Algoritmos e Estruturas de Dados

análise de complexidade

LECD 2023-2024

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 5ms e que o carregamento se processa a 300 Mb/s.

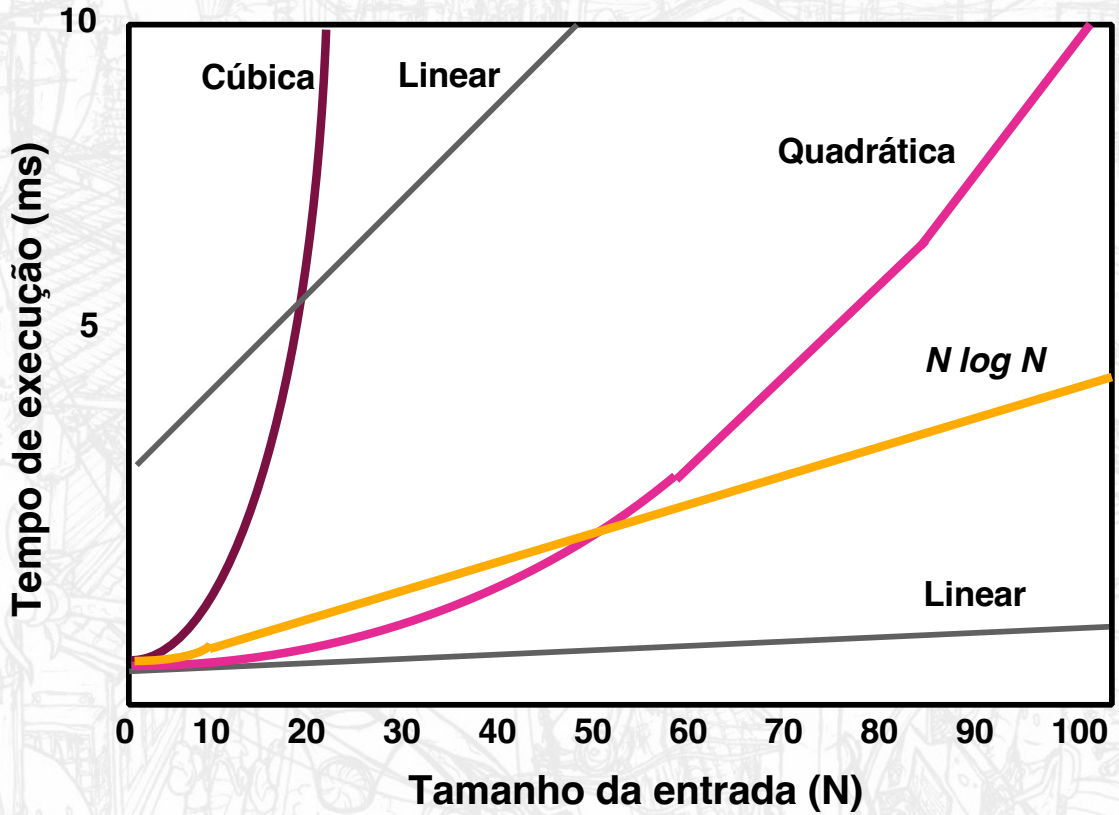
Sendo a dimensão do ficheiro N Mbits temos:

