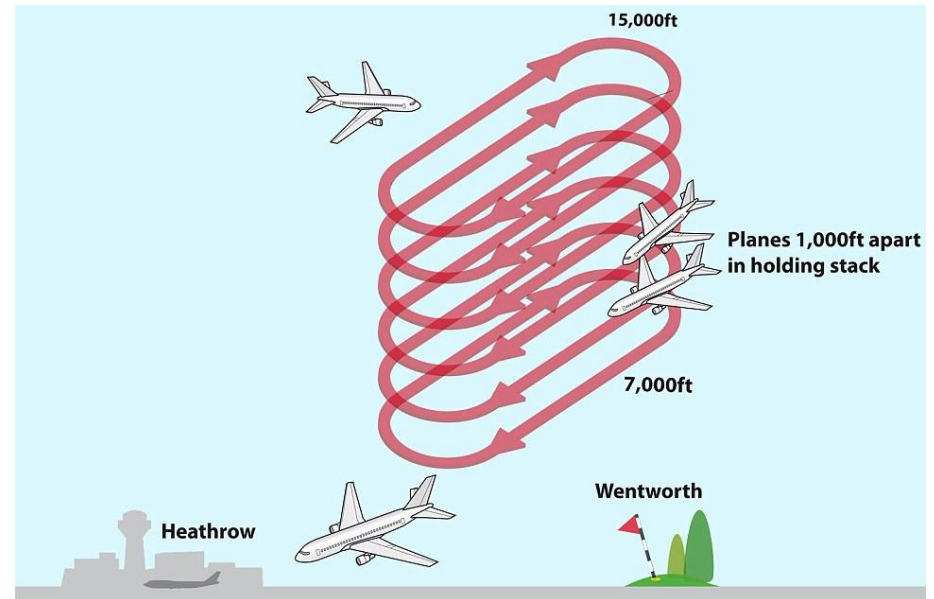


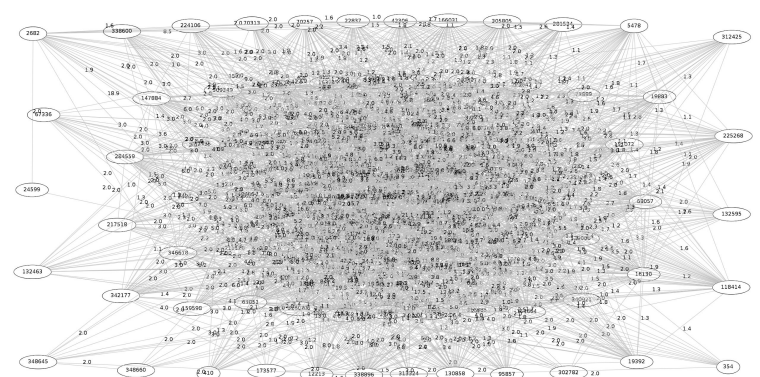
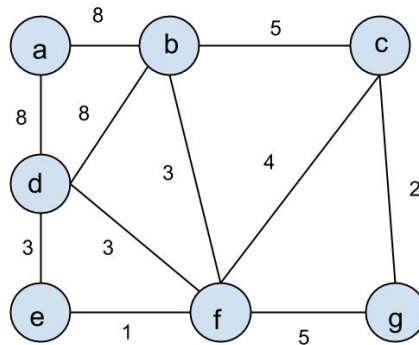
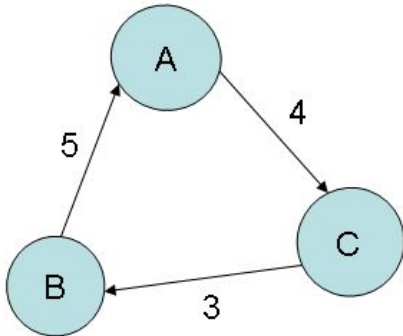
Introdução à análise de complexidade de algoritmos

Prof. Martín Vigil

Motivação



Motivação



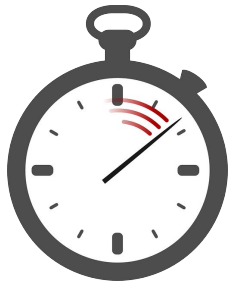
Motivação

- É importante analisar o *comportamento* dos algoritmos para:
 - Entender como as entradas *alteram* o comportamento
 - *Comparar* algoritmos similares quanto ao comportamento

