

# Algoritmos e Estruturas de Dados

análise de complexidade

2010/2011

Carlos Lisboa Bento

## Análise de Complexidade

Conceitos

- Depois de determinarmos a correcção de um algoritmo o passo seguinte é determinar:
  - o tempo
  - o espaço
- requeridos para a sua execução.

Em geral

mais dados de entrada



mais tempo de processamento

# Análise de Complexidade

Conceitos

Exemplo: consideremos o problema de carregar um ficheiro via INTERNET.

Consideremos que o tempo de ligação são 2 seg e que o carregamento se processa a 30 Mb/seg.

Sendo a dimensão do ficheiro N Mbits temos:

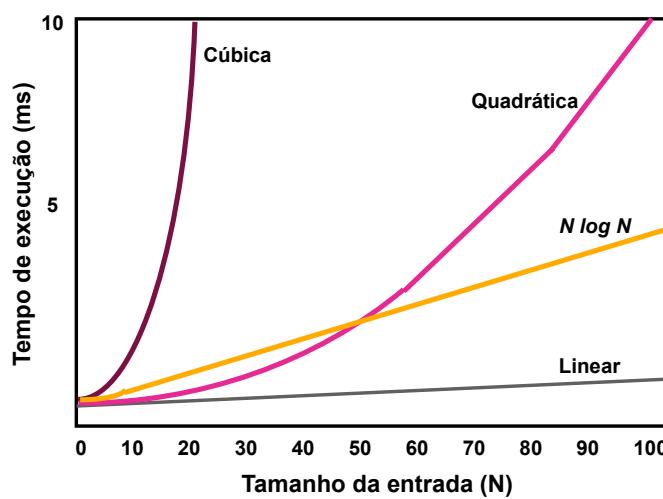
$$T(N) = N/30 + 2$$

(complexidade linear! É o que em geral desejamos... ou ainda melhor logarítmica)

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

Conceitos



© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Conceitos

Funções por ordem crescente de taxa de crescimento

$c$	constante
$\log N$	logarítmica
$\log^2 N$	logarítmica quadr
$N$	linear
$N \log N$	$N \log N$
$N^2$	quadrática
$N^3$	cúbica
$2^N$	exponencial

Algoritmos quadráticos são impraticáveis para entradas que excedam alguns milhares de elementos

Algoritmos cúbicos são impraticáveis para entradas que excedam algumas centenas de elementos

CONCLUSÃO: antes de consumir tempo optimizando o código é mais útil procurar optimizar o algoritmo

# Análise de Complexidade

## Conceitos

Exemplo de função cúbica

$$10N^3 + N^2 + 40N + 80$$

Consideremos os caso de

$N=2$

$$10N^3 + N^2 + 40N + 80 = 244$$

$$10N^3 = 80$$

$N=10$

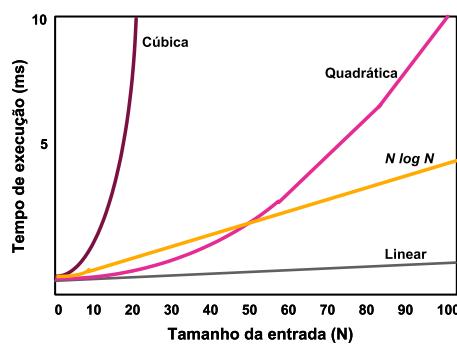
$$10N^3 + N^2 + 40N + 80 = 10.580$$

$$10N^3 = 10.000$$

$N=1000$

$$10N^3 + N^2 + 40N + 80 = 10.001.040.080$$

$$10N^3 = 10.000.000.000$$



CONCLUSÃO: para valores elevados de N o termo  $N^3$  domina o valor da função >> interessa-nos então ter uma medida de complexidade assintótica.

A notação O-grande é usada para representar a taxa de crescimento.

# Análise de Complexidade

## Conceitos

O – Grande (Paul Bachmann 1894)

Definição

$f(n)$  é  $O(g(n))$  se existirem valores  $c$  e  $N$  tais que  $f(n) \leq cg(n)$  para todo o  $n \geq N$ .