$T(N) = N/300 + 0.005$

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

Análise de Complexidade

Conceitos



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

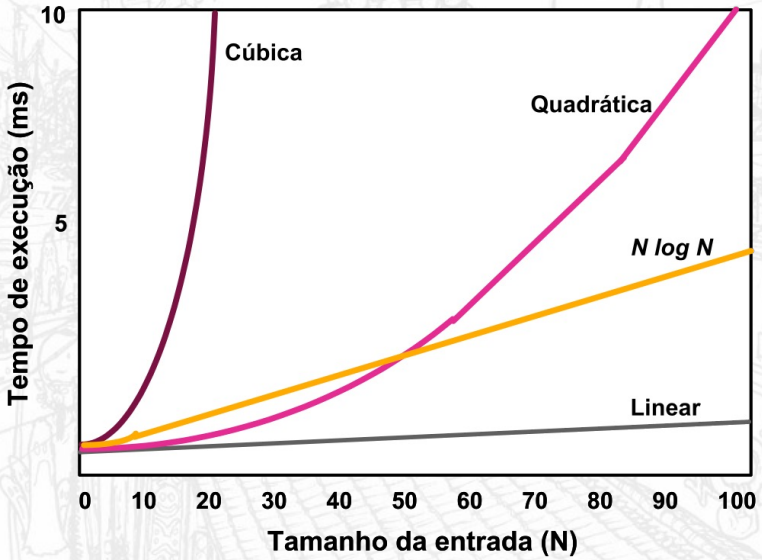
$10N^3 + N^2 + 40N + 80 = 10.580$

$10N^3 = 10.000$

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 assimptótica.

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

Análise de Complexidade

Conceitos

Funções por ordem crescente de taxa de crescimento

c	constante
log N	logaritmica
log² N	logaritmica quadr
N	linear
N log N	N log N
N^2	quadrática
N^3	cúbica
2^N	exponencial

Algoritmos de complexidade quadrática são impraticáveis para entradas que excedam alguns milhares de elementos

Algoritmos de complexidade cúbica são impraticáveis para entradas que excedam algumas centenas de elementos

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

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)≤cg(n) para todo o n≥N.

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 an^k é $O(n^k)$.
- A função n^k é $O(n^{k+j})$ para todo o j positivo.

Segue-se que

$f(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$ 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.

Tempo de execução?

- Linear *O(N)*

Análise de Complexidade

Exemplos

Prob #2: PONTO MAIS PRÓXIMO NUM PLANO

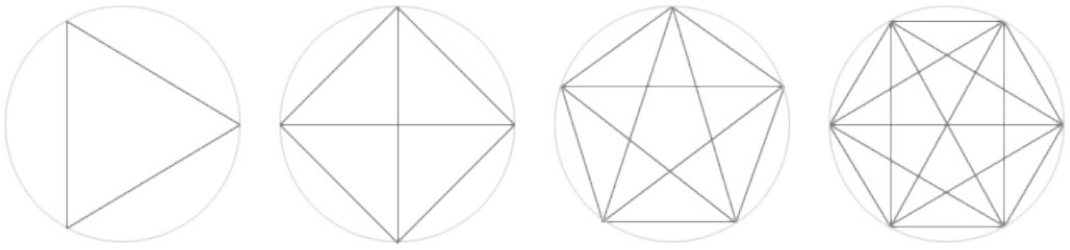
Dados N pontos num plano (um sistema de coordenadas x-y) encontrar o par de pontos mais próximos.

(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?



