Algoritma *Divide and Conquer* (Bagian 2)

Bahan Kuliah

IF2251 Strategi Algoritmik

Oleh: Rinaldi Munir

(c) Quick Sort

- Termasuk pada pendekatan sulit membagi, mudah menggabung *(hard split/easy join)*

- Tabel A dibagi (istilahnya: dipartisi) menjadi $A1$ dan $A2$ sedemikian sehingga elemen-elemen $A1 \leq$ elemen-elemen $A2$.

Partisi:

A1

4	2	3	1

A2

9	21	5	12

Sort:

A1

1	2	3	4

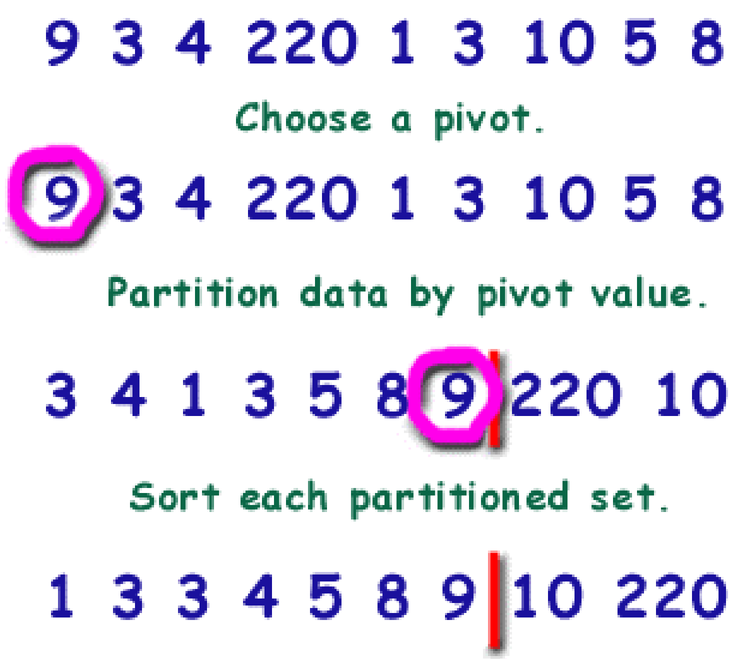
A2

5	9	12	21

Combine:

A

1	2	3	4	5	9	12	21



Teknik mem-partisi tabel:

(i) pilih $x \in \{ A[1], A[2], ..., A[n] \}$ sebagai *pivot*,

(ii) pindai tabel dari kiri sampai ditemukan $A[p] \geq x$

(iii) pindai tabel dari kanan sampai ditemukan $A[q] \leq x$

(iv) pertukarkan $A[p] \Leftrightarrow A[q]$

(v) ulangi (ii), dari posisi $p + 1$, dan (iii), dari posisi $q - 1$, sampai kedua pemindaian bertemu di tengah tabel

Contoh 4.6. Misalkan tabel *A* berisi elemen-elemen berikut:

8 1 4 6 9 3 5 7

Langkah-langkah partisi:

(i):	8	1	4	$\boxed{6}$ *pivot*	9	3	5	7
	$\rightarrow$							$\leftarrow$
(ii) & (iii):	8	1	4	6	9	3	5	7
	$\uparrow p$						$\uparrow q$	
(iv):	5	1	4	6	9	3	8	7

		→				←		
(ii) & (iii):	5	1	4	6	9	3	8	7
				↑p		↑q		

(iv):	5	1	4	3	9	6	8	7

				→←				
(ii) & (iii):	5	1	4	3	9	6	8	7
				↑q	↑p	($q < p$, berhenti)		

Hasil partisi pertama:

kiri:	5	1	4	3	(< 6)
kanan:	9	6	8	7	(≥ 6)

5 1 4 3

↑p ↑*q*

9 6 8 7

↑*p* ↑*q*

1 5 4 3

6 9 8 7

1 5 4 3

↑*q* ↑*p*

(*q* > *p* , berhenti)

6 9 8 7

↑*q* ↑p

(*q* > *p* , berhenti)

