



**Ciências
ULisboa**

Faculdade
de Ciências
da Universidade
de Lisboa

Análise de Algoritmos

- ▶ Algoritmo e Problema
- ▶ Correção e Eficiência
- ▶ Análise Experimental
- ▶ Análise Assintótica
- ▶ Notação O

Algoritmo e Problema

Um **algoritmo** é um método apropriado para resolver um determinado **problema** de forma computacional.

Exemplo:
problema da pesquisa

Algoritmos e Estruturas de Dados (201415 / S2)					
Num	Nome	Turma (T)	Turma (TP)	Sem	Curso
42624	Diogo Baltazar Walter De Freitas Pereira Marques	T21	TP25	U.C.Isoladas	S2
40513	Patrícia Maria Glória Ferreira	T21	TP22	Normal	S2
38977	João Pedro Aleixo Correia	T22	TP29	Normal	S2
36235	Emanuel João Robarts Pires	T21	TP25	Normal	S2
42238	Miguel Canhoto Nobre	T21	TP29	Normal	S2
43622	João Miguel Felício Dos Reis	T21	TP21	Normal	S2
43533	Henrique Miguel Parreira Barradas	T22	TP25	Normal	S2
47347	Inês Alexandra Mendonça Carriço	T22	TP23	Normal	S2
47303	Diogo Miguel Pereira Vasconcelos	T21	TP22	Normal	S2
47340	Pedro Jorge Amado Marques	T21	TP29	Normal	S2
47325	Ivaldo Humberto Semedo	T21	TP22	Normal	S2
45691	Francisco Branco Pires Marques Farinha	T21	TP26	Normal	S2
47160	Pedro Alexandre Morais De Andrade	T21	TP26	Normal	S2
47140	Ana Sofia Afonso Só	T22	TP27	Normal	S2
45687	Alexandre Miguel Nunes Tomé	T21	TP22	Normal	S2
47084	Joana Correia Magalhães Sousa	T22	TP24	Normal	S2
47083	João Manoel Fernandes	T22	TP21	Normal	S2
47082	Luís Miguel Barros Costa	T22	TP28	Normal	S2
47091	Rodrigo Soares Dos Santos	T21	TP24	Normal	S2
47086	João Pedro Santiago Polido	T22	TP29	Normal	S2
47090	Noel António Rato Ramos Cabeça	T22	TP27	Normal	S2
47079	Patrícia Raquel Silva Dias	T22	TP28	Normal	S2
47067	André Miguel Silva Dias	T22	TP28	Normal	S2
47073	Pedro Miguel Silva Dias	T22	TP28	Normal	S2
47072	Miguel Miguel Silva Dias	T22	TP28	Normal	S2
47069	João Miguel Silva Dias	T22	TP28	Normal	S2
47074	Márcio Miguel Silva Dias	T22	TP28	Normal	S2
47061	Diogo Miguel Silva Dias	T22	TP28	Normal	S2

Find

Find what:
40623

Find Next

Close

Exemplo

Problema da Pesquisa

Dada uma sequência X averiguar se um dado x existe nos primeiros n elementos de X , com $n \leq |X|$

Algoritmo da Procura Linear

input: sequência X , elemento x e natural n com $n \leq |X|$

output: true se x é um dos primeiros n elementos de X e false caso contrário

Percorrer a sequência desde o 1º até ao n -ésimo elemento, comparando cada elemento com x . Terminar quando se encontra elemento igual a x (output é true) ou quando já se comparou com o n -ésimo elemento (output é false)

Problemas, instâncias e algoritmos

- Um **problema** é definido por uma descrição genérica dos seus parâmetros e por uma condição que as soluções têm de satisfazer.
- Uma **instância de um problema** obtém-se especificando valores particulares para todos os parâmetros desse problema.
- Um **algoritmo que resolve um problema** é uma descrição, passo a passo, de uma forma genérica de encontrar uma solução de qualquer instância do problema.
 - Os parâmetros do problema são o **input** do algoritmo
 - A solução calculada é o **output** do algoritmo

Representação dos algoritmos

- Precisamos de formas de descrever a sequência de passos que compreende um algoritmo
 - **Linguagem natural**
 - Acessível a todos
 - Grande poder expressivo
 - **Pseudo-código**
 - Descrições mais estruturadas e rigorosas mas ainda de alto nível
 - Apropriadas para serem processadas por humanos
- É comum começar por expressar as ideias principais de um algoritmo em linguagem natural e depois passar para pseudo-código para tornar mais claros e precisos os detalhes do algoritmo

Exemplo: Procura linear

input: sequência X , elemento x e natural n com $n \leq |X|$

output: true se x é um dos primeiros n elementos de X e false caso contrário

Percorrer a sequência desde o 1º até ao n -ésimo elemento, comparando cada elemento com x . Termina quando encontra elemento igual a x (output é true) ou quando já comparou com o n -ésimo elemento (output é false)

Linguagem natural

Exemplo: Procura linear

input: sequência X , elemento x e natural n com $n \leq |X|$

output: true se x é um dos primeiros n elementos de X e false caso contrário

Percorrer a sequência desde o 1º até ao n -ésimo elemento, comparando cada elemento com x . Termina quando encontra elemento igual a x (output é true) ou quando já comparou com o n -ésimo elemento (output é false)

Linguagem natural

```
Algorithm linearSearch( $X, x, n$ ){  
   $i := 1$   
  found := false  
  while  $i \leq n$  and not found  
    if  $X(i) = x$  then  
      found := true  
    else  
       $i := i + 1$   
  return found  
}
```

Pseudo-código

Correção e Eficiência

Os algoritmos devem ser corretos e eficientes.

- **Correção:** Para todas as instâncias do problema, o que é calculado pelo algoritmo é de facto uma solução do problema.
- **Eficiência:** Para todas as instâncias do problema, a solução é calculada com um consumo aceitável de recursos. Em geral, mede-se a eficiência relativamente a dois recursos computacionais — **espaço** e **tempo** — e como uma **função do tamanho do *input***.

Mais sobre problemas e correção de algoritmos

A noção de problema e de correção de algoritmo para um problema anteriormente apresentada é simples mas estrita.

- **Problemas que envolvem aproximações**

- Dados 3 pontos num plano, saber se são colineares
- Dado um número x , encontrar a sua raiz quadrada

- **Problemas probabilísticos**

- Dado um número natural n , gerar uma permutação aleatória dos números entre 1 e n
- Baralhar uma sequência de números X

No 1º caso é necessário definir o que é uma boa aproximação.
No 2º caso é necessário considerar a probabilidade com que cada solução para uma instância é gerada.

