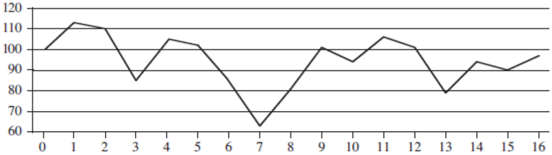
# Divisão e Conquista

Prof. Daniel Kikuti

Universidade Estadual de Maringá

# O problema de encontrar o segmento máximo

- Histórico de preço das ações de uma empresa no mercado.
- Comprar uma ação em um dia e vender em outro.
- Objetivo: maximizar o lucro (“comprar barato e vender caro”).




| Day | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Price | 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 |
| Change | | 13 | −3 | −25 | 20 | −3 | −16 | −23 | 18 | 20 | −7 | 12 | −5 | −22 | 15 | −4 | 7 |

**Como resolver o problema???**

# Força bruta

## Solução em alto nível

1. Gerar todos os pares de datas de modo que a data de compra preceda a data de venda.
2. Calcular o lucro para cada par.
3. Devolver o par que produz o maior lucro.

## Complexidade

# Força bruta

## Solução em alto nível

1. Gerar todos os pares de datas de modo que a data de compra preceda a data de venda.
2. Calcular o lucro para cada par.
3. Devolver o par que produz o maior lucro.

## Complexidade

- Um período de $n$ dias possui $\binom{n}{2}$ pares ($\Theta(n^2)$).
- Custo para avaliar cada par é $\Theta(1)$.
- Portanto, custo total desta abordagem é pelo menos $\Omega(n^2)$.

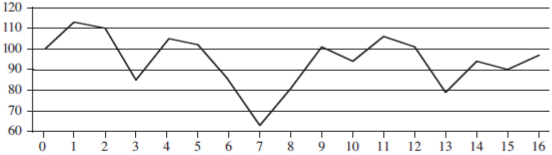
## Exercício

[CLRS 4.1-2] Escreva um pseudo-código para o método força bruta que resolve o problema de encontrar o segmento máximo. Seu algoritmo deve executar em tempo $\Theta(n^2)$.

# É possível desenvolver um algoritmo $o(n^2)$?

## Ver o problema maneira diferente

- Encontrar uma sequência de dias na qual a cadeia de variações do primeiro dia até o último é máxima.
- Considerar as variações diárias de preço (diferença do preço do dia $i$ e $i-1$).
- Dado um vetor $A$ contendo as variações, queremos encontrar um subvetor não vazio, contiguo, cuja soma dos elementos seja máxima.




| Day | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Price | 100 | 113 | 110 | 85 | 105 | 102 | 86 | 63 | 81 | 101 | 94 | 106 | 101 | 79 | 94 | 90 | 97 |
| Change | | 13 | −3 | −25 | 20 | −3 | −16 | −23 | 18 | 20 | −7 | 12 | −5 | −22 | 15 | −4 | 7 |

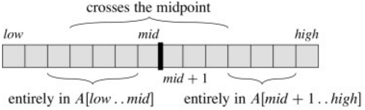
# Uma solução usando Divisão e Conquista

- Queremos encontrar a subsequência máxima do subvetor $A[\text{low} \ldots \text{high}]$.
- Dividir o problema em dois subproblemas de tamanho aproximadamente iguais ($A[\text{low} \ldots \text{mid}]$ e $A[\text{mid}+1 \ldots \text{high}]$).
- Seja $A[i \ldots j]$ a subsequência máxima do subvetor $A[\text{low} \ldots \text{high}]$ e seja mid o elemento do meio do subvetor $A$, como podemos determinar os índices $i$ e $j$, considerando a divisão em subproblemas?

# Uma solução usando Divisão e Conquista

- Queremos encontrar a subsequência máxima do subvetor $A[\text{low} \ldots \text{high}]$.
- Dividir o problema em dois subproblemas de tamanho aproximadamente iguais ($A[\text{low} \ldots \text{mid}]$ e $A[\text{mid}{+}1 \ldots \text{high}]$).
- Seja $A[i \ldots j]$ a subsequência máxima do subvetor $A[\text{low} \ldots \text{high}]$ e seja mid o elemento do meio do subvetor $A$, como podemos determinar os índices $i$ e $j$, considerando a divisão em subproblemas?

