

6.2 - Divide and Conquer (part 2)

[KOMS120403]

Desain dan Analisis Algoritma (2022/2023)

Dewi Sintari

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 6 (March 2023)

Daftar isi

- Teorema Master
- Matrix multiplication
- Perkalian matriks Strassen
- Perkalian bilangan besar
- Perkalian Karatsuba

Bagian 1. Teorema Master

Bagaimana menangani
lama perhitungan kompleksitas waktu?

Prinsip dasar Teorema Master

Saat menganalisis algoritma, ingatlah bahwa kita hanya peduli pada perilaku asimtotik.

Teorema Master dapat digunakan untuk menentukan notasi asimtotik kompleksitas waktu dalam bentuk relasi perulangan dengan mudah tanpa harus menyelesaikannya secara iteratif.

Teorema (Teorema Master)

*Ingat kembali fungsi kompleksitas waktu: $T(n) = aT(n/b) + f(n)$.
Jika $f(n) \in \Theta(n^d)$ dimana $d \geq 0$, maka:*

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

Hasil yang serupa juga berlaku untuk notasi Ω dan \mathcal{O} .

Teorema Master: Contoh 1

Pada Merge Sort/Quick Sort,

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

- $T(n) = aT(n/b) + cn^d$
- $a = 2, b = 2, d = 1$
- $a = b^d$ terpenuhi (yakni $2 = 2^1$)

Jadi relasi perulangan $T(n) = aT(n/b) + cn^d$ memenuhi *kondisi ke-2* dari fungsi berikut.

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

Jadi, $T(n) \in \mathcal{O}(n \log n)$.

Teorema Master: Contoh 2

Dalam algoritma **Powering** untuk menghitung X^n ,

$$T(n) \in \begin{cases} 1, & \text{untuk } n = 0 \\ T(n/2) + 1, & \text{untuk } n > 0 \end{cases}$$

- $T(n) = aT(n/b) + cn^d$
- $a = 1, b = 2, d = 0$
- $a = b^d$ terpenuhi (yakni $1 = 2^0$)

Jadi relasi perulangan $T(n) = aT(n/b) + cn^d$ memenuhi *kondisi ke-2* dari fungsi berikut.

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

Jadi, $T(n) \in \mathcal{O}(n^0 \log n) = \mathcal{O}(\log n)$.

Teorema Master: Contoh 3

Algoritma **penjumlahan array berbasis Divide-and-Conquer**, mengingat ukuran masukan adalah $n = 2^k$, sehingga fungsi kompleksitas waktunya adalah:

$$T(n) = 2T(n/2) + 1$$

sebab:

- pada setiap langkah, masalah dibagi menjadi 2 sub-masalah dengan ukuran yang sama ($b = 2$), dan keduanya harus diselesaikan ($a = 2$).
- fungsi kompleksitas DIVIDE dan COMBINE adalah $f(n) \in \Theta(1) = \Theta(n^0)$

Sehingga,

$$T(n) \in \Theta\left(n^{\log_b a}\right) = \left(n^{\log_2 2}\right) = \Theta(n)$$

Teorema Master: Contoh 4

Misalkan $T(n) = 2T(\frac{n}{4}) + \sqrt{n} + 42$. Tentukan parameter a , b , dan d seperti pada teorema.

Teorema Master: Contoh 4

Misalkan $T(n) = 2T(\frac{n}{4}) + \sqrt{n} + 42$. Tentukan parameter a , b , dan d seperti pada teorema.

$$a = 2; \quad b = 4; \quad d = \frac{1}{2}$$

Kondisi manakah dari Teorema Master yang memenuhi?

Teorema Master: Contoh 4

Misalkan $T(n) = 2T(\frac{n}{4}) + \sqrt{n} + 42$. Tentukan parameter a , b , dan d seperti pada teorema.

$$a = 2; \quad b = 4; \quad d = \frac{1}{2}$$

Kondisi manakah dari Teorema Master yang memenuhi?

Karena $2 = 4^{\frac{1}{2}}$, kasus kedua Teorema Master berlaku. Karenanya,

$$T(n) \in \Theta(n^d \log n) = \Theta(\sqrt{n} \log n)$$

Teorema Master: kelebihan dan kebenaran teorema

Divide-and-Conquer + Teorema Master = ?

