Projeto e Análise de Algoritmos

Aula 01 – Complexidade de Algoritmos

Edirlei Soares de Lima
<edirlei@iprj.uerj.br>

O que é um algoritmo?

- Um conjunto de **instruções** executáveis para resolver um **problema** (são as ideias por detrás dos programas)
  - O **problema** é a motivação para o algoritmo.
  - Geralmente existem **vários algoritmos** para um mesmo problema.
    - Como escolher?
  - Algoritmos são **independentes** da linguagem de programação, da máquina, da plataforma, etc.
  - Algoritmos são representados através da **descrição** das instruções de forma suficiente para que a audiência os entenda.

O que é um algoritmo?

- Um problema é caracterizado pela descrição de uma **entrada** e **saída**.

- **Exemplo:** problema de ordenação

Entrada: uma sequência $\{a_1; a_2; \ldots ; a_n\}$ de n números
Saída: uma permutação dos números $\{a'_1; a'_2; \ldots ; a'_n\}$ tal que $a'_1 \leq a'_2 \leq \ldots \leq a'_n$

Entrada: 6 3 7 9 2 4
Saída: 2 3 4 6 7 9

Propriedades Desejadas em um Algoritmo

- **Eficácia:** deve ser capaz de resolver corretamente todas as instâncias do problema.

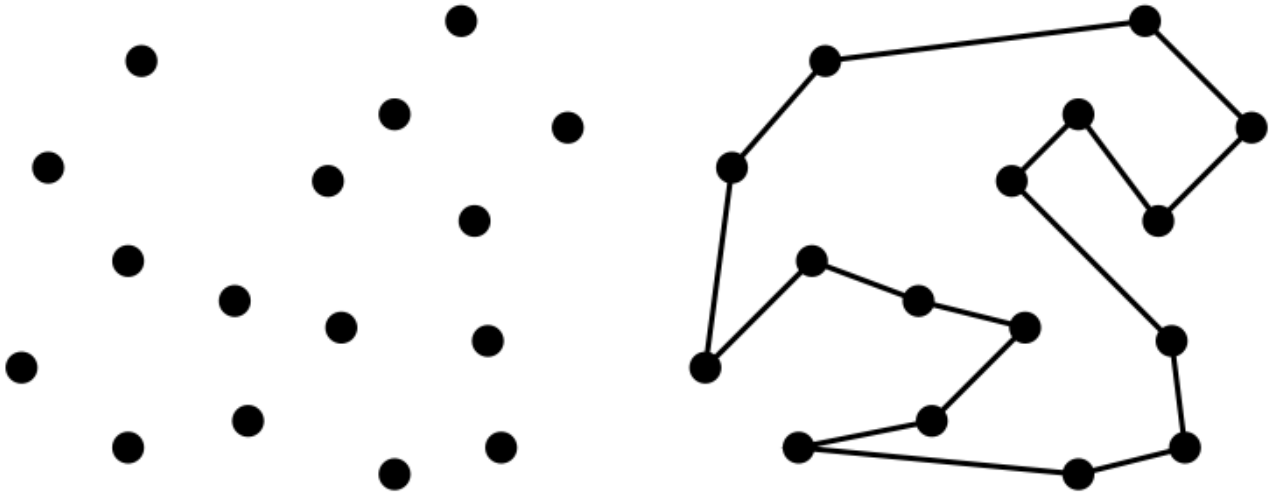
- **Eficiência:** a sua performance (tempo e memória) tem de ser adequada.

Exemplo: Eficácia de um Algoritmo

- Problema do Caixeiro Viajante:

  Entrada: um conjunto S de n pontos no plano
  Saída: o ciclo mais curto que visita todos os pontos de S

- Exemplo:



