LAPORAN PRAKTIKUM 5

PROBLEM-PROBLEM DENGAN PEMECAHAN MASALAH MENGGUNAKAN PARADIGMA DIVIDE & CONQUER

MATA KULIAH
ANALISIS ALGORITMA
D10G.4205 & D10K.0400601

PENGAJAR : (1) MIRA SURYANI, S.Pd., M.Kom
(2) INO SURYANA, Drs., M.Kom
(3) R. SUDRAJAT, Drs., M.Si
FAKULTAS : MIPA
SEMESTER : IV dan VI

DISUSUN OLEH : AHMAD FAAIZ A (140810180023)

PROGRAM STUDI S-1 TEKNIK INFORMATIKA
DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN

APRIL 2019

Studi Kasus (Lanjutan)

Studi Kasus 5: Mencari Pasangan Tititk 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.

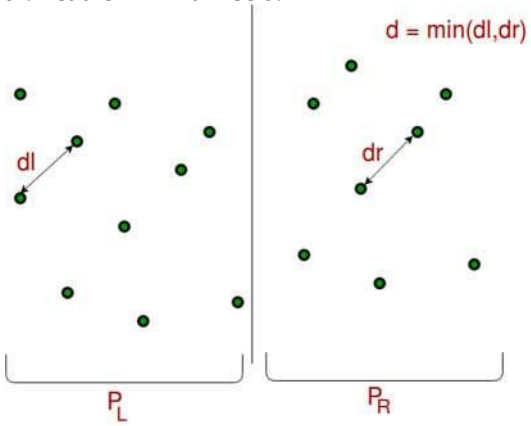
$$\|pq\| = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$$

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 (nLogn) menggunakan strategi Divide and Conquer**. Ikuti algoritma berikut:

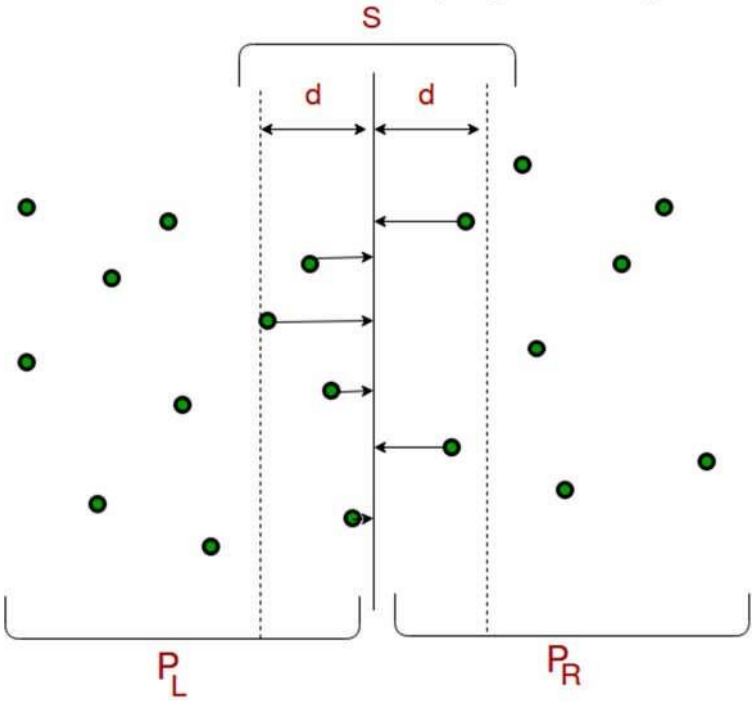
Input: An array of n points P[]
Output: The smallest distance between two points in the given array.