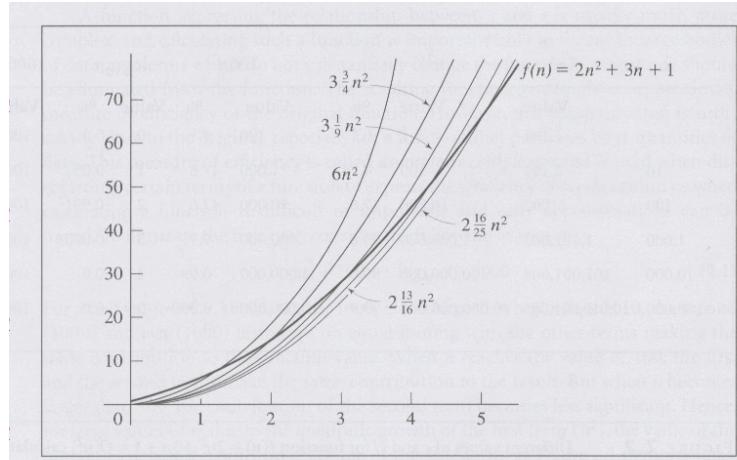
© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Conceitos

O – Grande

$f(n)$  é  $O(g(n))$  se existirem valores  $c$  e  $N$  tais que  $f(n) \leq cg(n)$  para todo o  $n \geq N$ .



© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Conceitos

### O – Grande (propriedades)

- Se  $f(n)$  é  $O(g(n))$  e  $g(n)$  é  $O(h(n))$ , então  $f(n)$  é  $O(h(n))$ .
- Se  $f(n)$  é  $O(h(n))$  e  $g(n)$  é  $O(h(n))$ , então  $f(n)+g(n)$  é  $O(h(n))$ .
- A função  $a_n^k$  é  $O(n^k)$ .
- A função  $n^k$  é  $O(n^{k+j})$  para todo o  $j$  positivo.

Segue-se que

$$f(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0 \text{ é } O(n^k)$$

# Análise de Complexidade

## Conceitos

### O – Grande (cálculo)

- Se  $f(n)$  é um polinómio de grau  $d$ , então  $f(n)$  é  $O(n^d)$ :
  - eliminar no polinómio termos de ordem inferior;
  - eliminar no polinómio constantes
- Usar as classes de funções de ordem mais baixa possível:
  - considerar “ $2n$  é  $O(N)$  e não “ $2n$  é  $O(N^2)$ ”
- Usar a expressão mais simples da classe de funções:
  - considerar “ $3n+5$  é  $O(N)$  e não “ $3n+5$  é  $O(3N)$ ”

# Análise de Complexidade

## Exemplos

### Prob #1: MENOR ELEMENTO NUM ARRAY

Dado um array de N elementos encontrar o mínimo elemento.

*(Este problema é de grande importância nas ciências da computação)*

Um possível algoritmo:

1. Criar uma variável min que guarda o menor elemento.
2. Inicializar min com o primeiro elemento do array.
3. Fazer uma pesquisa sequencial no array e actualizar a variável min de acordo.

# Análise de Complexidade

## Exemplos

### Prob #2: PONTO MAIS PRÓXIMO NUM PLANO

Dados N pontos num plano (um sistema de coordenadas x-y) encontrar o par de pontos mais próximos.

*(Este problema é de grande importância em processamento gráfico)*

Um possível algoritmo:

1. Calcular a distância entre cada par de pontos.
2. Guardar a distância mínima.

Tempo de execução?

- Existem  $N(N-1)/2$  pares de pontos -> Complexidade quadrática  $O(N^2)$
- Existe um algoritmo melhorado que corre em  $O(N \log N)$
- Existe ainda um algoritmo cujo tempo esperado é de  $O(N)$

# Análise de Complexidade

## Exemplos

### Prob #3: PONTOS COLINEARES NUM PLANO

Dados N pontos num plano (um sistema de coordenadas x-y) determinar se existem três pontos sobre uma mesma linha.

*(Este problema é de grande importância em muitos algoritmos de processamento gráfico. Isto porque a existência de pontos colineares introduz um caso degenerado que necessita de tratamento especial)*

Um possível algoritmo:

1. Considerar todos os grupos de três pontos.

Tempo de execução?