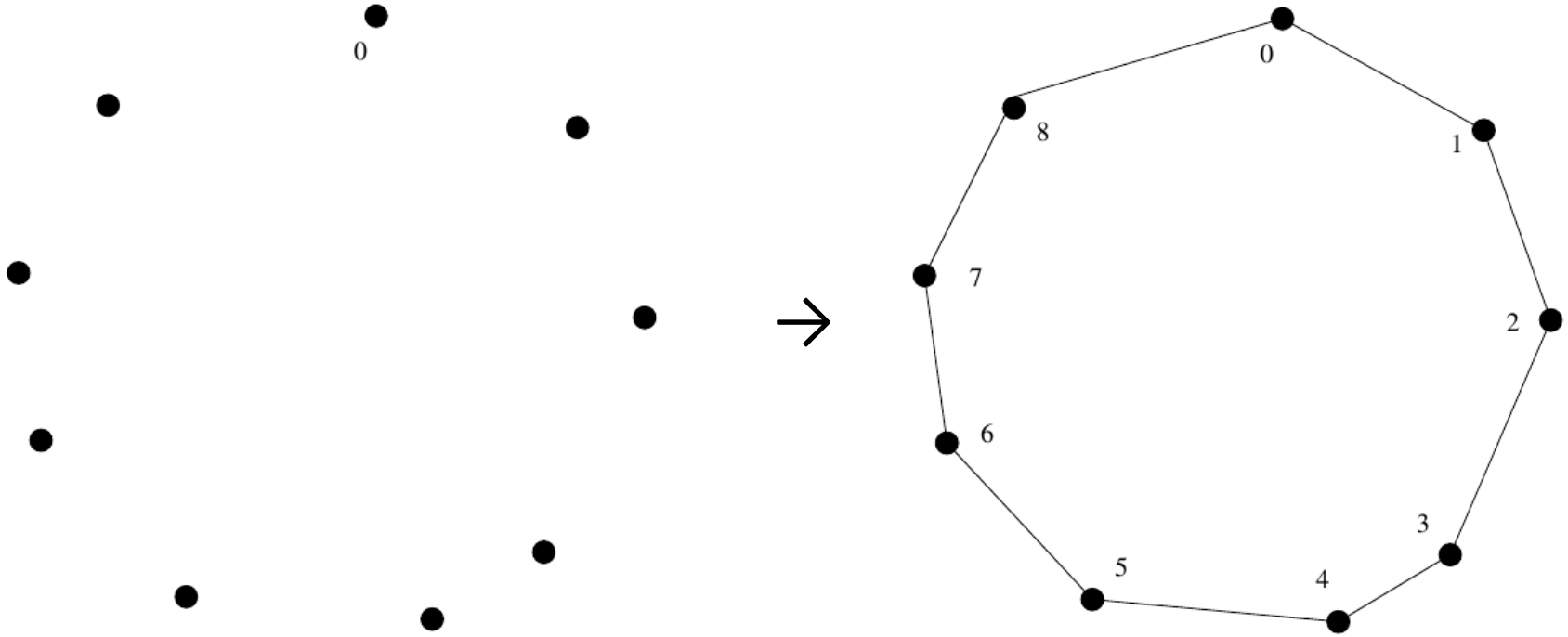
Exemplo: Eficácia de um Algoritmo

- Problema do Caixeiro Viajante
  - Um primeiro possível algoritmo:

1. $p_1 \leftarrow$ ponto inicial escolhido aleatoriamente
2. $i \leftarrow 1$
3. enquanto (existirem pontos por visitar) fazer
4. $\quad i \leftarrow i + 1$
5. $\quad p_i \leftarrow$ vizinho não visitado mais próximo de p_i-1
6. retorna caminho $p_1 \rightarrow p_2 \rightarrow \ldots \rightarrow p_n \rightarrow p_1$

