

LAPORAN PRAKTIKUM 5
Analisis algoritma



Sarah Navianti Dwi Sutisna
140810180021
Kelas A

PROGRAM STUDI S1 TEKNIK INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN
2020

Studi Kasus (Lanjutan)

Studi Kasus 5: Mencari Pasangan Titik Terdekat (Closest Pair of Points)

Identifikasi Problem:

Diberikan array n poin dalam bidang kartesius, dan problemnya adalah mencari tahu pasangan poin terdekat dalam bidang tersebut dengan merepresentasikannya ke dalam array. Masalah ini muncul di sejumlah aplikasi. Misalnya, dalam kontrol lalu lintas udara, kita mungkin ingin memantau pesawat yang terlalu berdekatan karena ini mungkin menunjukkan kemungkinan tabrakan. Ingat rumus berikut untuk jarak antara dua titik p dan q .

$$\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

Solusi

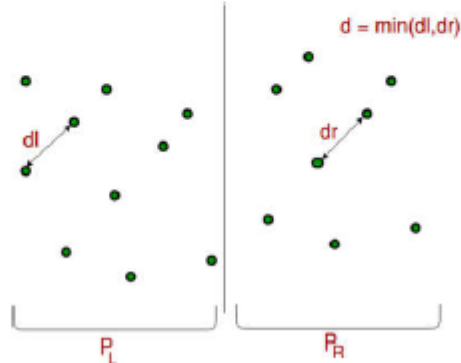
Solusi umum dari permasalahan tersebut adalah menggunakan algoritma **Brute force** dengan $O(n^2)$ hitung jarak antara setiap pasangan dan kembalikan yang terkecil. Namun, kita dapat menghitung jarak terkecil dalam waktu **$O(n \log n)$** menggunakan strategi **Divide and Conquer**. Ikuti algoritma berikut:

Input: An array of n points $P[]$

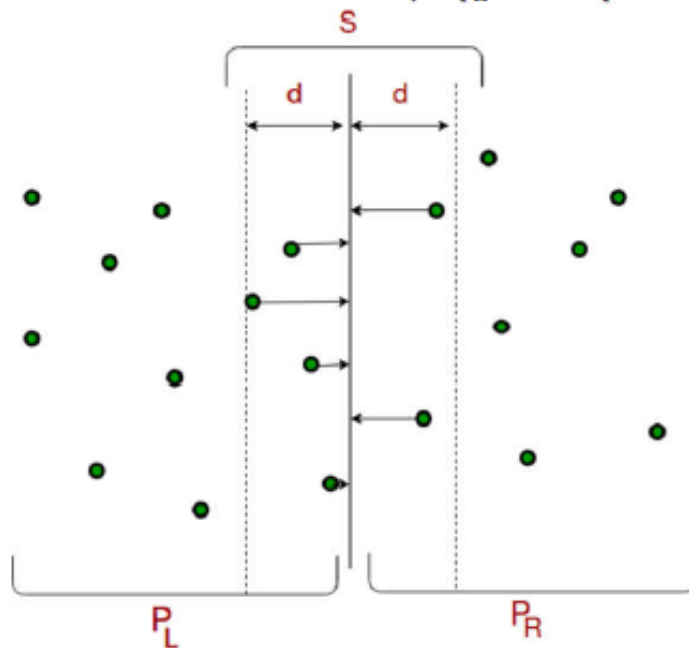
Output: The smallest distance between two points in the given array.

As a pre-processing step, input array is sorted according to x coordinates.

- 1) Find the middle point in the sorted array, we can take $P[n/2]$ as middle point.
- 2) Divide the given array in two halves. The first subarray contains points from $P[0]$ to $P[n/2]$. The second subarray contains points from $P[n/2+1]$ to $P[n-1]$.
- 3) Recursively find the smallest distances in both subarrays. Let the distances be d_l and d_r . Find the minimum of d_l and d_r . Let the minimum be d .