Análise de Algoritmos

- Existem diferentes técnicas de análise de algoritmos
- Em AED o *focus* será em técnicas de análise que permitem:
 - Prever o desempenho
 - Comparar algoritmos relativamente ao seu desempenho
 - Dar garantias relativamente ao desempenho



Análise de Algoritmos: Experimentalmente

- Pode-se fazer um estudo experimental dos recursos consumidos (observações empíricas)
- É preciso fixar uma implementação, um ambiente de execução, uma máquina, e um conjunto de instâncias (testes de *input*)
- Por exemplo, em Java, para medir o tempo de execução podemos usar `System.currentTimeMillis()`

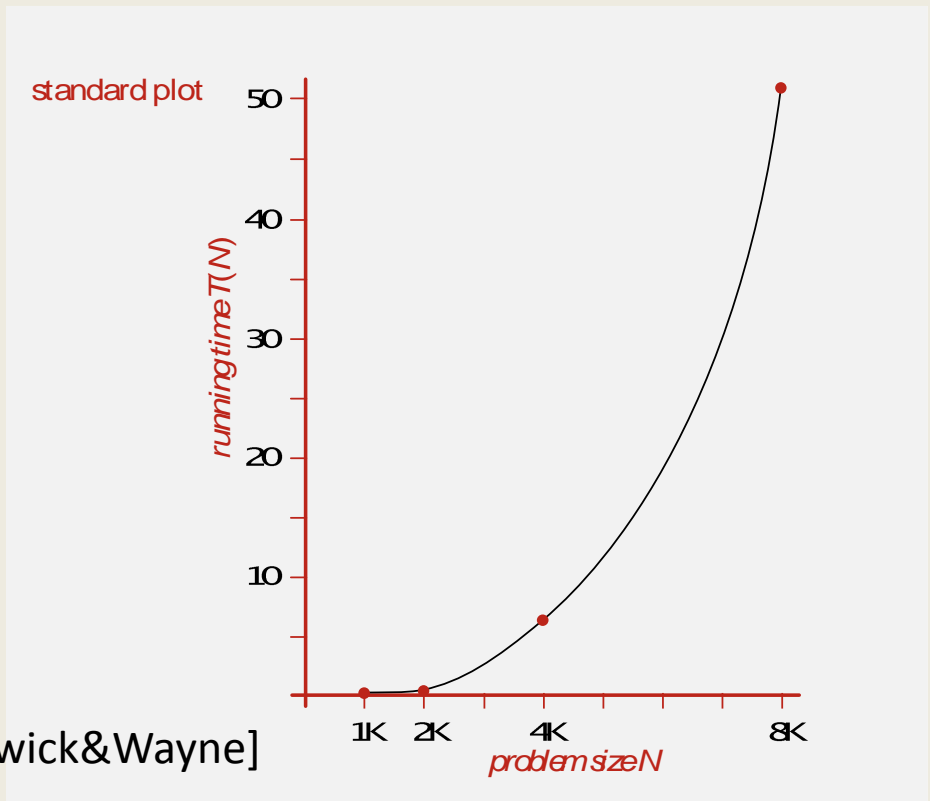
```
public class Stopwatch {  
    private final long start;  
    public Stopwatch() {  
        start = System.currentTimeMillis();  
    }  
  
    // return time (in seconds) since this object was created  
    public double elapsedTime() {  
        long now = System.currentTimeMillis();  
        return (now - start) / 1000.0;  
    }  
}
```



Análise de Algoritmos: Experimentalmente

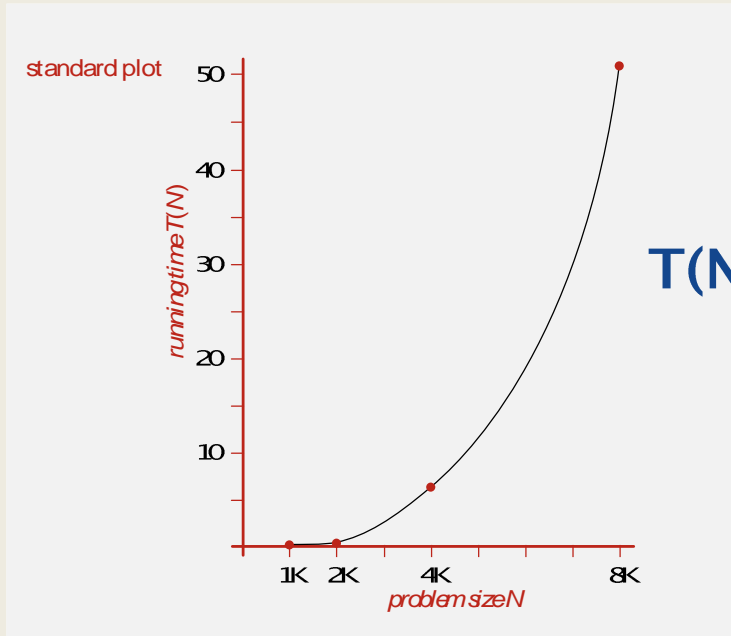
- Pode ser usado o método científico para encontrar uma função $T(N)$ que modele o tempo que o programa consome a resolver um problema de tamanho N
- Corre-se o programa para vários tamanhos de *input* e mede-se o seu tempo de execução para formular uma hipótese

N	time(seconds) †
250	0.0
500	0.0
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1
16,000	?



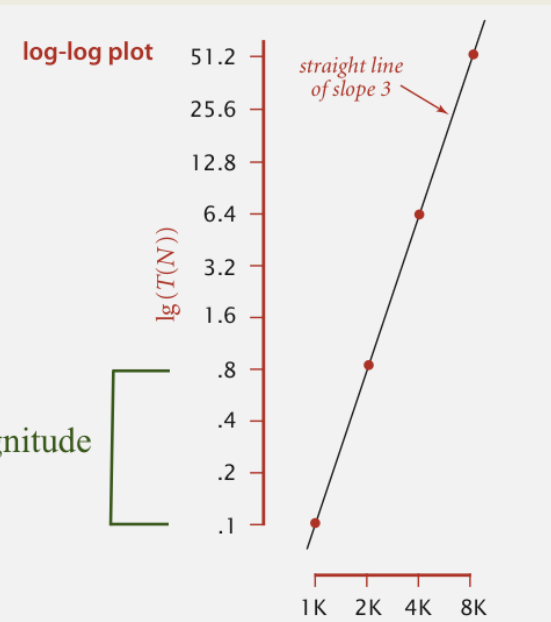
Análise de Algoritmos: Experimentalmente

- Para formular a hipótese pode ser útil recorrer antes a uma representação *log-log* dos pontos recolhidos