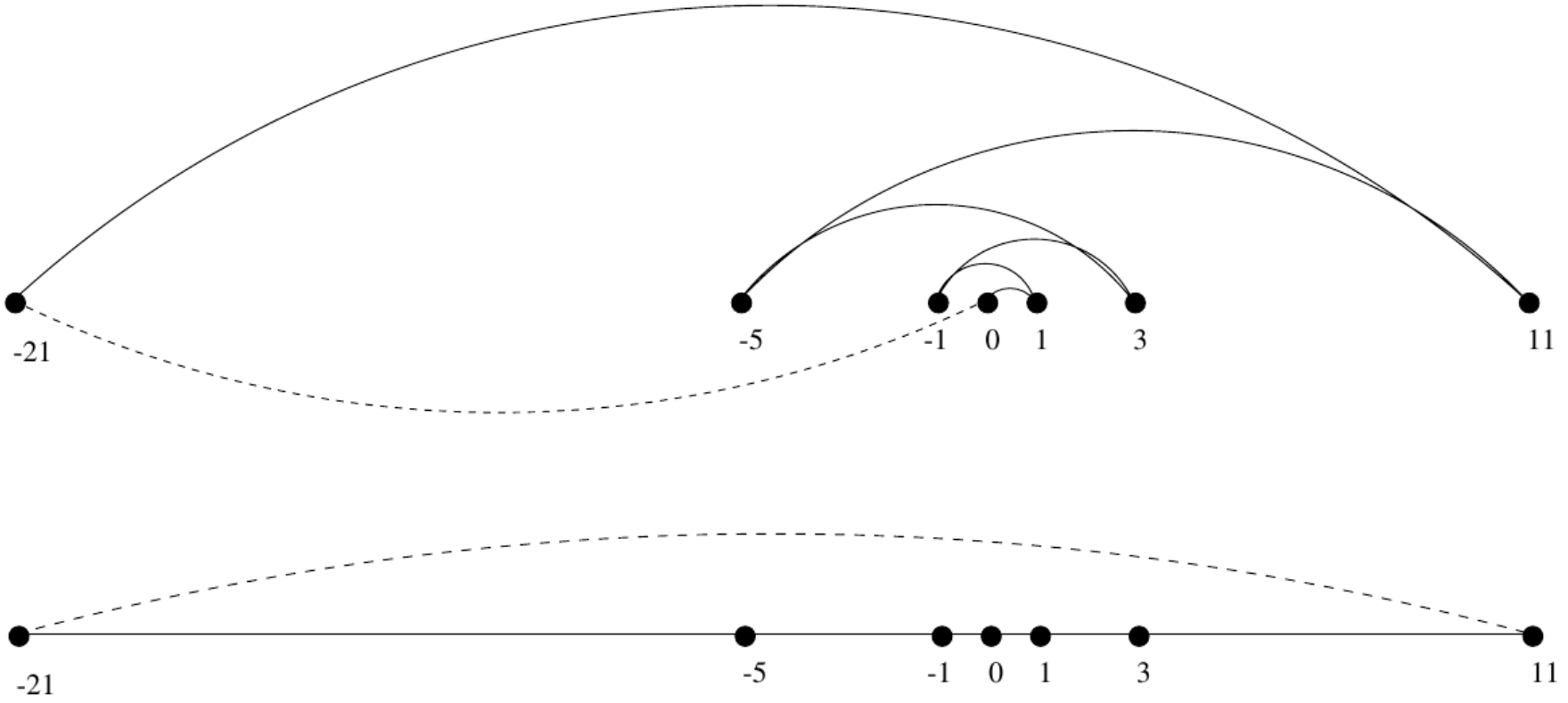
Exemplo: Eficácia de um Algoritmo

- Problema do Caixeiro Viajante
  - Parece funcionar…



Exemplo: Eficácia de um Algoritmo

- Problema do Caixeiro Viajante
  - Mas não funciona para todas as instâncias do problema!



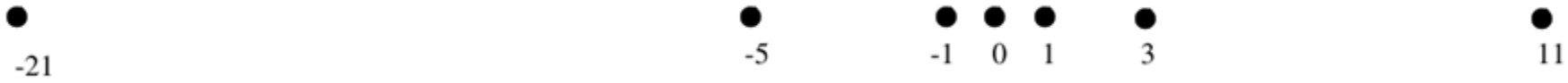
Exemplo: Eficácia de um Algoritmo

- Problema do Caixeiro Viajante
  - Um segundo possível algoritmo:

```
1. Para i ← 1 até (n - 1) fazer
2.   Adiciona ligação ao par de pontos mais próximo tal
     que os pontos estão em componentes conexas (cadeias
     de pontos) diferentes
3. Adiciona ligação entre dois pontos dos extremos da
   cadeia ligada
4. retorna o ciclo que formou com os pontos
```

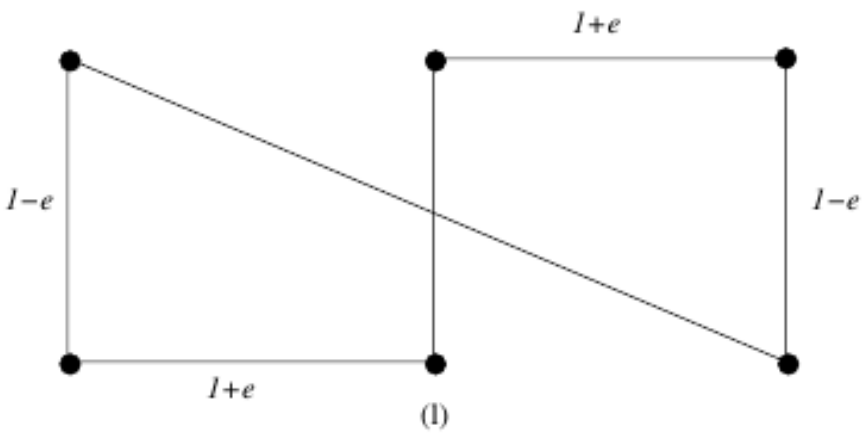
Exemplo: Eficácia de um Algoritmo

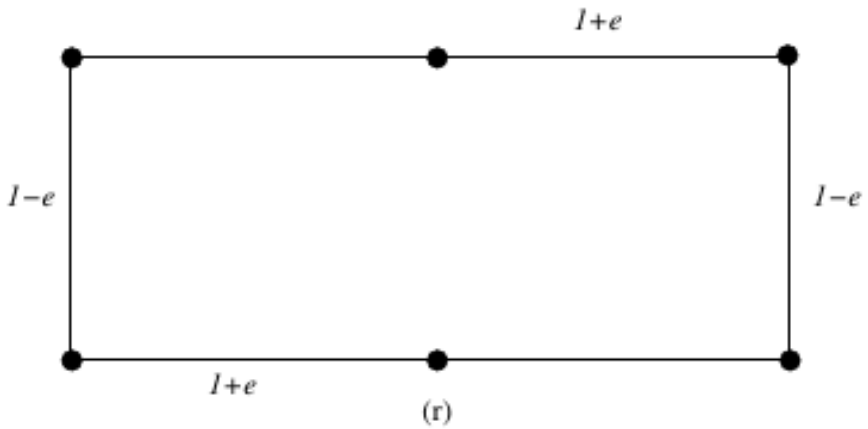
- Problema do Caixeiro Viajante
  - Parece funcionar…



Exemplo: Eficácia de um Algoritmo

- Problema do Caixeiro Viajante
  - Mas não funciona para todas as instâncias do problema!





Exemplo: Eficácia de um Algoritmo

- Problema do Caixeiro Viajante
  - Um terceiro possível algoritmo (força bruta):

```
1. P_min ← uma qualquer permutação dos pontos de S
2. Para P_i ← cada uma das permutações de pontos de S
3.    Se (custo(P_i) < custo(P_min)) Então
4.       P_min ← P_i
5. retorna caminho formado por P_min
```

  - O algoritmo é eficaz/correto, mas extremamente lento!
    - P(n) = n! = n × (n - 1) × ... × 1 (n fatorial)
    - Exemplo: P(20) = 2,432,902,008,176,640,000

Exemplo: Eficácia de um Algoritmo

- O problema apresentado é uma versão restrita (euclidiana) de um dos problemas mais "clássicos": **Travelling Salesman Problem** (TSP).