# Análise de Complexidade

## Conceitos

Cálculo da Complexidade Assimptótica → Análise assimptótica

- A análise assimptótica de um algoritmo determina o tempo de execução na notação O-grande.
- Para realizar a análise assimptótica:
  - Calculamos a função que descreve o número de operações primitivas.
  - Exprimimos esta função em termos da notação O-grande.
- Exemplo: um determinado algoritmo executa  $6n^2 - 3n - 2$  operações primitivas. Dizemos então que o algoritmo tem complexidade O-grande  $O(N^2)$ .
- Como constantes e termos de ordem inferior não são considerados podemos não os tomar em conta quando da contagem do número de funções primitivas.

# Análise de Complexidade

## Exemplos

Cálculo da complexidade assintótica

Em geral pretendemos saber a COMPLEXIDADE TEMPORAL relativa ao número de atribuições e comparações realizadas durante a execução do programa

PARA JÁ VAMOS SÓ CONSIDERAR O NÚMERO DE ATRIBUIÇÕES

Exemplo #1

```
for(i = sum = 0; i < n; i++)
    sum += a[i];
```

Data Structures and Algorithms in JAVA, Adam Drozdek

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

$$O(n)$$

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Exemplos

Cálculo da complexidade assintótica

Exemplo #2

```
for(i = 0; i < n; i++) {
    for(j = 1, sum = a[0]; j <= i; j++)
        sum += a[j];
    System.out.println ("sum for subarray 0 through "+i+" is" + sum);
}
```

Data Structures and Algorithms in JAVA, Adam Drozdek

$$T(n) = 1 + 3n + \sum_{i=1}^{n-1} 2i =$$

$$1 + 3n + 2(1+2+\dots+n-1) =$$

$$1 + 3n + n(n-1)$$

$$O(1) + O(n) + O(n^2) = O(n^2)$$

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Exemplos

Cálculo da complexidade assintótica

Exemplo #3

```
for(i = 4; i < n; i++) {  
    for(j = i-3, sum = a[i-4]; j <= i; j++)  
        sum += a[j];  
    System.out.println ("sum for subarray "+(i - 4)+" through "+i+" is"+ sum);  
}
```

Data Structures and Algorithms in JAVA, Adam Drozdek

$$T(n) = 1 + (3 + 2^*4)(n - 4) = 1 + 11(n - 4)$$

$O(n)$

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Exemplos

Cálculo da complexidade assintótica

Exemplo #1 – Soma dos  $n$  elementos de um array

```
for(i = sum = 0; i < n; i++)  
    sum += a[i];
```

Data Structures and Algorithms in JAVA, Adam Drozdek

- Nos exemplos 1 a 3 → o número de vezes que os ciclos são executados não depende da ordem porque estão os elementos no array

Exemplo #2 – Sequência aditiva para os elementos 0, 0..1, 0..2, ..., 0..n

```
for(i = 0; i < n; i++) {  
    for(j = 1, sum = a[0]; j <= i; j++)  
        sum += a[j];  
    System.out.println ("sum for subarray 0 through "+i+" is" + sum);  
}
```

Data Structures and Algorithms in JAVA, Adam Drozdek

Exemplo #3 – Sequência aditiva para os subarrays 0..4, 1..5, ..., n-4..n

```
for(i = 4; i < n; i++) {  
    for(j = i-3, sum = a[i-4]; j <= i; j++)  
        sum += a[j];  
    System.out.println ("sum for subarray "+(i - 4)+" through "+i+" is"+ sum);  
}
```

Data Structures and Algorithms in JAVA, Adam Drozdek

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Exemplos

### Cálculo da complexidade assintótica

Exemplo #4 – procura da maior sequência crescente num array de inteiros

```
for(i = 0, length = 1; i < n-1; i++) {  
    for(il = i2 = k = i; k < n-1 && a[k] < a[k+1]; k++,i2++);  
    if(length < i2 - il + 1)  
        length = i2 - il + 1;  
    System.out.println ("the length of the longest ordered subarray is" + length);  
}  
.
```

Data Structures and Algorithms in JAVA, Adam Drozdek

**Todos os números por ordem decrescente**

**O (n)**

**Todos os números por ordem crescente**

**O (n<sup>2</sup>)**

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Exemplos

### Cálculo da complexidade assintótica

Exemplo #5 – busca binária num array ordenado

```
int binarySearch(int[] arr, int key) {  
    int lo = 0, mid, hi = arr.length-1;  
    while (lo <= hi) {  
        mid = (lo + hi)/2;  
        if (key < arr[mid])  
            hi = mid - 1;  
        else if (arr[mid] < key)  
            lo = mid + 1;  
        else return mid; // success: return the index of  
    } // the cell occupied by key;  
    return -1; // failure: key is not in the array;  
}
```

