UDESC

Complexidade de Algoritmos

Prof. Rafael Alceste Berri
rafaelberri@gmail.com

Prof. Diego Buchinger
diego.buchinger@outlook.com

Prof. Cristiano Damiani Vasconcellos
cristiano.vasconcellos@udesc.br

UDESC

Análise de Algoritmos

Analisar um algoritmo significa **prever os recursos** que algoritmo necessita. Por exemplo, **memória**, **largura de banda** e mais frequentemente o **tempo** de computação.

Para analisar um algoritmo é necessário definir um **modelo de computação**. O modelo de computação do computador tradicional é o RAM (*Random Access Machine*) onde as instruções são executadas em sequência, sem concorrência, e os dados são armazenados em células de memória com **acesso aleatório**.

UDESC

Análise de Algoritmos

- Fazer uma média do tempo de execução?
- Contar o número de todas as instruções que são executadas:

  Por exemplo: m **load**, n **store**, o **add**, p **sub**, q **div**, r **mul**, s **call**, t **ret**, u **cmp**, v **jump**, etc.

O tempo de execução depende do processador, compilador, velocidade de acesso à memória, tamanho de memória (cache e ram) etc.

UDESC

Qual o tempo de execução?

```
int pesquisa(Estrutura *v, int n, int chave){
	int i;
	for (i = 0; i < n; i++)
		if (v[i].chave == chave)
			return i;
	return -1;
}
```

UDESC

Qual o tempo de execução?

Comparação de desempenho na resolução de **sistemas lineares** considerando tempos de operações de um computador real:

n	Método de Cramer	Método de Gauss
2	22 µs	50 µs
3	100 µs	159 µs
4	463 µs	353 µs
5	2,15 ms	666 µs
10	4,62 s	4,95 ms
20	247 dias	38,63 ms
40	$1,45 * 10^{13}$ anos	0,315 s

Qual o tempo de execução?

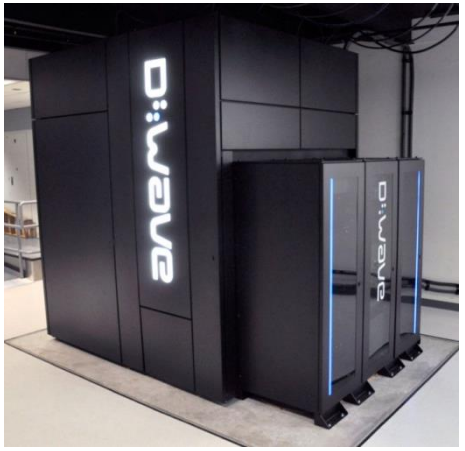
- Ok, mas e o avanço tecnológico, produzindo máquinas cada vez mais rápidas não faz o estudo de complexidade perder importância?

Computador 19xx

100x mais rápido

Computador 2016

2^b mais rápido



Computador quântico

Qual o tempo de execução?

- Análise de impacto do aumento de velocidade dos computadores para o Método de Cramer:

n	Computador 19xx	Computador 2016
3	100 µs	1 µs
5	2,15 ms	21,5 µs
7	46,274 ms	463 µs
10	4,62 s	46,2 ms
12	1,66 min	1 s
15	2,76 horas	1,656 min
20	247 dias	2,47 dias
40	$1,45 * 10^{13}$ anos	$1,45 * 10^{11}$ anos

UDESC

Análise de Algoritmos

A área de Complexidade de Algoritmos tenta **prever os recursos** de que o algoritmo necessitará

A **complexidade** vem ganhando destaque a ponto de que alguns autores dizem que este tema é o **coração da Computação** [Toscani e Veloso, 2001].

- Complexidade na fase de **projeto do algoritmo**
- Intratabilidade de problemas:
  - Problemas **NP-Completos** e **NP-Difícil**
  - Soluções alternativas (**aproximações**), ou uso de programação dinâmica.

Atividade 1

- Elabore o melhor algoritmo para receber uma sequencia de ‘*n*’ números inteiros. Depois o algoritmo deve receber um número ‘*m*’ e deve trazer como saída o número de vezes que o valor ‘*m*’ apareceu nesta sequência.