- Este problema tem **inúmeras aplicações** (mesmo na forma "pura")
  - Exemplos: produção industrial, rotas de veículos, análise genômica...

- **Não é conhecida nenhuma solução eficiente para este problema** (que gere resultados ótimos, e não apenas "aproximados")

- A solução apresentada tem complexidade temporal: **O(n!)**

- O TSP pertence a classe dos problemas **NP-hard**
  - A versão de decisão pertence à classes dos problemas **NP-completos**

Eficiência de um Algoritmo

- Quantas operações simples um computador pode realizar por segundo? (aproximação em ordem de grandeza)
- Assumindo que um computador realizar 1 milhão de operações simples por segundo, quanto tempo demorariam as seguintes quantidades de instruções?

Quantidade	**100**	**1,000**	**10,000**	**100,000**	**1,000,000**
N	< 1 seg	< 1 seg	< 1 seg	< 1 seg	1 seg
N^2	< 1 seg	1 seg	2 min	3 horas	12 dias
N^3	1 seg	18 min	12 dias	32 anos	31,710 anos
2^N	10^{17} anos	muito tempo	muito tempo	muito tempo	muito tempo
N!	muito tempo	muito tempo	muito tempo	muito tempo	muito tempo

muito tempo > 10^{25} anos

Eficiência de um Algoritmo

- Voltando as permutações do algoritmo para o problema do caixeiro viajante:
  - $P(n) = n! = n \times (n - 1) \times ... \times 1$

Exemplo de 6 permutações de {1, 2, 3}:

1 2 3

1 3 2

2 1 3

2 3 1

3 1 2

3 2 1

Eficiência de um Algoritmo

- Quanto tempo demora um programa que passa por todas as permutações de n números? (tempos aproximados considerando cerca de 10^7 permutações por segundo)

  n = 8: 0,003s

  n = 9: 0,026s

  n = 10: 0,236s

  n = 11: 2,655s

  n = 12: 33,923s

  …

  n = 20: 5000 anos !

Eficiência de um Algoritmo

- Um computador mais rápido adiantaria alguma coisa?

- Se n = 20 = 5000 anos, hipoteticamente:
  - 10x mais rápido ainda demoraria 500 anos;
  - 5,000x mais rápido ainda demoraria 1 ano;
  - 1,000,000x mais rápido demoraria quase dois dias mas:
    - n = 21 já demoraria mais de um mês;
    - n = 22 já demoraria mais de dois anos!

- A taxa de crescimento do algoritmo é muito importante!

Eficiência de um Algoritmo

- Como conseguir prever o tempo que um algoritmo demora?

- Como conseguir comparar dois algoritmos diferentes?

Random Access Machine (RAM)

- Precisamos de um modelo que seja genérico e independente da máquina/linguagem usada.

- Vamos considerar uma Random Access Machine (RAM)
  - Cada operação simples (+, -, / , *) demora 1 passo;
  - Cada acesso á memória custa também 1 passo;
  - Tempo de execução: número de passos executados em relação ao tamanho da entrada (T(n)).

- As operações são simplificadas, mas mesmo assim isto é útil (somar dois inteiros não custa o mesmo que dividir dois reais, mas veremos que esses valores não são importantes em uma visão global)

Exemplo de Contagem de Operações

- Um programa simples:

```
int count = 0;
int i;
for (i=0; i<n; i++)
  if (v[i] == 0)
    count++;
```

- Número de operações simples:

Declarações de variáveis	2
Atribuições	2
Comparação “menor que”	n + 1
Comparação “igual a”	n
Acesso ao vetor	n
Incremento	Entre n e 2n (dependendo dos zeros)

Exemplo de Contagem de Operações

- Um programa simples:

```
int count = 0;
int i;
for (i=0; i<n; i++)
  if (v[i] == 0)
    count++;
```

- Total de operações no **pior caso**:

  $T(n) = 2 + 2 + (n + 1) + n + n + 2n = 5 + 5n$

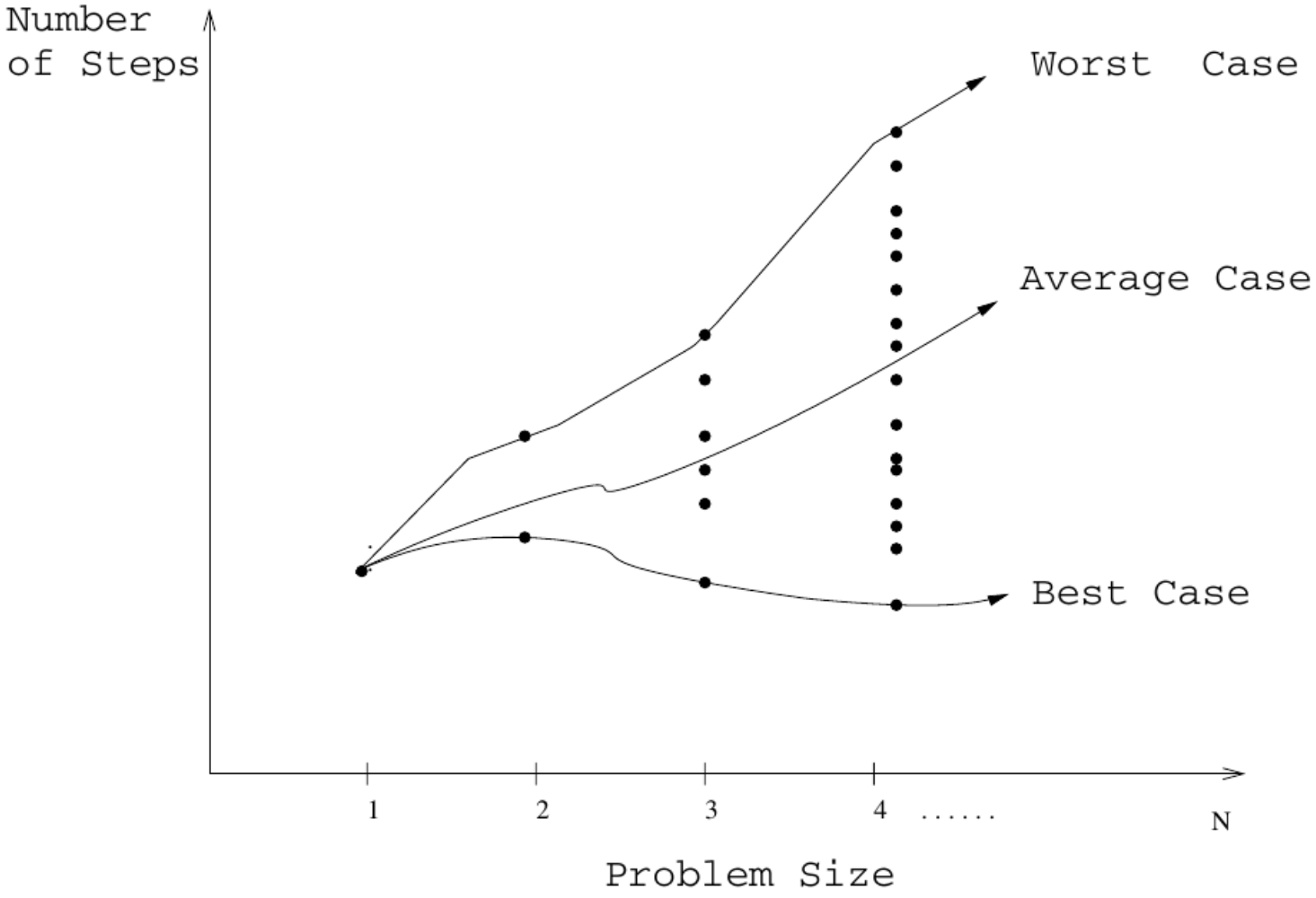
- Total de operações no **melhor caso**:

  $T(n) = 2 + 2 + (n + 1) + n + n + n = 5 + 4n$

Tipos de Análises de um Algoritmo

- **Análise do Pior Caso:**
  - T(n) = tempo máximo do algoritmo para uma entrada qualquer de tamanho n.
- **Análise Caso Médio:**
  - T(n) = tempo médio do algoritmo para todos as entradas de tamanho n.
    - Implica conhecer a distribuição estatística das entradas.
- **Análise do Melhor Caso:**
  - T(n) = avaliação ingênua de um algoritmo que é rápido para algumas entradas.

Tipos de Análises de um Algoritmo



Análise Assintótica

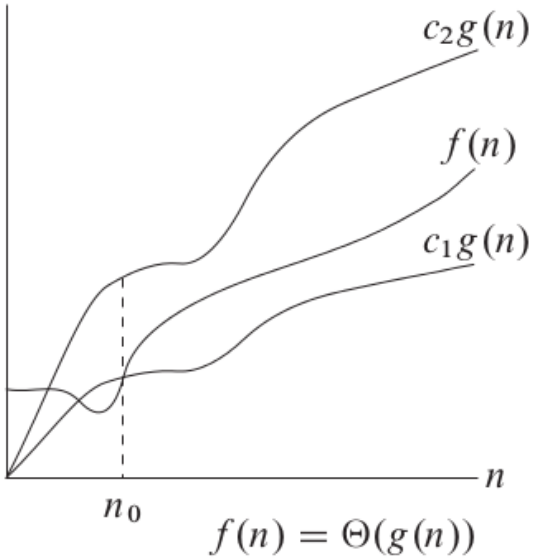
- Precisamos de ferramenta matemática para comparar **funções**
- Na análise de algoritmos usa-se a **Análise Assintótica**
  - Estudo do comportamento de algoritmos para entradas arbitrariamente grandes ou a "descrição" da taxa de crescimento.
- Usa-se uma notação especifica: O, Ω, Θ (*big O*, *omega*, *theta*)
- Permite "simplificar" expressões como a mostrada anteriormente focando apenas nas ordens de grandeza.

