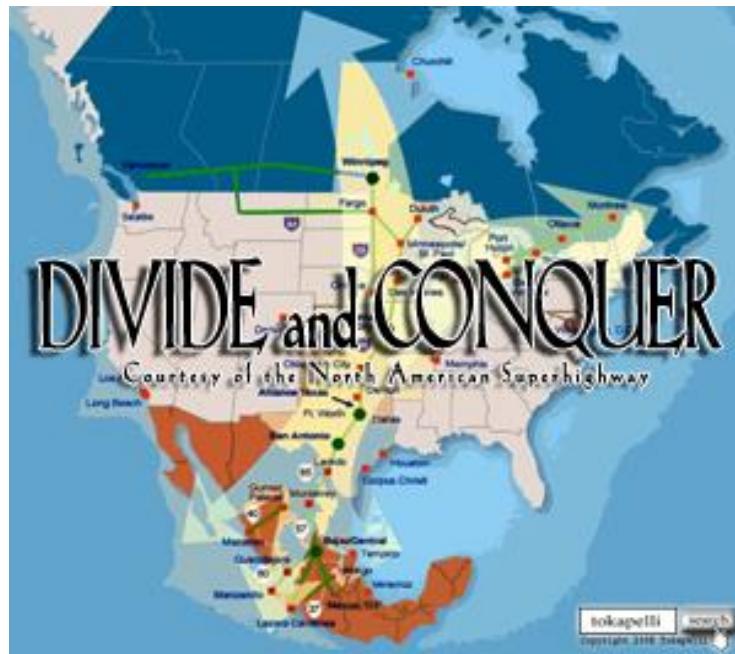


# Algoritma *Divide and Conquer*

Bahan Kuliah IF2211 Strategi Algoritma

(Bagian 3)

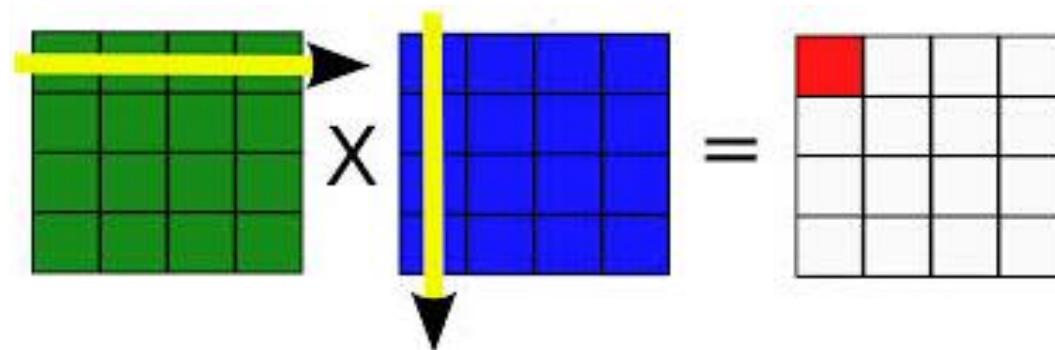
Oleh: Rinaldi Munir



Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika ITB  
2025

## 7. Perkalian Matriks

- Misalkan  $A$  dan  $B$  dua buah matrik berukuran  $n \times n$ .
- Perkalian matriks:  $C = A \times B$ , matriks  $C$  juga berukuran  $n \times n$



Elemen-elemen hasilnya:  $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^n a_{ik}b_{kj}$

# Penyelesaian secara *Brute Force*

Elemen-elemen hasilnya:  $c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = \sum_{k=1}^n a_{ik}b_{kj}$

Algoritma *brute force*: kalikan setiap vektor baris  $i$  dari matriks  $A$  dengan setiap vektor kolom  $j$  dari matriks  $B$ .

```
function KaliMatriks( $A, B : \text{Matriks}, n : \text{integer}$ )  $\rightarrow \text{Matriks}$ 
{ Mengalikan matriks  $A$  dan  $B$  yang berukuran  $n \times n$ , menghasilkan matriks  $C$  yang juga berukuran  $n \times n$  }
```

**Deklarasi**

$i, j, k : \text{integer}$

**Algoritma:**

```
for  $i \leftarrow 1$  to  $n$  do
  for  $j \leftarrow 1$  to  $n$  do
     $C[i, j] \leftarrow 0$  { inisialisasi penjumlah }
    for  $k \leftarrow 1$  to  $n$  do
       $C[i, j] \leftarrow C[i, j] + A[i, k]*B[k, j]$ 
    endfor
  endfor
endfor
return  $C$ 
```

Kompleksitas waktu algoritma:  $O(n^3)$

# Penyelesaian dengan *Divide and Conquer*

Matriks  $A$  dan  $B$  dibagi menjadi 4 buah matriks bujur sangkar.  
Masing-masing matriks bujur sangkar berukuran  $n/2 \times n/2$ :

