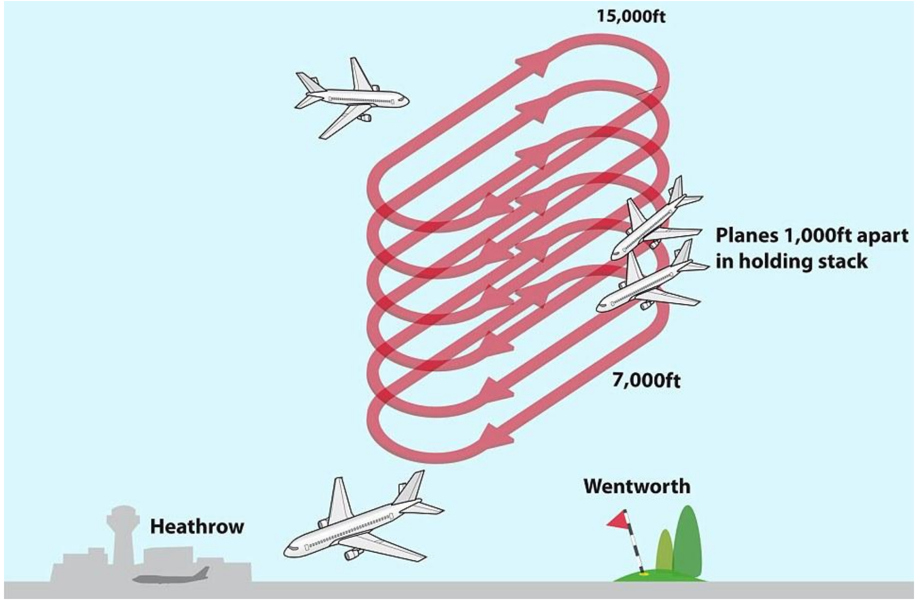
Introdução à análise de complexidade de algoritmos

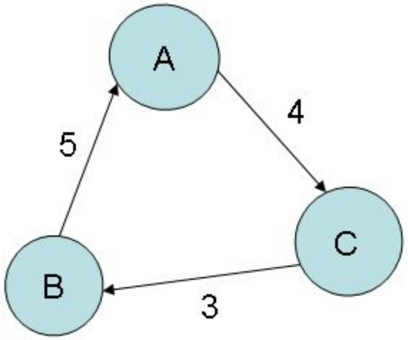
Prof. Martín Vigil

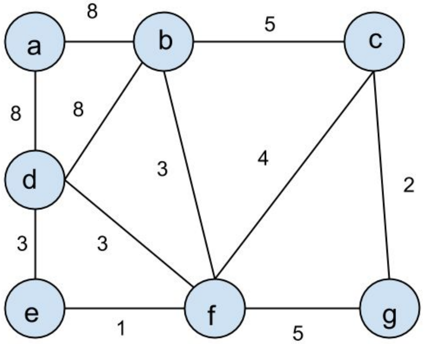
Motivação

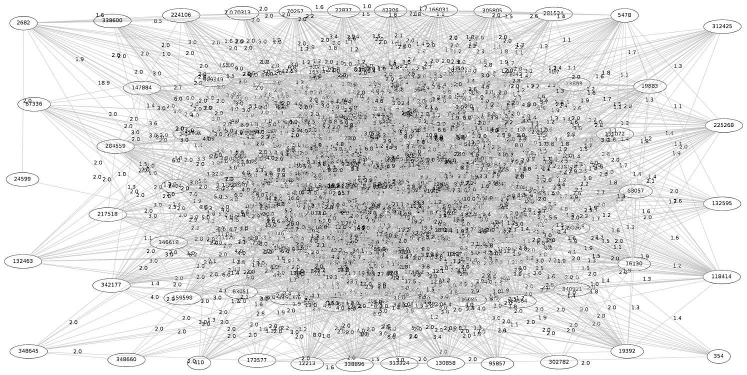

15,000ft
Planes 1,000ft apart
in holding stack
7,000ft
Wentworth
Heathrow

Motivação


Resultado da bus
Opção 1
11:01
30 min
Opção 2
11:11
34 min
Opção 3
11:21
38 min
Adicionar aos favoritos
ETA
11:31
Siga para ponto de ônibus
R. Anhaia - 705
em 3 min
Pesquisar
Paulista - 2319
Linha 715M-10 pela SPTrans
Dados em tempo real
indisponíveis
Exibir no mapa