- 4) From above 3 steps, we have an upper bound d of minimum distance. Now we need to consider the pairs such that one point in pair is from left half and other is from right half. Consider the vertical line passing through $P[n/2]$ and find all points whose x coordinate is closer than d to the middle vertical line. Build an array `strip[]` of all such points.



- 5) Sort the array `strip[]` according to y coordinates. This step is $O(n \log n)$. It can be optimized to $O(n)$ by recursively sorting and merging.
- 6) Find the smallest distance in `strip[]`. This is tricky. From first look, it seems to be a $O(n^2)$ step, but it is actually $O(n)$. It can be proved geometrically that for every point in `strip`, we only need to check at most 7 points after it (note that `strip` is sorted according to Y coordinate). See [this](#) for more analysis.
- 7) Finally return the minimum of d and distance calculated in above step (step 6)

Tugas:

- 1) Buatlah program untuk menyelesaikan problem closest pair of points menggunakan algoritma divide & conquer yang diberikan. Gunakan bahasa C++

Jawab :

```
/*
    Nama      : Sarah Navianti
    NPM       : 140810180021
    Kelas     : A
    Program   : Closest Pair of Point
*/

// A divide and conquer program in C++
// to find the smallest distance from a
// given set of points.

#include <bits/stdc++.h>
using namespace std;

// A structure to represent a Point in 2D plane
class Point {
```

```

    public:
        int x, y;
};

/* Following two functions are needed for library function qsort().
Refer: http://www.cplusplus.com/reference/cstdlib/qsort/ */

// Needed to sort array of points
// according to X coordinate
int compareX(const void* a, const void* b){

    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->x - p2->x);
}

// Needed to sort array of points according to Y coordinate
int compareY(const void* a, const void* b){

    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->y - p2->y);
}

// A utility function to find the
// distance between two points
float dist(Point p1, Point p2){
    return sqrt( (p1.x - p2.x)*(p1.x - p2.x) +
                (p1.y - p2.y)*(p1.y - p2.y)
                );
}

// A Brute Force method to return the
// smallest distance between two points
// in P[] of size n
float bruteForce(Point P[], int n){

    float min = FLT_MAX;
    for (int i = 0; i < n; ++i)
        for (int j = i+1; j < n; ++j)
            if (dist(P[i], P[j]) < min)
                min = dist(P[i], P[j]);
    return min;
}

// A utility function to find
// minimum of two float values
float min(float x, float y){

    return (x < y)? x : y;
}

```

```

}

// A utility function to find the
// distance between the closest points of
// strip of given size. All points in
// strip[] are sorted according to
// y coordinate. They all have an upper
// bound on minimum distance as d.
// Note that this method seems to be
// a  $O(n^2)$  method, but it's a  $O(n)$ 
// method as the inner loop runs at most 6 times

float stripClosest(Point strip[], int size, float d) {

    float min = d; // Initialize the minimum distance as d

    qsort(strip, size, sizeof(Point), compareY);

    // Pick all points one by one and try the next points till the difference
    // between y coordinates is smaller than d.
    // This is a proven fact that this loop runs at most 6 times
    for (int i = 0; i < size; ++i)
        for (int j = i+1; j < size && (strip[j].y - strip[i].y) < min; ++j)
            if (dist(strip[i], strip[j]) < min)
                min = dist(strip[i], strip[j]);

    return min;
}

// A recursive function to find the
// smallest distance. The array P contains
// all points sorted according to x coordinate

float closestUtil(Point P[], int n){

    // If there are 2 or 3 points, then use brute force
    if (n <= 3)
        return bruteForce(P, n);

    // Find the middle point
    int mid = n/2;
    Point midPoint = P[mid];

    // Consider the vertical line passing
    // through the middle point calculate
    // the smallest distance dl on left
    // of middle point and dr on right side
    float dl = closestUtil(P, mid);