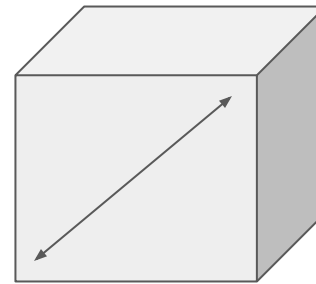
Comportamento do Algoritmo → Complexidade do Algoritmo

Complexidade de Algoritmos

- É a **quantidade de recursos** que um algoritmo consome para resolver um determinado problema



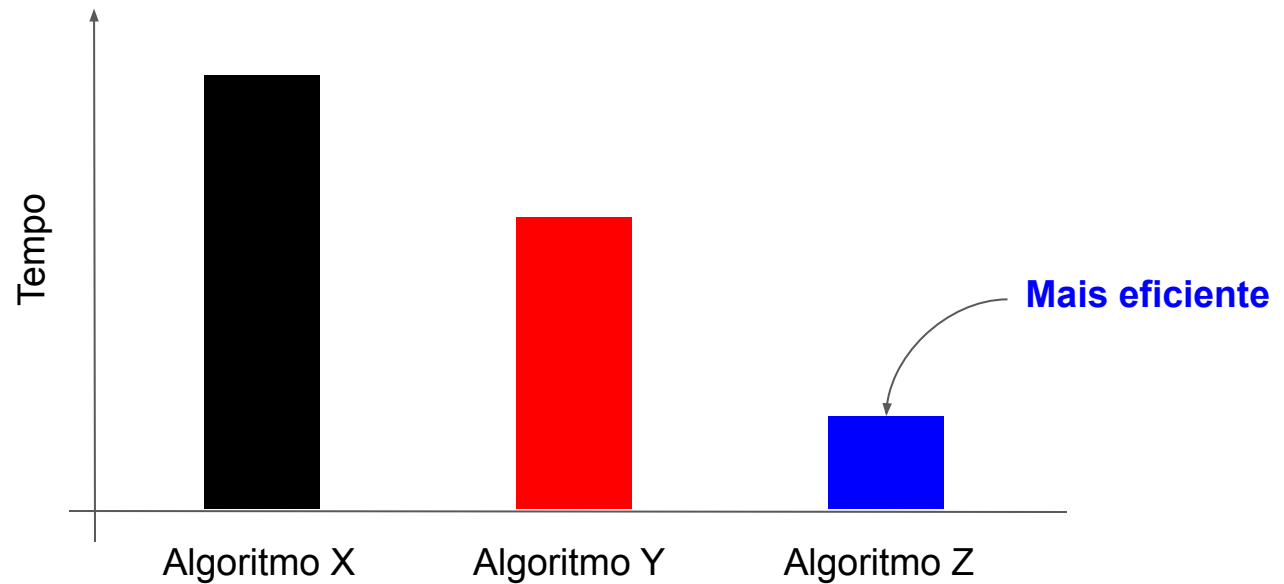
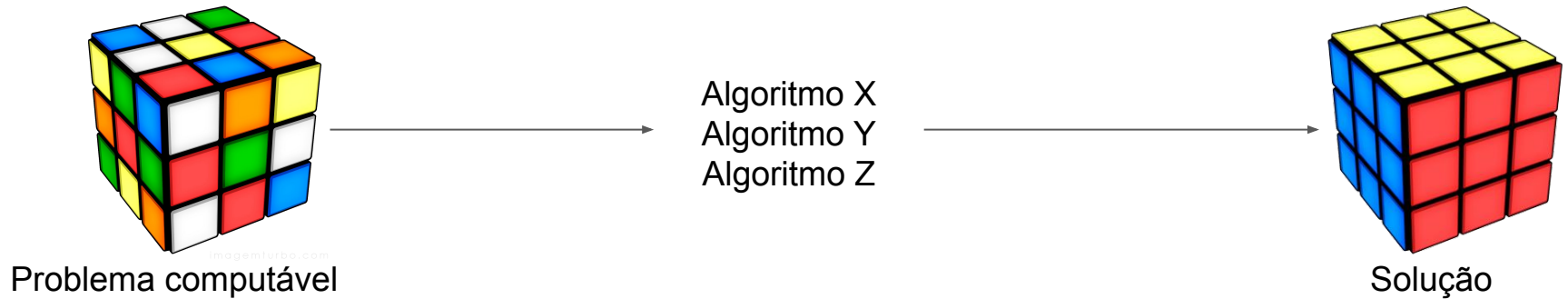
Complexidade Temporal