Teorema Master: kelebihan dan kebenaran teorema

Divide-and-Conquer + Teorema Master = ?

Kombinasi keduanya memberikan kita kemampuan untuk melakukan iterasi dengan sangat cepat antara **desain algoritma** dan **analisis kompleksitas**-nya.

Very **pro** way of algorithm development!

Teorema Master: kelebihan dan kebenaran teorema

Divide-and-Conquer + Teorema Master = ?

Kombinasi keduanya memberikan kita kemampuan untuk melakukan iterasi dengan sangat cepat antara **desain algoritma** dan **analisis kompleksitas**-nya.

Very **pro** way of algorithm development!

Buktinya bisa dibaca di catatan kuliah ini (hal 3-4):

<https://web.stanford.edu/class/archive/cs/cs161/cs161.1182/Lectures/Lecture3/CS161Lecture03.pdf>

Teorema Master: Kelebihan & kekurangan

- Teorema master memungkinkan Anda beralih dari perulangan ke ikatan asimtotik dengan sangat cepat.
- Teorema Master biasanya bekerja dengan baik untuk algoritma Divide-and-Conquer.
- Tetapi teorema Master *tidak berlaku untuk semua perulangan*, seperti pada kondisi berikut:
 - ▶ $T(n)$ tidak monoton, misalnya: $T(n) = \sin n$
 - ▶ $f(n)$ bukan polinomial, misalnya: $T(n) = 2T(\frac{n}{2}) + 2^n$
 - ▶ b tidak dapat dinyatakan sebagai konstanta, misalnya: $T(n) = T(\sqrt{n})$
- Jika tidak berlaku, hal yang dapat dilakukan antara lain:
 - ▶ ambil batas atas/bawah (namun dengan konsekuensi mendapatkan batas yang lebih “lemah”)
 - ▶ terapkan metode substitusi

Bagian 2. Perkalian matriks

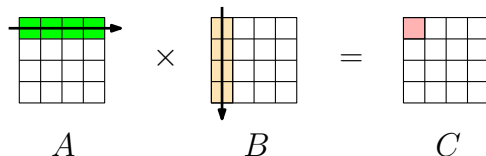
Perkalian matriks persegi (1)

Permasalahan

Diberikan dua matriks persegi A dan B . Hitunglah $A \times B$

Misalkan $A = [a_{ij}]$, $B = [b_{ij}]$ adalah matriks berukuran $n \times n$, dan $C = A \times B$.

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



Perkalian matriks persegi (2)

Pendekatan brute-force: hitung setiap elemen C satu per satu dengan mengalikan baris yang sesuai dari A dan kolom B .

Algorithm 1 Perkalian matriks persegi (*brute force*)

```
1: procedure MATRIXMULT( $A, B$ )
2:   for  $i \leftarrow 1$  to  $n$  do
3:     for  $j \leftarrow 1$  to  $n$  do
4:        $C[i, j] \leftarrow 0$ 
5:       for  $k \leftarrow 1$  to  $n$  do
6:          $C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$ 
7:       end for
8:     end for
9:   end for
10:  return  $C$ 
11: end procedure
```

Kompleksitas waktu: $\mathcal{O}(n^3)$

Perkalian matriks persegi (3)

Matriks A dan B masing-masing dipartisi menjadi empat submatriks dengan ukuran $\frac{n}{2} \times \frac{n}{2}$.

$$\begin{array}{ccc} \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array} \right] & \times & \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array} \right] = \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array} \right] \\ A & & B \qquad \qquad \qquad C \end{array}$$

Oleh karena itu, komponen matriks C dapat dihitung sebagai berikut:

- $C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$
- $C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$
- $C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$
- $C_{22} = A_{21} \cdot B_{22} + A_{22} \cdot B_{22}$

Perkalian matriks persegi (4)