A
B
C
5
4
3


a
b
c
d
e
f
g
8
5
8
8
3
4
2
3
3
1
5



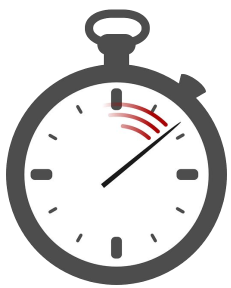
Motivação

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

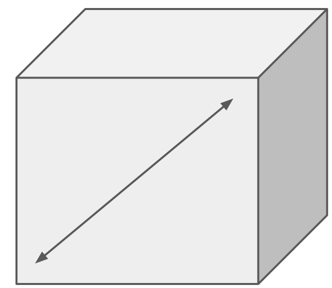
Comportamento do Algoritmo → Complexidade do Algoritmo

Complexidade de Algoritmos

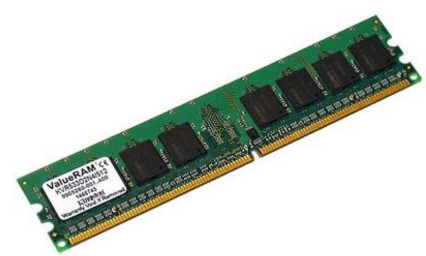
- É a quantidade de recursos que um algoritmo consome para resolver um determinado problema



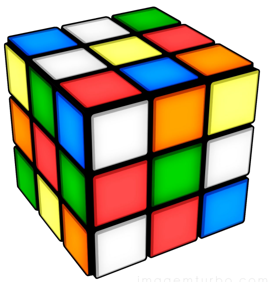
Complexidade Temporal



Complexidade de Espaço

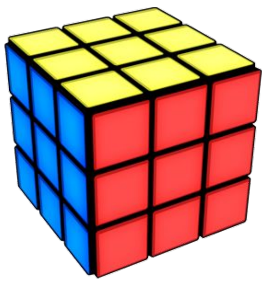


Comparando algoritmos

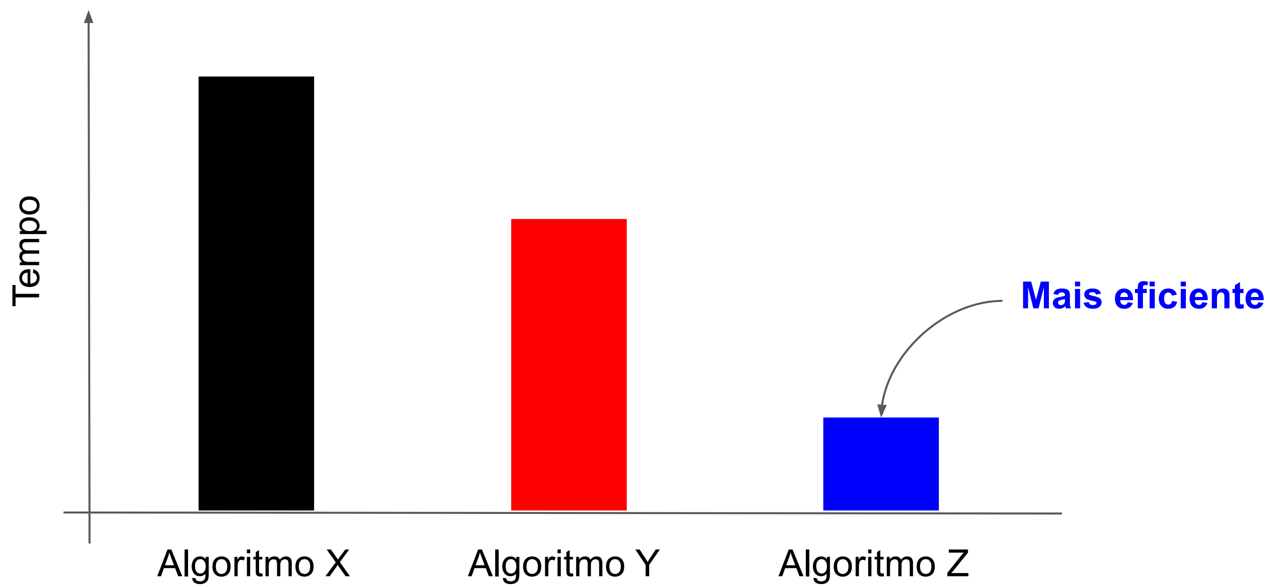


Problema computável

Algoritmo X
Algoritmo Y
Algoritmo Z



Solução



Avaliação empírica de complexidade

```
sorting — -bash — 53×14
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$ 
```