## Possíveis locais onde a subsequência máxima pode estar



(a)

# Algumas considerações

- Podemos computar a subsequência máxima dos subvetores $A[\text{low} \ldots \text{mid}]$ e $A[\text{mid}+1 \ldots \text{high}]$ recursivamente, pois estas são instâncias menores do problema de encontrar a sequencia máxima de um vetor.
- Precisamos focar no caso em que a solução $A[i \ldots j]$ cruza o meio. Note que esta não é uma instância menor do problema original, pois há a restrição que a sequência deve cruzar o elemento do meio.
- Como encontrar $A[i \ldots \text{mid}]$ e $A[\text{mid}+1 \ldots j]$ e combiná-las?

# Algoritmo para situação em que $A[i \dots j]$ cruza o meio

```
FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
left-sum ← −∞
sum ← 0
for i ← mid to low do
    sum ← sum + A[i]
    if sum > left-sum then
        left-sum ← sum
        max-left ← i
right-sum ← −∞
sum ← 0
for j ← mid + 1 to high do
    sum ← sum + A[i]
    if sum > right-sum then
        right-sum ← sum
        max-right ← j
return (max-left, max-right, left-sum + right-sum)
```

# Análise do Find-Max-Crossing-Subarray

## Correção

[Exercício] Demonstre os seguintes invariantes:

- left-sum contém a soma da subsequência máxima de $A[\mathrm{i{+}1} \ldots mid]$.
- right-sum contém a soma da subsequência máxima de $A[\mathrm{mid{+}1} \ldots \mathrm{j\text{-}1}]$.

## Complexidade

Seja $n =$ high - low $+$ 1:

- Cada iteração nos dois laços **for** custa $\Theta(1)$.
- O laço **for** das linhas 3–7 faz mid-low+1 iterações.
- O laço **for** das linhas 10–14 faz high-mid iterações.
- Portanto, (mid-low+1) + (high - mid) = high - low + 1 = $n$.
- Custo total $\Theta(n)$.

# O algoritmo de divisão e conquista

O algoritmo devolve uma tupla contendo os índices que delimitam uma subsequência máxima e a soma.

```
FIND-MAXIMUM-SUBARRAY(A, low, high)
if high = low then
    return (low, high, A[low])
else
    mid ← ⌊(low + high)/2⌋
    (left-low, left-high, left-sum) ←
     FIND-MAXIMUM-SUBARRAY(A, low, mid)
    (right-low, right-high, right-sum) ←
     FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
    (cross-low, cross-high, cross-sum) ←
     FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
    if left-sum ≥ right-sum and left-sum ≥ cross-sum then
        return (left-low, left-high, left-sum)
    else if right-sum ≥ left-sum and right-sum ≥ cross-sum then
        return (right-low, right-high, right-sum)
    else
        return (cross-low, cross-high, cross-sum)
```

# Análise do algoritmo

Exercício. Defina uma recorrência e compute o custo total do algoritmo FIND-MAXIMUM-SUBARRAY.

# Problema de multiplicação de inteiros longos

## Como computar $(2^{63} - 1) \times (2^{63} - 1)$?

- $9.223.372.036.854.775.807$ é o maior valor para um inteiro de 64 bits com sinal.
- Usando uma calculadora temos: $8.5070592 \times 10^{37}$.
- O valor desejado: $85.070.591.730.234.615.847.396.907.784.232.501.249$

## Pra que se incomodar com inteiros grandes?

- Usar pontos flutuantes (reais) nem sempre é uma opção razoável, pois operações aritméticas com tais números nem sempre são precisas.
- Aplicação em criptografia e outras áreas.

# Solução trivial

## Multiplicação de inteiros – algoritmo elementar

Dados dois números inteiros $x$ e $y$ com $n$ dígitos cada, calcular o produto $x \times y$ fazendo:

- Compute um "produto parcial" multiplicando cada dígito de $x$ separadamente por $y$;
- Some todos os produtos parciais.