Contoh

Matriks bujur sangkar dapat dipecah sebagai berikut:

$$A = \begin{bmatrix} 1 & 21 & 15 & 7 \\ 11 & 3 & 10 & 31 \\ 52 & 31 & 2 & 17 \\ 2 & 9 & 23 & 3 \end{bmatrix}$$

$$A_{11} = \begin{bmatrix} 1 & 21 \\ 11 & 3 \end{bmatrix}$$

$$A_{12} = \begin{bmatrix} 15 & 7 \\ 10 & 31 \end{bmatrix}$$

$$A_{21} = \begin{bmatrix} 52 & 31 \\ 2 & 9 \end{bmatrix}$$

$$A_{22} = \begin{bmatrix} 2 & 17 \\ 23 & 3 \end{bmatrix}$$

Perkalian matriks persegi (5): Pseudocode (*part 1*)

Algorithm 2 Matrix multiplication

```
1: procedure MMUL( $A, B$ : matrices,  $n$ : integer)
2:   if  $n = 1$  then                                     ▷ The matrices are of size  $1 \times 1$ 
3:     return  $A * B$                                        ▷ Scalar multiplication
4:   else
5:     SPLIT( $A$ )
6:     SPLIT( $B$ )
7:      $C_{11} \leftarrow$  MSUM(MMUL( $A_{11}, B_{11}, \frac{n}{2}$ ), MMUL( $A_{12}, B_{21}, \frac{n}{2}$ ))
8:      $C_{12} \leftarrow$  MSUM(MMUL( $A_{11}, B_{12}, \frac{n}{2}$ ), MMUL( $A_{12}, B_{22}, \frac{n}{2}$ ))
9:      $C_{21} \leftarrow$  MSUM(MMUL( $A_{21}, B_{11}, \frac{n}{2}$ ), MMUL( $A_{22}, B_{21}, \frac{n}{2}$ ))
10:     $C_{22} \leftarrow$  MSUM(MMUL( $A_{21}, B_{12}, \frac{n}{2}$ ), MMUL( $A_{22}, B_{22}, \frac{n}{2}$ ))
11:   end if
12:   return  $C$                                              ▷  $C$  is the union of  $C_{11}, C_{12}, C_{21}, C_{22}$ 
13: end procedure
```

Perkalian matriks persegi (6): Pseudocode (*part 2*)

Prosedur MSUM yang digunakan dalam MMUL adalah sebagai berikut.

Algorithm 3 Sum of two matrices

```
1: procedure MSUM( $A, B$ : matrices,  $n$ : integer)
2:   for  $i \leftarrow 1$  to  $n$  do
3:     for  $j \leftarrow 1$  to  $n$  do
4:        $C[i, j] \leftarrow A[i, j] + B[i, j]$ 
5:     end for
6:   end for
7: end procedure
```

Kompleksitas waktu: $\mathcal{O}(n^2)$

Perkalian matriks persegi (7): Kompleksitas waktu

Rumus rekursif untuk kompleksitas waktu diberikan oleh:

- Dengan **Teorema Master**:

$$T(n) = aT\left(\frac{n}{b}\right) + cn^d$$

dimana $a = 8$, $b = 2$, $d = 2$.

- Relasi $a > b^d$ (yaitu $8 > 2^2$) terpenuhi.
- Jadi $T(n)$ memenuhi kondisi ke-3 dari Teorema Master. Sehingga:

$$T(n) \in \mathcal{O}(n^{\log_2 8}) = \mathcal{O}(n^3)$$

Ini memberi kompleksitas waktu dengan *urutan besarnya sama dengan brute force*. Jadi algoritmanya tidak cukup “powerful”. Dapatkah kita mendapatkan kompleksitas waktu yang lebih baik?

Bagian 3. Perkalian Matriks Strassen



Figure: Volker Strassen (born in 1936, German mathematician)

Perkalian matriks Strassen (1)

- Ide Volker Strassen adalah untuk mengurangi banyaknya 'perkalian' dalam prosedur. Karena biaya 'perkalian' lebih 'mahal' daripada 'penambahan' (lihat https://www.wikiwand.com/en/Computational_complexity_of_mathematical_operations).
- Operasi berikut terdiri dari 8 perkalian dan 4 penjumlahan:
 - ▶ $C_{11} = A_{11} \cdot B_{11} + A_{12} \cdot B_{21}$
 - ▶ $C_{12} = A_{11} \cdot B_{12} + A_{12} \cdot B_{22}$
 - ▶ $C_{21} = A_{21} \cdot B_{11} + A_{22} \cdot B_{21}$
 - ▶ $C_{22} = A_{21} \cdot B_{22} + A_{22} \cdot B_{22}$
- Strassen memodifikasi persamaan di atas untuk mengurangnya menjadi 7 perkalian tetapi dengan konsekuensi terdapat lebih banyak penjumlahan.