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$A$                        $B$                        $C$

Elemen-elemen matriks  $C$  adalah:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

$$A = \begin{bmatrix} A_{11} & & A_{12} \\ A_{21} & & A_{22} \end{bmatrix}$$

$$B = \begin{bmatrix} B_{11} & & B_{12} \\ B_{21} & & B_{22} \end{bmatrix}$$

$$C = \begin{bmatrix} C_{11} & & C_{12} \\ C_{21} & & C_{22} \end{bmatrix}$$

**Contoh 11.** Misalkan matriks  $A$  adalah sebagai berikut:

$$A = \left[ \begin{array}{cc|cc} 3 & 4 & 8 & 16 \\ 21 & 5 & 12 & 10 \\ \hline 5 & 1 & 2 & 3 \\ 45 & 9 & 0 & -1 \end{array} \right]$$

Matriks  $A$  dibagi menjadi 4 upa-matriks  $2 \times 2$ :

$$A_{11} = \begin{bmatrix} 3 & 4 \\ 21 & 5 \end{bmatrix} \quad A_{12} = \begin{bmatrix} 8 & 16 \\ 12 & 10 \end{bmatrix} \quad A_{21} = \begin{bmatrix} 5 & 1 \\ 45 & 9 \end{bmatrix} \quad A_{22} = \begin{bmatrix} 2 & 3 \\ 0 & -1 \end{bmatrix}$$

*Pseudo-code perkalian matriks dengan algoritma divide and conquer:*

```
function KaliMatriks2(A, B : Matriks, n : integer) → Matriks
{ Memberikan hasil kali matriks A dan B yang berukuran  $n \times n$ . }
Deklarasi
i, j, k : integer
A11, A12, A21, A22, B11, B12, B21, B22, C11, C12, C21, C22 : Matriks

Algoritma:
if n = 1 then { matriks berukuran  $1 \times 1$  atau sebagai scalar }
    return A * B { perkalian dua buah scalar biasa }
else
    Bagi A menjadi A11, A12, A21, dan A22 yang masing-masing berukuran  $n/2 \times n/2$ 
    Bagi B menjadi B11, B12, B21, dan B22 yang masing-masing berukuran  $n/2 \times n/2$ 
    C11 ← KaliMatriks2(A11, B11, n/2) + KaliMatriks2(A12, B21, n/2)
    C12 ← KaliMatriks2(A11, B12, n/2) + KaliMatriks2(A12, B22, n/2)
    C21 ← KaliMatriks2(A21, B11, n/2) + KaliMatriks2(A22, B21, n/2)
    C22 ← KaliMatriks2(A21, B12, n/2) + KaliMatriks2(A22, B22, n/2)
    return C { C adalah gabungan C11, C12, C21, C22 }
endif
```

$$\begin{aligned}C11 &= A11 \cdot B11 + A12 \cdot B21 \\C12 &= A11 \cdot B12 + A12 \cdot B22 \\C21 &= A21 \cdot B11 + A22 \cdot B21 \\C22 &= A21 \cdot B12 + A22 \cdot B22\end{aligned}$$

Pseudo-code untuk operasi penjumlahan (+):

**function** *Tambah*(*A, B : Matriks, n : integer*)  $\rightarrow$  *Matriks*  
*{ Memberikan hasil penjumlahkan dua buah matriks, A dan B, yang berukuran n × n }*

**Deklarasi**

*i, j, k : integer*

**Algoritma:**

```
for i  $\leftarrow$  1 to n do
    for j  $\leftarrow$  1 to n do
        C[i, j]  $\leftarrow$  A[i, j] + B[i, j]
    endfor
endfor

return C
```

Kompleksitas algoritmanya:  $O(n^2)$

- Kompleksitas waktu perkalian matriks dengan *divide and conquer*:

$$T(n) = \begin{cases} a & , n = 1 \\ 8T(n/2) + cn^2 & , n > 1 \end{cases}$$

$$\begin{aligned} C11 &= A11 \cdot B11 + A12 \cdot B21 \\ C12 &= A11 \cdot B12 + A12 \cdot B22 \\ C21 &= A21 \cdot B11 + A22 \cdot B21 \\ C22 &= A21 \cdot B12 + A22 \cdot B22 \end{aligned}$$

Menurut Teorema Master,  $T(n) = aT(n/b) + cn^d$ , diperoleh  $a = 8$ ,  $b = 2$ ,  $d = 2$ , dan memenuhi  $a > b^d$  (yaitu  $8 > 2^2$ ) maka relasi rekurens

$$T(n) = 8T(n/2) + cn^2$$

memenuhi *case 3* (jika  $a > b^d$ )

sehingga

$$\begin{cases} O(n^d) & \text{jika } a < b^d \\ O(n^d \log n) & \text{jika } a = b^d \\ O(n^{\log_b a}) & \text{jika } a > b^d \end{cases}$$

$$T(n) = O(n^{2\log 8}) = O(n^3)$$

- Hasil ini tidak memberi perbaikan kompleksitas dibandingkan dengan algoritma *brute force*.
- Dapatkah kita membuat algoritma perkalian matriks yang lebih baik?