Análise Assintótica – Notação

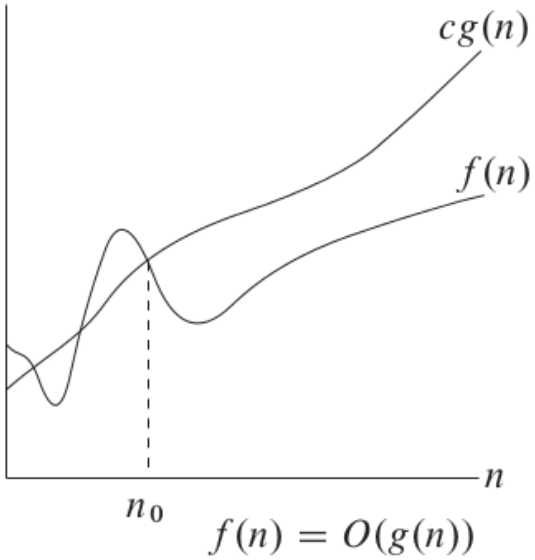
- **f(n) = O(g(n))**
  - Significa que C × g(n) é um limite superior de f(n)
- **f(n) = Ω(g(n))**
  - Significa que C × g(n) é um limite inferior de f(n)
- **f(n) = Θ(g(n))**
  - Significa que C1 × g(n) é um limite inferior de f(n) e C2 × g(n) é um limite superior de f(n)

Análise Assintótica – Notação

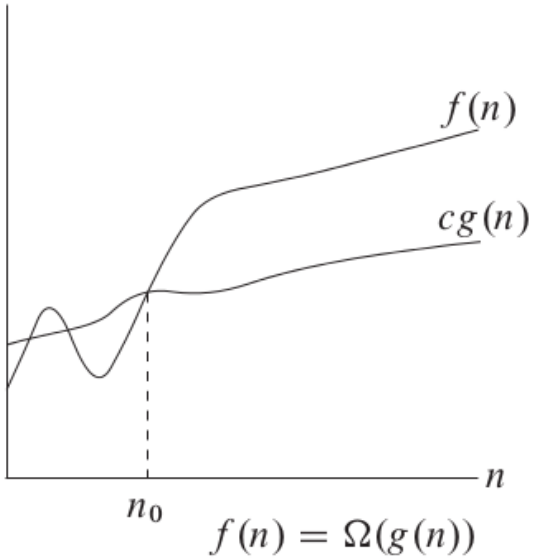
Θ



O



Ω



Análise Assintótica – Formalização

- **f(n) = O(g(n))** se existem constantes positivas n_0 e c tal que $f(n) \leq c \times g(n)$ para todo o $n \geq n_0$

- $f(n) = 3n^2 - 100n + 6$
  - $f(n) = O(n^2)$
  - $f(n) = O(n^3)$
  - $f(n) \neq O(n)$

Análise Assintótica – Formalização

- **$f(n) = \Omega(g(n))$** se existem constantes positivas n_0 e c tal que $f(n) \geq c \times g(n)$ para todo o $n \geq n_0$

- $f(n) = 3n^2 - 100n + 6$
  - $f(n) = \Omega(n^2)$
  - $f(n) \neq \Omega(n^3)$
  - $f(n) = \Omega(n)$

Análise Assintótica – Formalização

- **$f(n) = \Theta(g(n))$** se existem constantes positivas n_0, c_1 e c_2 tal que $c_1 \times g(n) \leq f(n) \leq c_2 \times g(n)$ para todo o $n \geq n_0$

- $f(n) = 3n^2 - 100n + 6$
  - $f(n) = \Theta(n^2)$
  - $f(n) \neq \Theta(n^3)$
  - $f(n) \neq \Theta(n)$

- $f(n) = \Theta(g(n))$ implica que $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$

Análise Assintótica

- O estudo assintótico nos permite ignorar constantes e termos que não são dominantes.

- Considerando a função $f(n) = 3n^2 + 10n + 50$

n	$3n^2 + 10n + 50$	$3n^2$
64	12978	12288
128	50482	49152
512	791602	786432
1024	3156018	3145728
2048	12603442	12582912
4096	50372658	50331648
8192	201408562	201326592
16384	805470258	805306368
32768	3221553202	3221225472

Análise Assintótica – Regras Praticas

- **Multiplicação por uma constante** não altera o comportamento:
  - $\Theta(c \times f(n)) = \Theta(f(n))$
  - $99 \times n^2 = \Theta(n^2)$

- Em um polinômio $a_x n^x + a_{x-1} n^{x-1} + \ldots + a_2 n^2 + a_1 n + a_0$ podemos nos focar na parcela com o **maior expoente**:
  - $3\mathbf{n^3} - 5n^2 + 100 = \Theta(n^3)$
  - $6\mathbf{n^4} - 20^2 = \Theta(n^4)$
  - $0.8\mathbf{n} + 224 = \Theta(n)$

- Em uma soma/subtração podemos nos focar na **parcela dominante**:
  - $\mathbf{2^n} + 6n^3 = \Theta(2^n)$
  - $\mathbf{n!} - 3n^2 = \Theta(n!)$
  - $n \log n + \mathbf{3n^2} = \Theta(n^2)$