Data Structures and Algorithms in JAVA, Adam Drozdek

**Chave no ponto central do array:**

**T(n) = cte**

**O (1)**

**Chave não existente no array:**

$$n, \frac{n}{2}, \frac{n}{2^2}, \dots, \frac{n}{2^m}$$

**m = log n**

**O (log N)**

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

Melhor, Médio e Pior Casos

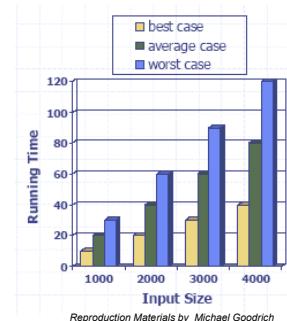
Para algoritmos como os que temos para os ex. 4 e ex. 5 necessitamos de analisar:

- Melhor Caso
- Caso Médio
- Pior Caso

$$\text{Caso Médio } C_M = \sum_i p(\text{entr}_i) * n\_passo(\text{entr}_i)$$

Exemplo:

Procura sequencial de uma chave num array não ordenado.



Reproduction Materials by Michael Goodrich

Melhor Caso – chave encontrada no primeiro elemento do array.

Pior Caso – chave encontrada no último elemento do array ou inexistente no array.

Caso Médio – vamos assumir que todas as posições do array têm a mesma probabilidade de terem a chave (distribuição uniforme de probabilidades)

$$p(\text{entr}_i) = \frac{1}{n} \text{ p/ } i = 1, \dots, n \quad n\_passo(\text{entr}_i) = i \quad C_M = \frac{1+2+\dots+n}{n} = \frac{n+1}{2} \quad O(n)$$

# Análise de Complexidade

Conceitos

$$T(N) = o(F(N)) \quad \text{crescimento de } T(N) < \text{crescimento de } F(N)$$

Oh pequeno

$$T(N) = O(F(N))$$

$$\text{crescimento de } T(N) \leq \text{crescimento de } F(N)$$

$$T(N) \leq c * F(N) \text{ p/ } N \geq N_0$$

Oh grande

$$T(N)$$

$$T(N) = \Theta(F(N))$$

$$\text{crescimento de } T(N) = \text{crescimento de } F(N)$$

Theta grande

$$T(N) = \Omega(F(N))$$

$$\text{crescimento de } T(N) \geq \text{crescimento de } F(N)$$

$$T(N) \geq c * F(N) \text{ p/ } N \geq N_0$$

Omega grande

Na definição de  $F(N)$  em Oh-grande eliminar constantes, termos de ordem inferior e conectivas relacionais.

Ex.:  $T(N) = 10N^3 + N^2 + 40N + 80$  é  $O(N^3)$

# Análise de Complexidade

## Aplicação

A análise da complexidade de um algoritmo permite-nos fazer algo de incontornável que é procurar algoritmos cuja complexidade seja computacionalmente aceitável.

### CONSIDEREMOS UM CASO PRÁTICO

Problema: MÁXIMA SUBSEQUÊNCIA CONTINUA ADITIVA

Dada uma sequência de inteiros (eventualmente negativos) A1,A2,...,AN, encontrar (e identificar a sequência correspondente a) máximo valor de  $\sum_{k=i}^j A_k$

A subsequência é zero se todos os inteiros forem negativos

Exemplo: { -2, 11, -4, 13, -5, 2 }    { 1, -3, 4, -2, -1, 6 }

# Análise de Complexidade

## Aplicação

A análise da complexidade de um algoritmo permite-nos fazer algo de incontornável que é procurar algoritmos cuja complexidade seja computacionalmente aceitável.

### CONSIDEREMOS UM CASO PRÁTICO

Este problema tem:

- solução óbvia em O(N<sup>3</sup>)
- ... menos óbvia em O(N<sup>2</sup>)
- ... mais elaborada em O(N)

>>> Estudar as várias soluções

>>> Fazer o paralelo com o TP1

# Análise de Complexidade

## Aplicação

A análise da complexidade de um algoritmo permite-nos fazer algo de incontornável que é procurar algoritmos cuja complexidade seja computacionalmente aceitável.