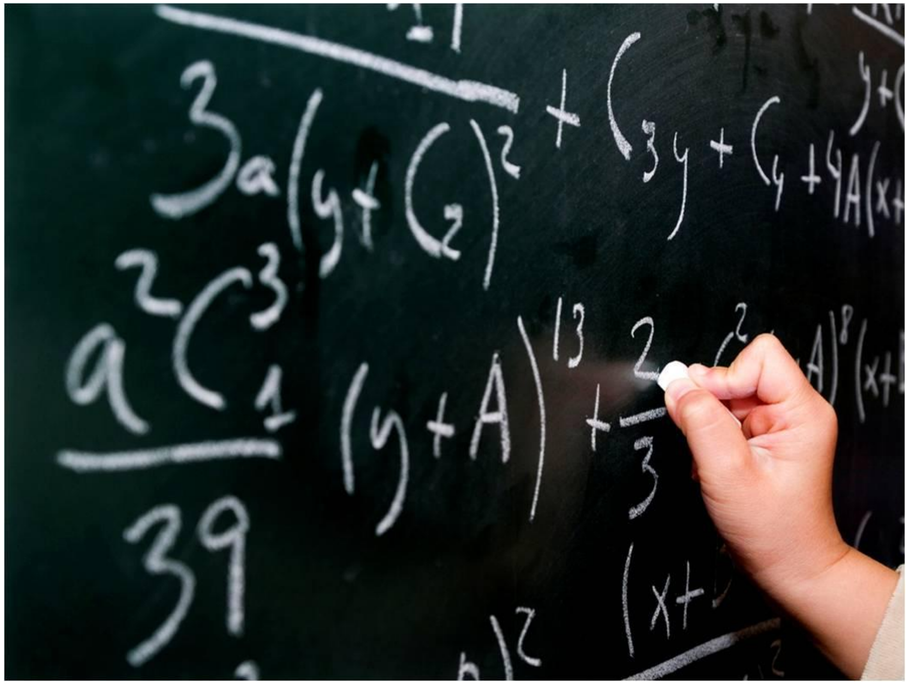
Implementar com LP
Executar em SO/PC
Medir tempos
Comparar tempos

Depende de hardware

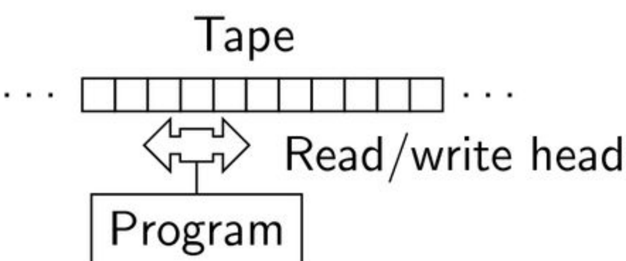
Depende de programador

Entradas limitadas

Análise matemática de complexidade

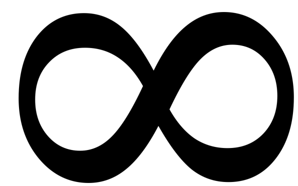


Nossa aula



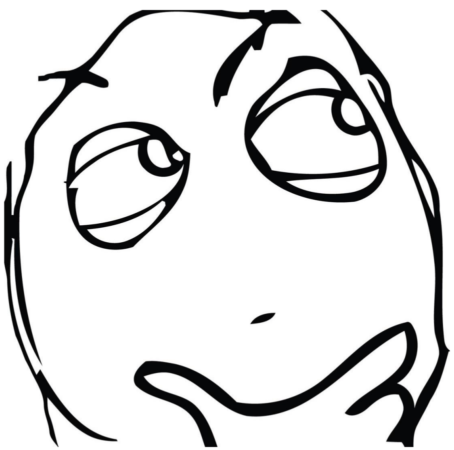
Abstrai hardware

Abstrai programador



Extrapola entradas

Como analisar matematicamente
a complexidade temporal
de um algoritmo?