3 mediators
3(3-1)/2
3 links

4 mediators
4(4-1)/2
6 links

5 mediators
5(5-1)/2
10 links

6 mediators
6(6-1)/2
15 links

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

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 assimptótica

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

Análise de Complexidade
Exemplos

Cálculo da complexidade assimptótica

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

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

$1 + 3n + 2\,(1+2+\ldots + n-1) =$

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

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

Análise de Complexidade
Exemplos

Cálculo da complexidade assimptótica

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);
}
```

n- 4

Data Structures and Algorithms in JAVA, Adam Drozdek

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

O (n)

Análise de Complexidade

Exemplos

Cálculo da complexidade assimptótica

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

Data Structures and Algorithms in JAVA, Adam Drozdek

Análise de Complexidade

Exemplos

Cálculo da complexidade assimptótica

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: $n, \frac{n}{2}, \frac{n}{2^2}, \ldots, \frac{n}{2^m}$ **m = log n** **O (log N)**

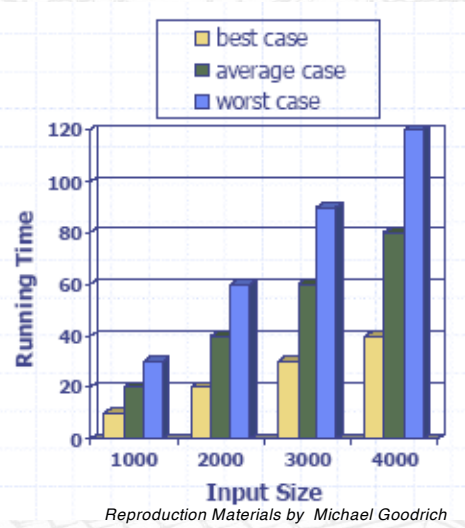
Análise de Complexidade

Melhor, Médio e Pior Casos

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

- Melhor Caso
- Caso Médio
- Pior Caso

Caso Médio $C_M = \sum_i p(entr_i) * n\_passo(entr_i)$



Reproduction Materials by Michael Goodrich

Exemplo:

Procura sequencial de uma chave num array não ordenado.

Melhor Caso – chave encontrada no primeiro elemento do array.

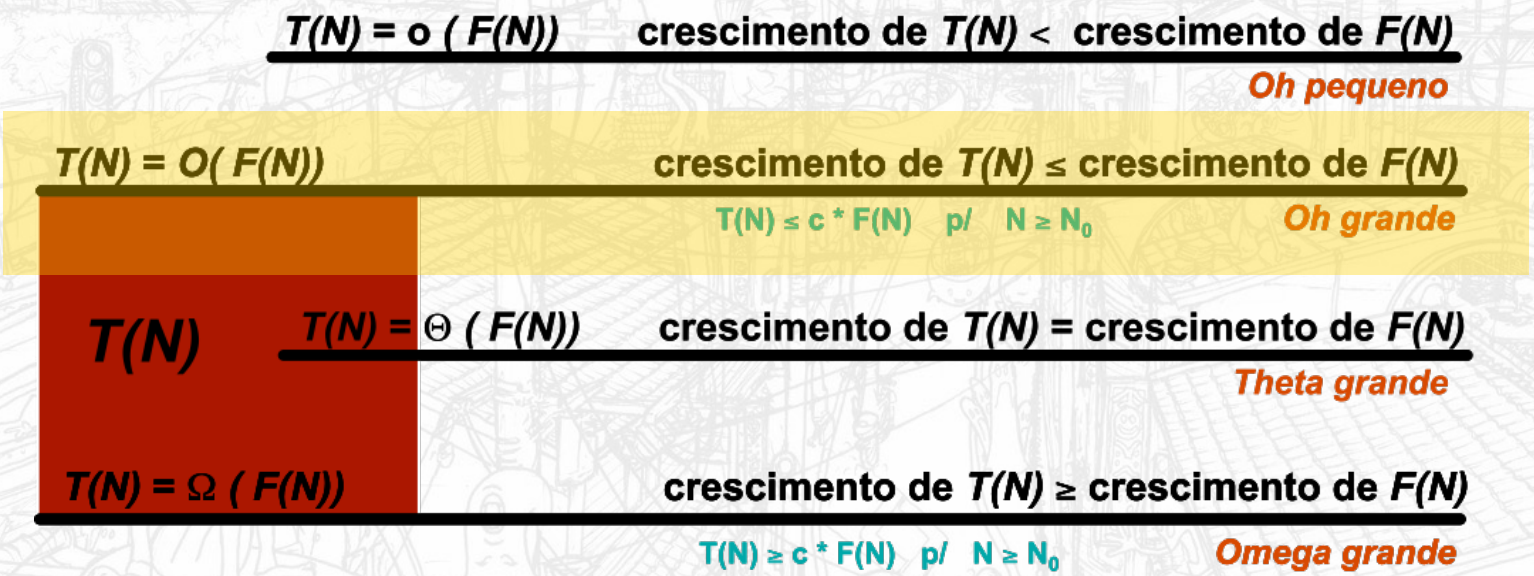
Pior Caso – chave encontrada no último elemento do array ou inexistente no array.

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

$$p(entr_i) = \frac{1}{n} \text{ p/ } i = 1,\ldots,n \qquad n\_passo(entr_i) = i \qquad C_M = \frac{1+2\ldots+n}{n} = \frac{n+1}{2} \qquad O(n)$$

Análise de Complexidade

Conceitos



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 $O(N^3)$

Design and Analysis of Algorithms I

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(N3)
- ... menos óbvia em O(N2)
- ... mais elaborada em O(N)

>>> Estudar as várias soluções

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 $\sum_{k=i}^{j} A_K$

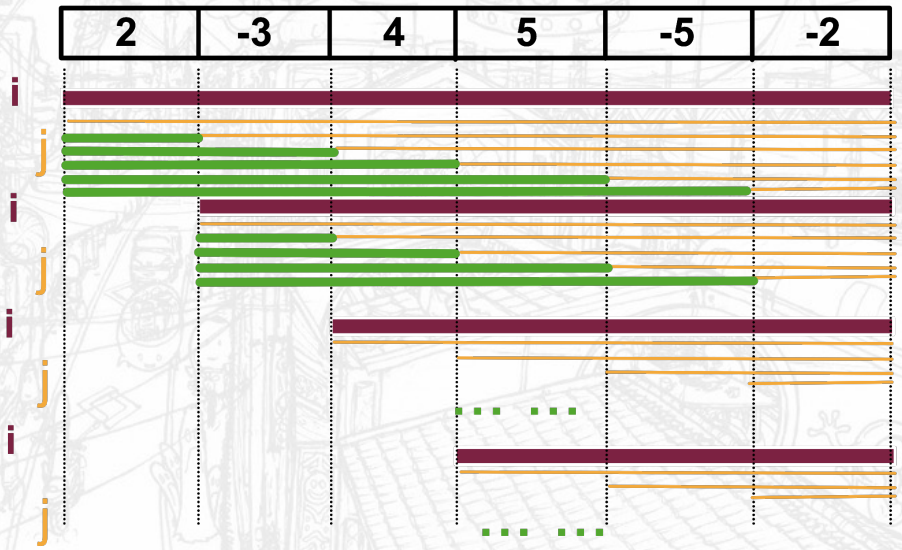
A subsequência é zero se todos os inteiros forem negativos

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

Análise de Complexidade

Aplicação

