TUGAS 5 PRAKTIKUM

PEMECAHAN MASALAH MENGGUNAKAN

PARADIGMA ALGORITMA DIVIDE & CONQUER

Disusun sebagai salah satu tugas

mata kuliah Analisis Algoritma



Patricia Joanne

140810160065

Dikumpulkan tanggal

24 April 2019

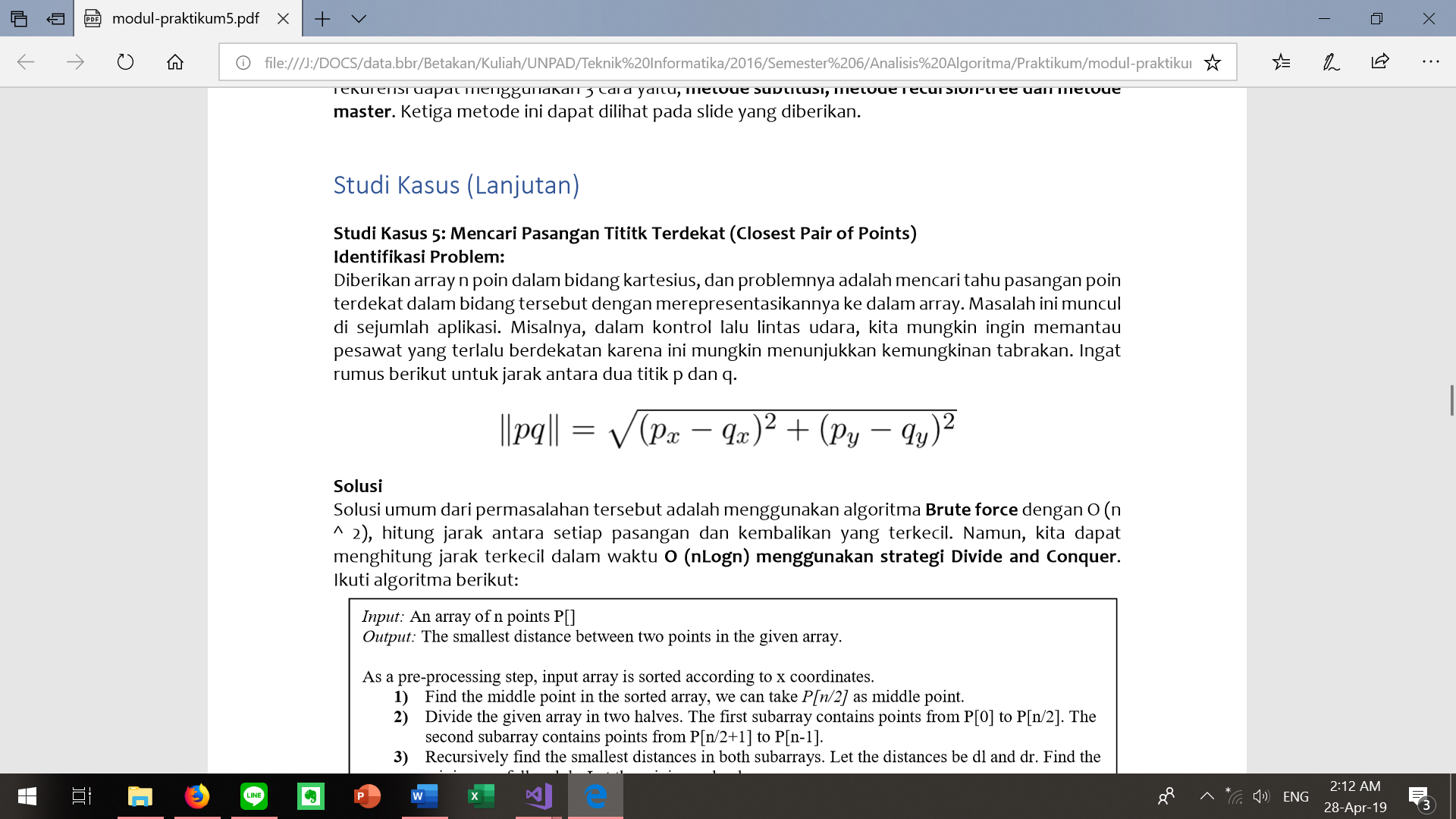
PROGRAM STUDI S-1 TEKNIK INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS PADJADJARAN

2019

**Nomor 1**



Jawab:

* Program: [cpp.sh/5eb52](http://cpp.sh/5eb52)

#include <bits/stdc++.h>

#include <chrono>

using namespace std;

using namespace std::chrono;

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; //Inisiasi jarak minimum = 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){

//Jika ada 2 atau 3 points, gunakan brute force

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(){

high\_resolution\_clock::time\_point t1 = high\_resolution\_clock

::now();

Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10},

{3, 4}};

int n = sizeof(P) / sizeof(P[0]);

cout<<"P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10},

{3, 4}};"<<endl<<endl;

cout<<"Jarak terkecil = "<<closest(P, n);

high\_resolution\_clock::time\_point t2 = high\_resolution\_clock

::now();

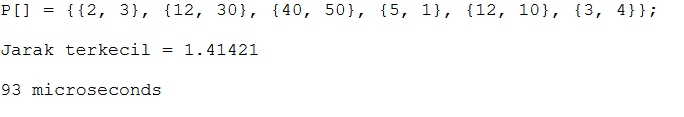
auto duration = duration\_cast<microseconds>

( t2 - t1 ).count();

cout<<endl<<endl<<duration<<" microseconds" <<endl;

}

* Output (input **P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}}**):



* Kompleksitas waktu:

Durasi waktu yang dibutuhkan untuk 6 titik input: 93 ms

Pembuktian dari algoritma:

Input: Array n poin P []

Output: Jarak terkecil antara dua titik dalam array yang diberikan.

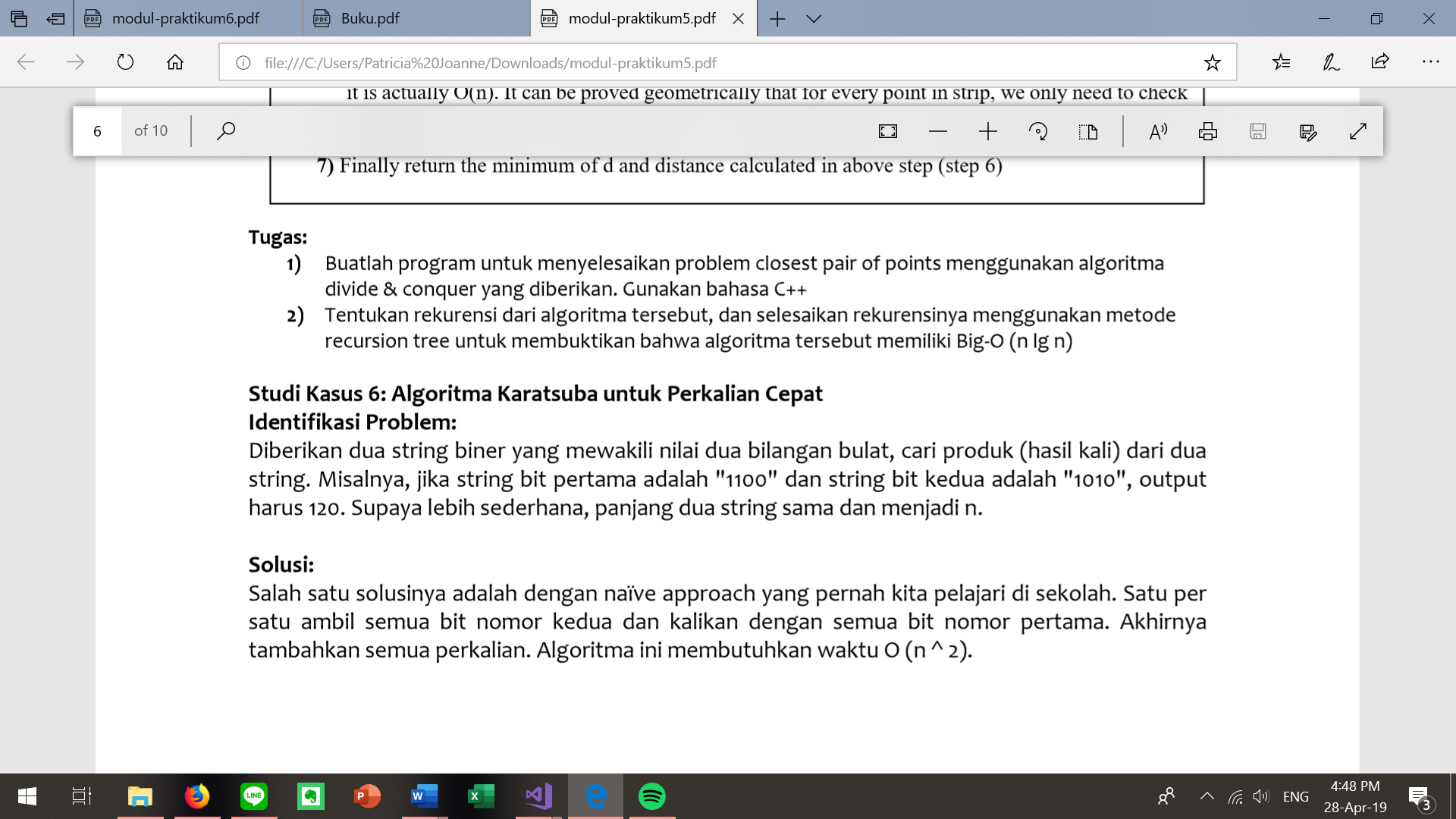
* + - 1. Array input diurutkan sesuai dengan koordinat x.
      2. Temukan titik tengah dalam array yang diurutkan, kita dapat mengambil P [n / 2] sebagai titik tengah.
      3. Bagi array yang diberikan dalam dua bagian. Subarray pertama berisi poin dari P [0] ke P [n / 2]. Subarray kedua berisi poin dari P [n / 2 + 1] ke P [n-1].
      4. Secara rekursif temukan jarak terkecil di kedua sub-layar. Tentukan jarak menjadi dl dan dr lalu temukan minimum dl dan dr. Tentukan minimum menjadi d.
      5. Sekarang kita perlu mempertimbangkan pasangan sedemikian sehingga satu titik berpasangan berasal dari setengah kiri dan lainnya adalah dari setengah kanan. Pertimbangkan garis vertikal yang melewati P [n / 2] dan temukan semua titik yang koordinat xnya lebih dekat daripada d ke garis vertikal tengah. Buat strip array [] dari semua titik tersebut.
      6. Urutkan strip array [] sesuai dengan koordinat y. Langkah ini adalah O (n \* log n). Itu dapat dioptimalkan untuk O (n) dengan menyortir dan menggabungkan secara rekursif.
      7. Temukan jarak terkecil di jalur []. Dari tampilan pertama, sepertinya ini adalah langkah O (n ^ 2), tetapi sebenarnya adalah O (n). Dapat dibuktikan secara geometris bahwa untuk setiap titik dalam strip, kita hanya perlu memeriksa paling banyak 7 poin setelahnya (perhatikan bahwa strip diurutkan berdasarkan koordinat Y).
      8. Terakhir kembalikan minimum d dan jarak yang dihitung pada langkah di atas (langkah 7).