Modelo Computacional Abstrato

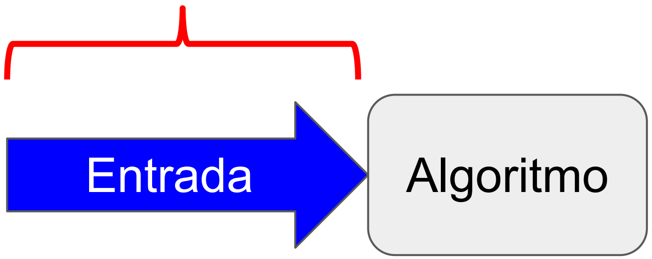
- ≠ tecnologias de CPU, ≠ tempos para executar um algoritmo
- Modelo *Random Access Machine*
  - 1 CPU de 1 core, memória ilimitada e sem hierarquia de cache
  - Uma instrução simples (+,-, *,/, etc) é executada em **1 passo**
  - Um acesso a memória (ler, escrever) é executado em **1 passo**
  - Loop ou subrotina é executado em **n>0 passo(s)**
    - Exemplo: ordenar um vetor

Modelo Computacional Abstrato

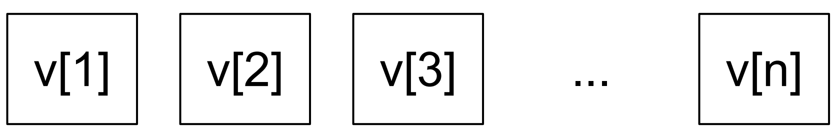
- Pró: permite análises simples e não equivocadas
- Cons: pode contradizer a realidade das tecnologias atuais
  - Multiplicação leva mais ***tempo*** que soma para executar
  - Exponenciação pode ou não executar em ***tempo*** constante (ex 2^k)
  - Cache afeta significativamente acesso na memória

Critérios para a análise de complexidade

- Tamanho n≥0 da entrada do algoritmo



- Exemplo: tamanho n≥0 de um vetor v a ser ordenado por um algoritmo



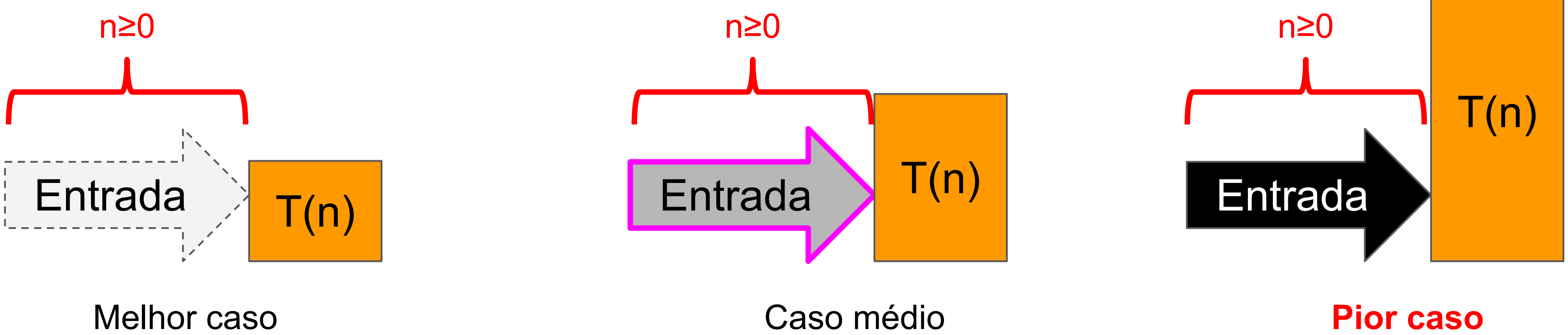
Critérios para a análise de complexidade

- Quantidade T(n) de instruções a executar como função do tamanho n da entrada

$$\#\text{instruções} = T(n)$$

Critérios para a análise de complexidade

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



- 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

```
n=5
f=1
para i=1,2,…,n
    f = f*i
```

- Instruções nem sempre executadas
  - Instruções dependentes de condicionais
  - Repetições não conhecidas a priori

```
se <condição>
    <bloco de instruções A>
senão
    <bloco de instruções B>
```

```
enquanto <condição>
    <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

```
se n é par
    < m instruções >
senão
    < m^2 instruções >
```

Código 2

```
k = sorteiaNúmeroNatural()
enquanto k < n
    <m instruções >
    k = k + 1
```

Exercício: Fatorial ou dobro

```
fatorialOuDobro(n)
    se n é par
        f = 1
        para i=1,2,…,n
            f = f * i
        retorne f
    senão
        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≥0 números
- Identificar o melhor e pior casos

```
função buscaLinear(x, v, n)
  para i=1,2,…,n
      se x == v[i]
        retorne i
  retorne -1
```