1

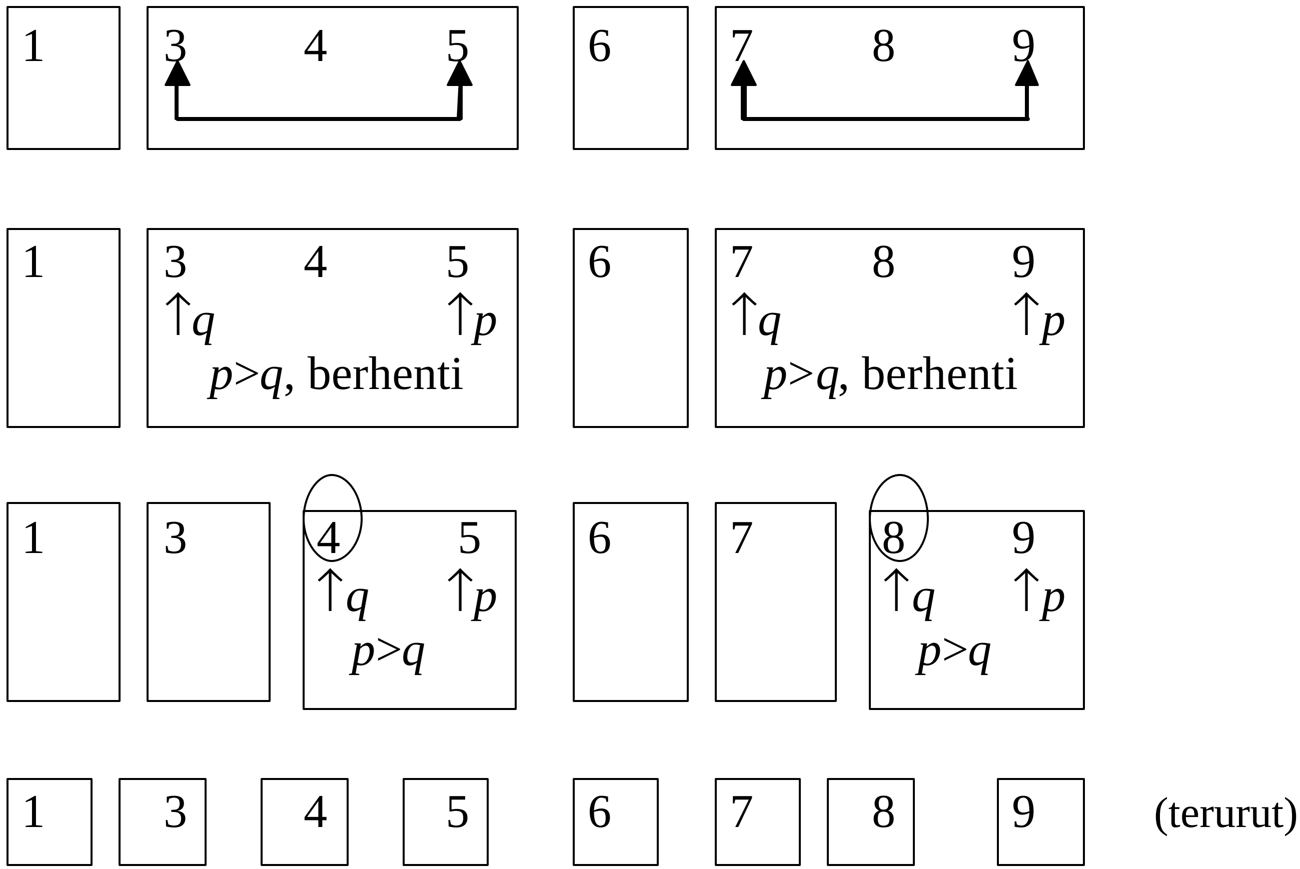
5 4 3

↑*p* ↑*q*

6

9 8 7

↑*p* ↑*q*


1
3 4 5
6
7 8 9
1
3 4 5
↑q ↑p
p>q, berhenti
6
7 8 9
↑q ↑p
p>q, berhenti
1
3
4 5
↑q ↑p
p>q
6
7
8 9
↑q ↑p
p>q
1 3 4 5 6 7 8 9 (terurut)

Pseudo-code Quick Sort:

```
procedure QuickSort(input/output A : TabelInt, input i,j: integer)

{ Mengurutkan tabel A[i..j] dengan algoritma Quick Sort.
  Masukan: Tabel A[i..j] yang sudah terdefinisi elemen-elemennya.
  Keluaran: Tabel A[i..j] yang terurut menaik.
}
Deklarasi
   k : integer

Algoritma:
    if  i < j then               { Ukuran(A) > 1 }
       Partisi(A, i, j, k)    { Dipartisi pada indeks k }
       QuickSort(A, i, k)     { Urut A[i..k] dengan Quick Sort }
       QuickSort(A, k+1, j)   { Urut A[k+1..j] dengan Quick Sort }
    endif
```

```
procedure Partisi(input/output A : TabelInt, input i, j : integer,
                  output q : integer)

{ Membagi tabel A[i..j] menjadi upatabel A[i..q] dan A[q+1..j]
  Masukan: Tabel A[i..j]yang sudah terdefinisi harganya.
  Keluaran upatabel A[i..q] dan upatabel A[q+1..j] sedemikian sehingga
           elemen tabel A[i..q] lebih kecil dari elemen tabel A[q+1..j]
}
Deklarasi
   pivot, temp : integer

Algoritma:
   pivot←A[(i + j) div 2]    { pivot = elemen tengah}
   p ← i
   q ← j
   repeat
      while A[p] < pivot do
        p ← p + 1
      endwhile
      { A[p] >= pivot}

      while A[q] > pivot do
        q ← q – 1
      endwhile
      { A[q] <= pivot}

      if p ≤ q  then
        {pertukarkan A[p] dengan A[q] }
         temp ← A[p]
         A[p] ← A[q]
         A[q] ← temp

        {tentukan awal pemindaian berikutnya }
        p ← p + 1
        q ← q - 1
      endif
   until p > q
```

Cara pemilihan *pivot*:

1. *Pivot* = elemen pertama/elemen terakhir/elemen tengah tabel

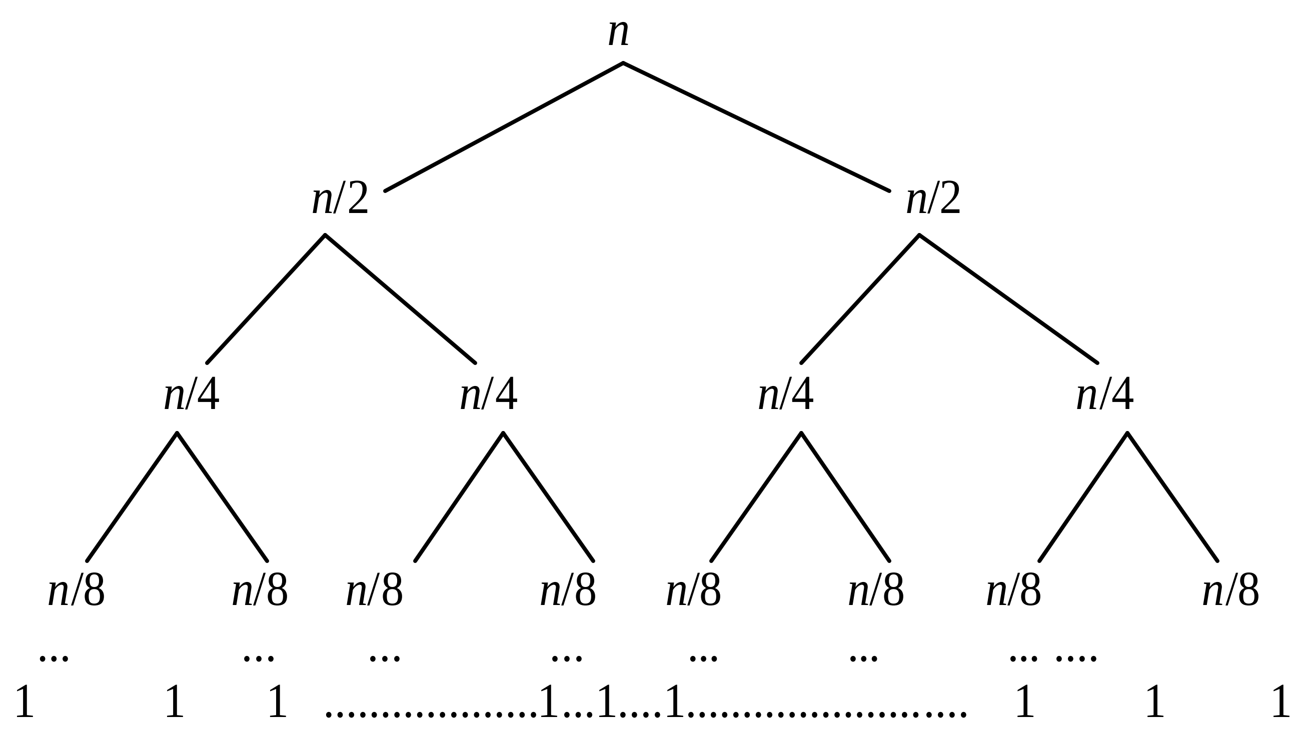
2. *Pivot* dipilih secara acak dari salah satu elemen tabel.

3. *Pivot* = elemen median tabel

Kompleksitas Algoritma Quicksort:

1. *Kasus terbaik (best case)*

- Kasus terbaik terjadi bila *pivot* adalah elemen median sedemikian sehingga kedua upatabel berukuran relatif sama setiap kali pempartisian.


n
n/2
n/2
n/4
n/4
n/4
n/4
n/8
n/8
n/8
n/8
n/8
n/8
n/8
n/8
1 1 1 ....1...1....1......................... 1 1 1

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

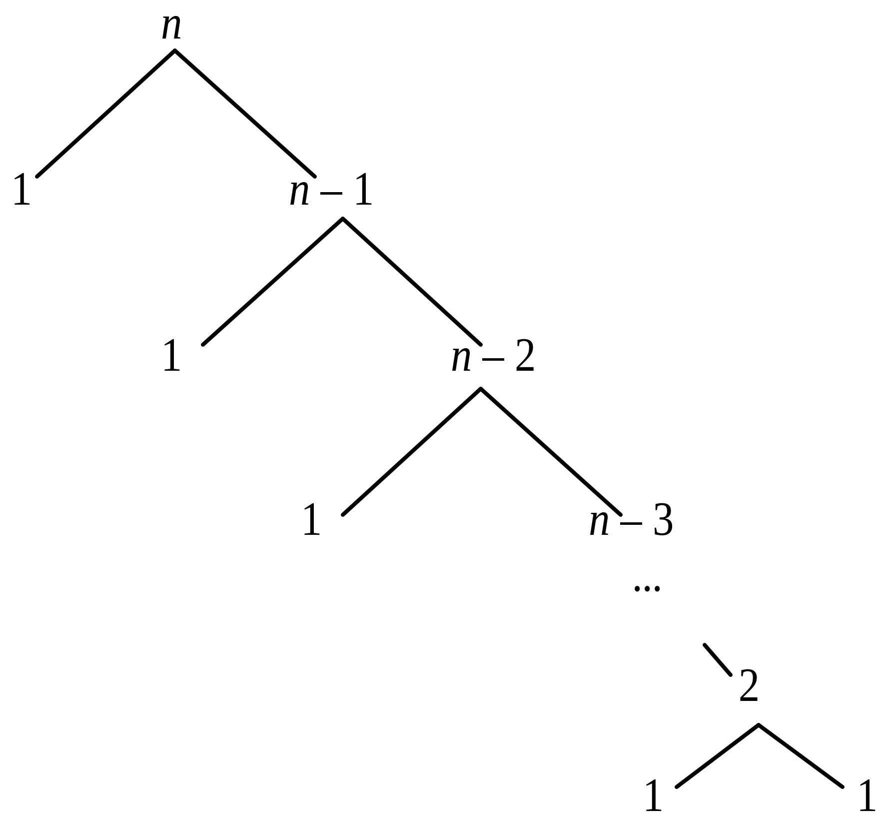
Penyelesaian (seperti pada *Merge Sort*):

$$T(n) = 2T(n/2) + cn = na + \mathrm{cn}\ {}^{2}\log n = O(n\ {}^{2}\log n).$$

2. *Kasus terburuk (worst case)*

- Kasus ini terjadi bila pada setiap partisi *pivot* selalu elemen maksimum (atau elemen minimum) tabel.

- Kasus jika tabel sudah terurut menaik/menurun


n
1
$n-1$
1
$n-2$
1
$n-3$
...
2
1
1

Kompleksitas waktu pengurutan:

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

Penyelesaian (seperti pada *Insertion Sort*):

$$T(n) = T(n - 1) + cn = O(n^2).$$

3. *Kasus rata-rata (average case)*

- Kasus ini terjadi jika *pivot* dipilih secara acak dari elemen tabel, dan peluang setiap elemen dipilih menjadi *pivot* adalah sama.

- $T_{avg}(n) = O(n^2 \log n)$.