# Algoritma Perkalian Matriks Strassen

- Ditemukan oleh Volker Strassen, seorang matematikawan Jerman
- Ideanya adalah mengurangi jumlah operasi kali. Operasi kali lebih ‘mahal’ ongkos komputasinya dibandingkan dengan operasi penjumlahan.
- Empat persamaan perkalian upamatriks (*sub-matrix*) terdahulu:

$$C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$$

$$C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$$

$$C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$$

$$C_{22} = A_{21} \cdot B_{12} + A_{22} \cdot B_{22}$$

terdapat 8 operasi perkalian ( $\cdot$ ) dan 4 operasi penjumlahan ( $+$ ) upamatriks.

- Strassen memanipulasi empat persamaan di atas sedemikian rupa sehingga jumlah operasi kali berkurang menjadi 7 (dengan konsekuensi operasi penjumlahan menjadi bertambah).

- Caranya adalah melakukan perhitungan intermediate sebagai berikut:

$$M_1 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$M_2 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$M_3 = (A_{11} - A_{21})(B_{11} + B_{12})$$

$$M_4 = (A_{11} + A_{12})B_{22}$$

$$M_5 = A_{11}(B_{12} - B_{22})$$

$$M_6 = A_{22}(B_{21} - B_{11})$$

$$M_7 = (A_{21} + A_{22})B_{11}$$

maka,

$$C_{11} = M_1 + M_2 - M_4 + M_6$$

$$C_{12} = M_4 + M_5$$

$$C_{21} = M_6 + M_7$$

$$C_{22} = M_2 - M_3 + M_5 - M_7$$

Terdapat 7 operasi  $\times$  dan 18 operasi  $+$



- **Volker Strassen** (born April 29, 1936) is a German mathematician, a professor emeritus in the department of mathematics and statistics at the University of Konstanz.
- In 2008 he was awarded the Knuth Prize for "seminal and influential contributions to the design and analysis of efficient algorithms."<sup>[5]</sup>

Sumber: Wikipedia

**function** *KaliMatriksStrassen(A, B : Matriks, n : integer) → Matriks*

{ Memberikan hasil kali matriks A dan B yang berukuran  $n \times n$ . }

**Deklarasi**

*i, j, k : integer*

*A11, A12, A21, A22, B11, B12, B21, B22, C11, C12, C21, C22, M1, M2, M3, M4, M5, M6, M7 : Matriks*

**Algoritma:**

**if** *n = 1 then* { matriks berukuran  $1 \times 1$  atau sebagai scalar }

**return** *A \* B* { perkalian dua buah scalar biasa }

**else**

Bagi A menjadi *A11, A12, A21, dan A22* yang masing-masing berukuran  $n/2 \times n/2$

Bagi B menjadi *B11, B12, B21, dan B22* yang masing-masing berukuran  $n/2 \times n/2$

*M1 ← KaliMatriksStrassen(A12 – A22, B21 + B22, n/2)*

*M2 ← KaliMatriksStrassen (A11 + A22, B11 + B22 , n/2)*

*M3 ← KaliMatriksStrassen (A11 – A21, B11 + B12 , n/2)*

*M4 ← KaliMatriksStrassen (A11 + A12, B22 , n/2)*

*M5 ← KaliMatriksStrassen (A11, B12 – B22 , n/2)*

*M6 ← KaliMatriksStrassen (A22, B21 – B11 , n/2)*

*M7 ← KaliMatriksStrassen (A21 + A22, B11 , n/2)*

*C11 ← M1 + M2 – M4 + M6*

*C12 ← M4 + M5*

*C21 ← M6 + M7*

*C22 ← M2 – M3 + M5 – M7*

**return** *C* { *C* adalah gabungan *C11, C12, C13, C14* }

**endif**

- Kompleksitas algoritmanya menjadi:

$$T(n) = \begin{cases} a & , n = 1 \\ 7T(n/2) + cn^2 & , n > 1 \end{cases}$$

Bila diselesaikan dengan Teorema Master,  $T(n) = aT(n/b) + cn^d$ , diperoleh  $a = 7$ ,  $b = 2$ ,  $d = 2$ , dan memenuhi  $a > b^d$  (yaitu  $7 > 2^2$ ) maka relasi rekurens

$$T(n) = 7T(n/2) + cn^2$$

memenuhi case 3 (jika  $a > b^d$ )       $\begin{cases} O(n^d) & \text{jika } a < b^d \\ O(n^d \log n) & \text{jika } a = b^d \\ O(n^{\log_b a}) & \text{jika } a > b^d \end{cases}$

sehingga

$$T(n) = O(n^{2\log 7}) = O(n^{2.81})$$

- Lebih baik dari perkalian matriks secara *divide and conquer* sebelumnya yang  $O(n^3)$

# 8. Perkalian Bilangan Bulat yang Besar

- Bilangan bulat yang besar (*big number*) adalah bilangan bulat dengan panjang  $n$  angka atau  $n$  bit.

