

Nama : Rizky Anugerah

NPM : 140810180049

Tugas 5

## 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

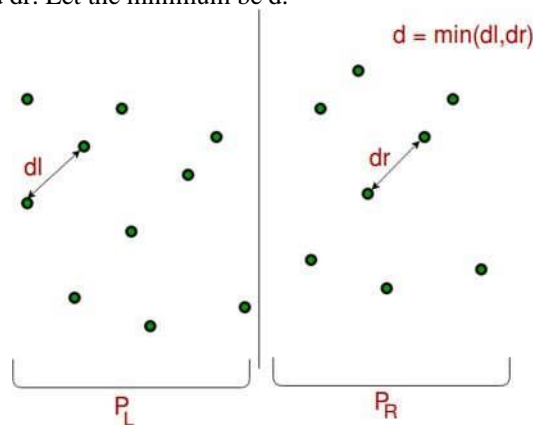
Solusi umum dari permasalahan tersebut adalah menggunakan algoritma **Brute force** dengan  $O(n^2)$ , hitung jarak antara setiap pasangan dan kembalikan yang terkecil. Namun, kita dapat menghitung jarak terkecil dalam waktu  **$O(n \log n)$**  menggunakan strategi **Divide and Conquer**. Ikuti algoritma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

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

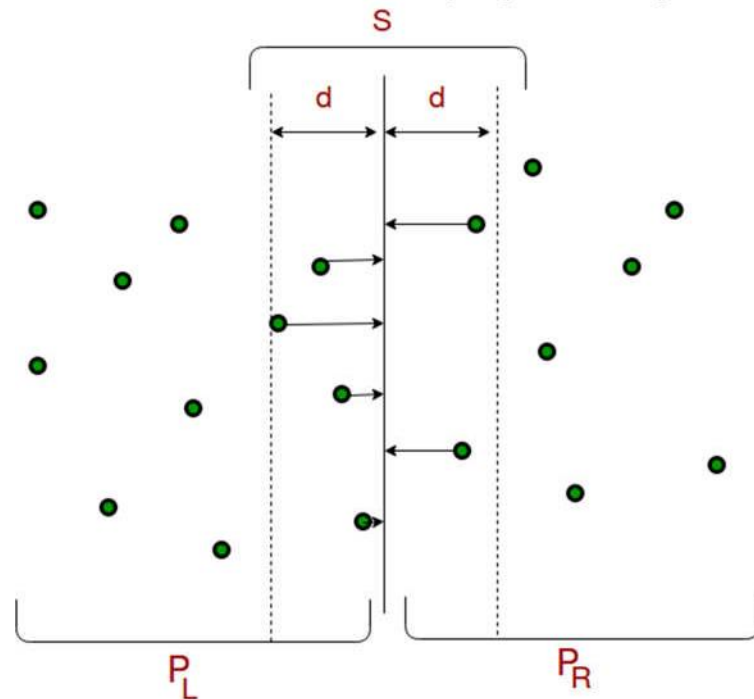
*Output:* The smallest distance between two points in the given array.

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      : Rizky Anugerah
NPM       : 140810180049
Kelas    : A
Program   : Closest Pair of Points using Divide and Conquer algorithm
*/

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

class Point
{
public:
    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;

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

    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 P[], int n)

```

```

{
    if (n <= 3)
        return bruteForce(P, n);

    int mid = n / 2;
    Point midPoint = P[mid];

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

    float d = min(dl, dr);

    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++;