(d) *Selection Sort*

```
procedure SelectionSort(input/output A : TabelInt, input i,j: integer)

{ Mengurutkan tabel A[i..j] dengan algoritma Selection Sort.
  Masukan: Tabel A[i..j] yang sudah terdefinisi elemen-elemennya.
  Keluaran: Tabel A[i..j] yang terurut menaik.
}

Algoritma:
   if  i < j then          { Ukuran(A) > 1 }
     Bagi(A, i, j)
     SelectionSort(A, i+1, j)
   endif
```

```
procedure Bagi(input/output A : TabInt, input i,j: integer)

{ Mencari elemen terkecil di dalam tabel A[i..j], dan menempatkan
elemen terkecil sebagai elemen pertama tabel.

  Masukan: A[i..j]
  Keluaran: A[i..j] dengan Ai adalah elemen terkecil.
}
Deklarasi
  idxmin, k, temp : integer

Algoritma:
  idxmin←i
  for k←i+1 to jdo
    if Ak < Aidxmin then
      idxmin←k
    endif
  endfor

 { pertukarkan Ai dengan Aidxmin }
 temp←Ai
 Ai←Aidxmin
 Aidxmin←temp
```

Contoh 4.5. Misalkan tabel *A* berisi elemen-elemen berikut:

4	12	3	9	1	21	5	2

Langkah-langkah pengurutan dengan *Selection Sort*:

4	12	3	9	1	21	5	2
1	12	3	9	4	21	5	2
1	2	3	9	4	21	5	12
1	2	3	9	4	21	5	12
1	2	3	4	9	21	5	12
1	2	3	4	5	21	9	12
1	2	3	4	5	9	12	21
1	2	3	4	5	9	12	21
1	2	3	4	5	9	12	21

Kompleksitas waktu algoritma:

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

Penyelesaian (seperti pada *Insertion Sort*):

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

4. Perpangkatan a^n

Misalkan $a \in R$ dan n adalah bilangan bulat tidak negatif:

$$a^n = a \times a \times \ldots \times a \quad (n \text{ kali}), \text{ jika } n > 0$$

$$= 1 \qquad , \text{ jika } n = 0$$

Penyelesaian dengan Algoritma Brute Force

```
function Exp1(input a, n : integer)→integer
{ Menghitung a^n, a > 0 dan n bilangan bulat tak-negatif
  Masukan: a, n
  Keluaran: nilai perpangkatan.
}
Deklarasi
  k, hasil : integer

Algoritma:
  hasil←1
  for k←1 to  n do
     hasil←hasil * a
  endfor

  return hasil
```

Kompleksitas waktu algoritma:

$$T(n) = n = O(n)$$

Penyelesaian dengan Divide and Conquer

Algoritma menghitung a^n:

1. Untuk kasus $n = 0$, maka $a^n = 1$.

2. Untuk kasus $n > 0$, bedakan menjadi dua kasus lagi:

   (i) jika n genap, maka $a^n = a^{n/2} \cdot a^{n/2}$

   (ii) jika n ganjil, maka $a^n = a^{n/2} \cdot a^{n/2} \cdot a$

Contoh 4.6. Menghitung 3^{16} dengan metode *Divide and Conquer*:

$$
\begin{aligned}
3^{16} = 3^8 \cdot 3^8 = (3^8)^2 \\
&= ((3^4)^2)^2 \\
&= (((3^2)^2)^2)^2 \\
&= ((((3^1)^2))^2)^2)^2 \\
&= ((((3^0)^2 \cdot 3)^2)^2)^2)^2 \\
&= ((((1)^2 \cdot 3)^2)^2)^2)^2 \\
&= ((((3)^2))^2)^2)^2 \\
&= (((9)^2)^2)^2 \\
&= (81)\ ^2)^2 \\
&= (6561)^2 \\
&= 43046721
\end{aligned}
$$

```
function Exp2(input a :real, n : integer) → real
{ mengembalikan nilai a^n, dihitung dengan metode Divide and Conquer }

Algoritma:
   if n = 0 then
      return 1
   else
      x←Exp2(a, n div 2)
      if odd(n) then         { fungsi odd memberikan true jika n ganjil }
       return x * x * a
      else
       return x * x
      endif
  endif
```

Kompleksitas algoritma:

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

Penyelesaian:

$$\begin{aligned} T(n) &= 1 + T(\lfloor n/2 \rfloor) \\ &= 1 + (1 + T(\lfloor n/4 \rfloor) = 2 + T(\lfloor n/4 \rfloor) \\ &= 2 + (1 + T(\lfloor n/8 \rfloor) = 3 + T(\lfloor n/8 \rfloor) \\ &= \ldots \\ &= k + T(\lfloor n/2^k \rfloor) \end{aligned}$$

Persamaan terakhir diselesaikan dengan membuat $n/2^k = 1$,

$$
\begin{aligned}
(n/2^k) = 1 \rightarrow \quad & \log (n/2^k) = \log 1 \\
& \log n - \log 2^k = 0 \\
& \log n - k \log 2 = 0 \\
& \log n = k \log 2 \\
& k = \log n / \log 2 = {}^2\log n
\end{aligned}
$$

sehingga

$$
\begin{aligned}
T(n) &= \lfloor {}^2\log n \rfloor + T(1) \\
&= \lfloor {}^2\log n \rfloor + 1 + T(0) \\
&= \lfloor {}^2\log n \rfloor + 1 + 0 \\
&= \lfloor {}^2\log n \rfloor + 1 \\
&= O({}^2\log n)
\end{aligned}
$$

5. Perkalian Matriks

- Misalkan A dan B dua buah matrik berukuran $n \times n$.

- Perkalian matriks: $C = A \times B$

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 dengan Algoritma Brute Force

```
function KaliMatriks1(input A,B: Matriks, input n : integer)→ Matriks
{ Memberikan hasil kali matriks A dan B yang berukuran n × n.
  Masukan: matriks integer A dan B, ukuran matriks (n)
  Keluaran: matriks C = A ´ B.
}

Deklarasi
  i, j, k : integer
  C : Matriks

Algoritma:
   for i←1 to n do
     for j←1 to n do
        Ci,j←0      { inisialisasi penjumlah }
        for k ← 1 to n do
           Ci,j ← Ci,j + Ai,k * Bk,j
        endfor
     endfor
   endfor

   return C
```

Kompleksitas algoritma: $T(n) = n^3 + n^2(n - 1) = O(n^3)$.