Complexidade de Espaço



Comparando algoritmos



Avaliação empírica de complexidade

```
sorting — -bash — 53x14
[machome:sorting martin$ make quicksort bubblesort
c++      quicksort.cpp    -o quicksort
c++      bubblesort.cpp   -o bubblesort
[machome:sorting martin$ time ./quicksort

real    0m0.005s
user    0m0.002s
sys     0m0.002s
[machome:sorting martin$ time ./bubblesort

real    0m0.092s
user    0m0.087s
sys     0m0.003s
machome:sorting martin$
```

Implementar com LP
Executar em SO/PC
Medir tempos
Comparar tempos



Depende de hardware

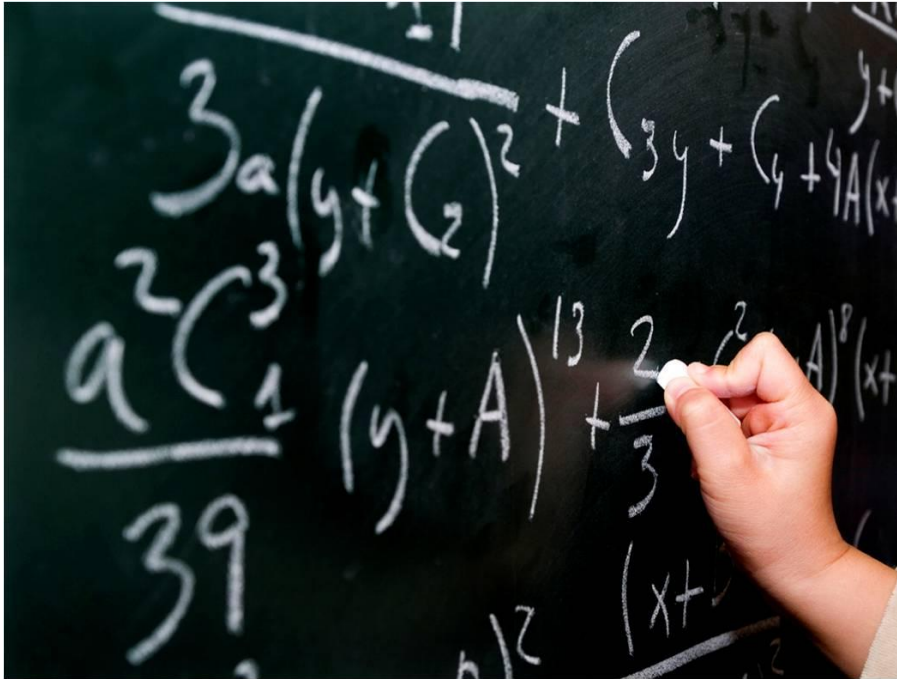


Depende de programador

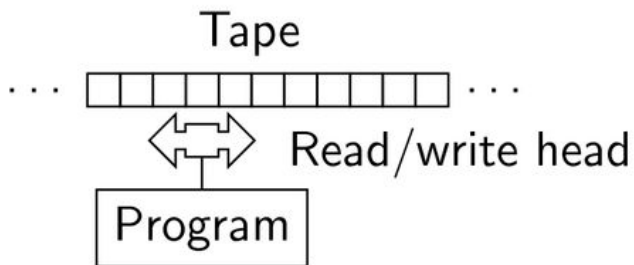


Entradas limitadas

Análise matemática de complexidade



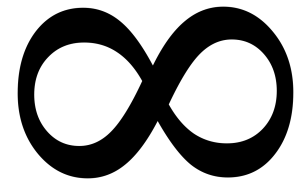
Nossa aula



Abstrai hardware

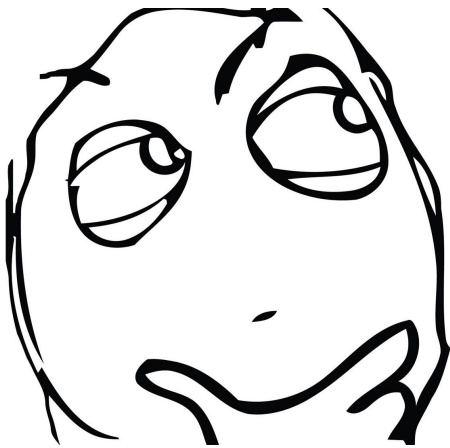


Abstrai programador



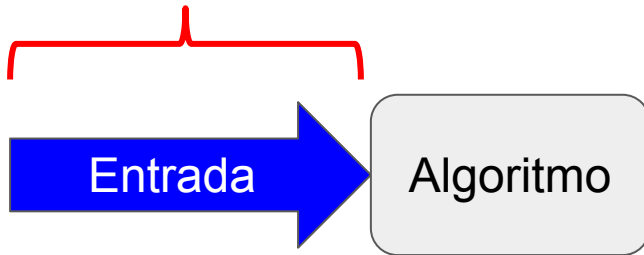
Extrapola entradas

Como analisar matematicamente
a complexidade temporal
de um algoritmo?

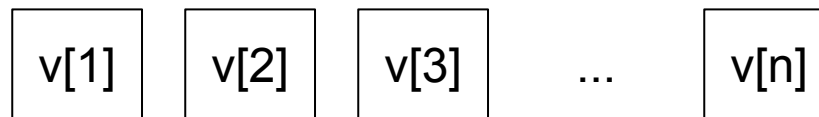


Critérios para a análise de complexidade

- **Tamanho** $n \geq 0$ da **entrada** do algoritmo



- Exemplo: **tamanho** $n \geq 0$ de um **vetor** v a ser ordenado por um algoritmo



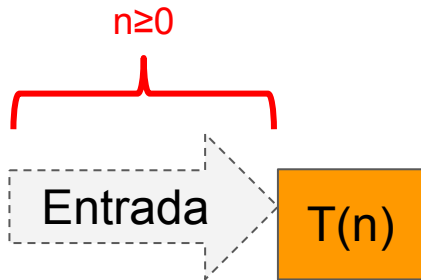
Cr terios para a an lise de complexidade

- Quantidade $T(n)$ de instru es a executar como fun  o do tamanho n da entrada