```
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;
  }
```



Caracteristicas:

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

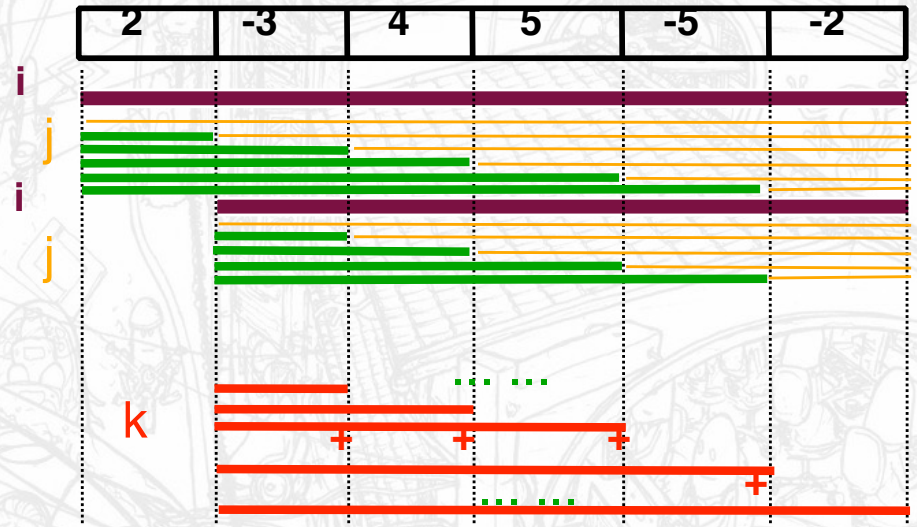
DESAFIO: Será que conseguimos melhorar o algoritmo em termos de complexidade temporal ?

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;
  }
```