Penyelesaian dengan Algoritma Divide and Conquer

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

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

$$\boldsymbol{A} \qquad\qquad\qquad \boldsymbol{B} \qquad\qquad\qquad \boldsymbol{C}$$

Elemen-elemen matriks C adalah:

$$C11 = A11 \cdot B11 + A12 \cdot B21$$
$$C12 = A11 \cdot B12 + A12 \cdot B22$$
$$C21 = A21 \cdot B11 + A22 \cdot B21$$
$$C22 = A21 \cdot B12 + A22 \cdot B22$$

Contoh 4.7. Misalkan matriks A adalah sebagai berikut:

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

Matriks A dibagi menjadi 4 upa-matriks 2 x 2:

$$A11 = \begin{bmatrix} 3 & 4 \\ 21 & 5 \end{bmatrix} \quad A12 = \begin{bmatrix} 8 & 16 \\ 12 & 10 \end{bmatrix} \quad A21 = \begin{bmatrix} 5 & 1 \\ 45 & 9 \end{bmatrix} \quad A22 = \begin{bmatrix} 2 & 3 \\ 0 & -1 \end{bmatrix}$$

```
function KaliMatriks2(input A,B: Matriks, input n : integer) → Matriks
{ Memberikan hasil kali matriks A dan B yang berukuran n × n.
  Masukan: matriks integer A dan B, ukuran matriks (n)
  Keluaran: matriks C = A ´ B.
}
Deklarasi
  i, j, k : integer
  A11, A12, A21, A22,
  B11, B12, B21, B22,
  C11, C12, C21, C22 : Matriks

Algoritma:
   if n = 1 then
      return A × B   { perkalian biasa }
   else
      Bagi A menjadi A11, A12, A21, dan A22 yang masing-masing
      berukuran n/2 × n/2
      Bagi B menjadi B11, B12, B21, dan B22 yang masing-masing
      berukuran n/2 × 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, C13, C14  }
   endif
```

Pseudo-code algoritma penjumlahan (+), $C = A + B$:

```
function Tambah(input A, B : Matriks, input n : integer) → Matriks
{ Memberikan hasil penjumlahkan dua buah matriks, A dan B, yang
  berukuran n × n.
  Masukan: matriks integer A dan B, ukuran matriks (n)
  Keluaran: matriks C = A + B
}
Deklarasi
  i, j, k : integer

Algoritma:
   for i←1 to n do
     for j←1 to n do
        Ci,j ← Ai,j + Bi,j
     endfor
   endfor
   return C
```

Kompleksitas waktu perkalian matriks seluruhnya adalah:

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

yang bila diselesaikan, hasilnya adalah:

$$T(n) = O(n^3)$$

Hasil ini tidak memberi perbaikan kompleksitas dibandingkan dengan algoritma *brute force*.

Dapatkah kita membuat algoritma perkalian matriks yang lebih baik?

Algoritma Perkalian Matriks Strassen

Hitung matriks antara:

$$M1 = (A12 - A22)(B21 + B22)$$

$$M2 = (\mathrm{A}11 + A22)(B11 + B22)$$

$$M3 = (A11 - A21)(B11 + B12)$$

$$M4 = (A11 + A12)B22$$

$$M5 = A11\,(B12 - B22)$$

$$M6 = A22\,(B21 - B11)$$

$$M7 = (A21 + A22)B11$$

maka,

$$C11 = M1 + M2 - M4 + M6$$

$$C12 = M4 + M5$$

$$C21 = M6 + M7$$

$$C22 = M2 - M3 + M5 - M7$$

Kompleksitas waktu algoritma perkalian matriks Strassen:

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

yang bila diselesaikan, hasilnya adalah

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

6. Perkalian Dua Buah Bilangan Bulat yang Besar

Persoalan: Misalkan bilangan bulat *X* dan *Y* yang panjangnya *n* angka

$$X = x_1x_2x_3 \ldots x_n$$
$$Y = y_1y_2y3\ldots y_n$$

Hitunglah hasil kali *X* dengan *Y*.

Contoh 4.8. Misalkan,

$$X = 1234 \quad (n = 4)$$
$$Y = 5678 \quad (n = 4)$$

Cara klasik mengalikan X dan Y:

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

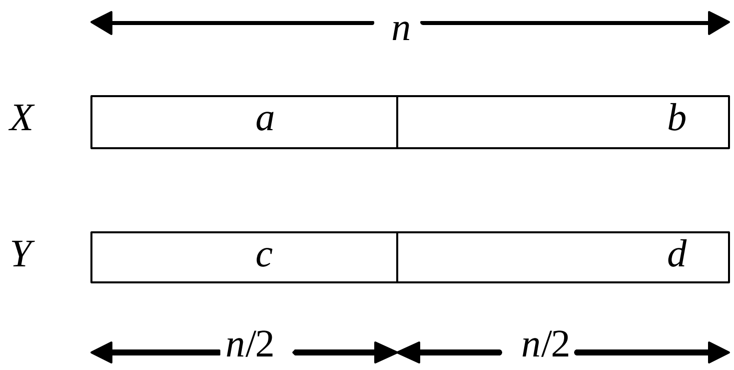
Pseudo-code algoritma perkalian matriks:

```
function Kali1(input X, Y : LongInteger, n : integer) → LongInteger
{ Mengalikan X dan Y, masing-masing panjangnya n digit dengan algoritma
brute force.
  Masukan: X dan Y yang panjangnya n angka
  Keluaran: hasil perkalian
}
Deklarasi
   temp, AngkaSatuan, AngkaPuluhan : integer

Algoritma:
   for setiap angka yi dari yn, yn-1, …, y1 do
      AngkaPuluhan ← 0
      for setiap angka xj dari xn, xn-1, …, x1 do
       temp ← xj * yi
       temp ← temp + AngkaPuluhan
       AngkaSatuan ← temp mod 10
       AngkaPuluhan ← temp div 10
       tuliskan AngkaSatuan
      endfor
   endfor
   Z ← Jumlahkan semua hasil perkalian dari atas ke bawah
   return Z
```

Kompleksitas algoritma: $O(n^2)$.