### CONSIDEREMOS UM CASO PRÁTICO

Problema: MÁXIMA SUBSEQUÊNCIA CONTINUA ADITIVA

Dada uma sequência de inteiros (eventualmente negativos) A1,A2,...,AN, encontrar (e identificar a sequência correspondente a) máximo valor de  $\sum_{k=i}^j A_k$

A subsequência é zero se todos os inteiros forem negativos

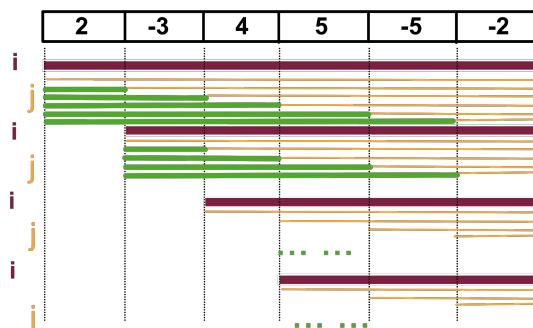
Exemplo: { -2, 11, -4, 13, -5, 2 }    { 1, -3, 4, -2, -1, 6 }

© DEI Carlos Lisboa Bento    ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Aplicação

```
public static int maxSubSum1( int [] a )
{
    int maxSum = 0;
    for( int i = 0; i < a.length; i++ )
        for( int j = i; j < a.length; j++ )
        {
            int thisSum = 0;
            for( int k = i; k <= j; k++ )
                thisSum += a[ k ];
            if( thisSum > maxSum )
            {
                maxSum = thisSum;
                seqStart = i;
                seqEnd = j;
            }
        }
    return maxSum;
}
```



#### Características:

- Pesquisa exaustiva
- Tempo de execução  $O(N^3)$

DESAFIO: Será que conseguimos melhorar o algoritmo em termos de complexidade temporal ?

© DEI Carlos Lisboa Bento    ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Aplicação

```
public static int maxSubSum1( int [ ] a )
{
    int maxSum = 0;

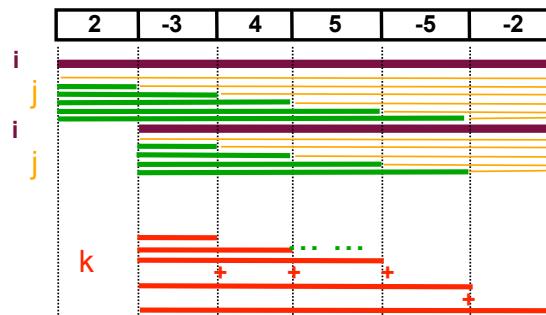
    for( int i = 0; i < a.length; i++ )
        for( int j = i; j < a.length; j++ )
        {
            int thisSum = 0;

            for( int k = i; k <= j; k++ )
                thisSum += a[ k ];

            if( thisSum > maxSum )
            {
                maxSum = thisSum;
                seqStart = i;
                seqEnd = j;
            }
        }

    return maxSum;
}
```

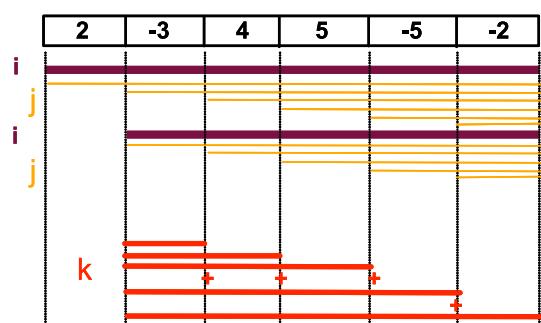
Em geral quando conseguimos eliminar o ciclo mais interno num algoritmo reduzimos a complexidade temporal



© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Aplicação



```
public static int maxSubSum1( int [ ] a )
{
    int maxSum = 0;

    for( int i = 0; i < a.length; i++ )

        int thisSum = 0;

        for( int j = i; j < a.length; j++ )
        {
            thisSum += a[ j ];

            if( thisSum > maxSum )
            {
                maxSum = thisSum;
                seqStart = i;
                seqEnd = j;
            }
        }

    return maxSum;
}
```

DESAFIO PARCIALMENTE ULTRAPASSADO: conseguimos reduzir a complexidade do algoritmo para  $O(N^2)$  !!!