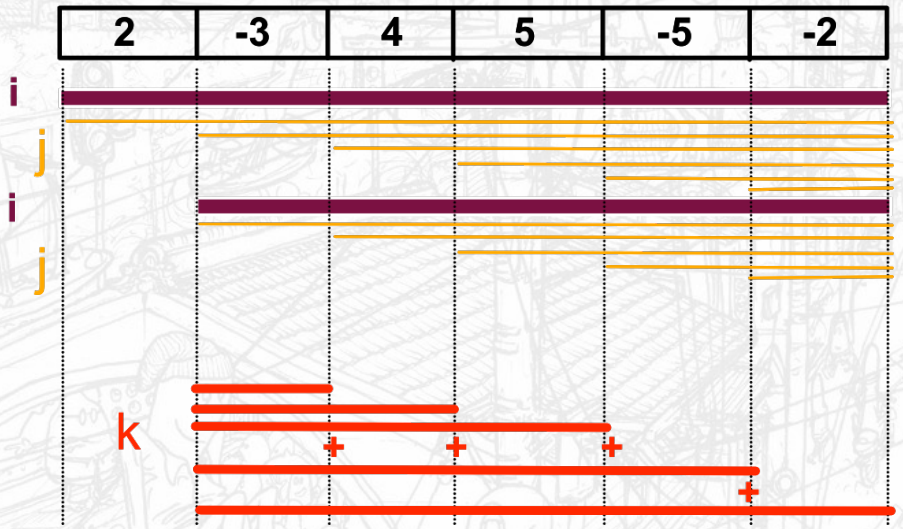
Em geral quando conseguimos eliminar o ciclo mais interno num algoritmo reduzimos a complexidade temporal



Hoooops !!!

Análise de Complexidade

Aplicação



Hoooops !!!

```
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(N²)* !!!

Análise de Complexidade

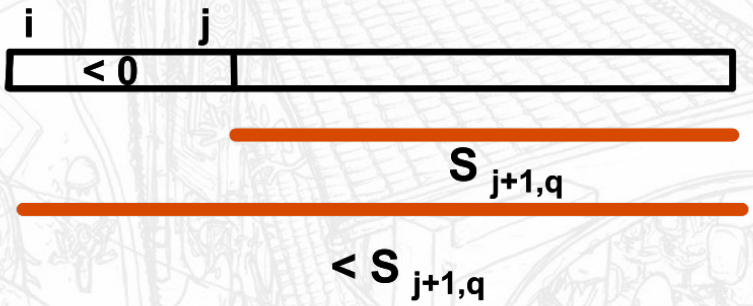
Aplicação

```
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 interessa calcular.

Vejamos !!