Penyelesaian dengan Algoritma Divide and Conquer



$s = n \ \underline{\text{div}}\ 2$

$a = X \ \underline{\text{div}}\ 10^s$
$b = X \ \underline{\text{mod}}\ 10^s$
$c = Y \ \underline{\text{div}}\ 10^s$
$d = Y \ \underline{\text{mod}}\ 10^s$

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

$X = a \cdot 10^s + b$
$Y = c \cdot 10^s + d$

Contoh,

$$X = 346769 = 346 \cdot 10^3 + 769$$
$$Y = 279431 = 279 \cdot 10^3 + 431$$

Perkalian X dengan Y dinyatakan sebagai

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

Pseudo-code perkalian *X* dan *Y*:

```
function Kali2(input X, Y : LongInteger, n : integer) → LongInteger
{ Mengalikan X dan Y, masing-masing panjangnya n digit dengan algoritma
Divide and Conquer.
  Masukan: X dan Y
  Keluaran: hasil perkalian X dan Y
}
Deklarasi
   a, b, c, d : LongInteger
   s : integer

Algoritma:
   if n = 1 then
      return X * Y    { perkalian 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
      return Kali2(a, c, s)*10^2s + Kali2(b, c, s)*10^s +
             Kali2(a, d, s)*10^s + Kali2(b, d, s)
  endif
```

Kompleksitas waktu algoritma:

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

- Penyelesaian:

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

- Ternyata, perkalian dengan algoritma *Divide and Conquer* seperti di atas belum memperbaiki kompleksitas waktu algoritma perkalian secara *brute force*.

- Adakah algoritma perkalian yang lebih baik?

Perbaikan (A.A Karatsuba, 1962):