## Exemplo

$x = 1234$ e $y = 5678$

$$
\begin{array}{r}
1\,2\,3\,4 \\
\times \quad 5\,6\,7\,8 \\
\hline
9\,8\,7\,2 \\
8\,6\,3\,8\phantom{\,0} \\
7\,4\,0\,4\phantom{\,0\,0} \\
6\,1\,7\,0\phantom{\,0\,0\,0} \\
\hline
7\,0\,0\,6\,6\,5\,2
\end{array}
$$

# Analisando a complexidade do algoritmo elementar

## Considerações

- Cada número possui $n$ dígitos;
- Assumimos base $10$ (embora o algoritmo elementar funcione exatamente da mesma maneira para base $2$);
- Operação elementar $=$ multiplicação e adição de dígitos.

## Complexidade de tempo $\approx$ número de operações elementares

- Para calcular cada produto parcial leva tempo $O(n)$.
- Adição de dígitos leva tempo $O(n)$.
- Como existem $n$ produtos parciais, então o algoritmo leva tempo $O(n^2)$.

# Analisando a complexidade do algoritmo elementar

## Algumas observações

Se $x$ e $y$ têm $n$ dígitos cada, então

- $x \times y$ tem no máximo $2n$ dígitos.
- $x + y$ tem no máximo $n + 1$ dígitos.

Exemplos:

- $9999 \times 9999 = 99980001$.
- $9999 + 9999 = 19998$.

## Um número quadrático de operações elementares é necessário?

**Desafio:** É possível fazer melhor? SIM!

# Algoritmo baseado em Divisão e Conquista

## Ideia base

Reescrever cada um dos números de forma a quebrar o produto em somas parciais de termos menores.

$$
\begin{aligned}
x &= \boxed{\quad x_L \quad} \; \boxed{\quad x_R \quad} = B^{n/2}x_L + x_R \\
y &= \boxed{\quad y_L \quad} \; \boxed{\quad y_R \quad} = B^{n/2}y_L + y_R \\
xy &= (B^{n/2}x_L + x_R)(B^{n/2}y_L + y_R) \\
&= B^n x_L y_L + B^{n/2}(x_L y_R + x_R y_L) + x_R y_R,
\end{aligned} \tag{1}
$$

onde:

- $x_L$ e $y_L$ são os $n/2$ dígitos mais significativos de $x$ e $y$;
- $x_R$ e $y_R$ são os $n/2$ dígitos menos significativos de $x$ e $y$;
- $B$ representa a base (que não importa de fato).

# Algoritmo baseado em Divisão e Conquista

## Exemplo

Calcule $1234 \times 5678$ usando o algoritmo de Karatsuba.

# Algoritmo baseado em Divisão e Conquista

## Exemplo

Calcule $1234 \times 5678$ usando o algoritmo de Karatsuba.

## Solução

- $x_L = 12$, $x_R = 34$, $y_L = 56$, $y_R = 78$.
- $x_L y_L = 12 \times 56 = 672$.
- $x_L y_R = 12 \times 78 = 936$.
- $x_R y_L = 34 \times 56 = 1904$.
- $x_R y_R = 34 \times 78 = 2652$.

$$\begin{aligned} xy &= B^n x_L y_L + B^{n/2}(x_L y_R + x_R y_L) + x_R y_R \\ &= 10^4 \times 672 + 10^2(936 + 1904) + 2652 \\ &= 7.006.652 \end{aligned}$$

# Algoritmo baseado em Divisão e Conquista

## Descrição em alto nível ($n$ é potência de 2)

1. Compute recursivamente o resultado para as quatro instâncias de tamanho $n/2$;
2. Combine-as usando a Equação 1;

$\textsc{Divisão-e-Conquista}(x, y)$

$n \leftarrow \max(\text{size of } x, \text{size of } y)$
**if** $n = 1$ **then** **return** $xy$
$m \leftarrow n/2$
$x_L \leftarrow \lfloor x/10^m \rfloor$ e $x_R \leftarrow x \bmod 10^m$
$y_L \leftarrow \lfloor y/10^m \rfloor$ e $y_R \leftarrow y \bmod 10^m$
$P_1 \leftarrow \textsc{Divisão-e-Conquista}(x_L, y_L)$
$P_2 \leftarrow \textsc{Divisão-e-Conquista}(x_L, y_R)$
$P_3 \leftarrow \textsc{Divisão-e-Conquista}(x_R, y_L)$
$P_4 \leftarrow \textsc{Divisão-e-Conquista}(x_R, y_R)$
**return** $10^n P_1 + 10^{n/2}(P_2 + P_3) + P_4$

# Algoritmo baseado em Divisão e Conquista

## Complexidade

O passo de combinar requer três adições de números de $n$ dígitos, portanto a complexidade de tempo é dada pela recorrência:

$$T(n) = 4T(n/2) + cn.$$

Qual a complexidade?