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

float closest(Point P[], int n)
{
    qsort(P, n, sizeof(Point), compareX);

    return closestUtil(P, n);
}

int main()
{
    Point P[] = {{1, 4}, {10, 20}, {45, 60}, {4, 3}, {11, 6}, {2, 4}};
    int n = sizeof(P) / sizeof(P[0]);

    cout << "P[] = {{1, 4}, {10, 20}, {45, 60}, {4, 3}, {11, 6}, {2, 4}}" << endl <
< endl;
    cout << "Jarak terkecilnya adalah : " << closest(P, n);
}

```

- 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:**

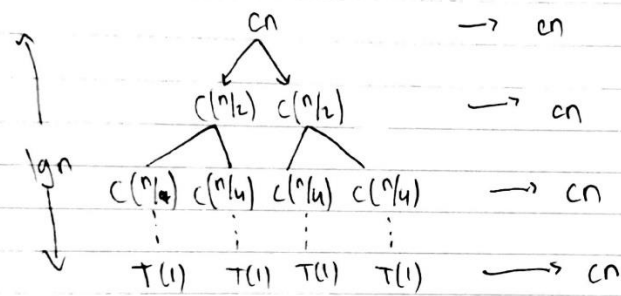
- Algoritma di atas membagi semua titik dalam dua set dan secara rekursif memanggil dua set ...  $2T(n/2)$
- Setelah membelah, ia menemukan strip dalam waktu ...  $O(n)$ ,
- membagi strip di garis tengah...  $O(n)$
- akhirnya menemukan titik terdekat dalam strip dalam waktu ...  $O(n)$ .

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

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

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

Recursion tree



$$T(n) = \underbrace{cn + cn + cn + \dots + cn}_{\lg n}$$

$$= \sum_{n=0}^{\lg n} cn$$

$$= cn \lg n$$

$$= O(n \lg n) \quad "$$

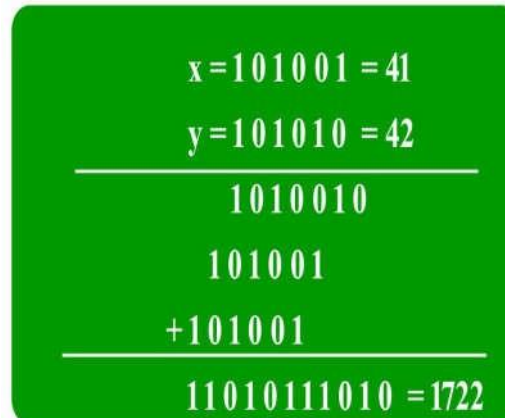
## Studi Kasus 6: Algoritma Karatsuba untuk Perkalian Cepat

### 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:

$\begin{aligned} X &= X_l * 2^{n/2} + X_r & [X_l \text{ and } X_r \text{ contain leftmost and rightmost } n/2 \text{ bits of } X] \\ Y &= Y_l * 2^{n/2} + Y_r & [Y_l \text{ and } Y_r \text{ contain leftmost and rightmost } n/2 \text{ bits of } Y] \end{aligned}$
--

- Produk XY dapat ditulis sebagai berikut:

$\begin{aligned} XY &= (X_l * 2^{n/2} + X_r) (Y_l * 2^{n/2} + Y_r) \\ &= 2^n X_l Y_l + 2^{n/2} (X_l Y_r + X_r Y_l) + X_r Y_r \end{aligned}$
---

- 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  $O(n^2)$ . 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.

$X_l Y_r + X_r Y_l = (X_l + X_r) (Y_l + Y_r) - X_l Y_l - X_r Y_r$
---

- Jadi nilai akhir XY menjadi:

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

- 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.

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

**Tugas:**

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

**Jawab:**

```
/*
Nama      : Rizky Anugerah
NPM       : 140810180049
Kelas    : A
Program   : Karatsuba algorithm for fast multiplication using Divide and Conquer algorithm
*/

#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; // If len1 >= len2
}

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

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

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

    int fh = n / 2;    // string bagian pertama, floor(n/2)
    int sh = (n - fh); // string bagian kedua, ceil(n/2)