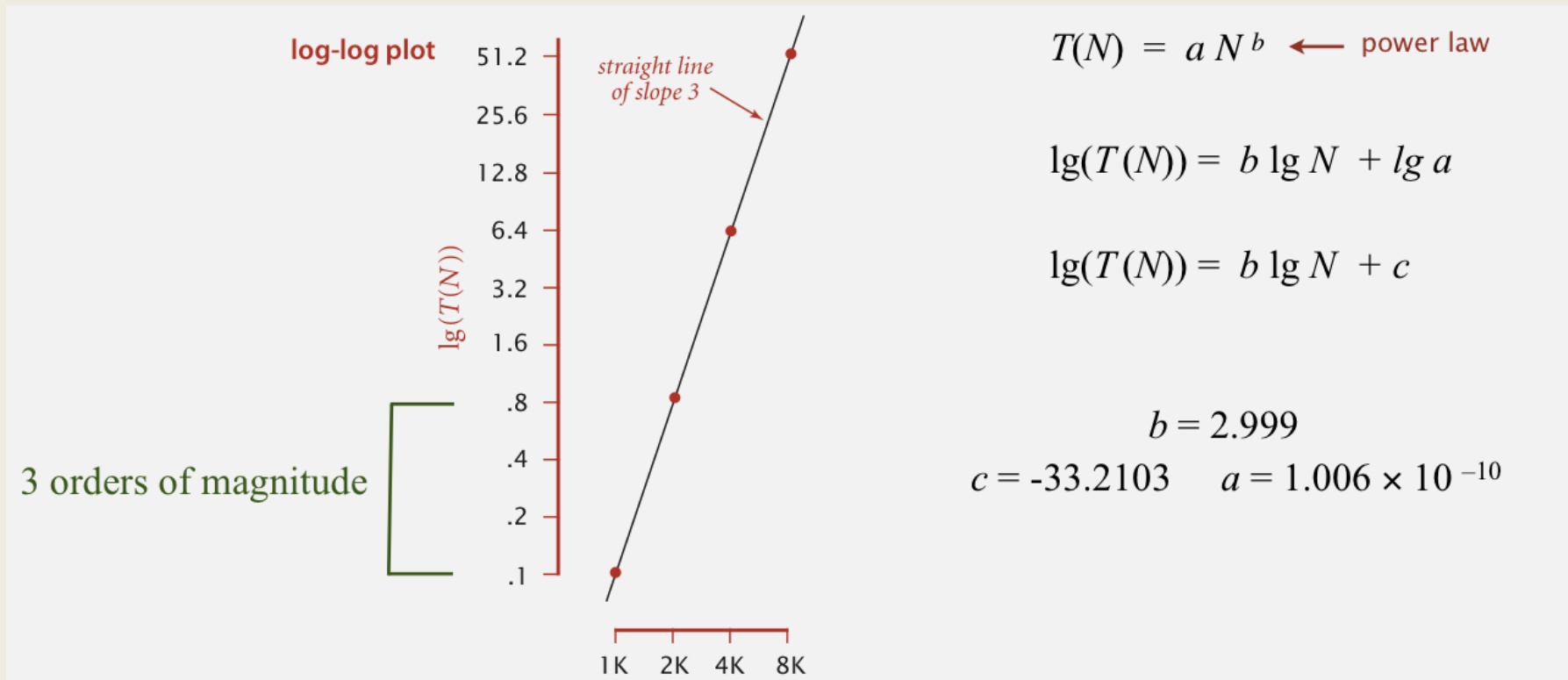
$$T(N) \sim a \cdot N^3$$

3 orders of magnitude



- Depois de ter a hipótese, usá-la para fazer previsões e ver se as observações a validam
- Caso necessário, reformular a hipótese a partir de mais dados

Análise de Algoritmos: Experimentalmente



[from Sedgewick&Wayne]

Análise de Algoritmos: Experimentalmente

Em resumo, para testar a hipótese

$$T(N) \sim a \cdot N^b$$

para uma dada implementação de um algoritmo:

1. Calcular $T(N_0)$ e $T(2 \cdot N_0)$ correndo o programa
2. Calcular b usando

$$\log_2(T(2 \cdot N_0)/T(N_0)) \rightarrow b \quad \text{qdo } N_0 \text{ cresce}$$

3. Calcular a usando

$$T(N_0)/N_0^b \rightarrow a \quad \text{qdo } N_0 \text{ cresce}$$

N	time (seconds) †
250	0.0
500	0.0
1,000	0.1
2,000	0.8
4,000	6.4
8,000	51.1
16,000	?

Análise de Algoritmos: Experimentalmente

– Limitações desta técnica de análise

- limitada às instâncias cobertas pelas experiências efectuadas
- não permite comparar com outros algoritmos sem os executar nas mesmas condições
- não se aplica a descrições de alto nível (temos que ter uma implementação)

– Vantagens desta técnica de análise

- Essencial para entender o desempenho de uma implementação específica

Análise de Algoritmos: Teoricamente

- Estimar os recursos necessários através de uma **análise assintótica**
- **Ideias chave:**
 - caracterizar o tempo e espaço consumidos em termos de duas funções **$T(n)$** e **$S(n)$** no tamanho n do input — independentemente da máquina, ambiente de execução, linguagem de programação
 - dar importância apenas às **taxas de crescimento**, ou seja, como crescem as funções à medida que se aumenta o n

Análise de Algoritmos: Teoricamente

- Contabilidade (modelo de custos):
 - Para a taxa de crescimento, as constantes aditivas e multiplicativas não são relevantes pelo que se considera uma contabilidade muito simplificada dos recursos consumidos
 - No caso do tempo, contam-se apenas o número de operações básicas, por exemplo
 - Avaliar uma expressão
 - Atribuir valor a variável
 - Aceder a posição de vector
 - Chamada a procedimento
 - Retorno de procedimento
 - É preciso ter em atenção que para o mesmo tamanho de *input* pode haver vários casos...

Análise por casos

- Pode-se fazer uma análise por casos:
 - **pior caso**: tomar o máximo para todos os *inputs* com um dado tamanho;
 - **melhor caso**: tomar o mínimo para todos os *inputs* com um dado tamanho;
 - **caso esperado**: a “média” para todos os *inputs* com um dado tamanho, baseada em certas hipóteses sobre a distribuição do input.
- A análise do caso esperado é usualmente difícil de obter, nomeadamente porque hipóteses realistas sobre a distribuição do *input* são frequentemente matematicamente intratáveis.

Exemplo

```
Algorithm linearSearch(X,x,n) {  
    i:=1  
    found:= false  
    while i ≤ n and not found  
        if X(i)=x then  
            found:=true  
        else  
            i:=i+1  
    return found  
}
```

Melhor caso ?

Pior caso ?

Ops ?

Exemplo

```
Algorithm linearSearch(X,x,n) {  
    i:=1  
    found:= false  
    while i ≤ n and not found  
        if X(i)=x then  
            found:=true  
        else  
            i:=i+1  
    return found  
}
```

operações primitivas

min

1

1

2

2

1

0

1

8

- **Melhor caso:** quando $X(1)=x$ e o ciclo é executada apenas uma vez

Exemplo

```
Algorithm linearSearch(X,x,n) {  
    i:=1  
    found:= false  
    while i ≤ n and not found  
        if X(i)=x then  
            found:=true  
        else  
            i:=i+1  
    return found  
}
```

operações primitivas

min	max
-----	-----

1	1
---	---

1	1
---	---

2	$n+1$
---	-------

2	$2n$
---	------

1	0
---	---

0	$2n$
---	------

1	1
---	---

-------	--

8	$5n+4$
---	--------

- **Melhor caso:** quando $X(1)=x$ e o ciclo é executada apenas uma vez
- **Pior caso:** quando o elemento não existe e o ciclo é executado n vezes