Contoh: 564389018149014329871520, 1000011011010100100110010101, ...

- Bahasa-bahasa pemrograman memiliki keterbatasan dalam merepresentasikan bilangan bulat yang besar.
- Dalam Bahasa C misalnya, tipe bilangan bulat hanya `char` (8 bit), `int`, (16 bit) dan `long` (32 bit).
- Untuk bilangan bulat yang lebih dari 32 bit, kita harus membuat tipe sendiri dan mendefinisikan primitif operasi-operasi aritmetika di dalamnya (+, -, \*, /, dll)

- Di sini hanya akan dibahas bagaimana algoritma melakukan operasi perkalian bilangan bulat yang besar.

Contoh:  $1765420875208345186 \times 75471199736308361736432 = ?$

- **Persoalan:** Misalkan bilangan bulat  $X$  dan  $Y$  yang panjangnya  $n$  angka (atau  $n$  bit):

$$X = x_1x_2x_3 \dots x_n$$

$$Y = y_1y_2y_3 \dots y_n$$

Hitunglah hasil kali  $X$  dengan  $Y$ .

**Contoh 12.** Misalkan,

$$X = 1234 \quad (n = 4)$$

$$Y = 5678 \quad (n = 4)$$

Cara klasik mengalikan  $X$  dan  $Y$  (*brute force*):

$$\begin{array}{r} X \times Y = 1234 \\ \underline{5678 \times} \\ 9872 \\ 8368 \\ 7404 \\ \hline 6170 \\ + \\ 7006652 \quad (7 \text{ angka}) \end{array}$$

## *Pseudo-code algoritma perkalian bilangan besar secara brute force:*

```
function Kali1(X, Y : LongInteger, n : integer) → LongInteger
{ Memberikan hasil perkalian X dan Y, masing-masing panjangnya n digit dengan algoritma brute force. }
```

### **Deklarasi**

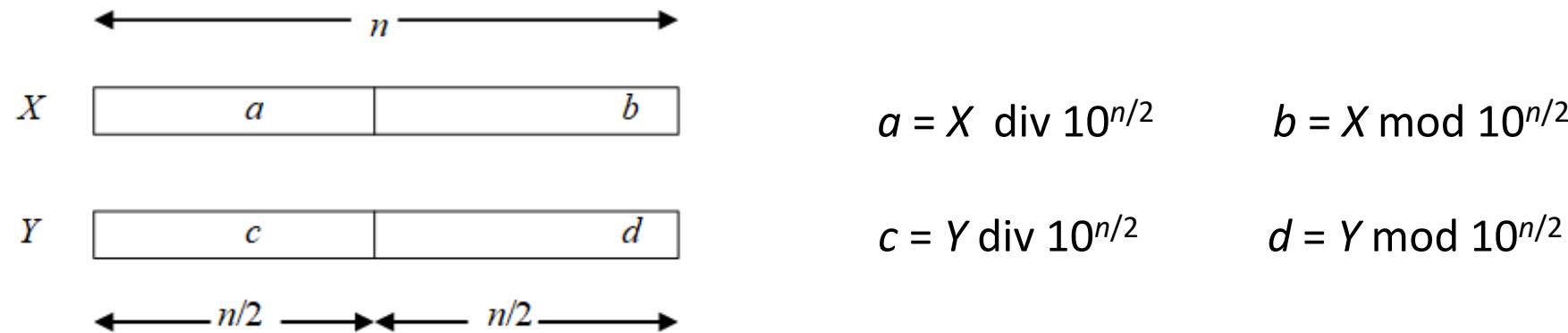
```
temp, AngkaSatuan, AngkaPuluhan : integer
```

### **Algoritma:**

```
for setiap angka  $y_i$  dari  $y_n, y_{n-1}, \dots, y_1$  do
    AngkaPuluhan  $\leftarrow 0
    for setiap angka  $x_j$  dari  $x_n, x_{n-1}, \dots, x_1$  do
        temp  $\leftarrow x_j * y_i
        temp  $\leftarrow temp + AngkaPuluhan
        AngkaSatuan  $\leftarrow temp \bmod 10
        AngkaPuluhan  $\leftarrow temp \bmod 10
        write(AngkaSatuan)
    endfor
endfor
Z  $\leftarrow$  Jumlahkan semua hasil perkalian dari atas ke bawah
return Z$$$$$ 
```