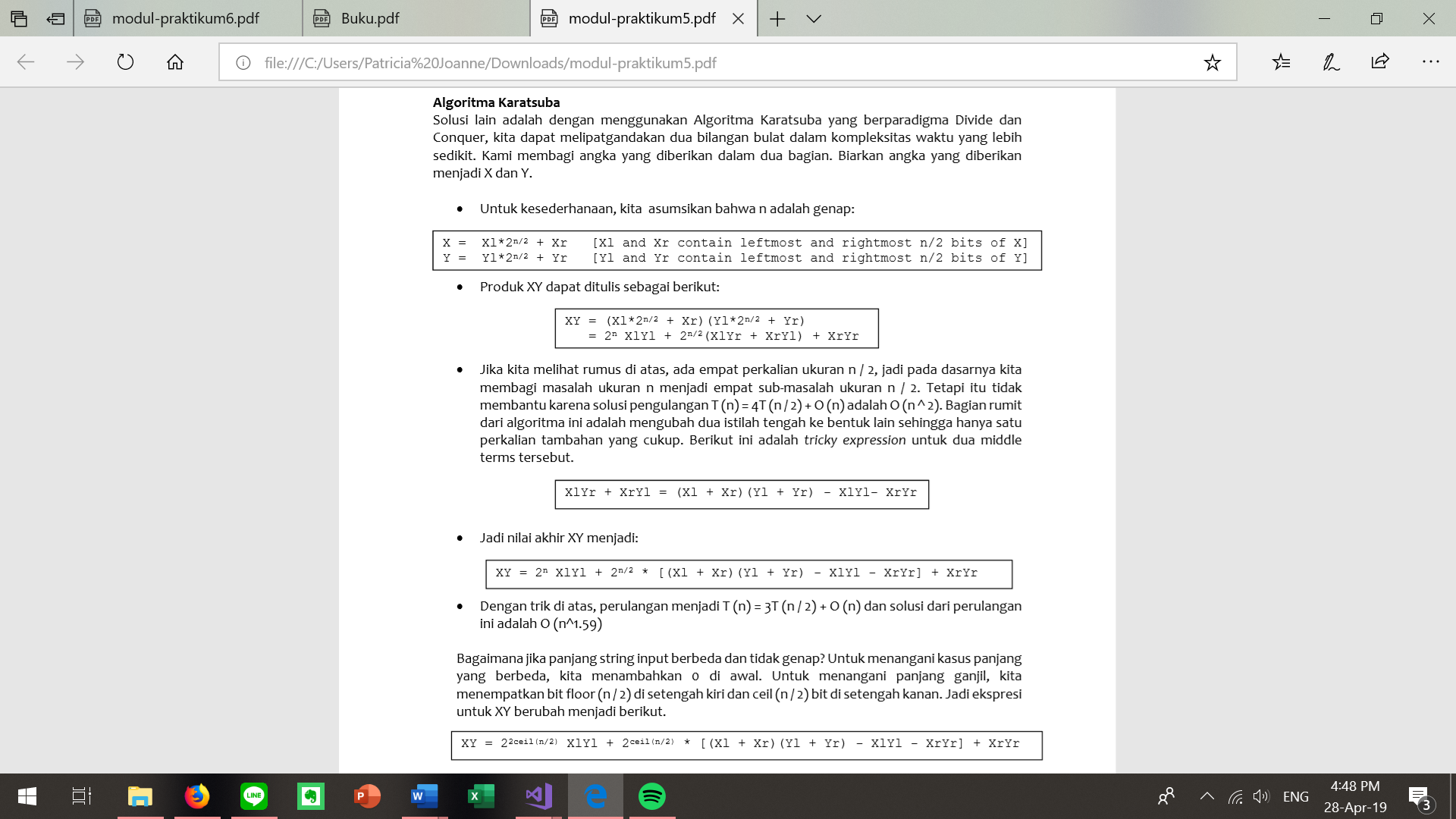
T(n) = 2T(n/2) + O(n) + O(n \* log n) + O(n)