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Aplicação

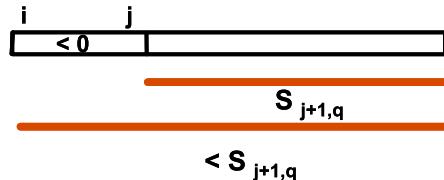
```
public static int maxSubSum1( int [ ] a )
{
    int maxSum = 0;

    for( int i = 0; i < a.length; i++ )
        int thisSum = 0;
        for( int j = i; j < a.length; j++ )
        {
            thisSum += a[ j ];
            if( thisSum > maxSum )
            {
                maxSum = thisSum;
                seqStart = i;
                seqEnd = j;
            }
        }
    return maxSum;
}
```

**Obs.:** o algoritmo quadrático ainda corresponde a uma pesquisa exaustiva

**Questão:** encontrar subsquências que não interessam calcular.

Vejamos !!



1. Todas as subsequências que confinam com a subsequência de soma máxima têm soma negativa ou igual a zero.

# Análise de Complexidade

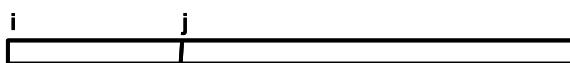
## Aplicação

Mais !!

... ...

$S_{i,j} > 0$

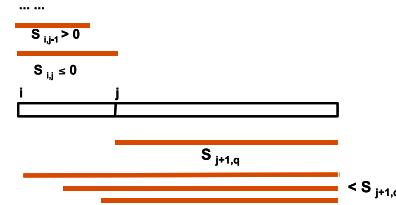
$S_{i,j} \leq 0$



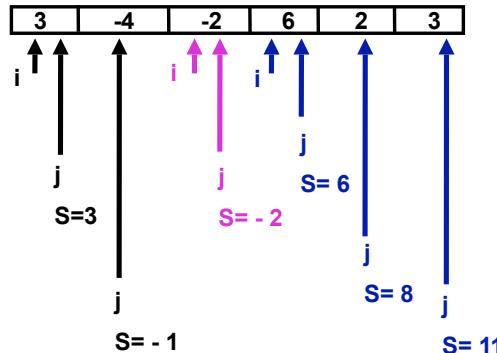
2. Qualquer destas subsequências não é uma subsequência de soma máxima ou é igual a uma subsequência de soma máxima já encontrada.

# Análise de Complexidade

Aplicação



Decorre...



← Este algoritmo tem complexidade linear ! Ufffff!!!

# Análise de Complexidade

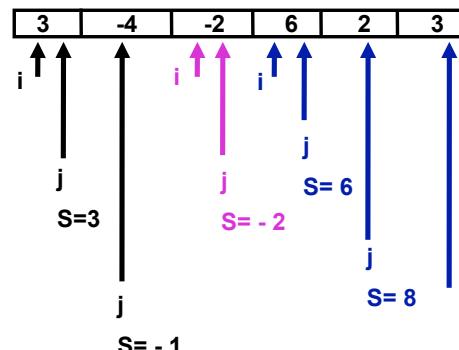
Aplicação

```
public static int maxSubSum3( int [] a )
{
    int maxSum = 0;
    int thisSum = 0;

    for( int i = 0, j = 0; j < a.length; j++ )
    {
        thisSum += a[ j ];

        if( thisSum > maxSum )
        {
            maxSum = thisSum;
            seqStart = i;
            seqEnd = j;
        }
        else if( thisSum < 0 )
        {
            i = j + 1;
            thisSum = 0;
        }
    }

    return maxSum;
}
```



# Análise de Complexidade

## Aplicação

Tempos de execução em segundos numa determinada máquina para os vários algoritmos de **máxima subsequência continua**

N	$O(N^3)$	$O(N^2)$	$O(N)$
10	0.00103	0.00045	0.00034
100	0.47015	0.01112	0.00064
1000	448.77	1.1233	0.00333
10000	ND	111.13	0.03042
100000	ND	ND	0.29832

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Aplicação

Tempos de execução em segundos para os vários algoritmos de **máxima subsequência contínua**

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Aplicação

... mais tempos de execução para diferentes complexidades, usando uma máquina com tempo de execução por instrução de 1  $\mu$ seg.