Kompleksitas algoritma:  $O(n^2)$

## Penyelesaian dengan algoritma *divide and conquer*



$X$  dan  $Y$  dapat dinyatakan dalam  $a$ ,  $b$ ,  $c$ , dan  $d$  sebagai

$$X = a \cdot 10^{n/2} + b \quad \text{dan} \quad Y = c \cdot 10^{n/2} + d$$

Perkalian  $X$  dengan  $Y$  dinyatakan sebagai

$$\begin{aligned} X \cdot Y &= (a \cdot 10^{n/2} + b) \cdot (c \cdot 10^{n/2} + d) \\ &= ac \cdot 10^n + ad \cdot 10^{n/2} + bc \cdot 10^{n/2} + bd \\ &= ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd \end{aligned}$$

**Contoh 13:** Misalkan  $n = 6$  dan  $X = 346769$  dan  $Y = 279431$ , maka

$$X = 346769 \rightarrow a = 346, b = 769 \rightarrow X = 346 \cdot 10^3 + 769$$

$$Y = 279431 \rightarrow c = 279, d = 431 \rightarrow Y = 279 \cdot 10^3 + 431$$

Perkalian  $X$  dengan  $Y$  dinyatakan sebagai

$$\begin{aligned}X \cdot Y &= (346 \cdot 10^3 + 769) \cdot (279 \cdot 10^3 + 431) \\&= (346)(279) \cdot 10^6 + (346)(431) \cdot 10^3 + (769)(279) \cdot 10^3 + (769)(431) \\&= (346)(279) \cdot 10^6 + ((346)(431) + (769)(279)) \cdot 10^3 + (769)(431)\end{aligned}$$

↑  
perkalian  
bilangan besar

↑  
perkalian  
bilangan besar

↑  
perkalian  
bilangan besar

↑  
perkalian  
bilangan besar

*Pseudo-code algoritma perkalian bilangan besar dengan algoritma divide and conquer:*

```
function Kali2(X, Y : LongInteger, n : integer) → LongInteger
{ Memberikan hasil perkalian X dan Y, masing-masing panjangnya n digit dengan algoritma divide and conquer. }
```

### Deklarasi

```
a, b, c, d : LongInteger
s : integer
```

### Algoritma:

```
if n = 1 then
    return X * Y { perkalian skalar biasa }
else
    s ← n div 2 { bagidua pada posisi s }
    a ← X div 10s
    b ← X mod 10s
    c ← Y div 10s
    d ← Y mod 10s
    return Kali2(a, c, s)*102s + Kali2(b, c, s)*10s + Kali2(a, d, s)*10s + Kali2(b, d, s)
endif
```

$$Kali2(a, c, s)*10^{2s} + Kali2(b, c, s)*10^s + Kali2(a, d, s)*10^s + Kali2(b, d, s)$$

- Kompleksitas algoritma *Kali2*:

$$T(n) = \begin{cases} a & , n = 1 \\ 4T(n/2) + cn & , n > 1 \end{cases}$$

Catatan: Menghitung  $10^s$  dan  $10^{2s}$  tidak perlu dilakukan dengan memangkatkan 10 sebanyak s kali atau  $2s$  kali, tetapi cukup dilakukan dengan menambahkan 0 sebanyak s kali atau  $2s$  kali

- Dengan menggunakan Teorema Master (tunjukkan!), diperoleh:

$$T(n) = O(n^2).$$

- Ternyata, perkalian dengan algoritma *Divide and Conquer* seperti di atas belum memperbaiki kompleksitas waktu algoritmanya, sama seperti perkalian secara *brute force*.
- Adakah algoritma perkalian yang lebih baik?

# Perbaikan: Algoritma Perkalian Karatsuba

- Ditemukan Anatolii Alexeevitch Karatsuba, seorang matematikawan Rusia pada tahun 1962
- Ideanya sama seperti pada perkalian matriks Strassen, yaitu dengan mengurangi jumlah operasi kali.
- Persamaan perkalian  $X$  dan  $Y$  terdahulu:

$$X \cdot Y = ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd$$

terdapat 4 operasi perkalian dan 3 operasi penjumlahan bilangan besar.

- Karatsuba memanipulasi persamaan di atas sedemikian rupa sehingga jumlah operasi kali berkurang menjadi 3 (dengan konsekuensi operasi penjumlahan menjadi bertambah).