Perkalian matriks Strassen (2)

Modifikasi perkalian matriks oleh Strassen adalah sebagai berikut:

- $M_1 = (A_{11} - A_{22})(B_{21} + B_{22})$
- $M_2 = (A_{11} + A_{22})(B_{11} + B_{22})$
- $M_3 = (A_{11} - A_{21})(B_{11} + B_{12})$
- $M_4 = (A_{11} + A_{12})B_{22}$
- $M_5 = A_{11}(B_{12} - B_{22})$
- $M_6 = A_{22}(B_{21} - B_{11})$
- $M_7 = (A_{21} + A_{22})B_{11}$

Sehingga:

- $C_{11} = M_1 + M_2 - M_4 + M_6$
- $C_{12} = M_4 + M_5$
- $C_{21} = M_6 + M_7$
- $C_{22} = M_2 - M_3 + M_5 - M_7$

Operasi ini terdiri dari 7 perkalian dan 18 penjumlahan.

Algorithm 4 Matrix multiplication

```
1: procedure STRASSEN( $A, B$ : matrices,  $n$ : integer)
2:   if  $n = 1$  then return  $A * B$ 
3:   else
4:     SPLIT( $A$ )
5:     SPLIT( $B$ )
6:      $M_1 \leftarrow \text{STRASSEN}(A_{12} - A_{22}, B_{21} + B_{22}, \frac{n}{2})$ 
7:      $M_2 \leftarrow \text{STRASSEN}(A_{11} + A_{22}, B_{11} + B_{22}, \frac{n}{2})$ 
8:      $M_3 \leftarrow \text{STRASSEN}(A_{11} - A_{21}, B_{11} + B_{12}, \frac{n}{2})$ 
9:      $M_4 \leftarrow \text{STRASSEN}(A_{11} + A_{12}, B_{22}, \frac{n}{2})$ 
10:     $M_5 \leftarrow \text{STRASSEN}(A_{11}, B_{12} - B_{22}, \frac{n}{2})$ 
11:     $M_6 \leftarrow \text{STRASSEN}(A_{22}, B_{21} - B_{11}, \frac{n}{2})$ 
12:     $M_7 \leftarrow \text{STRASSEN}(A_{21} + A_{22}, B_{11}, \frac{n}{2})$ 
13:     $C_{11} \leftarrow M_1 + M_2 - M_4 + M_6$ 
14:     $C_{12} \leftarrow M_4 + M_5$ 
15:     $C_{21} \leftarrow M_6 + M_7$ 
16:     $C_{22} \leftarrow M_2 - M_3 + M_5 - M_7$ 
17:   end if
18:   return  $C$ 
19: end procedure
```

▷ *Scalar multiplication*

▷ C is the union of $C_{11}, C_{12}, C_{21}, C_{22}$

Perkalian matriks Strassen (3)

Rumus rekursif untuk **kompleksitas waktu** diberikan oleh:

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

- Sesuai Teorema Master, $T(n) = aT\left(\frac{n}{b}\right) + cn^d$, dimana $a = 7$, $b = 2$, $d = 2$.
- Relasi $a > b^d$ (yakni $7 > 2^2$) terpenuhi.
- Jadi $T(n)$ memenuhi kasus ke-3 dari Teorema Master. Karenanya:

$$T(n) = \mathcal{O}(n \log_2 7) = \mathcal{O}(n^{2.81})$$

Ini memberikan kompleksitas waktu yang lebih baik daripada algoritma DnC sebelumnya.

Bagian 4. Perkalian bilangan besar

Perkalian bilangan besar (1): definisi

Sebuah **bilangan besar** adalah bilangan yang terdiri dari n digit atau n bit.

Contoh: 564389018149014329871520, 1000011011010100100110010101,
...