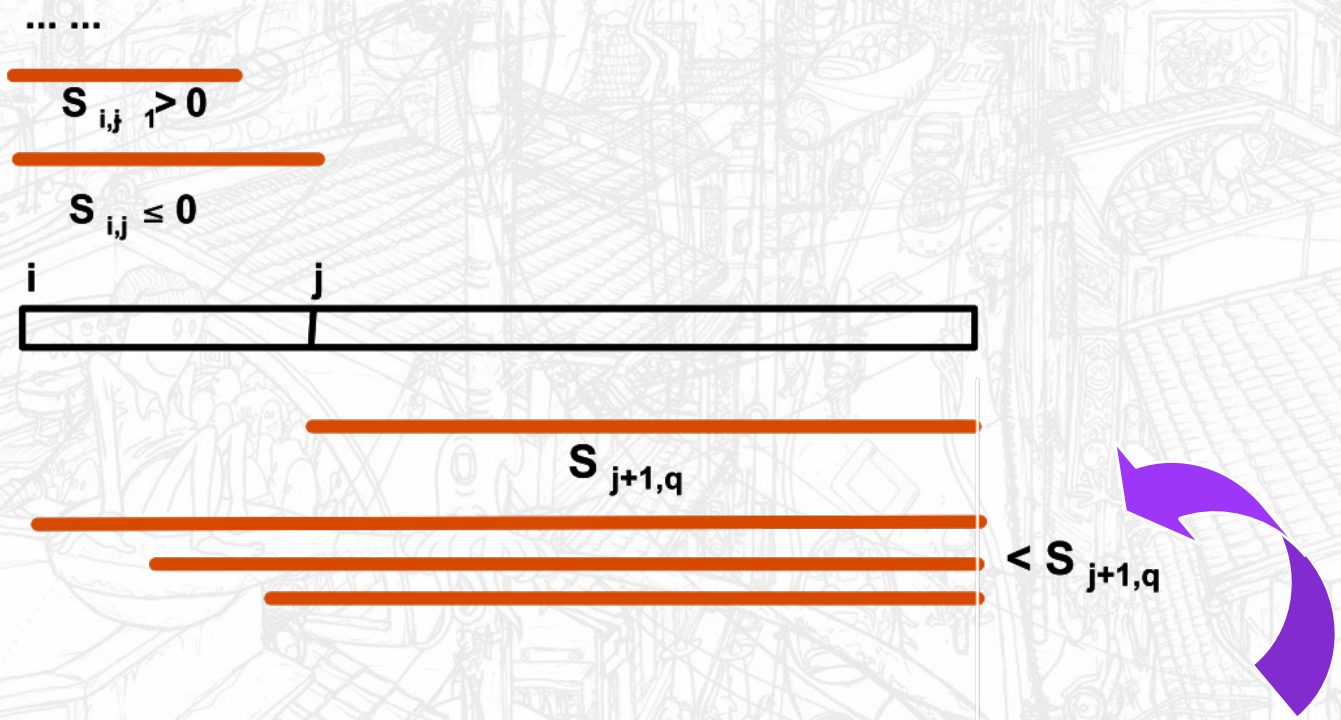


1. Todas as subsequências que confinam com a subsequência de soma máxima têm soma negativa ou igual a zero.

Análise de Complexidade

Aplicação

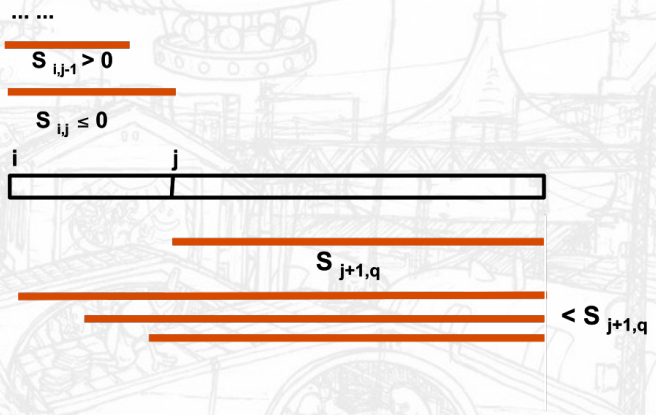
Mais !!