Exemplo

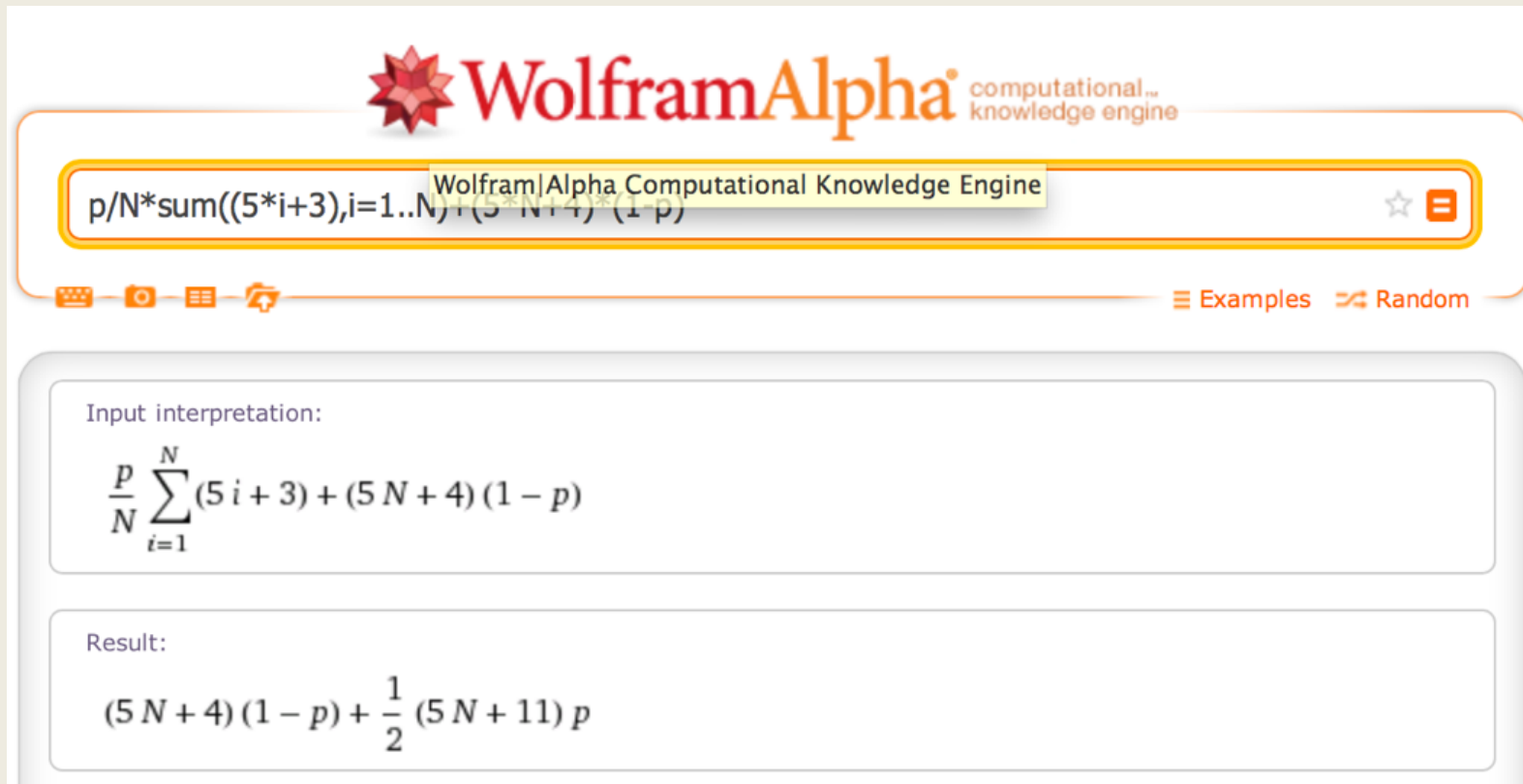
```
Algorithm linearSearch(X,x,n) {  
    i:=1  
    found:= false  
    while i ≤ n and not found  
        if X(i)=x then  
            found:=true  
        else  
            i:=i+1  
    return found  
}
```

Caso esperado: se assumirmos que há probabilidade p do valor procurado estar na sequência de input até n , e que não existem números repetidos, então

$$T(n) = \sum_{i=1}^n (5 \cdot i + 3) \cdot \frac{p}{n} + (5 \cdot n + 4) \cdot (1 - p)$$

Exemplo

A simplificação de $T(n)$ pode ser feita recorrendo a conhecimentos de matemática discreta ou a ferramentas como o *WolframAlpha* ou *Maple*



The screenshot shows the WolframAlpha interface. At the top is the WolframAlpha logo with the tagline "computational knowledge engine". Below the logo is a search bar containing the expression $p/N \cdot \sum_{i=1}^N (5i+3) + (5N+4)(1-p)$. To the right of the search bar are icons for a star and a menu. Below the search bar is a navigation bar with icons for keyboard, camera, list, and share, followed by links for "Examples" and "Random". The main content area is divided into two sections: "Input interpretation:" and "Result:". The "Input interpretation:" section shows the expression $\frac{p}{N} \sum_{i=1}^N (5i+3) + (5N+4)(1-p)$. The "Result:" section shows the simplified expression $(5N+4)(1-p) + \frac{1}{2}(5N+11)p$.

WolframAlpha computational knowledge engine

WolframAlpha Computational Knowledge Engine

$p/N \cdot \sum_{i=1}^N (5i+3) + (5N+4)(1-p)$

Examples Random

Input interpretation:

$$\frac{p}{N} \sum_{i=1}^N (5i+3) + (5N+4)(1-p)$$

Result:

$$(5N+4)(1-p) + \frac{1}{2}(5N+11)p$$

Outro Exemplo

input $n \geq 0$, v vector com números e $|v| \geq n$
output maior número no vector até $v[n-1]$

```
Algorithm arrayMax(v,n) {  
    currentMax:=v[0]  
    for i:=1 to n-1 do  
        if v[i]>currentMax then  
            currentMax:=v[i]  
        i:=i+1  
    return currentMax  
}
```

Melhor caso ?

Pior caso ?

Ops ?