    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()
{
    cout << "String 1 : 1111" << endl;
    cout << "String 2 : 1101" << endl;
    cout << "Hasil kali string adalah : " << multiply("1111", "1101");
}

```

- 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:**

Rekurensi dengan metode substitusi

$$T(n) = 3T(n/2) + O(n)$$

$$T(n) = 3T(n/2) + cn$$

$$T(n) \leq 3(c(n/2) \lg(n/2)) + cn$$

$$T(n) \leq 3/2 cn \lg(n/2) + cn$$

$$T(n) = cn \lg n - cn \lg 2 + cn$$

$$= cn \lg n - cn + cn$$

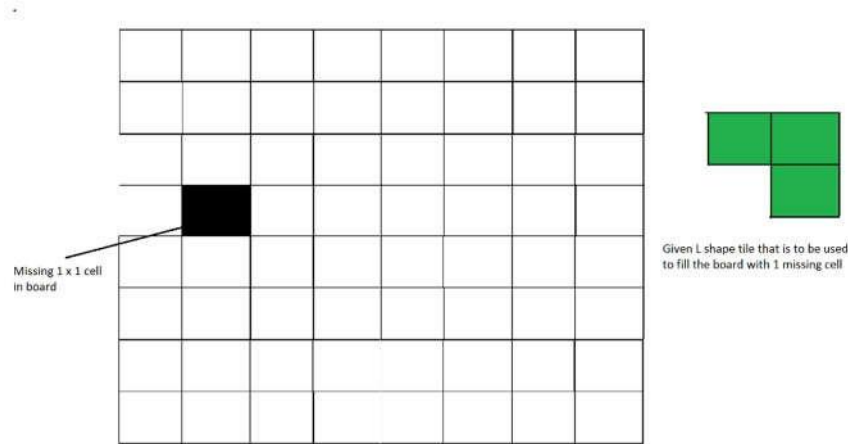
$$= cn \lg n$$

$$= O(n \lg n)$$

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

### Identifikasi Problem:

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



Gambar 2. Ilustrasi tiling problem

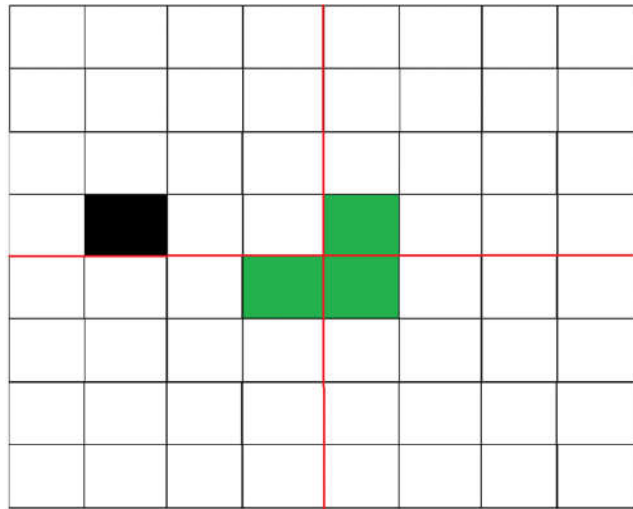
### Solusi:

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

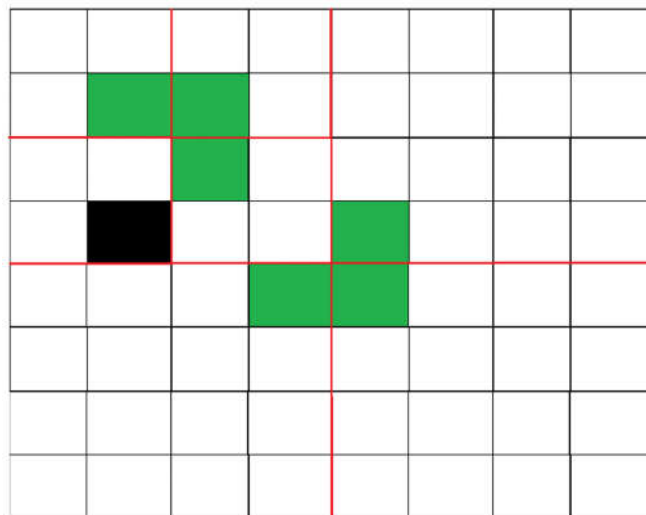
```
// n is size of given square, p is location of missing cell
Tile(int n, Point p)
```

- 1) Base case:  $n = 2$ , A  $2 \times 2$  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  $n/2 \times n/2$  subsquare that has a missing square. **Now all four subsquares of size  $n/2 \times n/2$  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  $p_1$ ,  $p_2$ ,  $p_3$  and  $p_4$  be positions of the 4 missing cells in 4 squares.
  - a) `Tile( $n/2$ ,  $p_1$ )`
  - b) `Tile( $n/2$ ,  $p_2$ )`
  - c) `Tile( $n/2$ ,  $p_3$ )`
  - d) `Tile( $n/2$ ,  $p_3$ )`

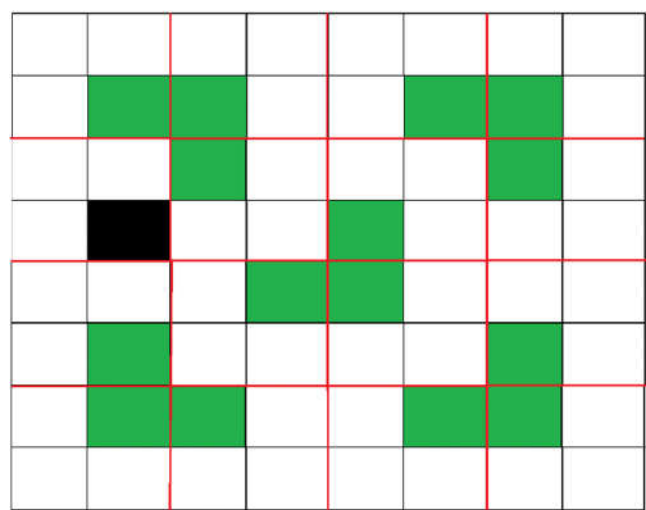
Gambar di bawah ini menunjukkan kerja algoritma di atas.



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++

**Jawab:**