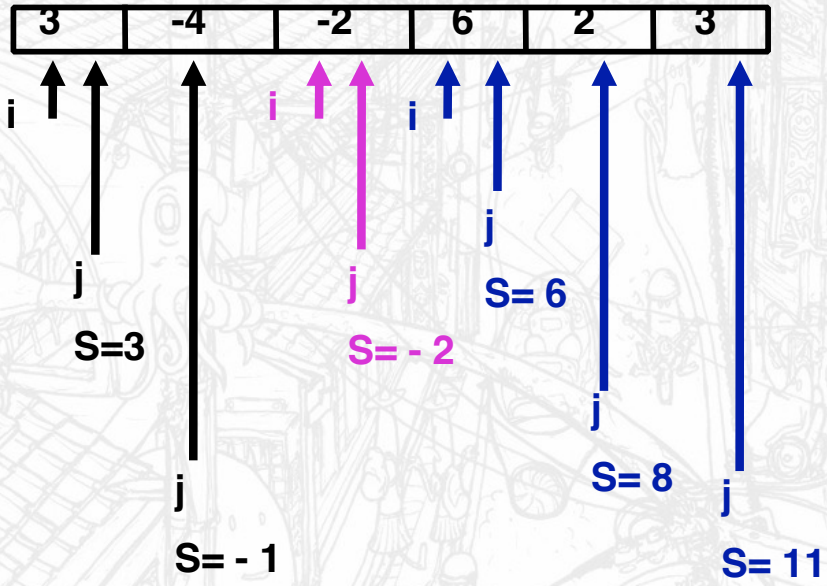
2. Qualquer destas subsequências não é uma subsquência de soma máxima ou é igual a uma subsequência de soma máxima já encontrada.

Análise de Complexidade

Aplicação



Decorre...

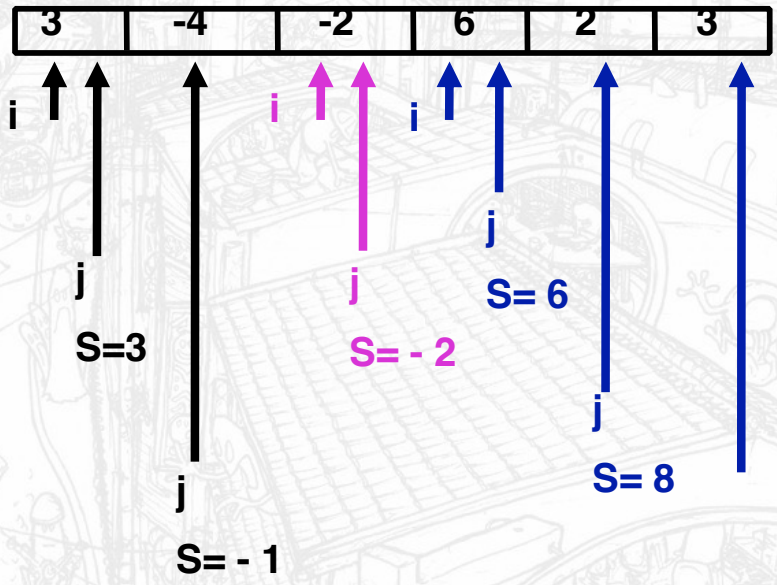


Este algoritmo tem complexidade linear ! Uffffff!!!

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

Análise de Complexidade
Aplicação

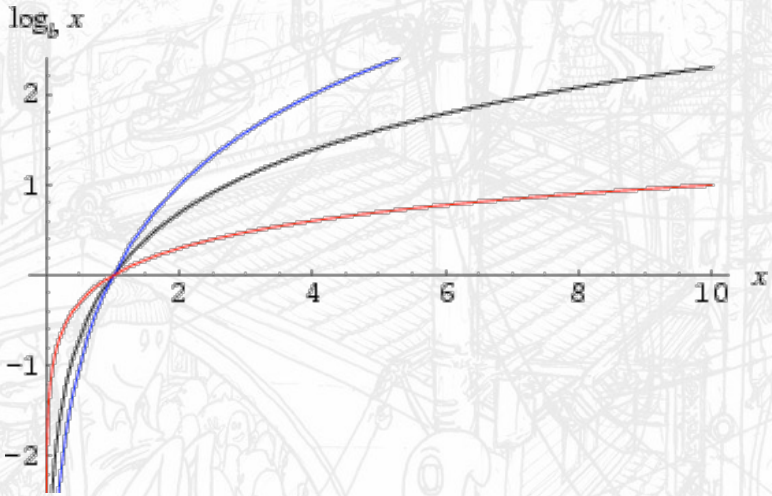
Outro exemplo

de uma entrevista de emprego na AMAZON :)

Análise de Complexidade

… fim

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/