Permasalahan bilangan besar

- Bahasa pemrograman memiliki keterbatasan dalam merepresentasikan angka yang besar
- Di C, tipe angka adalah `char` (8 bit), `int` (6 bit), dan `long` (32 bit)
- Untuk angka yang lebih besar dari 32 bit, kita harus mendefinisikan *new type* dan mendefinisikan operasi aritmatika primitif (+, -, *, /, dll.)

Perkalian bilangan besar (2): deskripsi permasalahan

Kita akan membahas bagaimana sebuah algoritma dapat melakukan perkalian dengan bilangan yang besar

Contoh: $1765420875208345186 \times 754711199736308361736432$

Permasalahan

Diberikan dua bilangan bulat X dan Y dari n digit (atau n bit):

$$X = x_1x_2x_3 \dots x_n$$

$$Y = y_1y_2y_3 \dots y_n$$

Hitunglah $X \times Y$

Perkalian bilangan besar (3): model perkalian konvensional

Contoh

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

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

Metode perkalian konvensional untuk $X \times Y$:

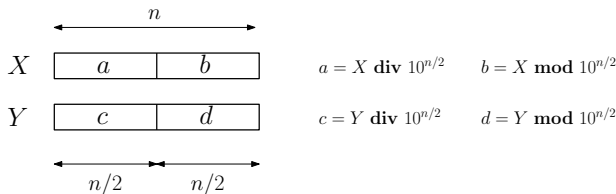
$$\begin{array}{r} X \times Y = 1234 \\ \quad \underline{5678 \times} \\ \quad 9872 \\ \quad 8368 \\ \quad 7404 \\ \underline{6170} \quad + \\ 7006652 \end{array}$$

Perkalian bilangan besar (4): pseudocode

Algorithm 5 Perkalian bilangan besar (*brute force*)

```
1: procedure MULT( $X, Y$ : long integer,  $n$ : integer)
2:   declaration
3:     temp, unit, tens: integer
4:   end declaration
5:   for setiap digit  $y_i$  of  $y_n, y_{n-1}, \dots, y_1$  do
6:     tens  $\leftarrow 0$ 
7:     for every digit  $x_j$  of  $x_n, x_{n-1}, \dots, x_1$  do
8:       temp  $\leftarrow x_j * y_i$ 
9:       temp  $\leftarrow$  temp + tens
10:      unit  $\leftarrow$  temp mod 10
11:      tens  $\leftarrow$  temp div 10
12:      print(unit)
13:    end for
14:  end for
15:   $Z \leftarrow$  jumlahkan semua hasil perkalian dari atas ke bawah
16:  return  $Z$ 
17: end procedure
```

Perkalian bilangan besar (5): pendekatan dengan DnC



X dan Y dapat direpresentasikan sebagai a , b , c , dan d :

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

Perkalian X dan Y direpresentasikan sebagai:

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

Perkalian bilangan besar (6): pendekatan dengan DnC

Contoh

Misalkan $n = 6$, $X = 346769$ 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 dan Y dapat ditulis sebagai:

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

Operasi ini melibatkan **empat** perkalian bilangan besar.

Perkalian bilangan besar (6): pseudocode DnC

Algorithm 6 Perkalian bilangan besar (DnC)

```
1: procedure MULT2( $X, Y$ : long integer,  $n$ : integer)
2:   declaration
3:      $a, b, c, d$ : Long integer,  $s$ : integer
4:   end declaration
5:   if  $n = 1$  then
6:     return  $X * Y$  ▷ scalar multiplication
7:   else
8:      $s \leftarrow n \text{ div } 2$ 
9:      $a \leftarrow X \text{ div } 10^s$ 
10:     $b \leftarrow X \text{ mod } 10^s$ 
11:     $c \leftarrow Y \text{ div } 10^s$ 
12:     $d \leftarrow Y \text{ mod } 10^s$ 
13:    return  $\text{MULT2}(a, c, s) * 10^{2s} + \text{MULT2}(b, c, s) * 10^s + \text{MULT2}(a, d, s) * 10^s + \text{MULT2}(b, d, s)$ 
14:  end if
15: end procedure
```

Perkalian bilangan besar (6): kompleksitas waktu

Kompleksitas waktu dari MULT2

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

Catatan. Prosedur untuk menghitung 10^s dan 10^{2s} pada algoritma dapat dilakukan dengan menambahkan nol sebanyak s atau sebanyak $2s$.