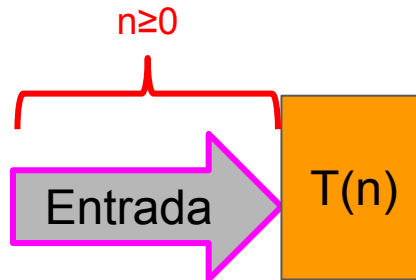
$$\# \text{instru  es} = T(n)$$

Critérios para a análise de complexidade

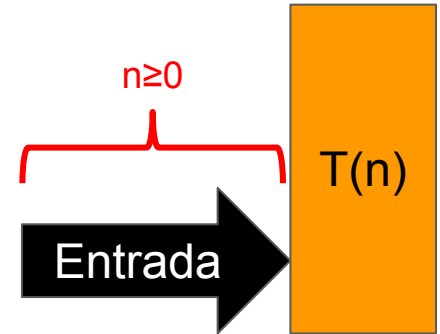
- O **estado** da **entrada** pode interferir em $T(n)$



Melhor caso



Caso médio



Pior caso

- Qual caso priorizar no projeto de um sistema?

Critérios para a análise de complexidade

- Instruções **sempre** executadas

- Instruções **independentes** de condicionais
- **Repetições conhecidas** a priori

```
1. n=5
2. f=1
3. para i=1,2,...,n
4.     f = f*i
```

- Instruções **nem sempre** executadas

- Instruções **dependentes** de condicionais
- **Repetições não conhecidas** a priori

```
1. se <condição>
2.     <bloco de instruções A>
3. senão
4.     <bloco de instruções B>
```

```
1. enquanto <condição>
2.     <bloco de instruções C>
```


Critérios para a análise de complexidade

- Instruções **sempre** executadas
 - Contabilizar normalmente

```
1.  n=5
2.  f=1
3.  para i=1,2,...,n
4.      f = f*i
```

Critérios para a análise de complexidade

- Instruções **nem sempre** executadas

Caso	Condicionais	Repetições não conhecidas a priori
Melhor	A alternativa menos complexa	O mínimo de repetições
Pior	A alternativa mais complexa	O máximo de repetições

Código 1

```
1. se n é par
2.     < m instruções >
3. senão
4.     < m2 instruções >
```