Exemplo

```
Algorithm arrayMax(v,n) {  
    currentMax:=v[0]  
    for i:=1 to n-1 do  
        if v[i]>currentMax then  
            currentMax:=v[i]  
        i:=i+1  
    return currentMax  
}
```

operações primitivas

min

2

1+n

2 (n-1)

0

2 (n-1)

1

5n

- **Melhor caso:** quando $v[0]$ é o maior elemento

Exemplo

```
Algorithm arrayMax(v,n) {  
    currentMax:=v[0]  
    for i:=1 to n-1 do  
        if v[i]>currentMax then  
            currentMax:=v[i]  
        i:=i+1  
    return currentMax  
}
```

operações primitivas

min	max
2	2
1+n	1+n
2 (n-1)	2 (n-1)
0	2 (n-1)
2 (n-1)	2 (n-1)
1	1

5n	7n-2

- **Melhor caso:** quando $v[0]$ é o maior element
- **Pior caso:** quando os elementos estão ordenados de forma crescente

Pior caso

- A análise do pior caso é a mais estudada pois
 - é matematicamente mais tratável e universalmente aplicável
 - dá uma informação importante — **um valor que é garantido não ser ultrapassado por nenhuma execução do algoritmo**
 - em alguns algoritmos o pior caso ocorre frequentemente
 - o caso esperado é frequentemente da mesma ordem que o pior caso
- Assim, a partir de agora as referências a **$T(n)$** referem-se em geral ao tempo de execução no pior caso
- Isto é equivalente a focar a análise nos **limites superiores** do tempo de execução dos algoritmos

Notação O (big-O)

- Uma notação apropriada para raciocinar acerca de **limites superiores** de taxa de crescimento de uma função; compara quão depressa crescem duas funções quando $n \rightarrow \infty$

- **Definição**

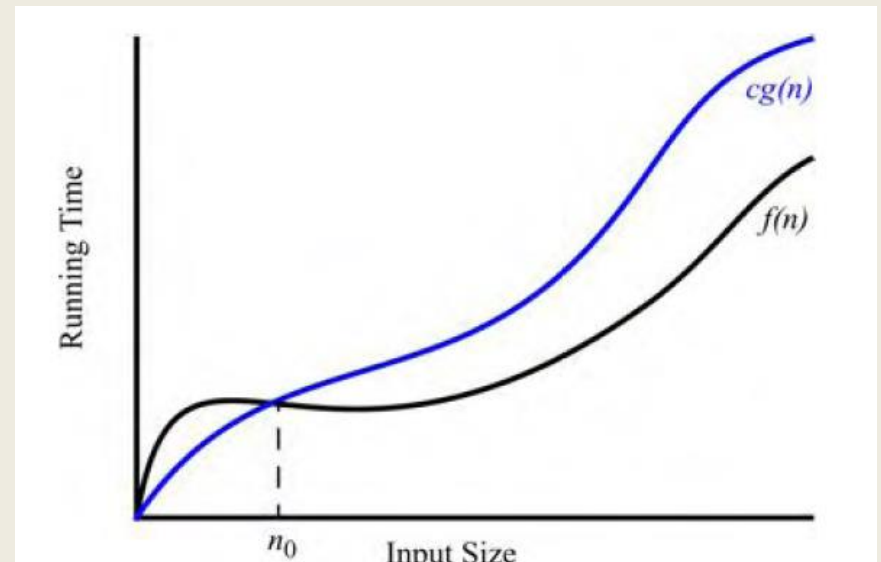
Sejam $f, g: N \rightarrow R^+$. f é $O(g)$

sse

existe n_0 e c em R^+ tal que $f(n) \leq c \cdot g(n)$ para todo o $n \geq n_0$

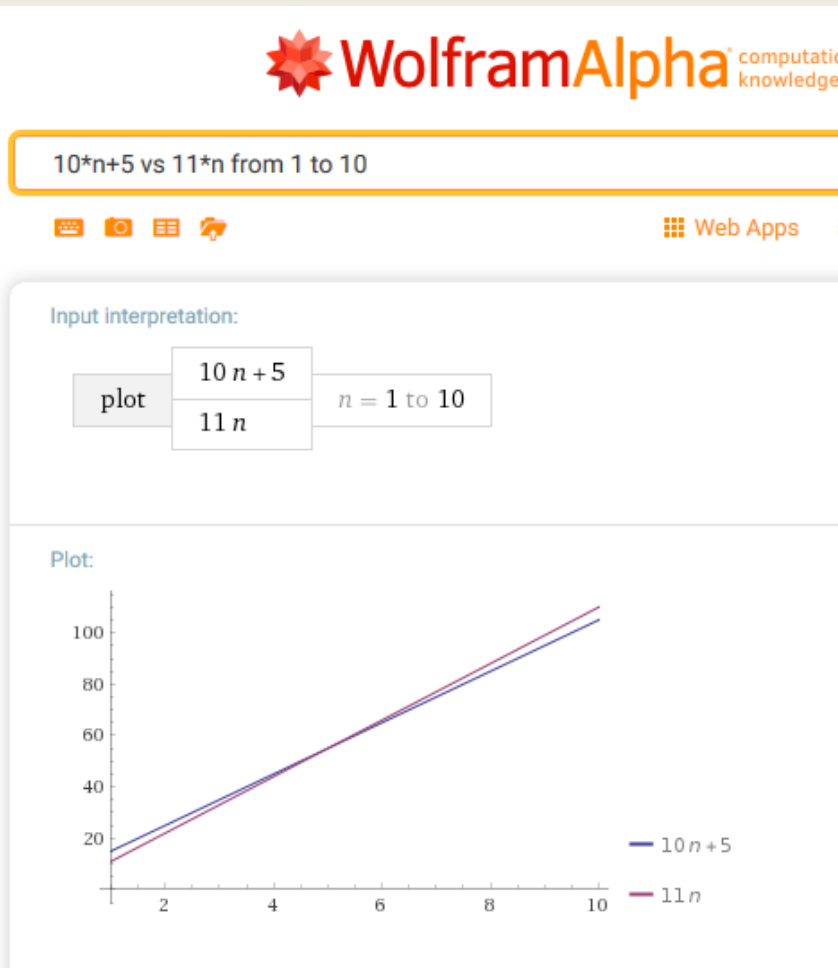
- **Interpretação**

- Se f é $O(g)$ então f cresce como g ou mais devagar.