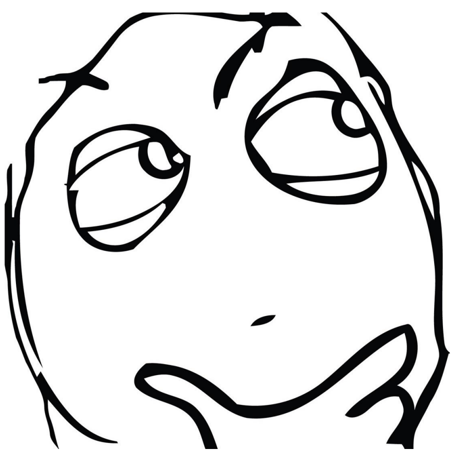
Síntese da aula

Introdução Notação Assintótica

Prof. Martín Vigil

$$\#\text{instruções} = T(n)$$

Se o tamanho n≥0 da entrada
for significativamente GRANDE?



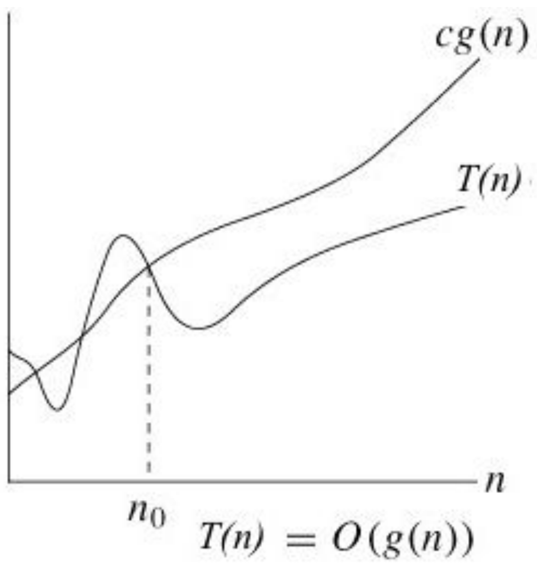
Notação assintótica

- 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 $\boldsymbol{\Omega}$ (Ômega)
    c. Limite restrito $\boldsymbol{\Theta}$ (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 $n>n_0$



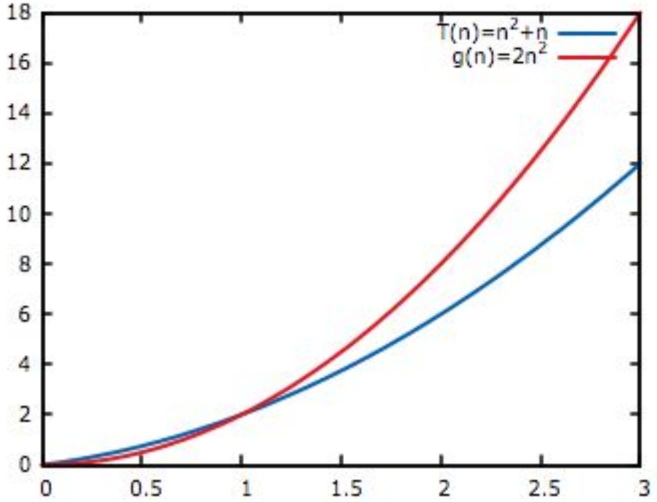
O algoritmo executa no máximo cg(n) instruções