Código 2

```
1. k = sorteiaNúmeroNatural()
2. enquanto k < n
3.     < m instruções >
4.     k = k + 1
```

Exercício: Fatorial ou dobro

```
1. fatorialOuDobro(n)
2.   se n é par
3.     f = 1
4.     para i=1,2,...,n
5.       f = f * i
6.     retorne f
7.   senão
8.     retorne 2*n
```

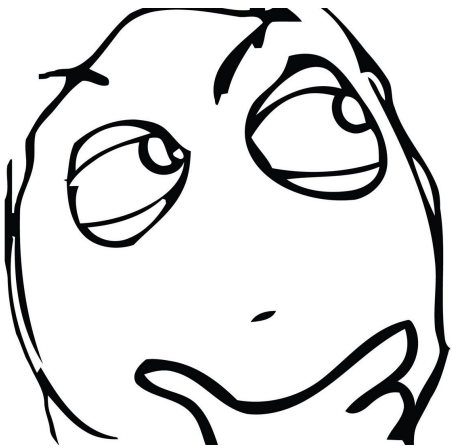
- Encontre $T(n)$ para o melhor e pior casos

Exemplo: Busca linear

- Buscar o número x em um vetor v de $n \geq 0$ números
- Identificar o melhor e pior casos

```
1. função buscaLinear(x, v, n)
2.   para i=1,2,...,n
3.     se x == v[i]
4.       retorne i
5.   retorne -1
```

Se o tamanho $n \geq 0$ da entrada
for significativamente GRANDE?



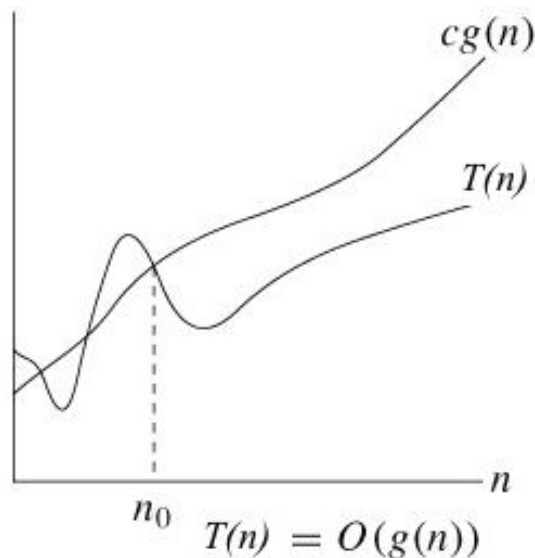
Notação assintótica

- Descreve o **limite de crescimento** de uma **função** quando o **tamanho** da entrada **cresce significativamente**
- **Prioriza** as instruções cuja quantidade tem **maior magnitude**
- Os limites de crescimento são três:
 - a. Limite superior O (O-grande)
 - b. Limite inferior Ω (Ômega)
 - c. Limite restrito Θ (Theta)

Limite assintótico superior O

- Identifica o **limite superior** de crescimento de uma função $T(n)$
- Este limite é dado por outra função $g(n)$

$T(n)=O(g(n))$ se existem $c, n_0 > 0$ tais que $T(n) \leq cg(n)$ para $n > n_0$



Exercícios: Encontrar limite superior O

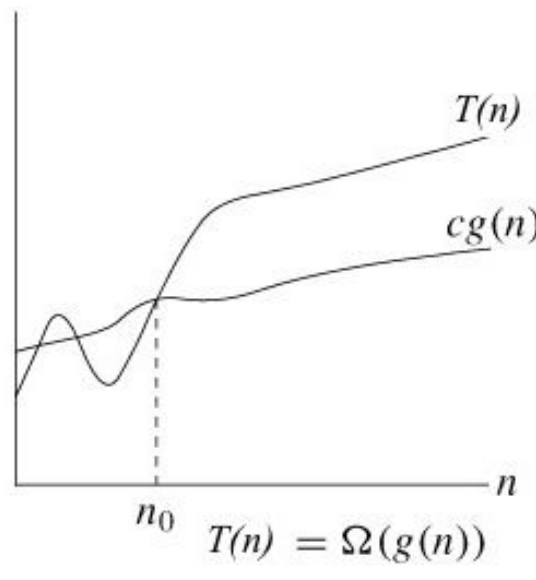
- $T(n) = n^2 + n$
 - $g(n) = ?$
 - $c = ?$
 - $n_0 = ?$