Dengan Teorema Master, diperoleh (**coba Anda buktikan!**):

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

Secara asimtotik, algoritma ini memiliki kompleksitas yang sama dengan algoritma brute force. Bisakah kita mendesain algoritma untuk masalah ini dengan kompleksitas waktu yang lebih baik?

Bagian 5. Perkalian Karatsuba



Figure: Anatoly Alexeyevich Karatsuba (1937-2008, Russian mathematician)

Perkalian Karatsuba (1): definisi

Perbaikan dari algoritma perkalian sebelumnya

Idenya mirip dengan *Perkalian matriks Strassen*, dengan mengurangi banyaknya perkalian.

Algoritma sebelumnya memberikan:

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

Karatsuba memanipulasi persamaan di atas sehingga hanya membutuhkan **3 perkalian**, tetapi akibatnya, dibutuhkan **lebih banyak penjumlahan**.

Perkalian Karatsuba (2): algoritma

Misalkan

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

Maka

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

Jadi, perkalian $X \cdot Y$ dapat ditulis sebagai:

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

Sekarang algoritma tersebut hanya berisi 3 perkalian, yaitu untuk menghitung nilai p , q , dan r .

Perkalian Karatsuba (3): pseudocode

Algorithm 7 Perkalian Karatsuba

```
1: procedure MULT3( $X, Y$ : long integer,  $n$ : integer)
2:   declaration
3:      $a, b, c, d, p, q, r$ : Long integer,  $s$ : integer
4:   end declaration
5:   if  $n = 1$  then
6:     return  $X * Y$ 
7:   else
8:      $s \leftarrow n \text{ div } 2$ 
9:      $a \leftarrow X \text{ div } 10^s$ 
10:     $b \leftarrow X \bmod 10^s$ 
11:     $c \leftarrow Y \text{ div } 10^s$ 
12:     $d \leftarrow Y \bmod 10^s$ 
13:     $p \leftarrow \text{MULT3}(a, c, s)$ 
14:     $q \leftarrow \text{MULT3}(b, d, s)$ 
15:     $r \leftarrow \text{MULT3}(a + b, c + d, s)$ 
16:    return  $p * 10^{2s} + (r - p - q) * 10^s + q$ 
17:   end if
18: end procedure
```

▷ scalar multiplication

Perkalian Karatsuba (3): kompleksitas waktu

Kompleksitas waktu MULT3

Misalkan $T(n)$ adalah kompleksitas waktu dari tiga perkalian bilangan bulat $n/2$ digit + penjumlahan bilangan bulat $n/2$ digit

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

Dari $T(n) = 3T(n/2) + cn$, kita memperoleh $a = 3$, $b = 2$, $d = 1$, dan $a > b^d$ (yaitu $3 > 2^1$).

Jadi rumus perulangan memenuhi kondisi ke-3 Teorema Master (yaitu $a > b^d$). Jadi:

$$T(n) = \mathcal{O}(n^{\log_2 3}) = \mathcal{O}(n^{1.59})$$

Ini lebih baik dari MULT2 (yaitu $\mathcal{O}(n^2)$).

Rangkuman

Kelebihan dari metode DnC

- **Memecahkan masalah yang sulit:** Ini adalah metode ampuh untuk memecahkan masalah yang sulit. Membagi masalah menjadi submasalah sehingga submasalah dapat digabungkan kembali merupakan kesulitan utama dalam mendesain algoritma baru. Untuk banyak masalah seperti itu, algoritma ini memberikan solusi sederhana.
- **Sifat paralel:** Karena ini memungkinkan kita untuk menyelesaikan submasalah secara mandiri, ini memungkinkan eksekusi dalam mesin multi-prosesor, khususnya sistem memori bersama di mana komunikasi data antar prosesor tidak perlu direncanakan sebelumnya, karena submasalah yang berbeda dapat dijalankan pada prosesor yang berbeda.

Kekurangan dari metode DnC

- **Rekursi berjalan lambat:** Ini karena tumpang tindih dari pemanggilan berulang dari submasalah. Algoritma juga membutuhkan *stack* untuk menyimpan *recursive call*-nya. (Tapi sebenarnya ini tergantung pada cara implementasinya.)

end of slide...