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 dl and dr. Find the minimum of dl and dr. 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 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(nLogn). 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 st rip 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++

```
#include <iostream>
#include <float.h>
#include <stdlib.h>
#include <math.h>
using namespace std;

struct Point
{
    int x, y;
};

int compareX(const void *a, const void *b)
{
```

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

int compareY(const void *a, const void *b)
{
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->y - p2->y);
}

float dist(Point p1, Point p2)
{
    return sqrt((p1.x - p2.x) * (p1.x - p2.x) +
                (p1.y - p2.y) * (p1.y - p2.y));
}

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;
}

float min(float x, float y)
{
    return (x < y) ? x : y;
}

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

    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;
}

float closestUtil(Point Px[], Point Py[], int n)
{
    if (n <= 3)
        return bruteForce(Px, n);
```

```
    int mid = n / 2;
    Point midPoint = Px[mid];


    Point Pyl[mid + 1];
    Point Pyr[n - mid - 1];
    int li = 0, ri = 0;
    for (int i = 0; i < n; i++)
    {
        if (Py[i].x <= midPoint.x)
            Pyl[li++] = Py[i];
        else
            Pyr[ri++] = Py[i];
    }

    float dl = closestUtil(Px, Pyl, mid);
    float dr = closestUtil(Px + mid, Pyr, n - mid);
    float d = min(dl, dr);

    int j = 0;
    for (int i = 0; i < n; i++)
        if (abs(Py[i].x - midPoint.x) < d)
            strip[j] = Py[i], j++;

    return min(d, stripClosest(strip, j, d));
}

float closest(Point P[], int n)
{
    Point Px[n];
    Point Py[n];
    for (int i = 0; i < n; i++)
    {
        Px[i] = P[i];
        Py[i] = P[i];
    }
    qsort(Px, n, sizeof(Point), compareX);
    qsort(Py, n, sizeof(Point), compareY);
    return closestUtil(Px, Py, n);
}

int main()
{
    Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};
    int n = sizeof(P) / sizeof(P[0]);
    cout << "The smallest distance is " << closest(P, n);
    return 0;
}
```

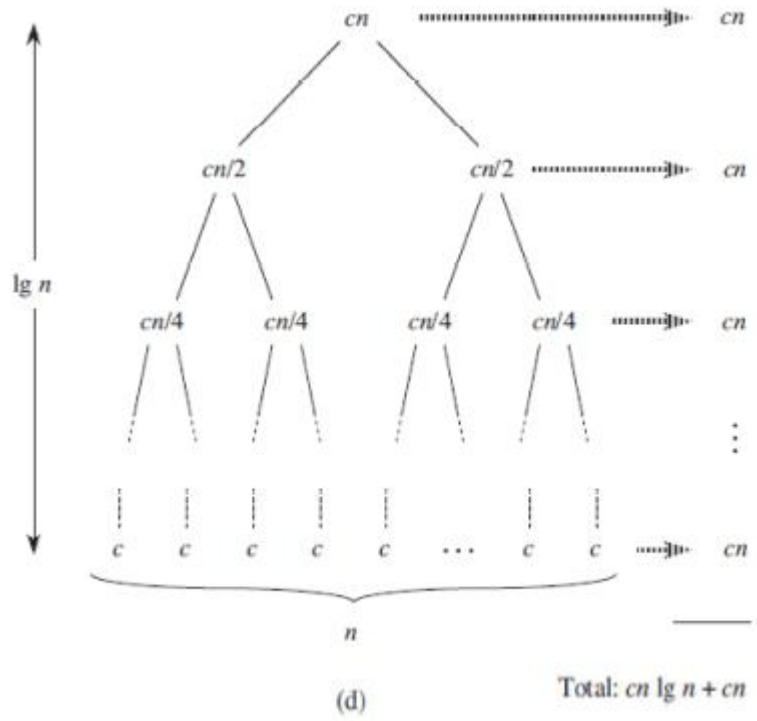
Screenshot:

```
The smallest distance is 1.41421
```

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

Rekurensi:

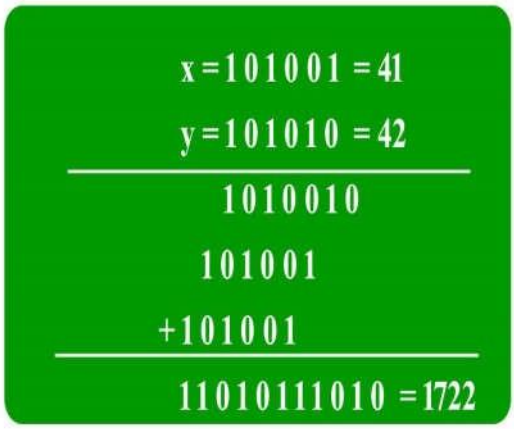
$$T(n) = \begin{cases} \theta(1) & ,n = 1 \\ 2T(n/2) + \theta(n) & ,n > 1 \end{cases}$$



Oleh karena itu, total cost keseluruhannya adalah cn lg n + cn atau $\theta(n\ lg\ n)$

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).



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:

$X = Xl*2^{n/2} + Xr$ [Xl and Xr contain leftmost and rightmost n/2 bits of X]
$Y = Yl*2^{n/2} + Yr$ [Yl and Yr contain leftmost and rightmost n/2 bits of Y]

- Produk XY dapat ditulis sebagai berikut:

$$XY = (Xl*2^{n/2} + Xr)(Yl*2^{n/2} + Yr)$$
$$= 2^n XlYl + 2^{n/2}(XlYr + XrYl) + XrYr$$

- 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 ^ 2). 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.

$$XlYr + XrYl = (Xl + Xr)(Yl + Yr) - XlYl - XrYr$$

- Jadi nilai akhir XY menjadi:

$$XY = 2^{n} XlYl + 2^{n/2} * [(Xl + Xr)(Yl + Yr) - XlYl - XrYr] + XrYr$$

- 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.

$$XY = 2^{2ceil(n/2)} XlYl + 2^{ceil(n/2)} * [(Xl + Xr)(Yl + Yr) - XlYl - XrYr] + XrYr$$

Tugas:

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

```
#include<iostream>
#include<stdio.h>

using namespace std;

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;
}

string addBitStrings( string first, string second )
{
    string result;

    int length = makeEqualLength(first, second);
    int carry = 0;

    for (int i = length-1 ; i >= 0 ; i--)
    {
```

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

        int sum = (firstBit ^ secondBit ^ carry)+'0';

        result = (char)sum + result;

        carry = (firstBit&secondBit) | (secondBit&carry) | (firstBit&carry);
    }

    if (carry)  result = '1' + result;

    return result;
}

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

long int multiply(string X, string Y)
{
    int n = makeEqualLength(X, Y);

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

    int fh = n/2
    int sh = (n-fh);

    string Xl = X.substr(0, fh);
    string Xr = X.substr(fh, sh);

    string Yl = Y.substr(0, fh);
    string Yr = Y.substr(fh, sh);

    long int P1 = multiply(Xl, Yl);
    long int P2 = multiply(Xr, Yr);
    long int P3 = multiply(addBitStrings(Xl, Xr), addBitStrings(Yl, Yr));

    return P1*(1<<(2*sh)) + (P3 - P1 - P2)*(1<<sh) + P2;
}

int main()
{
    printf ("%ld\n", multiply("1100", "1010"));
    printf ("%ld\n", multiply("110", "1010"));
    printf ("%ld\n", multiply("11", "1010"));
    printf ("%ld\n", multiply("1", "1010"));
    printf ("%ld\n", multiply("0", "1010"));
```

```
    printf ("%ld\n", multiply("111", "111"));
    printf ("%ld\n", multiply("11", "11"));
}
```

Screenshot:

```
120
60
30
10
0
49
9
```

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)

$T(n) = 3T\left(\frac{n}{2}\right) + n$

$= 3\left(3T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n = 9T\left(\frac{n}{4}\right) + \frac{5n}{2}$

$= 9\left(3T\left(\frac{n}{8}\right) + \frac{n}{4}\right) + \frac{5n}{2} = 27T\left(\frac{n}{8}\right) + \frac{19n}{4}$

$= \cdots$

$= 3^k T\left(\frac{n}{2^k}\right) + n + (k-1)\left(\frac{3^{k-1}n}{2^{k-1}}\right)$, diperoleh dengan rumus baris aritmatika, k=1,2,3,…

Berhenti jika ukuran table terkecil, $n = 1$

$T\left(\frac{n}{2^k}\right) = T(1) \quad \Rightarrow \quad \frac{n}{2^k} = 1 \quad \Rightarrow \quad k = \log_2 n$