```

```

float dr = closestUtil(P + mid, n - mid);

// Find the smaller of two distances
float d = min(dl, dr);

// Build an array strip[] that contains
// points close (closer than d)
// to the line passing through the middle point
Point strip[n];
int j = 0;
for (int i = 0; i < n; i++)
    if (abs(P[i].x - midPoint.x) < d)
        strip[j] = P[i], j++;

// Find the closest points in strip.
// Return the minimum of d and closest
// distance is strip[]
return min(d, stripClosest(strip, j, d) );
}

// The main function that finds the smallest distance
// This method mainly uses closestUtil()
float closest(Point P[], int n){

    qsort(P, n, sizeof(Point), compareX);

    // Use recursive function closestUtil()
    // to find the smallest distance

    return closestUtil(P, n);
}

// Driver code
int main(){

    Point P[] = {{8, 4}, {15, 30}, {80, 90}, {10,10}, {16,28}} ;
    int n = sizeof(P) / sizeof(P[0]);
    cout << "====Closest Pair Of Point==== " <<endl;
    cout << "\nThe smallest distance is " << closest(P, n);
    return 0;
}

```

```
F:\semester4\Analgo\AnalgoKu\AnalgoKu-5\closest.exe
====Closest Pair Of Point====

The smallest distance is 2.23607
-----
Process exited after 0.3673 seconds with return value 0
Press any key to continue . . .
```

- 2) Tentukan rekurensi dari algoritma tersebut, dan selesaikan rekurensinya menggunakan metode recursion tree untuk membuktikan bahwa algoritma tersebut memiliki Big-O ($n \lg n$)

Jawab:

Kompleksitas Waktu

Biarkan kompleksitas waktu dari algoritma di atas menjadi $T(n)$. Mari kita asumsikan bahwa kita menggunakan algoritma pengurutan $O(n \lg n)$. Algoritma di atas membagi semua titik dalam dua set dan secara rekursif memanggil dua set. Setelah membelah, ia menemukan strip dalam waktu $O(n)$, mengurutkan strip dalam waktu $O(n \lg n)$ dan akhirnya menemukan titik terdekat dalam strip dalam waktu $O(n)$.

Jadi $T(n)$ dapat dinyatakan sebagai berikut :

$$T(n) = 2T(n/2) + O(n) + O(n \lg n) + O(n)$$

$$T(n) = 2T(n/2) + O(n \lg n)$$

$$T(n) = T(n \times \lg n \times \lg n)$$

Catatan

- 1) Kompleksitas waktu dapat ditingkatkan menjadi $O(n \lg n)$ dengan mengoptimalkan langkah 5 dari algoritma di atas.
- 2) Kode menemukan jarak terkecil. Dapat dengan mudah dimodifikasi untuk menemukan titik dengan jarak terkecil.
- 3) Kode ini menggunakan pengurutan cepat yang bisa $O(n^2)$ dalam kasus terburuk. Untuk memiliki batas atas sebagai $O(n (\lg n)^2)$, algoritma pengurutan $O(n \lg n)$ seperti pengurutan gabungan atau pengurutan tumpukan dapat digunakan

Studi Kasus 6: Algoritma Karatsuba untuk Perkalian

Cepat Identifikasi Problem:

Diberikan dua string biner yang mewakili nilai dua bilangan bulat, cari produk (hasil kali) dari dua string. Misalnya, jika string bit pertama adalah "1100" dan string bit kedua adalah "1010", output harus 120. Supaya lebih sederhana, panjang dua string sama dan menjadi n .

Solusi:

Salah satu solusinya adalah dengan naïve approach yang pernah kita pelajari di sekolah. Satu per satu ambil semua bit nomor kedua dan kalikan dengan semua bit nomor pertama. Akhirnya tambahkan semua perkalian. Algoritma ini membutuhkan waktu $O(n^2)$.

$$\begin{array}{r}
 x = 101001 = 41 \\
 y = 101010 = 42 \\
 \hline
 1010010 \\
 101001 \\
 + 101001 \\
 \hline
 11010111010 = 1722
 \end{array}$$

Algoritma Karatsuba

Solusi lain adalah dengan menggunakan Algoritma Karatsuba yang berparadigma Divide dan Conquer, kita dapat melipatgandakan dua bilangan bulat dalam kompleksitas waktu yang lebih sedikit. Kami membagi angka yang diberikan dalam dua bagian. Biarkan angka yang diberikan menjadi X dan Y.