# Algoritmo baseado em Divisão e Conquista

## Complexidade

O passo de combinar requer três adições de números de $n$ dígitos, portanto a complexidade de tempo é dada pela recorrência:

$$T(n) = 4T(n/2) + cn.$$

Qual a complexidade? $O(n^2)$.

## UAUUU!!! Não ganhamos nada????

Olhando para a recorrência, o que poderíamos tentar fazer para diminuir a complexidade?

# Truque de Gauss (e Karatsuba)

Considere a equação:

$$xy = B^n x_L y_L + B^{n/2}(x_L y_R + x_R y_L) + x_R y_R$$

- Seja $P = (x_L + x_R) \times (y_L + y_R) = x_L y_L + x_L y_R + x_R y_L + x_R y_R$.
- Podemos obter o valor de $(x_L y_R + x_R y_L)$ subtraindo de $P$ os valores de $x_L y_L$ e $x_R y_R$.
- Assim:

$$xy = B^n x_L y_L + B^{n/2}(P - x_L y_L - x_R y_R) + x_R y_R$$

# Complexidade desta nova versão?

## Quantas chamadas recursivas são feitas?

Considerando os números $u = 1234$ e $v = 5678$, temos:

- $x_L = 12$, $x_R = 34$, $y_L = 56$, $y_R = 78$.
- $x_L y_L = 12 \times 56 = 672$.
- $x_R y_R = 34 \times 78 = 2652$.
- $P = (x_L + x_R) \times (y_L + y_R) = (12 + 34) \times (56 + 78) = 46 \times 134 = 6164$.
- $P - x_L y_L - x_R y_R = 2840$.

Portanto:

$$
\begin{aligned}
xy &= B^n x_L y_L + B^{n/2}(P - x_L y_L - x_R y_R) + x_R y_R \\
&= 10^4 \times 672 + 10^2 \times 2840 + 2652 \\
&= 7.006.652
\end{aligned}
$$

# O algoritmo

$\textsc{Karatsuba}(x, y)$

$n \leftarrow \max(\text{size of } x, \text{size of } y)$
**if** $n = 1$ **then** **return** $xy$
$m \leftarrow n/2$
$x_L \leftarrow \lfloor x/10^m \rfloor$ e $x_R \leftarrow x \bmod 10^m$
$y_L \leftarrow \lfloor y/10^m \rfloor$ e $y_R \leftarrow y \bmod 10^m$
$P_1 \leftarrow \textsc{Karatsuba}(x_L, y_L)$
$P_2 \leftarrow \textsc{Karatsuba}(x_R, y_R)$
$P_3 \leftarrow \textsc{Karatsuba}(x_L + x_R, y_L + y_R)$
**return** $10^n P_1 + 10^{n/2}(P_3 - P_1 - P_2) + P_2$

# Complexidade desta nova versão?

## Número de operações elementares?

- 3 multiplicações de números de tamanho n/2;
- Recorrência: $T(n) = 3\,T(n/2) + cn$;
- Complexidade: $O(n^{\lg 3})$ $(\lg 3 \approx 1,59)$.

## Curiosidades

- A. A. Karatsuba (1937–2008), matemático russo homepage;
- Existem algoritmos ainda mais rápidos que o de Karatsuba:
  - Toom-Cook: $O(n^{1.465})$.
  - Schönhage-Strassen: $O(n \lg n \lg n)$.

# Multiplicação de matrizes

## O problema

Dadas duas matrizes quadradas $n \times n$, $A$ e $B$, determinar o produto $C = A \cdot B$.
Cada célula da matriz é calculada como:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}.$$

## Algoritmo elementar

$\textsc{Square-Matrix-Multiply}(A, B)$

$n \leftarrow A.rows$
Let $C$ be a new matrix $n \times n$
**for** $i \leftarrow 1$ **to** $n$ **do**
    **for** $j \leftarrow 1$ **to** $n$ **do**
        $c_{ij} \leftarrow 0$
        **for** $k \leftarrow 1$ **to** $n$ **do**
            $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$
**return** $C$

# Como quebrar o problema em subproblemas?

Supondo que $n$ é potência de $2$, quebraremos em quatro matrizes de dimensões $n/2 \times n/2$.