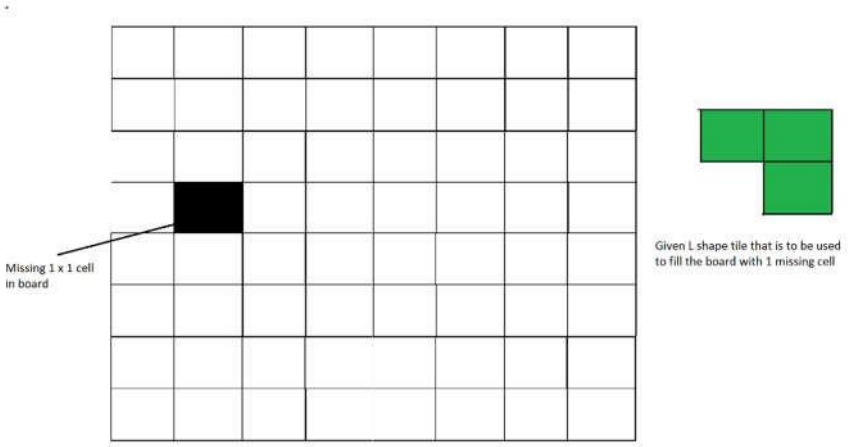
Sehingga,

$T(n) = nT(1) + cn\log_2 n$

$= na + cn\log_2 n$

$= O(n\log_2 n)$

Studi Kasus 7: Permasalahan Tata Letak Keramik Lantai (Tilling Problem)
Identifikasi Problem:
Diberikan papan berukutan n x n dimana n adalah dari bentuk 2k dimana k ≥ 1 (Pada dasarnya n adalah pangkat dari 2 dengan nilai minimumnya 2). Papan memiliki satu sel yang hilang (ukuran 1 x 1). Isi papan menggunakan ubin berbentuk L. Ubin berbentuk L berukuran 2 x 2 persegi dengan satu sel berukuran 1 × 1 hilang.

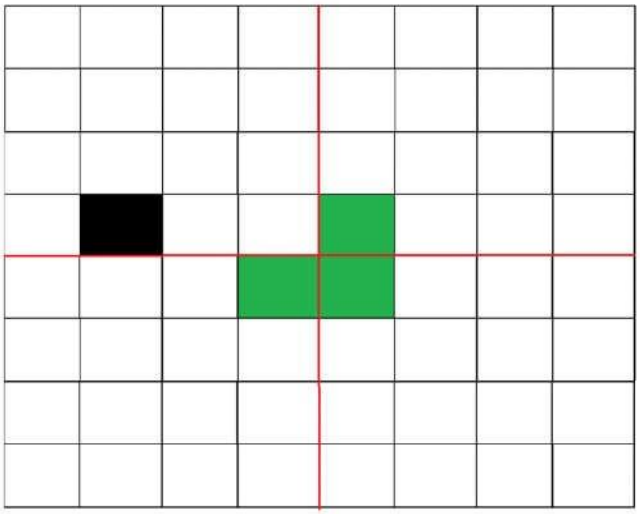


Gambar 2. Ilustrasi tilling problem

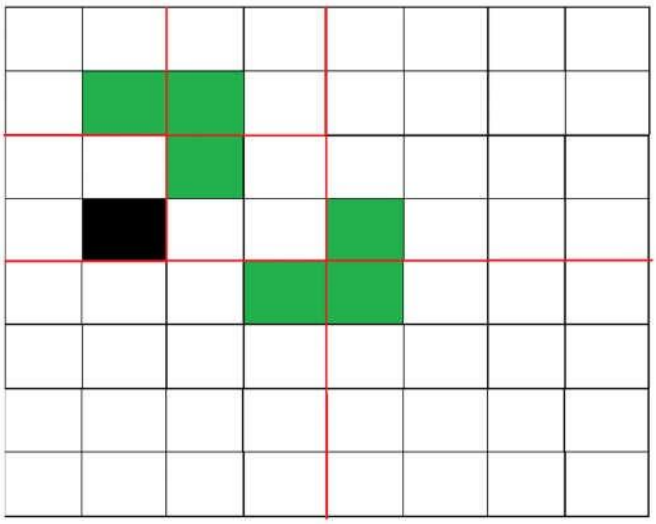
Solusi:
Masalah ini dapat diselesaikan menggunakan Divide and Conquer. Di bawah ini adalah algoritma rekursifnya

```
// 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)
```