Exercícios: Encontrar limite superior O

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

Exercícios: Encontrar limite superior O

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

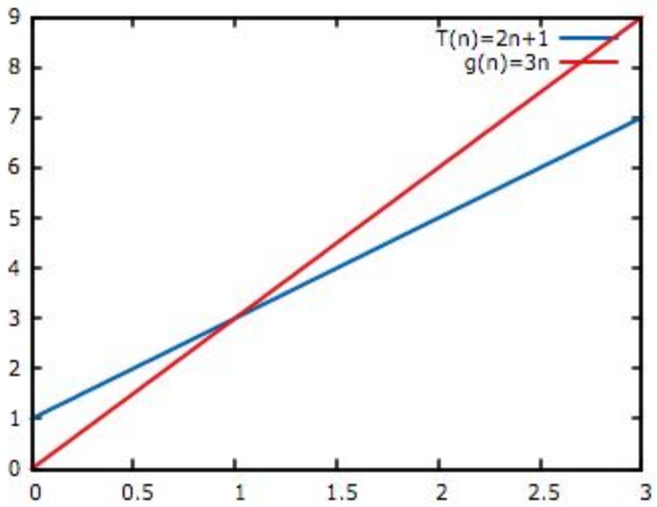


- $T(n) = 2n + 1$
  - $g(n) = ?$
  - $c = ?$
  - $n_0 = ?$

Exercícios: Encontrar limite superior O

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

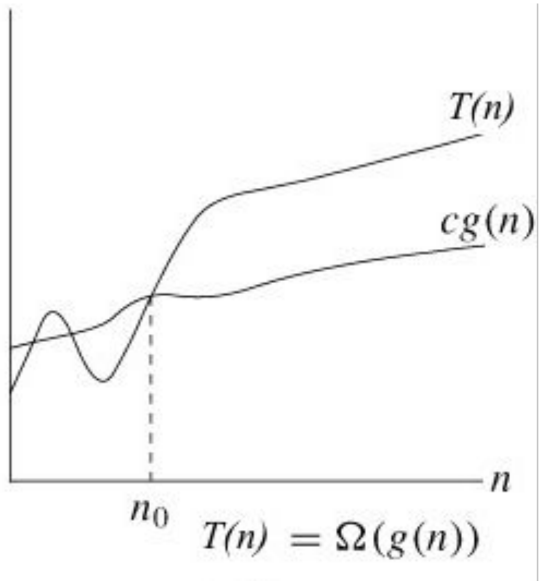
- $T(n) = 2n + 1$
  - $g(n) = ?$
  - $c = ?$
  - $n_0 = ?$



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

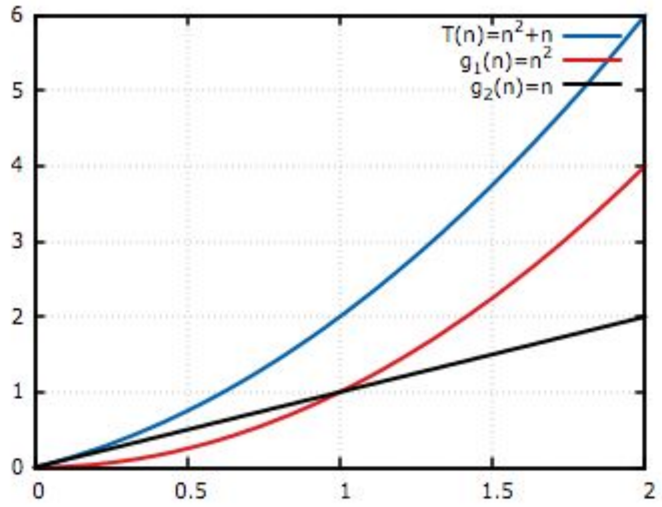
Exemplo: Encontrar limite inferior Ω

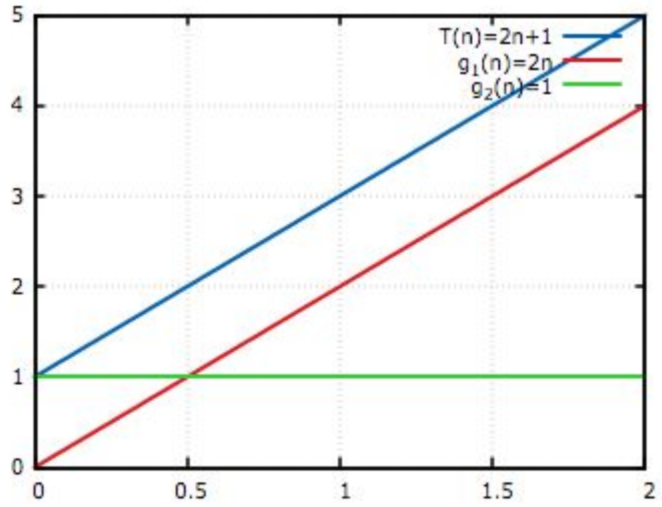
- $T(n) = n^2+n$
  - $g(n) = ?$
  - $c = ?$
  - $n_0 = ?$
- $T(n) = 2n + 1$
  - $g(n) = ?$
  - $c = ?$
  - $n_0 = ?$