- $T(n) = 2n + 1$
 - $g(n) = ?$
 - $c = ?$
 - $n_0 = ?$

Limite assintótico inferior Ω

- Identifica o **limite inferior** de crescimento de uma função $T(n)$
- Este limite é dado por outra função $g(n)$

$T(n) = \Omega(g(n))$ se existem $c, n_0 > 0$ tais que $T(n) \geq cg(n)$ para $n > n_0$



O algoritmo executa **no mínimo** $cg(n)$ instruções

Exemplo: Encontrar limite inferior Ω

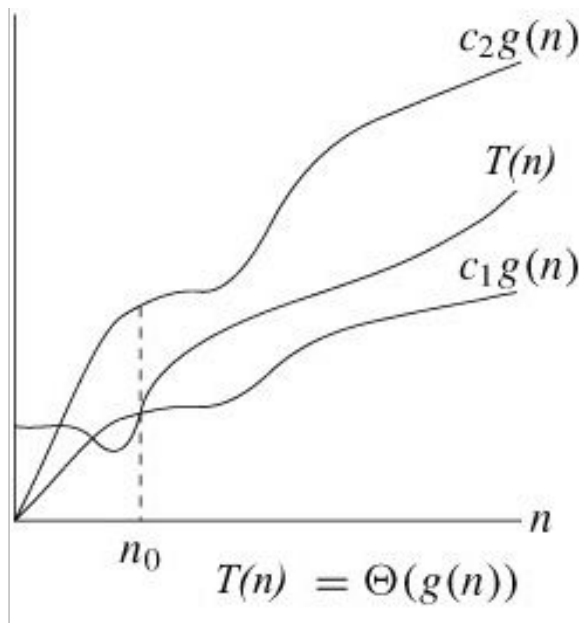
- $T(n) = n^2 + n$
 - $g(n) = ?$
 - $c = ?$
 - $n_0 = ?$

- $T(n) = 2n + 1$
 - $g(n) = ?$
 - $c = ?$
 - $n_0 = ?$

Limite assintótico restrito Θ

- Este limite também é dado por um função $g(n)$

$T(n) = O(g(n))$ se existem $c_1, c_2, n_0 > 0$ tais que
 $c_1 g(n) \leq T(n) \leq c_2 g(n)$ para $n > n_0$



O algoritmo executará entre
 $c_1 g(n)$ e $c_2 g(n)$ operações

Exemplo: Encontrar limite restrito Θ

- $T(n) = n^2 + n$
 - $g(n) = ?$
 - $c = ?$
 - $n_0 = ?$

- $T(n) = 2n + 1$
 - $g(n) = ?$
 - $c = ?$
 - $n_0 = ?$

Exercício

- Encontrar $g(n)$, n_1 e n_2 e n_0 para
 - a. $\Theta(g(n)) = x^2 + x$
 - b. $\Theta(g(n)) = 2x^2+1$ para n par, $3x+2$ para n ímpar

Exercício

- Encontrar os limites assintóticos para o algoritmo
 - Melhor caso: $T(n) = 2$
 - Pior caso: $T(n) = 2n+2$

```
1. fatorialOuDobro(n)
2.   se n % 2 == 0
3.     f = 1
4.     para i=1,2,...,n
5.       f = f * i
6.     retorne f
7.   senão
8.     retorne 2*n
```

Exercício

- Encontrar os limites assintóticos para o algoritmo
 - Melhor casos: $T(n) = 2$
 - Pior caso: $T(n) = n + 1$

```
1. função buscaLinear(x, v, n)
2.   para i=1,2,...,n
3.     se x == v[i]
4.       retorne i
5.   retorne -1
```


Análise assintótica do Exercício 3

- Encontrar os limites assintóticos para o algoritmo
 - Melhor caso: $T(n) = 1$
 - Pior caso: $T(n) = 5n^{0.5} + 3$

```
1.  função ehPrimo :  
2.      se n < 2  
3.          retorne FALSO  
4.      senão se n == 2  
5.          retorne VERDADE  
6.      senão  
7.          i = 2;  
8.          primo = VERDADE;  
9.          enquanto i < n E primo :  
10.             primo = (n % i ≠ 0)  
11.             i = i + 1;  
12.      retorne primo
```

Síntese da aula