```
/*
Nama      : Rizky Anugerah
NPM       : 140810180049
Kelas    : A
Program   : Problem Tilling
*/
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <climits>
using namespace std;

int **tromino_tile;

int power_2(int k)
{
    int result = 1;
    for (int i = 0; i < k; i++)
    {
        result = result * 2;
    }
    return result;
}

void allocate(int n, int x, int y)
{
    tromino_tile = new int *[n];
    for (int i = 0; i < n; i++)
    {
        tromino_tile[i] = new int[n];
    }
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            tromino_tile[i][j] = -1;
        }
    }
    tromino_tile[x][y] = 100;
}

void print_tromino(int n)
{

```

```

for (int i = 0; i < n; i++)
{
    cout << "\n";
    for (int j = 0; j < n; j++)
    {
        if (tromino_tile[i][j] == 100)
            cout << "\t"
                << "X";
        else
            cout << "\t" << tromino_tile[i][j];
    }
    cout << "\n";
}

void free_memory(int n)
{
    for (int i = 0; i < n; ++i)
    {
        delete[] tromino_tile[i];
    }
    delete[] tromino_tile;
}

void trominoTile(int n, int hole_row, int hole_col, int x, int y)
{
    int i;
    int half = n / 2;

    if (n == 2)
    {
        if (hole_row < x + half && hole_col < y + half)
        {
            i = rand() % 10;
            tromino_tile[x + half][y + half - 1] = i;
            tromino_tile[x + half][y + half] = i;
            tromino_tile[x + half - 1][y + half] = i;
        }

        else if (hole_row < x + half && hole_col >= y + half)
        {
            i = rand() % 10;
            tromino_tile[x + half][y + half - 1] = i;
            tromino_tile[x + half][y + half] = i;
            tromino_tile[x + half - 1][y + half - 1] = i;
        }