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

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

$$= \underbrace{ac}_{p} \cdot 10^{2s} + \{\underbrace{(a+b)(c+d)}_{r} - \underbrace{ac}_{p} - \underbrace{bd}_{q}\} \cdot 10^{s} + \underbrace{bd}_{q}$$

```
function Kali3(input X, Y : LongInteger, n : integer) → LongInteger
{ Mengalikan X dan Y, masing-masing panjangnya n digit dengan algoritma
Divide and Conquer.
  Masukan: X dan Y
  Keluaran: hasil perkalian X dan Y
}
Deklarasi
   a, b, c, d : LongInteger
   s : integer

Algoritma:
   if n = 1 then
      return X * Y    { perkalian 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 waktu algoritmanya:

$T(n)$ = waktu perkalian integer yang berukuran n/2 + waktu untuk perkalian dengan 10^s dan 10^{2s} dan waktu untuk penjumlahan

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

Bila relasi rekurens diselesaikan, diperoleh $T(n) = O(n^{\log 3}) = O(n^{1.59})$, lebih baik daripada kompleksitas waktu dua algoritma perkalian sebelumnya.

Masalah Lain

(yang dipecahkan dengan *D & C*)

The Polynomial Multiplication Problem

another divide-and-conquer algorithm

Problem:
Given two polynomials of degree n

$$\begin{aligned} A(x) &= a_0 + a_1 x + \cdots + a_n x^n \\ B(x) &= b_0 + b_1 x + \cdots + b_n x^n, \end{aligned}$$

compute the product $A(x)B(x)$.

Example:

$$\begin{aligned} A(x) &= 1 + 2x + 3x^2 \\ B(x) &= 3 + 2x + 2x^2 \\ A(x)B(x) &= 3 + 8x + 15x^2 + 10x^3 + 6x^4 \end{aligned}$$

Question: How can we efficiently calculate the coefficients of $A(x)B(x)$?
Assume that the coefficients a_i and b_i are stored in arrays $A[0 \ldots n]$ and $B[0 \ldots n]$.
Cost of any algorithm is number of scalar multiplications and additions performed.

Convolutions

Let $A(x) = \sum_{i=0}^{n} a_i x^i$ and $B(x) = \sum_{i=0}^{m} b_i x^i$.

Set $C(x) = \sum_{k=0}^{n+m} c_i x^i = A(x)B(x)$.

Then

$$c_k = \sum_{i=0}^{k} a_i b_{k-i}$$

for all $0 \leq k \leq m+n$.

Definition: The vector $(c_0, c_1, \ldots, c_{m+n})$ is the convolution of the vectors $(a_0, a_1, \ldots, a_n)$ and $(b_0, b_1, \ldots, b_m)$.

Calculating convolutions (and thus polynomial multiplication) is a major problem in digital signal processing.

The Direct (Brute Force) Approach

Let $A(x) = \sum_{i=0}^{n} a_i x^i$ and $B(x) = \sum_{i=0}^{n} b_i x^i$.

Set $C(x) = \sum_{k=0}^{2n} c_i x^i = A(x)B(x)$ with

$$c_k = \sum_{i=0}^{k} a_i b_{k-i}$$

for all $0 \le k \le 2n$.

The direct approach is to compute all c_k using the formula above. The total number of multiplications and additions needed are $\Theta(n^2)$ and $\Theta(n^2)$ respectively. Hence the complexity is $\Theta(n^2)$.

Questions: Can we do better?
Can we apply the divide-and-conquer approach to develop an algorithm?

The Divide-and-Conquer Approach

The Divide Step: Define

$$
\begin{aligned}
A_0(x) &= a_0 + a_1 x + \cdots + a_{\lfloor \frac{n}{2} \rfloor - 1} x^{\lfloor \frac{n}{2} \rfloor - 1}, \\
A_1(x) &= a_{\lfloor \frac{n}{2} \rfloor} + a_{\lfloor \frac{n}{2} \rfloor + 1} x + \cdots + a_n x^{n - \lfloor \frac{n}{2} \rfloor}.
\end{aligned}
$$

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

Similarly we define $B_0(x)$ and $B_1(x)$ such that

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

Then

$$
\begin{aligned}
A(x)B(x) &= A_0(x)B_0(x) + A_0(x)B_1(x)x^{\lfloor \frac{n}{2} \rfloor} + \\
&\quad A_1(x)B_0(x)x^{\lfloor \frac{n}{2} \rfloor} + A_1(x)B_1(x)x^{2\lfloor \frac{n}{2} \rfloor}.
\end{aligned}
$$

Remark: The original problem of size n is divided into 4 problems of input size $\frac{n}{2}$.

Example:

$$\begin{aligned} A(x) &= 2+5x+3x^2+x^3-x^4 \\ B(x) &= 1+2x+2x^2+3x^3+6x^4 \\ A(x)B(x) &= 2+9x+17x^2+23x^3+34x^4+39x^5 \\ &\quad +19x^6+3x^7-6x^8 \end{aligned}$$

$A_0(x) = 2+5x,\quad A_1(x) = 3+x-x^2,$

$A(x) = A_0(x) + A_1(x)x^2$

$B_0(x) = 1+2x,\quad B_1(x) = 2+3x+6x^2,$

$B(x) = B_0(x) + B_1(x)x^2$

$$\begin{aligned} A_0(x)B_0(x) &= 2+9x+10x^2 \\ A_1(x)B_1(x) &= 6+11x+19x^2+3x^3-6x^4 \\ A_0(x)B_1(x) &= 4+16x+27x^2+30x^3 \\ A_1(x)B_0(x) &= 3+7x+x^2-2x^3 \\ A_0(x)B_1(x)+A_1(x)B_0(x) &= 7+23x+28x^2+28x^3 \end{aligned}$$

$$A_0(x)B_0(x) + (A_0(x)B_1(x) + A_1(x)B_0(x))x^2 + A_1(x)B_1(x)x^4$$
$$= 2+9x+17x^2+23x^3+34x^4+39x^5+19x^6+3x^7-6x^8$$

The Divide-and-Conquer Approach

The Conquer Step: Solve the four subproblems, i.e., computing

$$A_0(x)B_0(x), \quad A_0(x)B_1(x),$$
$$A_1(x)B_0(x), \quad A_1(x)B_1(x)$$

by recursively calling the algorithm 4 times.

The Divide-and-Conquer Approach

The Combining Step: Adding the following four polynomials

$$
\begin{aligned}
&A_0(x)B_0(x)\\
&\quad + A_0(x)B_1(x)x^{\lfloor \frac{n}{2} \rfloor}\\
&\quad + A_1(x)B_0(x)x^{\lfloor \frac{n}{2} \rfloor}\\
&\quad + A_1(x)B_1(x)x^{2\lfloor \frac{n}{2} \rfloor}.
\end{aligned}
$$

takes $\Theta(n)$ operations. Why?

The First Divide-and-Conquer Algorithm

PolyMulti1$(A(x), B(x))$

{

$$A_0(x) = a_0 + a_1 x + \cdots + a_{\lfloor \frac{n}{2} \rfloor - 1} x^{\lfloor \frac{n}{2} \rfloor - 1};$$

$$A_1(x) = a_{\lfloor \frac{n}{2} \rfloor} + a_{\lfloor \frac{n}{2} \rfloor + 1} x + \cdots + a_n x^{n - \lfloor \frac{n}{2} \rfloor};$$

$$B_0(x) = b_0 + b_1 x + \cdots + b_{\lfloor \frac{n}{2} \rfloor - 1} x^{\lfloor \frac{n}{2} \rfloor - 1};$$