- Considere n < 1.000.000
- NOTA: existe alguma consideração diferente caso ‘*m*’ seja um inteiro entre 0 e 10.000, ou um inteiro entre 0 e 1.000.000.000.000?
- OBS: e se o valor de ‘m’ fosse informado antes da sequência de ‘n’ números?

Use entrada em CAL-01-atividade-entrada.zip
Envie rafaelberri@gmail.com A: “TC-CAL01”

UDESC

Conceitos Básicos de Complexidade

Medidas de Complexidade

- Como calcular a quantidade de trabalho requerido por um algoritmo, ou seja, sua complexidade?

```
int pesquisa(Estrutura *v, int n, int chave){
      int i;
      for (i = 0; i < n; i++)
            if (v[i].chave == chave)
                  return i;
      return -1;
}
```

Medidas de Complexidade – tam. entrada & val. de entrada

```
int pesquisa(Estrutura *v, int n, int chave){
	int i;
	for (i = 0; i < n; i++)
		if (v[i].chave == chave)
			return i;
	return -1;
}
```

Medidas de Complexidade

- Como calcular a quantidade de trabalho requerido por um algoritmo, ou seja, sua complexidade?
  - Depende do **tamanho da entrada**;
  - Depende dos **valores da entrada**;

Ex: ordenação de uma lista de '*n*' elementos:

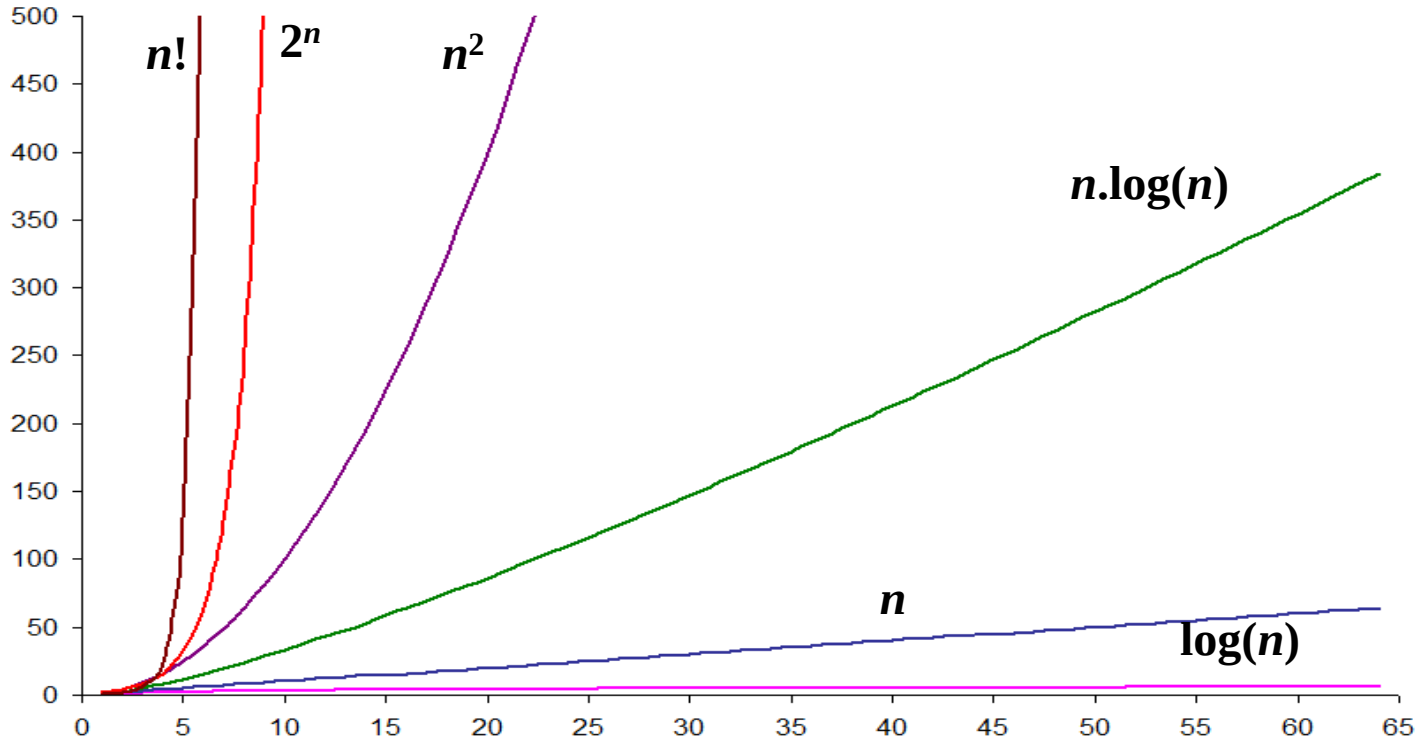
lista com elementos já ordenados

vs

lista com elementos totalmente desordenados

Medidas de Complexidade

CONSIDERAÇÃO I: trabalhar com valores grandes para ‘n’ (entrada). Assim, ordens de crescimento são destacadas.



Funções de Complexidade

Considere que cada operação leva 1ns em média em um determinado processador. Determine o tempo das funções abaixo para os seguintes valores de operações:

$f(n) / n$	$n=10$	$n=100$	$n=1.000$	$n=10.000$	$n=100.000$	$n=1.000.000$
$log_2 n$						
n						
$3n$						
$n \, log_2 n$						
n^2						
2^n						
$n!$						

Funções de Complexidade

Considere que cada operação leva 1ns em média em um determinado processador. Determine o tempo das funções abaixo para os seguintes valores de operações:

$f(n) / n$	*n=10*	*n=100*	*n=1.000*	*n=10.000*	*n=100.000*	*n=1.000.000*
$log_2 n$	3,321928	6,643856	9,965784	13,28771	16,60964	19,93157
n	10	100	1000	10000	100000	1000000
$3n$	30	300	3000	30000	300000	3000000
$n\ log_2 n$	33,21928	664,3856	9965,784	132877,1	1660964	19931569
n^2	100	10000	1000000	1E+08	1E+10	1E+12
2^n	1024	1,27E+30	1,1E+301	#NÚM!	#NÚM!	#NÚM!
$n!$	3628800	9,3E+157	#NÚM!	#NÚM!	#NÚM!	#NÚM!

Complexidade de um Algoritmo

- Quantidade de trabalho = Medir quantidade de vezes que a **operação fundamental** é executada. (as vezes mais de uma operação).
- Qual poderia ser uma operação fundamental deste algoritmo?

```
int pesquisa(Estrutura *v, int n, int chave){
      int i;
      for (i = 0; i < n; i++)
            if (v[i].chave == chave) //
                  return i;
      return -1;
}
```

Complexidade de um Algoritmo

- Função de execuções:

exec(a, d) := sequência de execuções de operações fundamentais efetuadas na execução do algoritmo **a**, com entrada **d**.

```
int pesquisa(Estrutura *v, int n, int chave){
      int i;
      for (i = 0; i < n; i++)
            if (v[i].chave == chave) //
                  return i;
      return -1;
}
```

Complexidade de um Algoritmo

- Função de custo:

custo(s) := comprimento da sequência **s**, definido conforme o peso estabelecido para as operações fundamentais.

```
int pesquisa(Estrutura *v, int n, int chave){
      int i;
      for (i = 0; i < n; i++)
            if (v[i].chave == chave) //
                  return i;
      return -1;
}
```

Complexidade de um Algoritmo

- Função de desempenho:

desemp(d) := custo(exec(a,d))

```
int pesquisa(Estrutura *v, int n, int chave){
      int i;
      for (i = 0; i < n; i++)
            if (v[i].chave == chave) //
                  return i;
      return -1;
}
```

Complexidade de um Algoritmo

- Ideia interessante, por que não avaliar em função do tamanho da entrada **d**?
  - tam(d) = n (não reversível)
  - Assim: aval(n) = desemp(d)
  - aval(n) envolve um conjunto de todos os desempenhos para entradas de tamanho n.
    - Critérios interessantes para avaliação de desempenho em tamanho n:
      - Complexidade máxima (complexidade pessimista ou pior caso);
      - Valor médio (complexidade média ou caso médio)