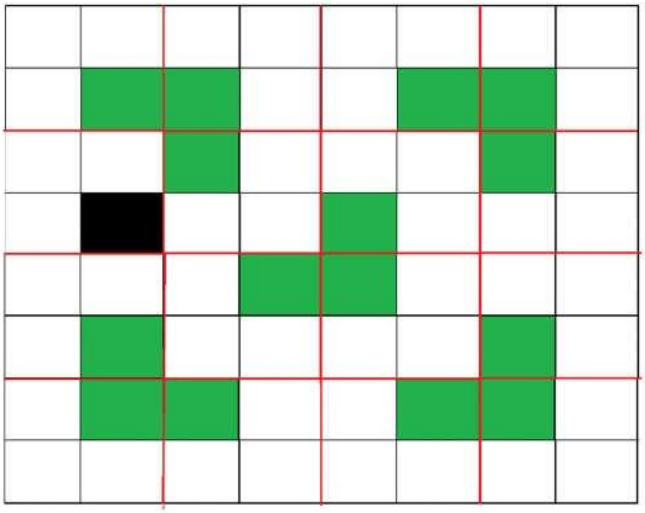
Gambar di bawah ini menunjukkan kerja algoritma di atas.



Gambar 3. Ilustrasi setelah tile pertama ditempatkan



Gambar 4 Perulangan untuk subsquare pertama.



Gambar 5. Memperlihatkan langkah pertama di keempat subsquares.

Tugas:

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

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
using namespace std;

int board[1000][1000];
int no = 0;
int quadrant = 0;

void trominoTile(int xBoard,
                 int yBoard,
                 int x_hole,
                 int y_hole,
                 int boardSize);

void trominoTile(int xBoard, int yBoard, int x_hole, int y_hole, int boardSize)
{
    int halfSize = boardSize / 2,
        xCenter = 0,
        yCenter = 0;

    xCenter = xBoard + halfSize - 1;
    yCenter = yBoard + halfSize - 1;

    if (boardSize == 2)
    {
        if (board[xBoard][yBoard + 1] == 0 && board[xBoard + 1][yBoard] == 0 && board[xBoar
d + 1][yBoard + 1] == 0)
        {
            no++;
            board[xBoard][yBoard + 1] = no;
            board[xBoard + 1][yBoard] = no;
            board[xBoard + 1][yBoard + 1] = no;
        }

        if (board[xBoard][yBoard] == 0 && board[xBoard + 1][yBoard] == 0 && board[xBoard +
1][yBoard + 1] == 0)
        {
            no++;
            board[xBoard][yBoard] = no;
            board[xBoard + 1][yBoard] = no;
            board[xBoard + 1][yBoard + 1] = no;
```

```
        }

        if (board[xBoard][yBoard + 1] == 0 && board[xBoard][yBoard] == 0 && board[xBoard +
1][yBoard + 1] == 0)
        {
            no++;
            board[xBoard + 1][yBoard + 1] = no;
            board[xBoard][yBoard + 1] = no;
            board[xBoard][yBoard] = no;
        }

        if (board[xBoard][yBoard + 1] == 0 && board[xBoard + 1][yBoard] == 0 && board[xBoar
d][yBoard] == 0)
        {
            no++;
            board[xBoard][yBoard] = no;
            board[xBoard][yBoard + 1] = no;
            board[xBoard + 1][yBoard] = no;
        }

        return;
    }

    if (x_hole <= xCenter)
    {
        if (y_hole <= yCenter)
        {
            if (board[xCenter][yCenter + 1] == 0 && board[xCenter + 1][yCenter] == 0 && boa
rd[xCenter + 1][yCenter + 1] == 0)
            {
                no++;
                board[xCenter][yCenter + 1] = no;
                board[xCenter + 1][yCenter] = no;
                board[xCenter + 1][yCenter + 1] = no;
                quadrant = 1;
            }
        }
        else
        {
            if (board[xCenter][yCenter] == 0 && board[xCenter + 1][yCenter] == 0 && board[x
Center + 1][yCenter + 1] == 0)
            {
                no++;
                board[xCenter][yCenter] = no;
                board[xCenter + 1][yCenter + 1] = no;
                board[xCenter + 1][yCenter] = no;
                quadrant = 2;
            }
```