Misalkan

$$r = (a + b)(c + d) = ac + (ad + bc) + bd$$

maka,

$$(ad + bc) = r - ac - bd = (a + b)(c + d) - ac - bd$$

Dengan demikian, perkalian  $X$  dan  $Y$  dimanipulasi menjadi

$$\begin{aligned} X \cdot Y &= ac \cdot 10^n + (ad + bc) \cdot 10^{n/2} + bd \\ &= \underbrace{ac}_p \cdot 10^n + \underbrace{\{(a + b)(c + d) - ac - bd\}}_r \cdot 10^{n/2} + \underbrace{bd}_q \end{aligned}$$

Sekarang hanya terdapat tiga operasi kali, yaitu  $p$ ,  $q$ , dan  $r$

*Pseudo-code algoritma perkalian bilangan besar dengan algoritma Karatsuba:*

```
function Kali3(X, Y : LongInteger, n : integer) → LongInteger
{ Memberikan hasil perkalian X dan Y, masing-masing panjangnya n digit dengan algoritma Karatsuba. }
```

### Deklarasi

*a, b, c, d, p, q, r : LongInteger*  
*s : integer*

### Algoritma:

```
if n = 1 then
    return X*Y { perkalian skalar biasa }
else
    s ← n div 2 { bagidua pada posisi s }
    a ← X div  $10^s$ 
    b ← X mod  $10^s$ 
    c ← Y div  $10^s$ 
    d ← Y mod  $10^s$ 
    p ← Kali3(a, c, s)
    q ← Kali3(b, d, s)
    r ← Kali3(a + b, c + d, s)
    return p* $10^{2s}$  + (r - p - q)* $10^s$  + q
endif
```

- Kompleksitas algoritmanya:

$T(n) = \text{tiga buah perkalian dua integer yang berukuran } n/2 + \text{penjumlahan integer berukuran } n/2$

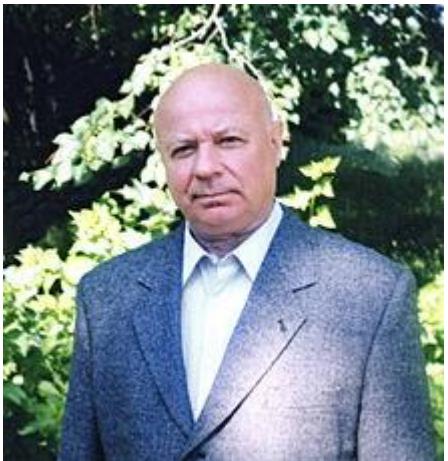
$$T(n) = \begin{cases} a & , n = 1 \\ 3T(n/2) + cn & , n > 1 \end{cases}$$

- Bila diselesaikan dengan Teorema Master,  $T(n) = aT(n/b) + cn^d$ , diperoleh  $a = 3$ ,  $b = 2$ ,  $d = 1$ , dan memenuhi  $a > b^d$  (yaitu  $3 > 2^1$ ) maka relasi rekurens  $T(n) = 3T(n/2) + cn$  memenuhi case 3 (jika  $a > b^d$ ), sehingga

$$T(n) = O(n^{2\log 3}) = O(n^{1.59})$$

- Ini lebih baik daripada kompleksitas waktu algoritma perkalian sebelumnya ( $O(n^2)$ ).

## Anatolii Alexevich Karatsuba



**Anatolii Alexeevitch Karatsuba** (Russian: Анатóлий Алексéевич Карацúба; Grozny, January 31, 1937 — Moscow, September 28, 2008) was a Russian mathematician, who authored the first fast multiplication method: the Karatsuba algorithm, a fast procedure for multiplying large numbers. (Sumber: Wikipedia)

# 9. Perkalian Polinom

- **Persoalan:**

Diberikan dua buah polinom derajat n

$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

$$B(x) = b_0 + b_1x + b_2x^2 + \dots + b_nx^n$$

Hitunglah perkalian  $A(x)B(x)$

## Contoh 14:

$$A(x) = 1 + 2x + 3x^2$$

$$B(x) = 3 + 2x + 2x^2$$

$$A(x)B(x) = (1 + 2x + 3x^2)(3 + 2x + 2x^2) = 3 + 8x + 15x^2 + 10x^3 + 6x^4$$

# Penyelesaian secara *brute force*

- Misalkan:

$$A(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n = \sum_{i=0}^n a_i x^i$$

$$B(x) = b_0 + b_1x + b_2x^2 + \dots + b_nx^n = \sum_{i=0}^n b_i x^i$$

- Maka algoritma *brute force* adalah mengalikan kedua polinom secara langsung:

$$C(x) = A(x)B(x) = \sum_{k=0}^{2n} c_k x^k , \text{ di sini } c_k = \sum_{0 \leq i,j \leq n, i+j=k} a_i b_j , 0 \leq k \leq 2n$$

- Dengan menggunakan formula perkalian di atas, ada dua buah kalang (*loop*), kalang pertama dari  $k = 0$  sampai  $2n$ , kalang kedua dari  $i = 0$  sampai  $k$ .
- Dapat dengan mudah ditunjukkan bahwa operasi perkalian adalah  $O(n^2)$  dan operasi penjumlahan  $O(n^2)$ . Kompleksitas algoritma seluruhnya adalah  $O(n^2)$ .