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

Podemos escrever então a equação $C = A \cdot B$ como:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \qquad (2)$$

# Como quebrar o problema em subproblemas?

A Equação 2 corresponde a quatro equações:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \tag{3}$$
$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \tag{4}$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \tag{5}$$
$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \tag{6}$$

## Um algoritmo recursivo

- Caso base: $n = 1$ o algoritmo devolve $c_{11} = a_{11} \cdot b_{11}$.
- Caso $n > 1$: Quantas chamadas recursivas o algoritmo faz?
- Recorrência:

# Como quebrar o problema em subproblemas?

A Equação 2 corresponde a quatro equações:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21} \tag{3}$$
$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22} \tag{4}$$
$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21} \tag{5}$$
$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22} \tag{6}$$

## Um algoritmo recursivo

- Caso base: $n = 1$ o algoritmo devolve $c_{11} = a_{11} \cdot b_{11}$.
- Caso $n > 1$: Quantas chamadas recursivas o algoritmo faz?
- Recorrência: $T(n) = 8\,T(n/2) + cn^2$.

# O método de Strassen

1. Divida as matrizes $A$ e $B$ em submatrizes $n/2 \times n/2$.
2. Crie 10 matrizes $n/2 \times n/2$, $S_1, S_2, \ldots, S_{10}$, pela soma ou diferença entre as matrizes criadas no passo anterior (complexidade $\Theta(n^2)$).
3. Usando as matrizes criadas no primeiro passo e as 10 matrizes do passo anterior, compute sete produtos entre matrizes $P_1, P_2, \ldots, P_7$.
4. Compute as submatrizes desejadas $C_{11}, C_{12}, C_{21}, C_{22}$ pela adição e subtração das várias combinações das matrizes $P_i$ (complexidade $\Theta(n^2)$).

# As submatrizes

$$\begin{aligned} S_1 &= B_{12} - B_{22}; & S_2 &= A_{11} + A_{12}; \\ S_3 &= A_{21} + A_{22}; & S_4 &= B_{21} - B_{11}; \\ S_5 &= A_{11} + A_{22}; & S_6 &= B_{11} + B_{22}; \\ S_7 &= A_{12} - A_{22}; & S_8 &= B_{21} + B_{22}; \\ S_9 &= A_{11} - A_{21}; & S_{10} &= B_{11} + B_{12}. \end{aligned}$$

$$\begin{aligned} P_1 &= A_{11} \cdot S_1; & P_2 &= S_2 \cdot B_{22}; \\ P_3 &= S_3 \cdot B_{11}; & P_4 &= A_{22} \cdot S_4; \\ P_5 &= S_5 \cdot S_6; & P_6 &= S_7 \cdot S_8; \\ P_7 &= S_9 \cdot S_{10}. \end{aligned}$$

# Como o algoritmo de Strassen computa a matriz C?

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6 \\
C_{12} &= P_1 + P_2 \\
C_{21} &= P_3 + P_4 \\
C_{22} &= P_5 + P_1 - P_3 - P_7
\end{aligned}
$$

# Conclusões

- Técnicas de projeto auxiliam no desenvolvimento de algoritmos;
- Algoritmos mais sofisticados podem ser mais rápidos (para $n$ grande);
- Matemática ajuda a projetar algoritmos mais sofisticados.

# Exercício

Multiplicação de matrizes (Algoritmo de Strassen).
Ler Capítulo 4.2 do Cormen (3a. edição).

# Referências


- Dasgupta, Papadimitriou and Vazirani. Algorithms, 2006.
- Brassard and Bratley. Algorithmics: Theory and Practice, 1988
- Kleinberg and Tardos. Algorithm Design, 2005
- Karatsuba algorithm (https://en.wikipedia.org/wiki/Karatsuba_algorithm)
- Paulo Feofiloff, http://www.ime.usp.br/ song/ufabc/multiplication-handout.pdf