$$B_1(x) = b_{\lfloor \frac{n}{2} \rfloor} + b_{\lfloor \frac{n}{2} \rfloor + 1} x + \cdots + b_n x^{n - \lfloor \frac{n}{2} \rfloor};$$

$U(x) = PolyMulti1(A_0(x), B_0(x));$
$V(x) = PolyMulti1(A_0(x), B_1(x));$
$W(x) = PolyMulti1(A_1(x), B_0(x));$
$Z(x) = PolyMulti1(A_1(x), B_1(x));$

$$\textbf{return} \left(U(x) + [V(x) + W(x)] x^{\lfloor \frac{n}{2} \rfloor} + Z(x) x^{2\lfloor \frac{n}{2} \rfloor} \right)$$

}

Running Time of the Algorithm

Assume n is a power of 2, $n = 2^h$. By substitution (expansion),

$$
\begin{aligned}
T(n) &= 4T\left(\frac{n}{2}\right) + cn \\
&= 4\left[4T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right] + cn \\
&= 4^2 T\left(\frac{n}{2^2}\right) + (1+2)cn \\
&= 4^2\left[4T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right] + (1+2)cn \\
&= 4^3 T\left(\frac{n}{2^3}\right) + (1+2+2^2)cn \\
&\vdots \\
&= 4^i T\left(\frac{n}{2^i}\right) + \sum_{j=0}^{i-1} 2^j cn \quad \text{(induction)} \\
&\vdots \\
&= 4^h T\left(\frac{n}{2^h}\right) + \sum_{j=0}^{h-1} 2^j cn \\
&= n^2 T(1) + cn(n-1) \\
&\quad \left(\text{since } n = 2^h \text{ and } \sum_{j=0}^{h-1} 2^j = 2^h - 1 = n - 1\right) \\
&= \Theta(n^2).
\end{aligned}
$$

The same order as the brute force approach!

Comments on the Divide-and-Conquer Algorithm

Comments: The divide-and-conquer approach makes no essential improvement over the brute force approach!

Question: Why does this happen.

Question: Can you improve this divide-and-conquer algorithm?

Problem: Given 4 numbers

$$A_0, A_1, B_0, B_1$$

how many multiplications are needed to calculate the three values

$$A_0B_0, A_0B_1 + A_1B_0, A_1B_1?$$

This can obviously be done using 4 multiplications but there is a way of doing this using only the following 3:

$$\begin{aligned} Y &= (A_0 + A_1)(B_0 + B_1) \\ U &= A_0B_0 \\ Z &= A_1B_1 \end{aligned}$$

U and Z are what we originally wanted and

$$A_0B_1 + A_1B_0 = Y - U - Z.$$

Improving the Divide-and-Conquer Algorithm

Define

$$\begin{aligned} Y(x) &= (A_0(x) + A_1(x)) \times (B_0(x) + B_1(x)) \\ U(x) &= A_0(x)B_0(x) \\ Z(x) &= A_1(x)B_1(x) \end{aligned}$$

Then

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

Hence $A(x)B(x)$ is equal to

$$U(x) + [Y(x) - U(x) - Z(x)]x^{\lfloor \frac{n}{2} \rfloor} + Z(x) \times x^{2\lfloor \frac{n}{2} \rfloor}$$

Conclusion: You need to call the multiplication procedure 3, rather than 4 times.

The Second Divide-and-Conquer Algorithm

PolyMulti2$(A(x), B(x))$
{

$A_0(x) = a_0 + a_1 x + \cdots + a_{\lfloor \frac{n}{2} \rfloor - 1} x^{\lfloor \frac{n}{2} \rfloor - 1}$;
$A_1(x) = a_{\lfloor \frac{n}{2} \rfloor} + a_{\lfloor \frac{n}{2} \rfloor + 1} x + \cdots + a_n x^{n - \lfloor \frac{n}{2} \rfloor}$;

$B_0(x) = b_0 + b_1 x + \cdots + b_{\lfloor \frac{n}{2} \rfloor - 1} x^{\lfloor \frac{n}{2} \rfloor - 1}$;
$B_1(x) = b_{\lfloor \frac{n}{2} \rfloor} + b_{\lfloor \frac{n}{2} \rfloor + 1} x + \cdots + b_n x^{n - \lfloor \frac{n}{2} \rfloor}$;

$Y(x) = PolyMulti2(A_0(x) + A_1(x), B_0(x) + B_1(x))$
$U(x) = PolyMulti2(A_0(x), B_0(x))$;
$Z(x) = PolyMulti2(A_1(x), B_1(x))$;

return $\left(U(x) + [Y(x) - U(x) - Z(x)] x^{\lfloor \frac{n}{2} \rfloor} + Z(x) x^{2 \lfloor \frac{n}{2} \rfloor} \right)$;
}

Running Time of the Modified Algorithm

Assume $n = 2^h$. Let $\lg x$ denote $\log_2 x$.
By the substitution method,

$$\begin{aligned} T(n) &= 3T\left(\frac{n}{2}\right) + cn \\ &= 3\left[3T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right] + cn \\ &= 3^2T\left(\frac{n}{2^2}\right) + \left(1 + \frac{3}{2}\right)cn \\ &= 3^2\left[3T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right] + \left(1 + \frac{3}{2}\right)cn \\ &= 3^3T\left(\frac{n}{2^3}\right) + \left(1 + \frac{3}{2} + \left[\frac{3}{2}\right]^2\right)cn \\ &\;\;\vdots \\ &= 3^hT\left(\frac{n}{2^h}\right) + \sum_{j=0}^{h-1}\left[\frac{3}{2}\right]^j cn. \end{aligned}$$

We have

$$3^h = (2^{\lg 3})^h = 2^{h \lg 3} = (2^h)^{\lg 3} = n^{\lg 3} \approx n^{1.585},$$

and

$$\sum_{j=0}^{h-1}\left[\frac{3}{2}\right]^j = \frac{(3/2)^h - 1}{3/2 - 1} = 2 \cdot \frac{3^h}{2^h} - 2 = 2n^{\lg 3 - 1} - 2.$$

Hence

$$T(n) = \Theta(n^{\lg 3}T(1) + 2cn^{\lg 3}) = \Theta(n^{\lg 3}).$$

Comments

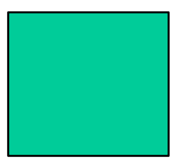
- The divide-and-conquer approach doesn't always give you the best solution.
  Our original D-A-C algorithm was just as bad as brute force.

- There is actually an $O(n \log n)$ solution to the polynomial multiplication problem.
  It involves using the *Fast Fourier Transform* algorithm as a subroutine.
  The FFT is another classic D-A-C algorithm (Chapt 30 in CLRS gives details).

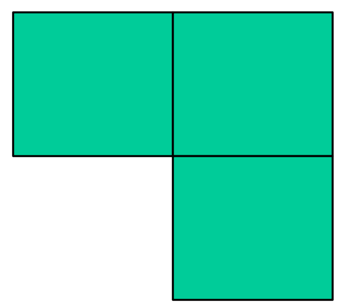
- The idea of using 3 multiplications instead of 4 is used in large-integer multiplications.
  A similar idea is the basis of the classic Strassen matrix multiplication algorithm (CLRS, Chapter 28).

Masalah Pengubinan

Masalah: Diberikan sebuah papan yang berukuran $2^k \times 2^k$. Tersedia sebuah ubin dan $2^{2k} - 1$ buah ubin yang terdiri dari kelompok 3-ubin berbentuk huruf L. Pasanglah semua ubin pada papan tersebut.



Ubin tunggal



Ubin berbentuk L (3-ubin)

Algoritma *D & C*:

- Bagi papan menjadi 4 bagian
- Tempatkan kelompok 3-ubin berbentuk L pada bagian tengah yang tidak ada ubin tinggal
- Ubin tunggal dapat ditaruh di mana saja.