## Penyelesaian dengan algoritma *divide and conquer*

- Bagidua  $A(x)$  menjadi  $A_0(x)$  dan  $A_1(x)$ , masing-masing  $n/2$  suku

$$A_0(x) = a_0 + a_1x + a_2x^2 + \dots + a_{\lfloor n/2 \rfloor - 1}x^{\lfloor n/2 \rfloor - 1}$$

$$A_1(x) = a_{\lfloor n/2 \rfloor} + a_{\lfloor n/2 \rfloor + 1}x + a_{\lfloor n/2 \rfloor + 2}x^2 + \dots + a_nx^{n - \lfloor n/2 \rfloor}$$

maka

$$A(x) = A_0(x) + A_1(x)x^{\lfloor n/2 \rfloor}$$

- Dengan cara yang sama untuk  $B(x)$  menjadi  $B_0(x)$  dan  $B_1(x)$  sedemikian sehingga

$$B(x) = B_0(x) + B_1(x)x^{\lfloor n/2 \rfloor}$$

- Maka, perkalian  $A(x)$  dan  $B(x)$  ditulis menjadi

$$A(x)B(x) = A_0(x)B_0(x) + \{A_0(x)B_1(x) + A_1(x)B_0(x)\}x^{\lfloor n/2 \rfloor} + A_1(x)B_1(x)x^{2\lfloor n/2 \rfloor}$$

→ Terdapat empat buah perkalian upa-polinom

**Contoh 15:**  $A(x) = 2 + 5x + 3x^2 + x^3 - x^4$  dan  $B(x) = 1 + 2x + 2x^2 + 3x^3 + 6x^4$

Bagidua  $A(x)$  menjadi:

$$A_0(x) = 2 + 5x \text{ dan } A_1(x) = 3 + x - x^2 \text{ sehingga } A(x) = A_0(x) + A_1(x)x^2$$

Bagidua  $B(x)$  menjadi:

$$B_0(x) = 1 + 2x \text{ dan } B_1(x) = 2 + 3x + 6x^2 \text{ sehingga } B(x) = B_0(x) + B_1(x)x^2$$

Maka

$$\begin{aligned} A(x)B(x) &= A_0(x)B_0(x) + (A_0(x)B_1(x) + A_1(x)B_0(x))x^2 + A_1(x)B_1(x)x^4 \\ &= (2 + 5x)(1 + 2x) + \{ (2 + 5x)(2 + 3x + 6x^2) + (3 + x - x^2)(1 + 2x) \}x^2 + \\ &\quad (3 + x - x^2)(2 + 3x + 6x^2)x^4 \\ &= 2 + 9x + 10x^2 + (4 + 16x + 27x^2 + 30x^3 + 3 + 7x + x^2 - 2x^3)x^2 + \\ &\quad 6 + 11x + 19x^2 + 3x^3 - 6x^4 \\ &= 2 + 9x + 17x^2 + 23x^3 + 34x^4 + 39x^5 + 19x^6 + 3x^7 - 6x^8 \end{aligned}$$

$$A(x)B(x) = A_0(x)B_0(x) + ( A_0(x)B_1(x) + A_1(x)B_0(x) ) x^{\lfloor n/2 \rfloor} + A_1(x) B_1(x) x^{2\lfloor n/2 \rfloor}$$

**function** *KaliPolinom(A, B : Polinom, n : integer) → Polinom*

{ Memberikan hasil perkalian polinom  $A(x)$  dan  $B(x)$ , masing-masing berderajat  $n$  dengan algoritma divide and conquer. }

### Deklarasi

*A0, A1, B0, B1 : Polinom*

*s : integer*

### Algoritma:

**if**  $n = 0$  **then**

**return**  $A * B$  { perkalian skalar biasa }

**else**

$s \leftarrow n \text{ div } 2$  { bagidua suku-suku polinom pada posisi  $s$  }

$A0 \leftarrow a_0 + a_1x + a_2x^2 + \dots + a_{s-1}x^{s-1}$

$A1 \leftarrow a_s + a_{s+1}x + a_{s+2}x^2 + \dots + a_nx^{n-s}$

$B0 \leftarrow b_0 + b_1x + b_2x^2 + \dots + b_{s-1}x^{s-1}$

$B1 \leftarrow b_s + b_{s+1}x + b_{s+2}x^2 + \dots + b_nx^{n-s}$

**return**  $KaliPolinom(A0, B0, s) + ( KaliPolinom(A0, B1, s) + KaliPolinom(A1, B0, s) ) * x^s +$   
 $KaliPolinom(A1, B1, s) * x^{2s}$

**endif**

## Kompleksitas algoritmanya:

- Tinjau lagi  $A(x)B(x) = A_0(x)B_0(x) + (A_0(x)B_1(x) + A_1(x)B_0(x))x^{\lfloor n/2 \rfloor} + A_1(x)B_1(x)x^{2\lfloor n/2 \rfloor}$
- Penjumlahan dua polinom berukuran  $n/2$  (operator  $+$  berwarna merah) kompleksitasnya  $O(n)$  atau  $cn$
- Kompleksitas algoritmanya:  $T(n) = \begin{cases} a & , n = 0 \\ 4T(n/2) + cn & , n > 0 \end{cases}$

- Dengan menggunakan Teorema Master,  $a = 4$ ,  $b = 2$ ,  $d = 1$ , dan memenuhi  $a > b^d$  (yaitu  $4 > 2^1$ ) maka relasi rekurens  $T(n) = 4T(n/2) + cn$  memenuhi case 3, sehingga

$$T(n) = O(n^{2\log 4}) = O(n^2)$$

- Hasil ini tidak memberi perbaikan dibandingkan dengan algoritma *brute force*.
- Dapatkah kita membuat algoritma perkalian polinom yang lebih baik?