Análise Assintótica – Exercício

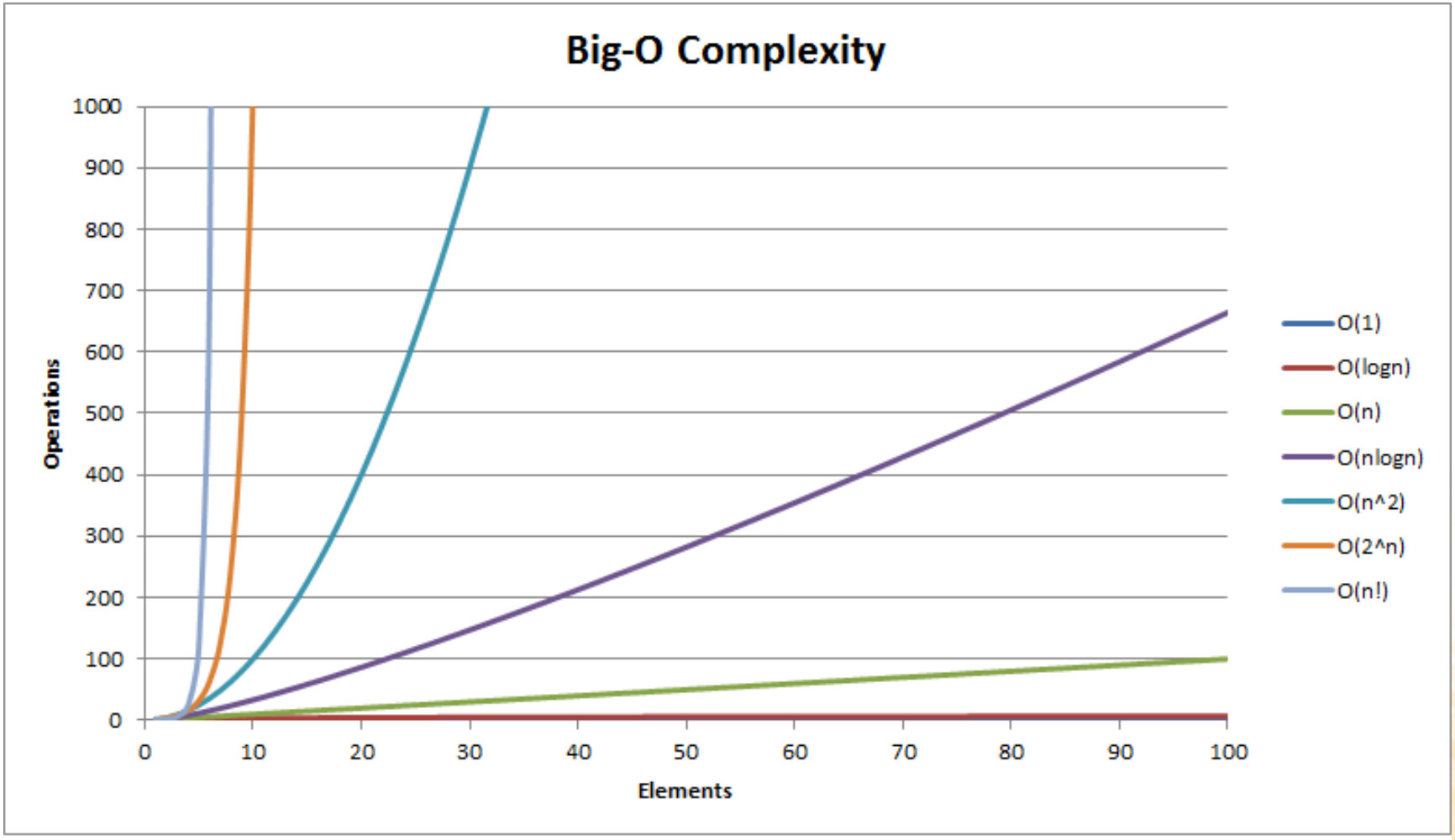
- $T(n) = 32n^2 + 17n + 32$

- Responda se T(n) é
  - $O(n^2)$? Sim
  - $O(n^3)$? Sim
  - $\Omega(n)$? Sim
  - $\Omega(n^2)$? Sim
  - $O(n)$? Não
  - $\Theta(n^2)$? Sim
  - $\Omega(n^3)$? Não
  - $\Theta(n)$? Não
  - $\Theta(n^4)$? Não

Crescimento Assintótico

Função	Nome	Exemplos
1	constante	Somar dois números
log n	logarítmica	Pesquisa binária, inserir um número em uma heap
n	linear	1 ciclo para buscar o valor máximo em um vetor
n log n	linearítimica	Ordenação (merge sort, heap sort)
n^2	quadrática	2 ciclos (bubble sort, selection sort)
n^3	cúbica	3 ciclos (Floyd-Warshall)
2^n	exponencial	Pesquisa exaustiva (subconjuntos)
n!	fatorial	Todas as permutações

Crescimento Assintótico



Análise Assintótica – Exemplos Praticos

- Um programa tem dois pedaços de código A e B, executados um a seguir ao outro, sendo que A corre em O(n log n) e B em $O(n^2)$.
  - O programa corre em $O(n^2)$, porque $n^2 > n \log n$
- Um programa chama n vezes uma função O(log n), e em seguida volta a chamar novamente n vezes outra função O(log n)
  - O programa corre em O(n log n)
- Um programa tem 5 ciclos, chamados sequencialmente, cada um deles com complexidade O(n)
  - O programa corre em O(n)
- Um programa P_1 tem tempo de execução proporcional a $100 \times n \log n$. Um outro programa P_2 tem $2 \times n^2$. Qual é o programa mais eficiente?
  - P_1 é mais eficiente porque $n^2 > n \log n$. No entanto, para um n pequeno, P_2 é mais rápido e pode fazer sentido ter um programa que chama P_1 ou P_2 de acordo com o valor de n.