Class	Complexity	Number of Operations and Execution Time (1 instr/ $\mu$ sec)			
	$n$	$10^0$	$10^2$	$10^3$	
constant	$O(1)$	1	1 $\mu$ sec	1	1 $\mu$ sec
logarithmic	$O(\lg n)$	3.32	3 $\mu$ sec	6.64	7 $\mu$ sec
linear	$O(n)$	10	10 $\mu$ sec	$10^2$	100 $\mu$ sec
$O(n \lg n)$	$O(n \lg n)$	33.2	33 $\mu$ sec	664	664 $\mu$ sec
quadratic	$O(n^2)$	$10^2$	100 $\mu$ sec	$10^4$	10 msec
cubic	$O(n^3)$	$10^3$	1 msec	$10^6$	1 sec
exponential	$O(2^n)$	1024	10 msec	$10^{30}$	$3.17 * 10^{17}$ yrs
	$n$	$10^4$	$10^5$	$10^6$	
constant	$O(1)$	1	1 $\mu$ sec	1	1 $\mu$ sec
logarithmic	$O(\lg n)$	13.3	13 $\mu$ sec	16.6	7 $\mu$ sec
linear	$O(n)$	$10^4$	10 msec	$10^5$	0.1 sec
$O(n \lg n)$	$O(n \lg n)$	$133 * 10^3$	133 msec	$166 * 10^4$	1.6 sec
quadratic	$O(n^2)$	$10^8$	1.7 min	$10^{10}$	16.7 min
cubic	$O(n^3)$	$10^{12}$	11.6 days	$10^{15}$	31.7 yr
exponential	$O(2^n)$	$10^{3010}$		$10^{30103}$	$10^{301030}$

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Equações de Recorrências

Algoritmos RECURSIVOS :: Como calcular a sua complexidade? :: EQUAÇÕES DE RECORRÊNCIA

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Equações de Recorrências

Eq1: programa que percorre recorrentemente os dados com eliminação de um elemento a cada passagem

$$C_N = C_{N-1} + N$$

...

$$C_N \text{ aprox. } N^2/2$$

$$C_N = C_{N-1} + N$$

$$= C_{N-2} + (N-1) + N$$

$$= C_{N-3} + (N-2) + (N-2) + N$$

...

$$= 1 + 2 + \dots + (N-1) + N$$

$$= N(N+1)/2$$

...

$$C_N \text{ aprox. } N^2/2$$

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

## Equações de Recorrências

Eq2: programa que recursivamente parte a entrada ao meio

$$C_N = C_{N/2} + 1$$

...

$$C_N \text{ aprox. } \lg N$$

$$C_{2^n} = C_{2^{n-1}} + 1$$

$$= C_{2^{n-2}} + 1 + 1$$

...

$$= n + 1$$

$$C_N \text{ aprox. } \lg N$$

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

Equações de Recorrências

Eq3: programa que recursivamente parte a entrada ao meio, mas que tem de examinar cada um dos elementos de entrada

$$C_N = C_{N/2} + N$$

...

$$C_N \text{ aprox. } 2N$$

Eq4: programa que percorre linearmente a entrada e que antes durante ou de seguida recursivamente parte a entrada em duas metades

$$C_N = 2C_{N/2} + N$$

...

$$C_N \text{ aprox. } N \lg N$$

# Análise de Complexidade

Equações de Recorrências

Eq5: programa que recursivamente parte a entrada em duas metades e que de seguida tem de realizar uma quantidade fixa de operações

$$C_N = 2C_{N/2} + 1$$

...

$$C_N \text{ aprox. } 2N$$

# Análise de Complexidade

Problemas NP Completos

## Problemas TRATÁVEIS (POLINOMIALMENTE SOLÚVEIS)

- Pior caso, com complexidade temporal na forma  $O(N^k)$ , com  $k$  tipicamente igual a 2 ou 3.
- Designam-se por problemas do tipo P ou TRATÁVEIS
- Um programa do tipo P que recorre a outro de tipo P mantém-se do tipo P
- Tratável >> tem uma solução escalável (tempos não crescem abruptamente, como acontece quando esse crescimento é exponencial)

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

Problemas NP-Completos

## Problemas INTRATÁVEIS

- Problemas NP-Completos são uma subclasse dos problemas intratáveis
  - Só se conhecem soluções para estes problemas em tempo exponencial (ex.:  $O(2^{O(n^k)})$  )
  - Se fosse possível resolver um problema NP-completo em tempo polinomial então seria possível resolver todos os problemas NP-completos em tempo polinomial
  - $P = NP?$  (um prémio de 1 M\$ para quem conseguir provar isto ou  $P \neq NP$ , [www.claymath.org/millennium](http://www.claymath.org/millennium))
  - $NP =$  Non-deterministic Polynomial Time