## Perbaikan:

- Ideya sama seperti pada metode perkalian Karatsuba, yaitu dengan mengurangi jumlah operasi kali.
- Persamaan perkalian  $A(x)$  dan  $B(x)$  sebelumnya:

$$A(x)B(x) = A_0(x)B_0(x) + (A_0(x)B_1(x) + A_1(x)B_0(x))x^{\lfloor n/2 \rfloor} + A_1(x)B_1(x)x^{2\lfloor n/2 \rfloor}$$

terdapat 4 operasi perkalian dan 3 operasi penjumlahan upa-polinom derajat  $n/2$ .

- Dengan memanipulasi persamaan di atas sedemikian rupa kita dapat mengurangi jumlah operasi kali berkurang menjadi 3 (dengan konsekuensi operasi penjumlahan menjadi bertambah).

- Definisikan

$$Y(x) = (A_0(x) + A_1(x)) \times (B_0(x) + B_1(x))$$

$$U(x) = A_0(x)B_0(x)$$

$$Z(x) = A_1(x)B_1(x)$$

- Maka

$$Y(x) - U(x) - Z(x) = A_0(x)B_1(x) + A_1(x)B_0(x)$$

sehingga

$$\begin{aligned} A(x)B(x) &= A_0(x)B_0(x) + (A_0(x)B_1(x) + A_1(x)B_0(x))x^{\lfloor n/2 \rfloor} + A_1(x)B_1(x)x^{2\lfloor n/2 \rfloor} \\ &= U(x) + \{Y(x) - U(x) - Z(x)\}x^{\lfloor n/2 \rfloor} + Z(x)x^{2\lfloor n/2 \rfloor} \end{aligned}$$

Sekarang hanya terdapat tiga operasi perkalian polinom, yaitu  $U(x)$ ,  $Y(x)$ , dan  $Z(x)$

$$A(x)B(x) = U(x) + \{ Y(x) - U(x) - Z(x) \} x^{\lfloor n/2 \rfloor} + Z(x) x^{2\lfloor n/2 \rfloor}$$

**function** *KaliPolinom2(A, B : Polinom, n : integer)*  $\rightarrow$  *Polinom*

{ Memberikan hasil perkalian polinom  $A(x)$  dan  $B(x)$ , masing-masing berderajat  $n$  dengan algoritma divide and conquer. }

### Deklarasi

$A0, A1, B0, B1, U, Y, Z : Polinom$

$s : \text{integer}$

### Algoritma:

**if**  $n = 0$  **then**

**return**  $A * B$  { perkalian skalar biasa }

**else**

$s \leftarrow n \text{ div } 2$  { bagidua suku-suku polinom pada posisi  $s$  }

$A0 \leftarrow a_0 + a_1x + a_2x^2 + \dots + a_{s-1}x^{s-1}$

$A1 \leftarrow a_s + a_{s+1}x + a_{s+2}x^2 + \dots + a_nx^{n-s}$

$B0 \leftarrow b_0 + b_1x + b_2x^2 + \dots + b_{s-1}x^{s-1}$

$B1 \leftarrow b_s + b_{s+1}x + b_{s+2}x^2 + \dots + b_nx^{n-s}$

$Y \leftarrow KaliPolinom2(A0 + A1, B0 + B1, s)$

$U \leftarrow KaliPolinom2(A0, B0, s)$

$Z \leftarrow KaliPolinom2(A1, B1, s)$

**return**  $U + (Y - U - Z) * x^s + Z * x^{2s}$

**endif**

## Kompleksitas algoritmanya:

$$T(n) = \begin{cases} a & , n = 0 \\ 3T(n/2) + cn & , n > 0 \end{cases}$$

- Dengan menggunakan Teorema Master,  $a = 3$ ,  $b = 2$ ,  $d = 1$ , dan memenuhi  $a > b^d$  (yaitu  $3 > 2^1$ ) maka relasi rekurens  $T(n) = 3T(n/2) + cn$  memenuhi case 3, sehingga

$$T(n) = O(n^{2\log 3}) = O(n^{1.59})$$

- Hasil ini lebih baik dibandingkan dengan algoritma *divide and conquer* sebelumnya

BERSAMBUNG