6.2 - Divide and Conquer (part 2)

[KOMS120403]

Desain dan Analisis Algoritma (2023/2024)

Dewi Sintiari

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 6 (March 2024)

Daftar isi

Bagian 1. Teorema Master

Bagaimana menangani
kesulitan pada 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:*

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

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

Teorema Master: Contoh 1

Pada Merge Sort/Quick Sort,

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

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

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

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

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

Teorema Master: Contoh 2

Dalam algoritma Powering untuk menghitung X^n,

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

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

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

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

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.

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

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.

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

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.

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: untuk fungsi $T(\sqrt{n})$
- Jika TM 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

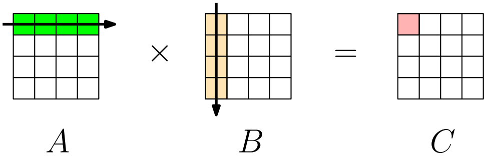
Perkalian matriks persegi (1)

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*)

```
procedure MatrixMult(A, B)
    for i ← 1 to n do
        for j ← 1 to n do
            C[i, j] ← 0
            for k ← 1 to n do
                C[i, j] ← C[i, j] + A[i, k] * B[k, j]
            end for
        end for
    end for
    return C
end procedure
```

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

Perkalian matriks persegi (3)

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

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

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:

$$A = \left[\begin{array}{cc|cc} 1 & 21 & 15 & 7 \\ 11 & 3 & 10 & 31 \\ \hline 52 & 31 & 2 & 17 \\ 2 & 9 & 23 & 3 \end{array}\right]$$

$$A_{11} = \begin{bmatrix} 1 & 21 \\ 11 & 3 \end{bmatrix} \quad A_{12} = \begin{bmatrix} 15 & 7 \\ 10 & 31 \end{bmatrix}$$

$$A_{21} = \begin{bmatrix} 52 & 31 \\ 2 & 9 \end{bmatrix} \quad A_{22} = \begin{bmatrix} 2 & 17 \\ 23 & 3 \end{bmatrix}$$

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

Algorithm 2 Matrix multiplication

procedure $\textsc{Mmul}(A, B\text{: matrices}, n\text{: integer})$
    if $n = 1$ **then** ▷ *The matrices are of size* 1×1
        return $A * B$ ▷ *Scalar multiplication*
    else
        $\textsc{Split}(A)$
        $\textsc{Split}(B)$
        $C_{11} \leftarrow \textsc{Msum}(\textsc{Mmul}(A_{11}, B_{11}, \frac{n}{2}), \textsc{Mmul}(A_{12}, B_{21}, \frac{n}{2}))$
        $C_{12} \leftarrow \textsc{Msum}(\textsc{Mmul}(A_{11}, B_{12}, \frac{n}{2}), \textsc{Mmul}(A_{12}, B_{22}, \frac{n}{2}))$
        $C_{21} \leftarrow \textsc{Msum}(\textsc{Mmul}(A_{21}, B_{11}, \frac{n}{2}), \textsc{Mmul}(A_{22}, B_{21}, \frac{n}{2}))$
        $C_{22} \leftarrow \textsc{Msum}(\textsc{Mmul}(A_{21}, B_{12}, \frac{n}{2}), \textsc{Mmul}(A_{22}, B_{22}, \frac{n}{2}))$
    end if
    return C ▷ *C is the union of $C_{11}, C_{12}, C_{21}, C_{22}$*
end procedure

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

Prosedur MSUM yang digunakan dalam MMUL adalah sebagai berikut.

Algorithm 3 Sum of two matrices

procedure MSUM(A, B: matrices, n: integer)
  for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** n **do**
      $C[i,j] \leftarrow A[i,j] + B[i,j]$
    end for
  end for
end procedure