        else if (hole_row >= x + half && hole_col < y + half)

```

```

{
    i = rand() % 10;
    tromino_tile[x + half - 1][y + half] = i;
    tromino_tile[x + half][y + half] = i;
    tromino_tile[x + half - 1][y + half - 1] = i;
}

else
{
    i = rand() % 10;
    tromino_tile[x + half - 1][y + half] = i;
    tromino_tile[x + half][y + half - 1] = i;
    tromino_tile[x + half - 1][y + half - 1] = i;
}
}
else
{
    if (hole_row < x + half && hole_col < y + half)
    {
        i = rand() % 10;
        tromino_tile[x + half][y + half - 1] = i;
        tromino_tile[x + half][y + half] = i;
        tromino_tile[x + half - 1][y + half] = i;

        trominoTile(half, hole_row, hole_col, x, y);
        trominoTile(half, x + half, y + half - 1, x + half, y);
        trominoTile(half, x + half, y + half, x + half, y + half);
        trominoTile(half, x + half - 1, y + half, x, y + half);
    }
    else if (hole_row < x + half && hole_col >= y + half)
    {
        i = rand() % 10;
        tromino_tile[x + half][y + half - 1] = i;
        tromino_tile[x + half][y + half] = i;
        tromino_tile[x + half - 1][y + half - 1] = i;

        trominoTile(half, hole_row, hole_col, x, y + half);
        trominoTile(half, x + half, y + half - 1, x + half, y);
        trominoTile(half, x + half, y + half, x + half, y + half);
        trominoTile(half, x + half - 1, y + half - 1, x, y);
    }
    else if (hole_row >= x + half && hole_col < y + half)
    {
        i = rand() % 10;
        tromino_tile[x + half - 1][y + half] = i;
        tromino_tile[x + half][y + half] = i;
        tromino_tile[x + half - 1][y + half - 1] = i;
    }
}

```

```

        trominoTile(half, hole_row, hole_col, x + half, y);
        trominoTile(half, x + half - 1, y + half, x, y + half);
        trominoTile(half, x + half, y + half, x + half, y + half);
        trominoTile(half, x + half - 1, y + half - 1, x, y);
    }
    else
    {
        i = rand() % 10;
        tromino_tile[x + half - 1][y + half] = i;
        tromino_tile[x + half][y + half - 1] = i;
        tromino_tile[x + half - 1][y + half - 1] = i;

        trominoTile(half, hole_row, hole_col, x + half, y + half);
        trominoTile(half, x + half - 1, y + half, x, y + half);
        trominoTile(half, x + half, y + half - 1, x + half, y);
        trominoTile(half, x + half - 1, y + half - 1, x, y);
    }
}
}

int main()
{
    int k = 3, hole_row = 3, hole_col = 3;

    srand(time(NULL));

    if (k < 1)
        cout << "\nK harus positif\n";
    else
    {
        if (hole_row == 0 || hole_col == 0)
            cout << "\nJumlah baris dan kolom harus positif integer\n";
        else
        {
            int n = power_2(k);

            if (n >= hole_row && n >= hole_col)
            {
                hole_row--;
                hole_col--;

                allocate(n, hole_row, hole_col);

                trominoTile(n, hole_row, hole_col, 0, 0);
                print_tromino(n);
                free_memory(n);
            }
            else

```

```

        cout << "\nJumlah baris dan kolom tidak boleh lebih dari 2^k\n";
    }
}
return 0;
}

```

- 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:**

Metode master

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

$$= 4T(n/2) + 1 \quad \Rightarrow \quad a=4, b=2, f(n)=1$$

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

$$f(n) = 1$$

$$= O(n^{\log_2 4 - \epsilon}) \quad \text{untuk } \epsilon = 2$$

Case 1 applies

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