- Untuk kesederhanaan, kita asumsikan bahwa n adalah genap:

$$\begin{array}{l}
 X = X_1 \cdot 2^{n/2} + X_r \quad [X_1 \text{ and } X_r \text{ contain leftmost and rightmost } n/2 \text{ bits of } X] \\
 Y = Y_1 \cdot 2^{n/2} + Y_r \quad [Y_1 \text{ and } Y_r \text{ contain leftmost and rightmost } n/2 \text{ bits of } Y]
 \end{array}$$

- Produk XY dapat ditulis sebagai berikut:

$$\begin{aligned}
 XY &= (X_1 \cdot 2^{n/2} + X_r) (Y_1 \cdot 2^{n/2} + Y_r) \\
 &= 2^n X_1 Y_1 + 2^{n/2} (X_1 Y_r + X_r Y_1) + X_r Y_r
 \end{aligned}$$

- Jika kita melihat rumus di atas, ada empat perkalian ukuran $n/2$, jadi pada dasarnya kita membagi masalah ukuran n menjadi empat sub-masalah ukuran $n/2$. Tetapi itu tidak membantu karena solusi pengulangan $T(n) = 4T(n/2) + O(n)$ adalah $O(n^4)$. Bagian rumit dari algoritma ini adalah mengubah dua istilah tengah ke bentuk lain sehingga hanya satu perkalian tambahan yang cukup. Berikut ini adalah *tricky expression* untuk dua middle terms tersebut.

$$X_1 Y_r + X_r Y_1 = (X_1 + X_r) (Y_1 + Y_r) - X_1 Y_1 - X_r Y_r$$

- Jadi nilai akhir XY menjadi:

$$XY = 2^n X_1 Y_1 + 2^{n/2} * [(X_1 + X_r) (Y_1 + Y_r) - X_1 Y_1 - X_r Y_r] + X_r Y_r$$

- Dengan trik di atas, perulangan menjadi $T(n) = 3T(n/2) + O(n)$ dan solusi dari perulangan ini adalah $O(n^{1.59})$

Bagaimana jika panjang string input berbeda dan tidak genap? Untuk menangani kasus panjang yang berbeda, kita menambahkan 0 di awal. Untuk menangani panjang ganjil, kita menempatkan bit floor ($n/2$) di setengah kiri dan ceil ($n/2$) bit di setengah kanan. Jadi ekspresi untuk XY berubah menjadi berikut.

$$XY = 2^{2\text{ceil}(n/2)} X_1 Y_1 + 2^{\text{ceil}(n/2)} * [(X_1 + X_r) (Y_1 + Y_r) - X_1 Y_1 - X_r Y_r] + X_r Y_r$$

Tugas:

- 1) Buatlah program untuk menyelesaikan problem *fast multiplication* menggunakan algoritma divide & conquer yang diberikan (Algoritma Karatsuba). Gunakan bahasa C++