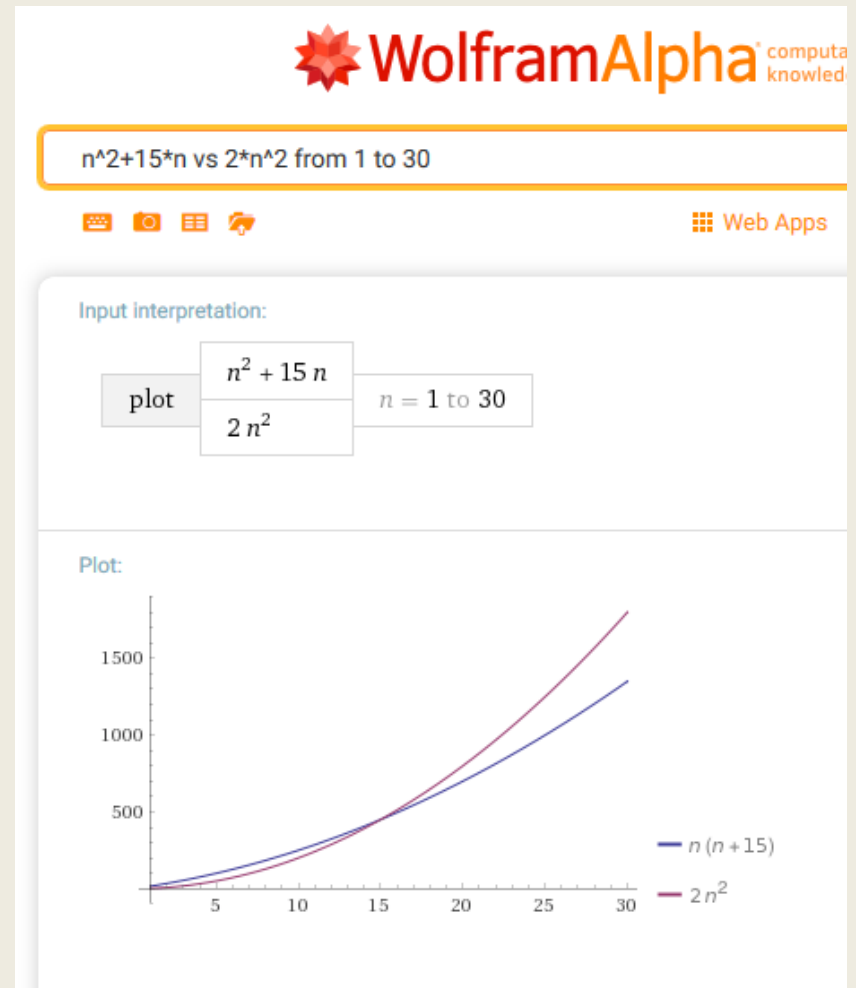


Notação O (big-O): Exemplos

- $10n+5$ é $O(n)$

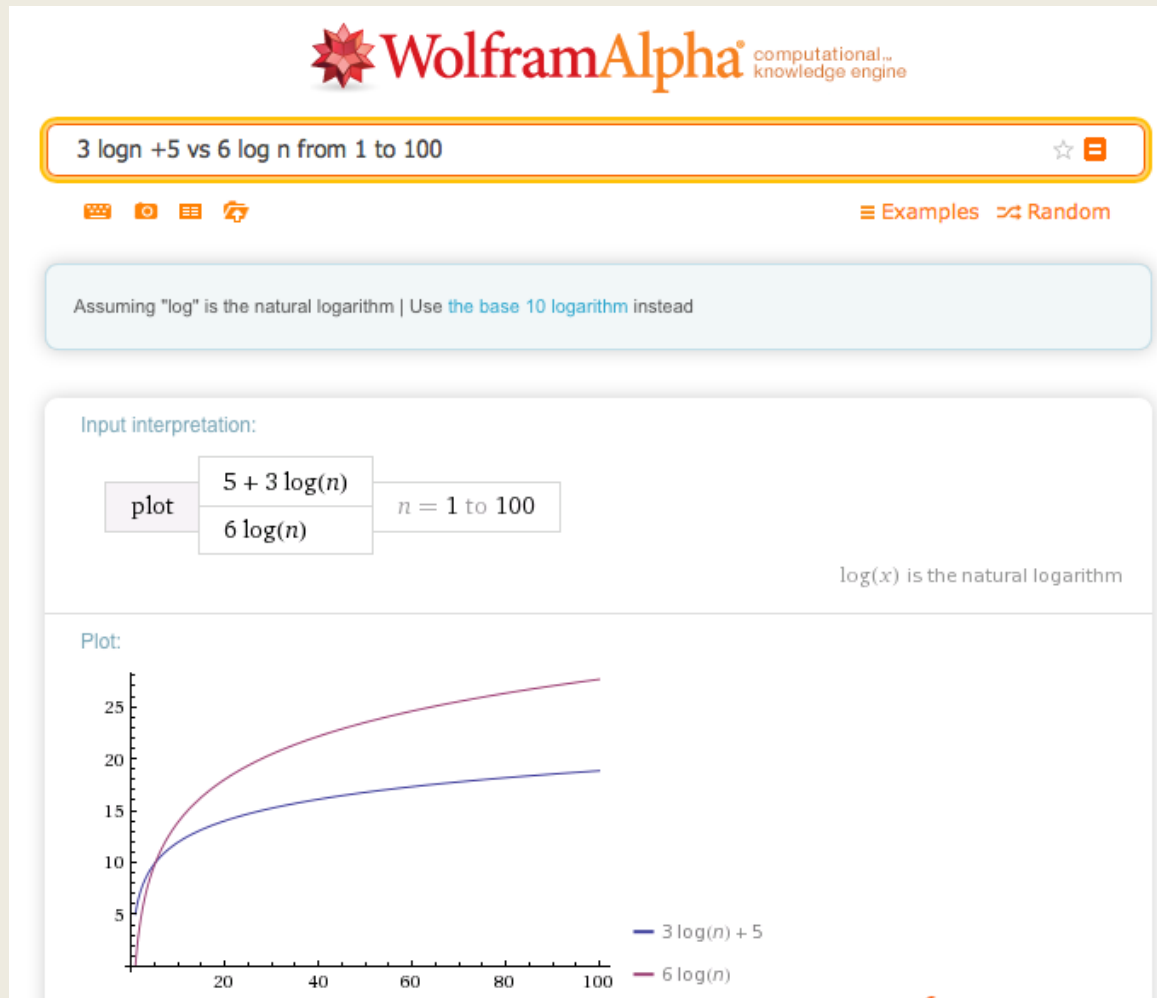


- $n^2 + 15n$ é $O(n^2)$



Notação O (big-O): Exemplos

- $3 \log n + 5$ é $O(\log n)$



Notação O (big-O): Algumas Regras Simples

- Se $f(n)$ é um polinómio de ordem d então $f(n)$ é $O(n^d)$
 - pode-se esquecer os termos de ordem inferior
 - pode-se esquecer o coeficiente de grau d
- Deve-se usar a classe de funções mais pequena possível
 - $2n$ é $O(n)$ é preferível a $2n$ é $O(n^2)$
- Deve-se usar o representante da classe mais simples
 - $3n+5$ é $O(n)$ é preferível a $3n+5$ é $O(3n)$

Notação Θ

- Outra notação muito usada, útil para classificar algoritmos relativamente à taxa de crescimento assintótica

- **Definição**

Sejam $f, g: \mathbb{N} \rightarrow \mathbb{R}^+$. f é $\Theta(g)$

sse

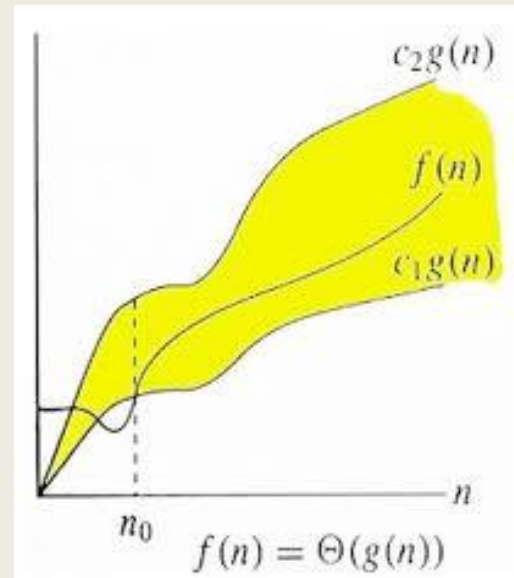
existe c_1, c_2, n_0 em \mathbb{R}^+ tal que

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \text{ para todo o } n \geq n_0$$