Exercícios

1) Escreva um algoritmo para verificar se um vetor contêm pelo menos dois valores duplicados (em qualquer lugar do vetor). Em seguida, analise a complexidade do algoritmo proposto.

```
bool hasDuplicate(int *vet, int n){
  int i, j;
  bool duplicate = false;
  for (i = 0; i < n; i++){
    for (j = 0; j < n; j++){
      if (i != j && A[i] == A[j])
        return true;
    }
  }
  return false;
}
```

$O(n^2)$

Exercícios

2) Qual a complexidade do seguinte algoritmo?

```
int i, j, k, x, result = 0;
for (i = 0; i < N; i++){
  for (j = i; j < N; j++){
    for (k = 0; k < M; k++){
      x = 0;
      while (x < N){
        result++;
        x += 3;
      }
    }
    for (k = 0; k < 2 * M; k++)
      if (k % 7 == 4)
        result++;
  }
}
```

n

m

m

n^2

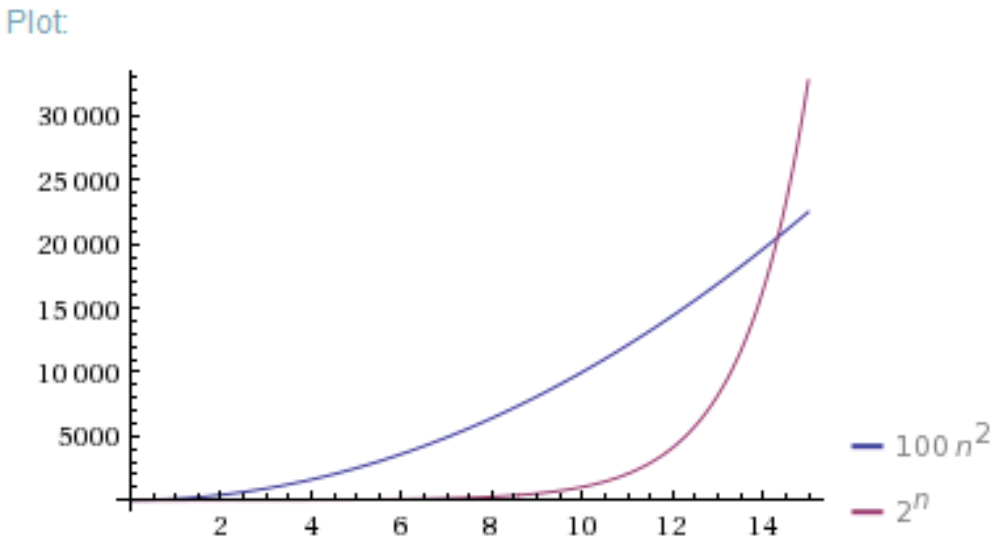
$O(n^2mn) = O(mn^3)$

Exercícios

3) Qual o menor valor de n tal que um algoritmo cujo tempo de execução é $100n^2$ funciona mais rápido que um algoritmo cujo tempo de execução é 2^n na mesma máquina?

Wolfram Alpha:

```
plot 100*n^2 from n=0 to 15, 2^n from n=0 to 15
```



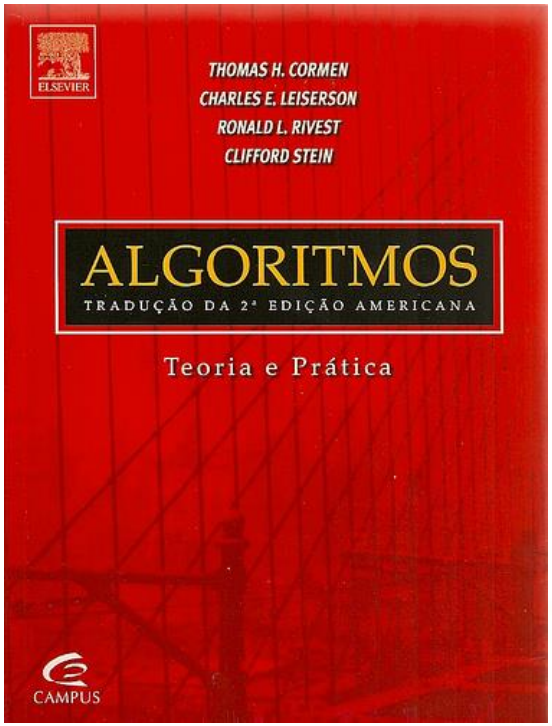
Exercícios

Lista de Exercícios 01 – Complexidade de Algoritmos

http://www.inf.puc-rio.br/~elima/paa/

Leitura Complementar

- Cormen, T., Leiserson, C., Rivest, R., e Stein, C. **Algoritmos – Teoria e Prática** (tradução da 2ª. Edição americana), Editora Campus, 2002.



- **Capítulo 1: A função dos Algoritmos na Computação**
- **Capítulo 2: Conceitos Básicos**
- **Capítulo 3: Crescimento de Funções**