Jawab :

```
/*
    Nama      : Sarah Navianti
    NPM       : 140810180021
    Kelas     : A
    Program   : Karatsuba Algorithm
*/

// C++ implementation of Karatsuba algorithm for bit string multiplication.
#include<iostream>
#include<stdio.h>

using namespace std;

// FOLLOWING TWO FUNCTIONS ARE COPIED FROM http://goo.gl/q00hZ
// Helper method: given two unequal sized bit strings, converts them to
// same length by adding leading 0s in the smaller string. Returns the
// the new length
int makeEqualLength(string &str1, string &str2){

    int len1 = str1.size();
    int len2 = str2.size();
    if (len1 < len2){

        for (int i = 0 ; i < len2 - len1 ; i++)
            str1 = '0' + str1;
        return len2;
    }
    else if (len1 > len2){

        for (int i = 0 ; i < len1 - len2 ; i++)
            str2 = '0' + str2;
    }
    return len1; // If len1 >= len2
}

// The main function that adds two bit sequences and returns the addition
string addBitStrings( string first, string second ){

    string result; // To store the sum bits

    // make the lengths same before adding
    int length = makeEqualLength(first, second);
    int carry = 0; // Initialize carry
```

```

// Add all bits one by one
for (int i = length-1 ; i >= 0 ; i--){

    int firstBit = first.at(i) - '0';
    int secondBit = second.at(i) - '0';

    // boolean expression for sum of 3 bits
    int sum = (firstBit ^ secondBit ^ carry)+'0';

    result = (char)sum + result;

    // boolean expression for 3-bit addition
    carry = (firstBit&secondBit) | (secondBit&carry) | (firstBit&carry);
}

// if overflow, then add a leading 1
if (carry) result = '1' + result;

return result;
}

// A utility function to multiply single bits of strings a and b

int multiplyiSingleBit(string a, string b) {
    return (a[0] - '0')*(b[0] - '0');
}

// The main function that multiplies two bit strings X and Y and returns
// result as long integer

long int multiply(string X, string Y){

    // Find the maximum of lengths of x and Y and make length
    // of smaller string same as that of larger string
    int n = makeEqualLength(X, Y);

    // Base cases
    if (n == 0) return 0;
    if (n == 1) return multiplyiSingleBit(X, Y);

    int fh = n/2; // First half of string, floor(n/2)
    int sh = (n-fh); // Second half of string, ceil(n/2)

    // Find the first half and second half of first string.
    // Refer http://goo.gl/1Lmgn for substr method
    string Xl = X.substr(0, fh);
    string Xr = X.substr(fh, sh);

    // Find the first half and second half of second string
    string Yl = Y.substr(0, fh);

```

```

string Yr = Y.substr(fh, sh);

// Recursively calculate the three products of inputs of size n/2
long int P1 = multiply(Xl, Yl);
long int P2 = multiply(Xr, Yr);
long int P3 = multiply(addBitStrings(Xl, Xr), addBitStrings(Yl, Yr));

// Combine the three products to get the final result.
return P1*(1<<(2*sh)) + (P3 - P1 - P2)*(1<<sh) + P2;
}

// Driver program to test above functions
int main(){

    printf ("%ld\n", multiply("1100", "1010"));
    printf ("%ld\n", multiply("1001", "0110"));
    printf ("%ld\n", multiply("1100", "0011"));
    printf ("%ld\n", multiply("0000", "1111"));
    printf ("%ld\n", multiply("0001", "1010"));
    printf ("%ld\n", multiply("0011", "0011"));
    printf ("%ld\n", multiply("1111", "1111"));
}

```

F:\semester4\Analgo\AnalgoKu\AnalgoKu-5\karatsuba.exe

```

120
54
36
0
10
9
225

-----
Process exited after 0.1425 seconds with return value 0
Press any key to continue . . .

```

- 2) Rekurensi dari algoritma tersebut adalah $T(n) = 3T(n/2) + O(n)$, dan selesaikan rekurensinya menggunakan metode substitusi untuk membuktikan bahwa algoritma tersebut memiliki Big-O ($n \lg n$)

JAWAB :

- Let's try divide and conquer.
 - Divide each number into two halves.
 - $x = x_H r^{n/2} + x_L$
 - $y = y_H r^{n/2} + y_L$
 - Then:

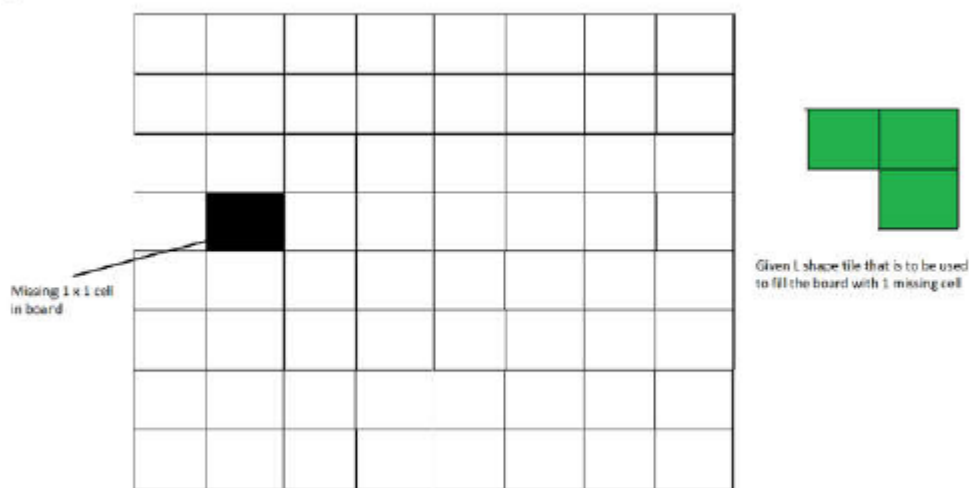
$$xy = (x_H r^{n/2} + x_L) y_H r^{n/2} + y_L$$

$$= x_H y_H r^n + (x_H y_L + x_L y_H) r^{n/2} + x_L y_L$$
 - Runtime?
 - $T(n) = 4 T(n/2) + O(n)$
 - $T(n) = O(n^2)$
- Instead of 4 subproblems, we only need 3 (with the help of clever insight).
- Three subproblems:
 - $a = x_H y_H$
 - $d = x_L y_L$
 - $e = (x_H + x_L) (y_H + y_L) - a - d$
- Then $xy = a r^n + e r^{n/2} + d$
- $T(n) = 3 T(n/2) + O(n)$
- $T(n) = O(n^{\log_2 3}) = O(n^{1.584...})$

Studi Kasus 7: Permasalahan Tata Letak Keramik Lantai (Tiling Problem)

Identifikasi Problem:

Diberikan papan berukuran $n \times n$ dimana n adalah dari bentuk $2k$ dimana $k \geq 1$ (Pada dasarnya n adalah pangkat dari 2 dengan nilai minimumnya 2). Papan memiliki satu sel yang hilang (ukuran 1×1). Isi papan menggunakan ubin berbentuk L. Ubin berbentuk L berukuran 2×2 persegi dengan satu sel berukuran 1×1 hilang.



Gambar 2. Ilustrasi tiling problem

Solusi:

Masalah ini dapat diselesaikan menggunakan Divide and Conquer. Di bawah ini adalah algoritma rekursifnya

```
// n is size of given square, p is location of missing cell
Tile(int n, Point p)

1) Base case: n = 2, A 2 x 2 square with one cell missing is nothing
   but a tile and can be filled with a single tile.

2) Place a L shaped tile at the center such that it does not cover
   the n/2 * n/2 subsquare that has a missing square. Now all four
   subsquares of size n/2 x n/2 have a missing cell (a cell that doesn't
   need to be filled). See figure 3 below.

3) Solve the problem recursively for following four. Let p1, p2, p3 and
   p4 be positions of the 4 missing cells in 4 squares.
   a) Tile(n/2, p1)
   b) Tile(n/2, p2)
   c) Tile(n/2, p3)
   d) Tile(n/2, p3)
```

(Gambar yang menunjukkan kerja algoritma di atas bisa dilihat di Modul Praktikum 5)

Tugas:

- 1) Buatlah program untuk menyelesaikan problem *tilling* menggunakan algoritma divide & conquer yang diberikan. Gunakan bahasa C++

JAWAB :

