se reduz um problema em subproblemas resolve estes separadamente e se combinam as soluções (se ***n*** é i a 1 milhão, ***n log n*** é perto de 20 milhões)
- ***n^2* :** tempo de execução quadrático (típico quando é necess processar todos os pares de dados de entrada; prático apenas pequenos problemas, ex: produto de vectores)
- ***n^3* :** tempo de execução cúbico (para ***n*** = 100, ***n^3*** = 1 mill ex: produto de matrizes)
- ***2^n* :** tempo de execução exponencial (provavelmente de po aplicação prática; típico em soluções de força bruta; para ***n*** = ***2^n*** = 1 milhão; se ***n*** duplica, o tempo passa a ser o quadrado)

Ordens de complexidade mais comuns



FUNÇÃO DE COMPLEXIDADE	n (tamanho do problema)		
	20	**40**	**60**
n	0.0002 s	0.0004 s	0.0006 s
$n \log_2 n$	0.0009 s	0.0021 s	0.0035 s
n^2	0.0040 s	0.0160 s	0.0360 s
n^3	0.0800 s	0.6400 s	2.1600 s
2^n	10.0000 s	27 dias	3660 séculos
3^n	580 minutos	38550 séculos	$1.3*10^{14}$ séculos

“O grande” ou “Big O”

- Na prática, é difícil (senão impossível) prever com rigor o tempo de execução de um algoritmo.
- Assim, identificam-se as **operações dominantes** (mais frequentes ou muito mais demoradas) e determina-se o número de vezes que são executadas.
- Exprime-se o resultado com a notação de “O grande” (“Big O”)
- Os termos de baixa ordem são desconsiderados e as constantes também.

“O grande” ou “Big O”

- Exemplos:
  - $c_k n^k + c_{k-1} n^{k-1} + \ldots + c_0 = O(n^k)$

    (c_i - constantes)
  - $log_2 n = O(log\ n)$

    (não se indica a base porque mudar de base é multiplicar por uma constante)
  - $4 = O(1)$

    (usa-se 1 para ordem constante)

Metodologia para determinar a complexidade

- ➔ Considere-se o seguinte código:

```
for (i = 0; i < n; i++)
{
   Instruções;
}
```

- ➔ A contabilização do número de instruções é simples:

  n iterações e, em cada uma, são executadas um número constante de instruções $\Rightarrow$ $\boldsymbol{O(n)}$

Metodologia para determinar a complexidade

➔ Considere-se o seguinte código:

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
    {
        Instruções;
    }
```

➔ A contabilização do número de instruções é ainda simples:

o ciclo interno (for j) é $\boldsymbol{O(n)}$ e é executado $\boldsymbol{n}$ vezes $\Rightarrow$ $\boldsymbol{O(n^2)}$

Exercícios

- Qual o melhor caso de uma busca sequencial em um vetor? Qual a complexidade?

- Qual o pior caso? Qual a complexidade?

Exercícios

- Qual o melhor caso de uma busca sequencial em um vetor? Qual a complexidade?
  - O melhor caso é quando o elemento buscado está na primeira posição. Assim não é preciso percorrer o vetor. Complexidade constante O(1).

- Qual o pior caso? Qual a complexidade?
  - O pior caso é quando o elemento buscado não existe no vetor, que terá que ser todo percorrido. Complexidade O(n).

Exercícios

- Quais as complexidades dos seguintes algoritmos?

1. $f(n) = n^2 + 1$
2. $f(n) = 517$
3. $f(n)=k2^n$
4. $f(n) = 2n + 10$
5. $f(n) = 2^{100}$
6. $f(n) = 20n^3 + 10n \log n + 5$

Exercícios

- Quais as complexidades dos seguintes algoritmos?

1.	$f(n) = n^2 + 1$	$O(n^2)$
2.	$f(n) = 517$	$O(1)$
3.	$f(n)=k2^n$	$O(2^n)$
4.	$f(n) = 2n + 10$	$O(n)$
5.	$f(n) = 2^{100}$	$O(1)$
6.	$f(n) = 20n^3 + 10n \log n + 5$	$O(n^3)$

Ordene-os por ordem crescente de eficiên

$O(1)$, $O(n)$, $O(n^2)$, $O(n^3)$, $O(2^n)$

Exercícios

- Considere dois programas A e B com tempos de execução $100n^2$ e $5n^3$, respectivamente. Qual é o mais eficiente ?

Exercícios

- Considere dois programas A e B com tempos de execução $100n^2$ e $5n^3$, respectivamente. Qual é o mais eficiente ?

  - Se considerarmos um conjunto de dados de tamanho $n < 20$, o programa B será mais eficiente que o programa A.

  - Entretanto, se o conjunto de dados for grande, a diferença entre os dois programas se tornará bastante significativa e o programa A se tornará melhor.

  - Se considerarmos o **comportamento assintótico** de ambos, o programa A é mais eficiente, já que possui complexidade quadrática ($O(n^2)$).

Exercícios

- O que o código abaixo faz? Qual sua complexidade?

```
int soma_acumulada(int n) {
    int i;
    int acumulador = 0;
    for(i = 0; i < n; i++) {
        acumulador += i;
    }
    return acumulador;
}
```

Exercícios

- O que o código abaixo faz? Qual sua complexidade?

```
-    int soma_acumulada(int n) {
-        int i;
1        int acumulador = 0;
n        for(i = 0; i < n; i++) {
n            acumulador += i;
-        }
1        return acumulador;
-    }
```

O código calcula o somatório de 0 a n-1. Sua complexidade é O(n).

Referências

- Macêdo, Autran. Notas de Aula – Teoria sobre Algoritmos – Complexidade, UFU.
- Adriana, Maria. Notas de Aula – ICC, FACOM, UFU.
- Cunha, Ítalo, Notas de aula – Análise de complexidade – UFMG
- Teoria da Computacao - Tiaraju Asmuz Diverio & Paulo Blauth Menezes - Cap.3 (3.4) e Cap.5