Exemplo: Encontrar limite inferior Ω

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

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

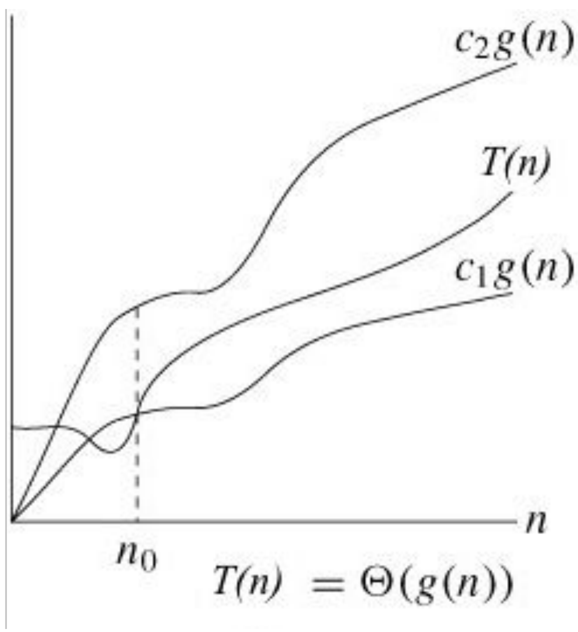




Limite assintótico restrito Θ

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

$T(n)=\Theta(g(n))$ 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 $c_1g(n)$ e $c_2g(n)$ operações

Exemplo: Encontrar limite restrito Θ

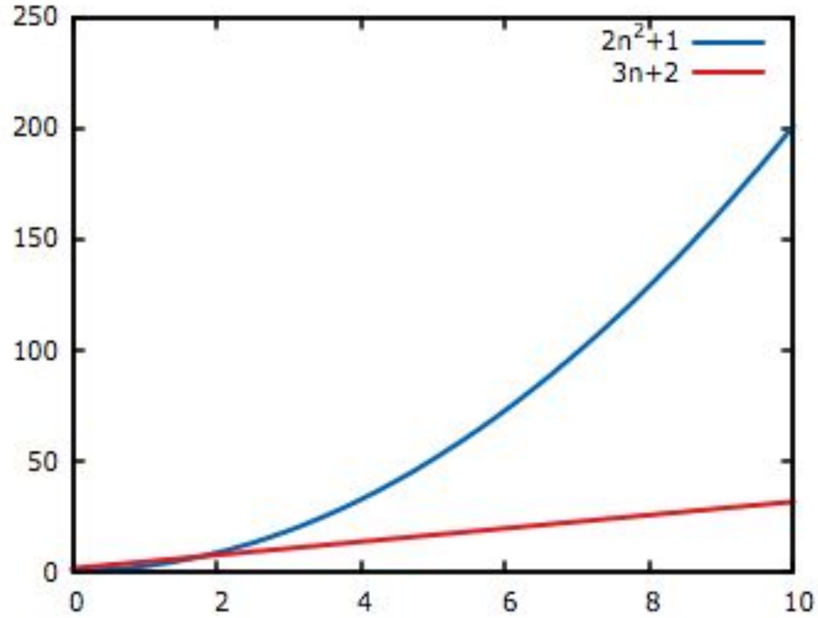
- $T(n) = n^2+n$
  - $g(n) = ?$
  - $c = ?$
  - $n_0 = ?$
- $T(n) = 2n + 1$
  - $g(n) = ?$
  - $c = ?$
  - $n_0 = ?$

Exercício

- Encontrar $g(n)$, c_1 e c_2 e n_0 para
    a. $T(n) = n^3 + n \in \Theta(g(n))$

Exercício

- Encontrar $g(n)$, c_1 e c_2 e n_0 para
  a. $T(n) = n^3 + n \in \Theta(g(n))$
  b. $T(n) = 2n^2+1$ para n par, $T(n)$ $3n+2$ para n ímpar e $T(n) \in \Theta(g(n))$



Exercício

- Encontrar os limites assintóticos para o algoritmo
  - Melhor caso:
  - Pior caso:

```
fatorialOuDobro(n)
    se n % 2 == 0
        f = 1
        para i=1,2,…,n
            f = f * i
        retorne f
    senão
        retorne 2*n
```

Exercício

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

```
fatorialOuDobro(n)
    se n % 2 == 0
        f = 1
        para i=1,2,…,n
            f = f * i
        retorne f
    senão
        retorne 2*n
```

Exercício

- Encontrar os limites assintóticos para o algoritmo
  - Melhor caso:
  - Pior caso:

```
função buscaLinear(x, v, n)
para i=1,2,…,n
    se x == v[i]
      retorne i
  retorne -1
```

Exercício

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

```
função buscaLinear(x, v, n)
para i=1,2,…,n
    se x == v[i]
      retorne i
  retorne -1
```

Análise assintótica do Exercício 3

- Encontrar os limites assintóticos para o algoritmo
  - Melhor caso:
  - Pior caso:

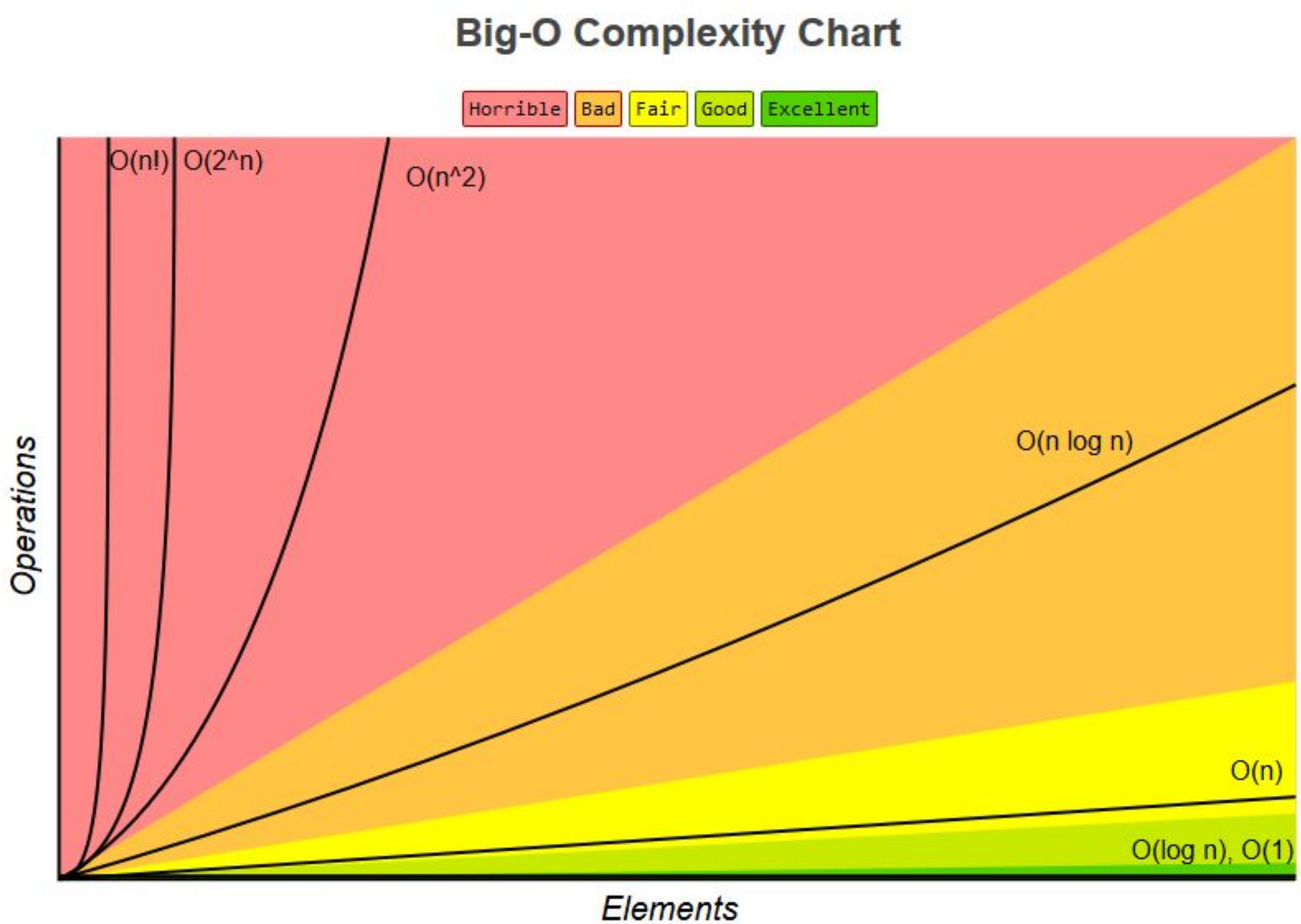
```
função ehPrimo :
    se n < 2
        retorne FALSO
    senão se n == 2
        retorne VERDADE
    senão
        i = 2;
        primo = VERDADE;
        enquanto i < n E primo :
            primo = (n % i ≠ 0)
            i = i + 1;
        retorne primo
```

Análise assintótica do Exercício 3

- Encontrar os limites assintóticos para o algoritmo
  - Melhor caso: $T(n) = 2$
  - Pior caso: $T(n) = 7n+9$

```
função ehPrimo :
    se n < 2
        retorne FALSO
    senão se n == 2
        retorne VERDADE
    senão
        i = 2;
        primo = VERDADE;
        enquanto i < n E primo :
            primo = (n % i ≠ 0)
            i = i + 1;
        retorne primo
```

Comparando classes assintóticas



Síntese da aula