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

Problemas NP-Completos

## Porque estudar este tipo de problemas?

- Se detectarmos que um problema é do tipo NP-Completo temos três possibilidades:

- aceitar que o nosso algoritmo tem comportamento temporalmente exponencial
- avançar para o estudo de uma solução aproximada em vez de procurar a solução óptima
- alterar as restrições do problema no sentido de que este se torne do tipo P

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

Problemas NP-Completos

## Como provamos que um problema é NP-completo?

- Pegar num problema NP-completo conhecido e reduzido ao nosso problema
- Alguns problemas do tipo NP-completo:
  - Boolean satisfiability problem (SAT)
  - N-puzzle
  - Knapsack problem
  - TSP
  - Subgraph isomorphism problem
  - Subset sum problem
  - Clique problem
  - Graph coloring problem

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

Problemas NP-Completos

Algumas técnicas para resolver problemas do tipo  
NP-completo

- APROXIMAÇÃO :: em vez de procurar uma solução óptima procurar uma quase óptima
- RESTRIÇÃO :: restringindo a estrutura da entrada (ex.: considerar só grafos planares) algoritmos mais rápidos podem ser aplicáveis
- ALEATORIZAÇÃO :: utilizar pesquisa aleatória para obter tempos médios de execução mais baixos, podendo o algoritmo falhar a procura de uma solução com uma probabilidade baixa (ex.: método de Monte Carlo)
- PARAMETRIZAÇÃO: por vezes encontramos um algoritmo mais rápido se fixarmos algumas parametrizações na entrada
- HEURÍSTICAS :: um algoritmo que se comporta razoavelmente bem em muitos casos, mas para o qual não existe prova de ser sempre rápido nem de produzir sempre bons resultados (ex.: uso de meta-heurísticas)

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

Problemas NP-Completos

... tipos de problemas intratáveis

- PSPACE :: precisam de espaço polinomial
- EXPTIME :: têm crescimento exponencial no tempo
- INDECIDÍVEIS :: não existe algoritmo para a sua solução independentemente do tempo disponibilizado

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# Análise de Complexidade

Em Destaque

- Antes de consumir tempo optimizando o código é mais útil procurar optimizar o algoritmo.
- No algoritmo de determinação da MÁXIMA SUBSEQUÊNCIA CONTINUA ADITIVA temos:
  - O algoritmo de base tem complexidade  $O(N^3)$ .
  - Eliminando a repetição desnecessária de adições temos  $O(N^2)$ .
  - Eliminando sequências que não contribuem para o resultado temos  $O(N)$  (corresponde a deixar de fazer uma pesquisa exaustiva)
- Para além da expressão  $O(F(N))$  temos:
  - $\Theta(F(N))$
  - $\Omega(F(N))$
- Para algoritmos recursivos usamos equações de recorrências para o cálculo da complexidade
- Para problemas NP-completos algumas abordagens possíveis passam por:  
aproximação; aleatorização; inclusão de restrições; parametrização; recurso a heurísticas

© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

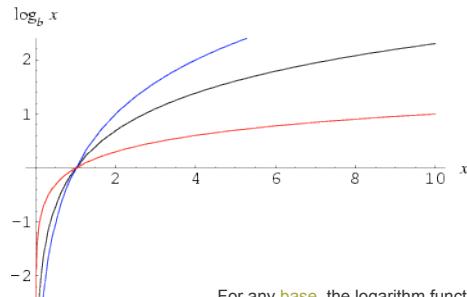
# Análise de Complexidade

... fim



© DEI Carlos Lisboa Bento ALGORITMOS E ESTRUTURAS DE DADOS

# [anexos]



For any base, the logarithm function has a singularity at  $x = 0$ .

In the above plot, the blue curve is the logarithm to base 2 ( $\log_2 x = \lg x$ ),  
the black curve is the logarithm to base e (the natural logarithm  $\log_e x = \ln x$  ),  
and the red curve is the logarithm to base 10 (the common logarithm, i.e., log,  $\log_{10} x = \log x$  ).

FONTE: <http://mathworld.wolfram.com/>