Program :

```
/*
  Nama      : Sarah Navianti
  NPM       : 140810180021
  Kelas     : A
  Program   : Tilling Problem
*/

#include <bits/stdc++.h>
using namespace std;

// function to count the total number of ways
int countWays(int n,int a)
{
    // table to store values
    // of subproblems
    int count[n-1];
    count[0] = 0;

    // Fill the table upto value n
    for (int i = 1; i <= n ;i++){
        // recurrence relation
```

```

        if (i > a)
            count[i] = count[i - 1] + count[i - a];

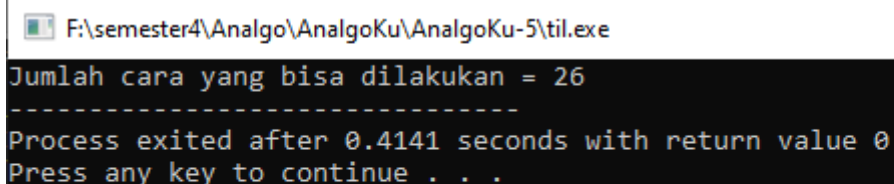
        // base cases
        else if (i < a )
            count[i] = 1;

        // i == a
        else
            count[i] = 2;
    }

    // required number of ways
    return count[n];
}

// Driver program to test above
int main()
{
    int n = 8 , a = 4;
    cout << "Jumlah cara yang bisa dilakukan = "
         << countWays(n,a);
    return 0;
}

```



```

F:\semester4\Analgo\AnalgoKu\AnalgoKu-5\tit.exe
Jumlah cara yang bisa dilakukan = 26
-----
Process exited after 0.4141 seconds with return value 0
Press any key to continue . . .

```

// n adalah ukuran kotak yang diberikan, p adalah lokasi sel yang hilang
Tile (int n, Point p)

- 1) Kasus dasar: $n = 2$, A 2×2 persegi dengan satu sel yang hilang tidak ada apa-apanya tapi ubin dan bisa diisi dengan satu ubin.
- 2) Tempatkan ubin berbentuk L di tengah sehingga tidak menutupi subsquare $n/2 * n/2$ yang memiliki kuadrat yang hilang. Sekarang keempatnya subskuen ukuran $n/2 \times n/2$ memiliki sel yang hilang (sel yang tidak perlu diisi). Lihat gambar 2 di bawah ini.
- 3) Memecahkan masalah secara rekursif untuk mengikuti empat. Biarkan p1, p2, p3 dan p4 menjadi posisi dari 4 sel yang hilang dalam 4 kotak.
 - a) Ubin ($n/2, p1$)

b) Ubin ($n / 2$, $p2$)

c) Ubin ($n / 2$, $p3$)

d) Ubin ($n / 2$, $p3$)

- 2) Relasi rekurensi untuk algoritma rekursif di atas dapat ditulis seperti di bawah ini. C adalah konstanta. $T(n) = 4T(n / 2) + C$. Selesaikan rekurensi tersebut dengan Metode Master!

JAWAB :

Kompleksitas Waktu:

Relasi perulangan untuk algoritma rekursif di atas dapat ditulis seperti di bawah ini. C adalah konstanta.

$$T(n) = 4T(n / 2) + C$$

Rekursi di atas dapat diselesaikan dengan menggunakan Metode Master dan kompleksitas waktu adalah $O(n^2)$

Bagaimana cara kerjanya?

Pengerjaan algoritma Divide and Conquer dapat dibuktikan menggunakan Mathematical Induction. Biarkan kuadrat input berukuran $2k \times 2k$ di mana $k \geq 1$.

Kasus Dasar: Kita tahu bahwa masalahnya dapat diselesaikan untuk $k = 1$. Kami memiliki 2×2 persegi dengan satu sel hilang.

Hipotesis Induksi: Biarkan masalah dapat diselesaikan untuk $k-1$.

Sekarang perlu dibuktikan untuk membuktikan bahwa masalah dapat diselesaikan untuk k jika dapat diselesaikan untuk $k-1$. Untuk k , ditempatkan ubin berbentuk L di tengah dan memiliki empat subsquare dengan dimensi $2k-1 \times 2k-1$ seperti yang ditunjukkan pada gambar 2 di atas. Jadi jika dapat menyelesaikan 4 subskuares, dapat menyelesaikan kuadrat lengkap.