- **Interpretação**

$g(n)$ é $O(f(n))$ e $f(n)$ é $O(g(n))$

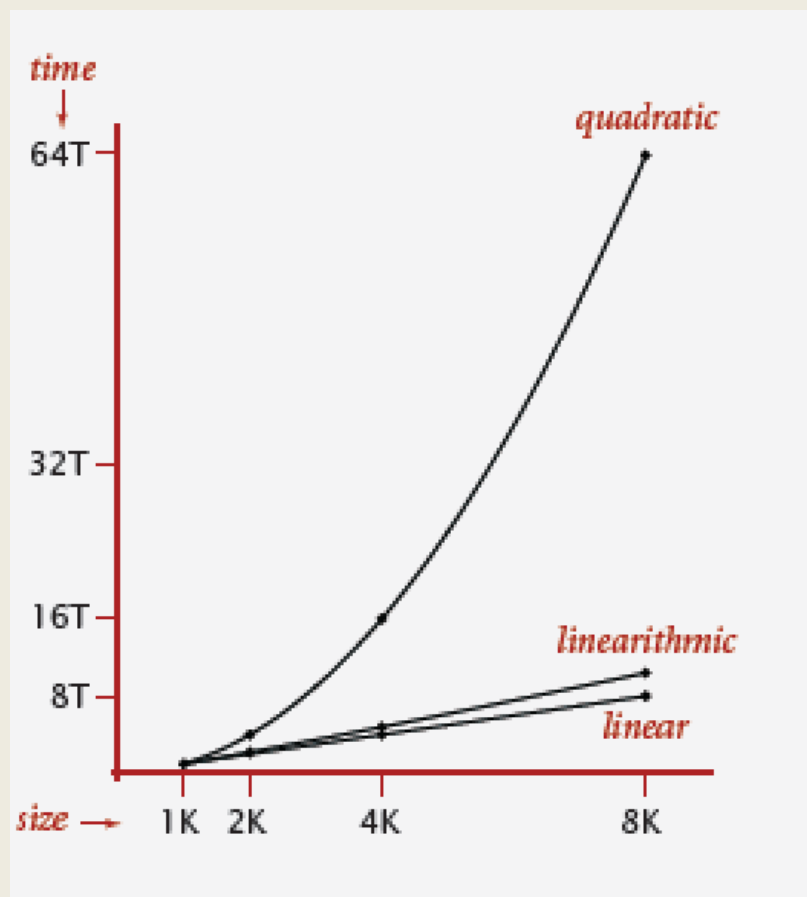
**a taxa de crescimento das
duas funções é a mesma**



Classes de funções importantes

[from Sedgewick&Wayne]

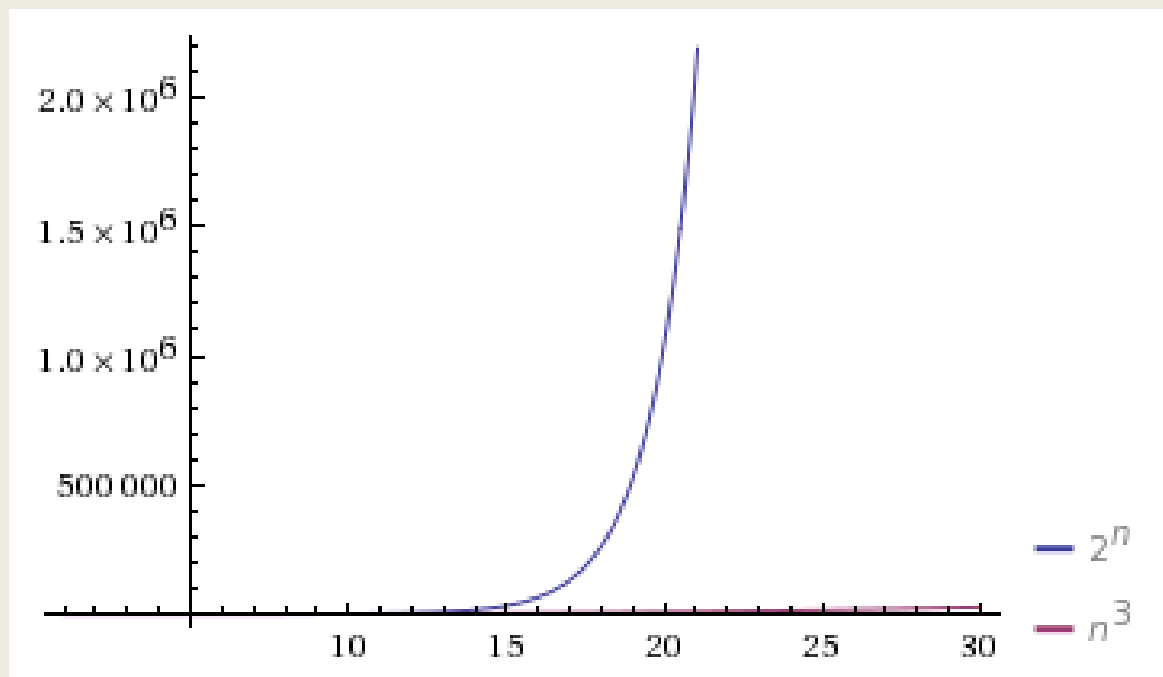
order of growth	name
1	constant
$\log N$	logarithmic
N	linear
$N \log N$	linearithmic
N^2	quadratic
N^3	cubic
2^N	exponential



Classes de funções importantes

[from Sedgewick&Wayne]

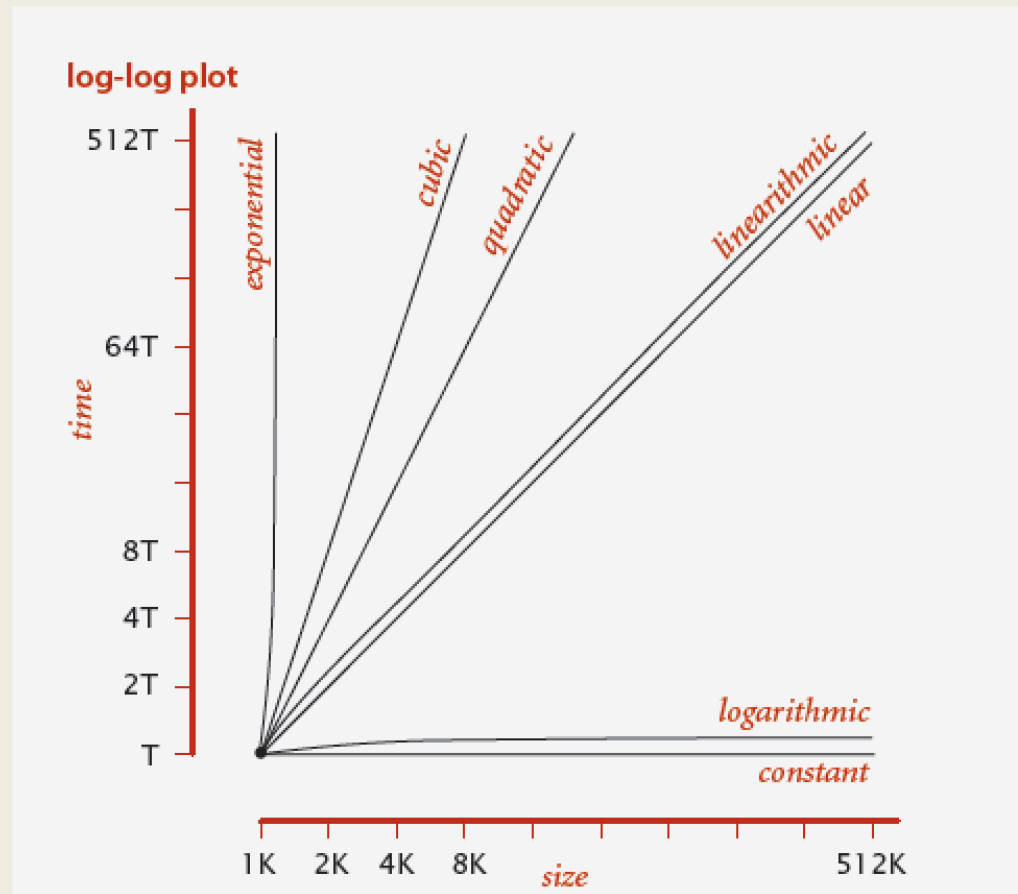
order of growth	name
1	constant
$\log N$	logarithmic
N	linear
$N \log N$	linearithmic
N^2	quadratic
N^3	cubic
2^N	exponential