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

Perkalian matriks persegi (7): Kompleksitas waktu

Misal *T(n): banyaknya perkalian matriks.*

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 menguranginya 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

procedure Strassen(A, B: matrices, n: integer)
    if $n = 1$ **then return** $A * B$ ▷ *Scalar multiplication*
    else
        Split(A)
        Split(B)
        $M_1 \leftarrow \textsc{Strassen}(A_{12} - A_{22}, B_{21} + B_{22}, \frac{n}{2})$
        $M_2 \leftarrow \textsc{Strassen}(A_{11} + A_{22}, B_{11} + B_{22}, \frac{n}{2})$
        $M_3 \leftarrow \textsc{Strassen}(A_{11} - A_{21}, B_{11} + B_{12}, \frac{n}{2})$
        $M_4 \leftarrow \textsc{Strassen}(A_{11} + A_{12}, B_{22}, \frac{n}{2})$
        $M_5 \leftarrow \textsc{Strassen}(A_{11}, B_{12} - B_{22}, \frac{n}{2})$
        $M_6 \leftarrow \textsc{Strassen}(A_{22}, B_{21} - B_{11}, \frac{n}{2})$
        $M_7 \leftarrow \textsc{Strassen}(A_{21} + A_{22}, B_{11}, \frac{n}{2})$
        $C_{11} \leftarrow M_1 + M_2 - M_4 + M_6$
        $C_{12} \leftarrow M_4 + M_5$
        $C_{21} \leftarrow M_6 + M_7$
        $C_{22} \leftarrow M_2 - M_3 + M_5 - M_7$
    end if
    return C ▷ *C is the union of $C_{11}, C_{12}, C_{21}, C_{22}$*
end procedure

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: $1765420875208345186 \times 7547111997363083617364 32$

Permasalahan

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

$$X = x_1 x_2 x_3 \ldots x_n$$
$$Y = y_1 y_2 y_3 \ldots y_n$$

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 $X \times Y$:

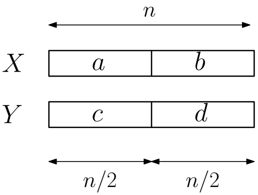
$$
\begin{array}{rr}
X \times Y = & 1234 \\
 & \underline{5678 \times} \\
 & 9872 \\
 & 8368 \\
 & 7404 \\
 & \underline{6170 +} \\
 & 7006652
\end{array}
$$

Perkalian bilangan besar (4): pseudocode

Algorithm 5 Perkalian bilangan besar (*brute force*)

```
procedure MULT(X, Y: long integer, n: integer)
    declaration
        temp, unit, tens: integer
    end declaration
    for setiap digit y_i of y_n, y_{n-1}, ..., y_1 do
        tens ← 0
        for every digit x_j of x_n, x_{n-1}, ..., x_1 do
            temp ← x_j * y_i
            temp ← temp + tens
            unit ← temp  mod 10
            tens ← temp div 10
            print(unit)
        end for
    end for
    Z ← jumlahkan semua hasil perkalian dari atas ke bawah
    return Z
end procedure
```

Perkalian bilangan besar (5): pendekatan dengan DnC



$a = X \textbf{ div } 10^{n/2}$ $\quad b = X \textbf{ mod } 10^{n/2}$

$c = Y \textbf{ div } 10^{n/2}$ $\quad d = Y \textbf{ mod } 10^{n/2}$

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)

```
procedure MULT2(X, Y: long integer, n: integer)
    declaration
        a, b, c, d: Long integer,  s: integer
    end declaration
    if n = 1 then
        return X * Y                                          ▷ scalar multiplication
    else
        s ← n div 2
        a ← X div 10^s
        b ← X mod 10^s
        c ← Y div 10^s
        d ← Y mod 10^s
        return MULT2(a, c, s)*10^(2s) + MULT2(b, c, s)*10^s + MULT2(a, d, s)*10^s + MULT2(b, d, s)
    end if
end procedure
```

Perkalian bilangan besar (6): kompleksitas waktu

Kompleksitas waktu dari $\textsc{Mult2}$

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

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:

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

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

Perkalian Karatsuba (3): pseudocode

Algorithm 7 Perkalian Karatsuba

```
procedure MULT3(X, Y: long integer, n: integer)
    declaration
        a, b, c, d, p, q, r: Long integer,  s: integer
    end declaration
    if n = 1 then
        return X * Y                                  ▷ scalar multiplication
    else
        s ← n div 2
        a ← X div 10^s
        b ← X mod 10^s
        c ← Y div 10^s
        d ← Y mod 10^s
        p ← MULT3(a, c, s)
        q ← MULT3(b, d, s)
        r ← MULT3(a + b, c + d, s)
        return p * 10^(2s) + (r − p − q) * 10^s + q
    end if
end procedure
```

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...