```
        }
    }
    else
    {
        if (y_hole <= yCenter)
        {
            if (board[xCenter][yCenter + 1] == 0 && board[xCenter][yCenter] == 0 && board[x
Center + 1][yCenter + 1] == 0)
            {
                no++;
                board[xCenter][yCenter] = no;
                board[xCenter][yCenter + 1] = no;
                board[xCenter + 1][yCenter + 1] = no;
                quadrant = 3;
            }
        }
        else
        {
            if (board[xCenter + 1][yCenter] == 0 && board[xCenter][yCenter] == 0 && board[x
Center][yCenter + 1] == 0)
            {
                no++;
                board[xCenter][yCenter] = no;
                board[xCenter][yCenter + 1] = no;
                board[xCenter + 1][yCenter] = no;
                quadrant = 4;
            }
        }
    }

    if (quadrant == 1)
    {
        trominoTile(xBoard, yBoard, x_hole, y_hole, halfSize);
        trominoTile(xBoard, yCenter + 1, xCenter, yCenter + 1, halfSize);
        trominoTile(xCenter + 1, yBoard, xCenter + 1, yCenter, halfSize);
        trominoTile(xCenter + 1, yCenter + 1, xCenter + 1, yCenter + 1,
                    halfSize);
    }

    if (quadrant == 2)
    {
        trominoTile(xBoard, yBoard, xCenter, yCenter, halfSize);
        trominoTile(xBoard, yCenter + 1, x_hole, y_hole, halfSize);
        trominoTile(xCenter + 1, yBoard, xCenter + 1, yCenter, halfSize);
        trominoTile(xCenter + 1, yCenter + 1, xCenter + 1, yCenter + 1,
                    halfSize);
    }
```

```
    if (quadrant == 3)
    {
        trominoTile(xBoard, yBoard, xCenter, yCenter, halfSize);
        trominoTile(xBoard, yCenter + 1, xCenter, yCenter + 1, halfSize);
        trominoTile(xCenter + 1, yBoard, x_hole, y_hole, halfSize);
        trominoTile(xCenter + 1, yCenter + 1, xCenter + 1, yCenter + 1,
                    halfSize);
    }

    if (quadrant == 4)
    {
        trominoTile(xBoard, yBoard, xCenter, yCenter, halfSize);
        trominoTile(xBoard, yCenter + 1, xCenter, yCenter + 1, halfSize);
        trominoTile(xCenter + 1, yBoard, xCenter + 1, yCenter, halfSize);
        trominoTile(xCenter + 1, yCenter + 1, x_hole, y_hole, halfSize);
    }
}

int main()
{
    int boardSize, x_hole, y_hole;
    do
    {
        printf("\n-------------------------------------");
        printf("\nEnter size of board (0 to quit): ");
        scanf("%d", &boardSize);
        if (boardSize)
        {
            printf("\nEnter coordinates of missing hole: ");
            scanf("%d%d", &x_hole, &y_hole);
            for (int i = 1; i <= pow(2, boardSize); i++)
            {
                for (int j = 1; j <= pow(2, boardSize); j++)
                {
                    board[i][j] = 0;
                }
            }
            board[x_hole][y_hole] = -1;

            trominoTile(1, 1, x_hole, y_hole, pow(2, boardSize));

            for (int i = 1; i <= pow(2, boardSize); i++)
            {
                for (int j = 1; j <= pow(2, boardSize); j++)
                {
                    if (i == x_hole && j == y_hole)
                    {
                        board[i][j] == -1;
```

```
                        printf("%4s", "X");
                    }
                    else
                    {
                        printf("%4d", board[i][j]);
                    }
                }
                cout << endl;
            }
        }
        no = 0;
    }

    while (boardSize);

    return EXIT_SUCCESS;
}
```

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

$T\ (n) =\ 4T\ \left(\frac{n}{2}\right)\ +\ C$

$a = 4, \qquad b = 2, \qquad f(n) = 1$

$n^{\log_b a} = n^{\log_2 4} = n^2$

$f(n) = 1 = O(n^{\log_2 4 - \epsilon})$ untuk $\epsilon = 1$

Case 1 applies

Maka, solusinya adalah

$T(n) = O\left(n^{\log_b a}\right) = O(n^{\log_2 4}) = O(n^2)$