Classes de funções importantes

[from Sedgewick&Wayne]

order of growth	name
1	constant
$\log N$	logarithmic
N	linear
$N \log N$	linearithmic
N^2	quadratic
N^3	cubic
2^N	exponential



O que isto significa na prática

Interpretando o valor de f em microsegundos ($10^{-6}s$) temos:

$f \setminus n$	20	30	40	60	1000	100000
n	.00002 s	.00003 s	.00004 s	.00006 s	.001 s	.1 s
n^2	.0004 s	.0009 s	.0016 s	.0036 s	1 s	3 h
n^3	.008 s	.027 s	.064 s	.216 s	17 mn	31 anos
n^5	3.2 s	24.3 s	1.7 mn	13 mn	31 anos	10^{10} sec
2^n	1 s	7.9 mn	12.7 dias	366 sec
3^n	58 mn	6.5 anos	3855 sec	10^{13} sec

- Ter um computador “100x mais rapido” é equivalente a dividir por 100 a função f
- Inventar algoritmos melhores dá melhor resultado do que arranjar *hardware* mais potente

Classes de funções importantes

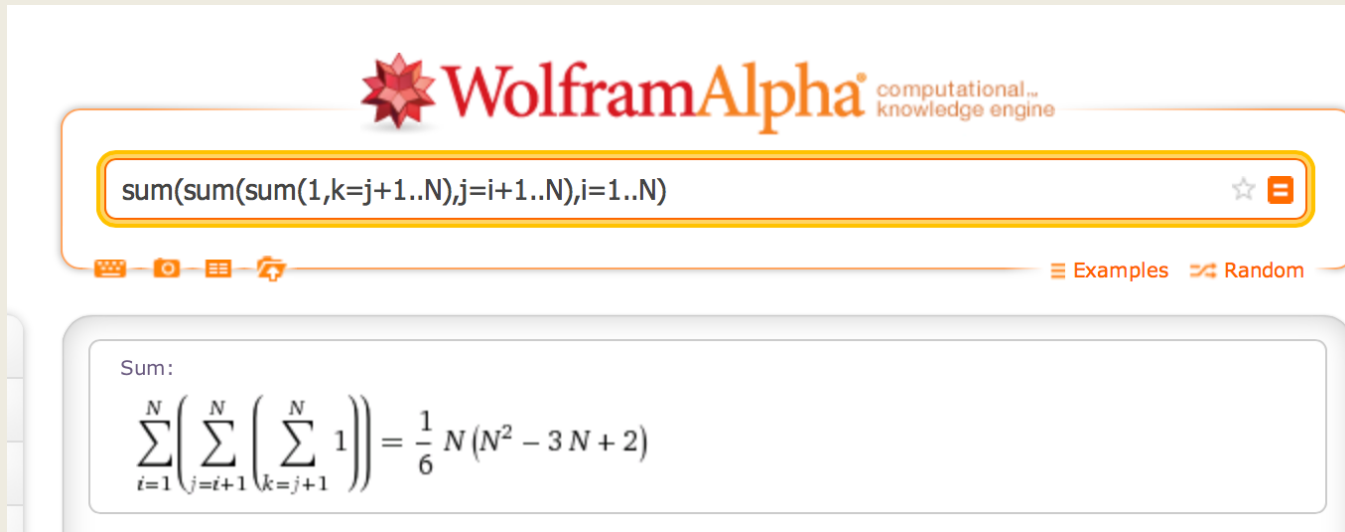
[from Sedgewick&Wayne]

order of growth	name	typical code framework	description	example	$T(2N) / T(N)$
1	constant	<pre>a = b[0] + b[1];</pre>	statement	add two array elements	1
$\log N$	logarithmic	<pre>while (N > 1) { N = N / 2; ... }</pre>	divide in half	binary search	~ 1
N	linear	<pre>for (int i = 0; i < N; i++) { ... }</pre>	loop	find the maximum	2
$N \log N$	linearithmic	[see mergesort lecture]	divide and conquer	mergesort	~ 2
N^2	quadratic	<pre>for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) { ... }</pre>	double loop	check all pairs	4
N^3	cubic	<pre>for (int i = 0; i < N; i++) for (int j = 0; j < N; j++) for (int k = 0; k < N; k++) { ... }</pre>	triple loop	check all triples	8
2^N	exponential	[see combinatorial search lecture]	exhaustive search	check all subsets	$T(N)$

Importante relembrar

$$\sum_{i=0}^n i = \frac{n \cdot (n + 1)}{2}$$

$$\sum_{i=1}^n a^i = \frac{a^{n+1} - a}{a - 1}$$



The screenshot shows the WolframAlpha interface. The input field contains the expression `sum(sum(sum(1,k=j+1..N),j=i+1..N),i=1..N)`. Below the input field, the result is displayed as a nested sum formula:
$$\sum_{i=1}^N \left(\sum_{j=i+1}^N \left(\sum_{k=j+1}^N 1 \right) \right) = \frac{1}{6} N (N^2 - 3N + 2)$$

$$x = \log_b n \text{ sse } b^x = n$$

$$\log_2 n = \log n \quad \log_b a = \frac{\log_c a}{\log_c b}$$

$$\sum_{i=1}^N \frac{1}{i} < \log_e N + 1$$

Análise Teórica vs Experimental

- Os limites teóricos são usados como referência e são úteis em muitas situações
- Mas não servem para dar estimativas do desempenho de uma implementação
 - ignoradas constantes multiplicativas
 - não captura aspetos que influenciam fortemente o desempenho