T(n) = 2T(n/2) + O(n \* log n)

T(n) = T(n \* log n \* log n)

**Nomor 2**





Jawab:

* Program: [cpp.sh/33ou7](http://cpp.sh/33ou7)

#include<iostream>

#include<chrono>

#include<stdio.h>

using namespace std;

using namespace std::chrono;

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 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(){

high\_resolution\_clock::time\_point t1 = high\_resolution\_clock

::now();

cout<<"String 1: 1100, String 2: 1010"<<endl;

cout<<"Hasil kali: "<<multiply("1100", "1010");

high\_resolution\_clock::time\_point t2 = high\_resolution\_ clock::now();

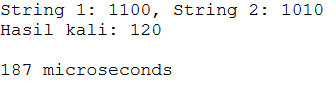
auto duration = duration\_cast<microseconds>

( t2 - t1 ).count();

cout<<endl<<endl<<duration<<" microseconds" <<endl;

}

* Output (input **1100 dan 1010**):



* Kompleksitas waktu:

Durasi waktu yang dibutuhkan untuk 6 titik input: 187 ms

Pembuktian dari algoritma:

Menggunakan algoritma Divide dan Conquer, kita dapat melipatgandakan dua bilangan bulat dalam kompleksitas waktu yang lebih sedikit. Bagi angka yang diberikan dalam dua bagian. Biarkan angka yang diberikan menjadi X dan Y.

Untuk kesederhanaan, mari kita asumsikan bahwa n adalah genap:

X = Xl\*2n/2 + Xr [Xl dan Xr mengandung n/2 bit paling kiri dan paling kanan X]

Y = Yl\*2n/2 + Yr [Yl dan Yr mengandung n/2 bit paling kiri dan paling kanan Y]

Hasilnya seperti ini:

XY = (Xl\*2n/2 + Xr)(Yl\*2n/2 + Yr)

= 2n XlYl + 2n/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 bagian tengah ke bentuk lain sehingga hanya satu perkalian tambahan yang cukup. Berikut ini adalah ekspresi sulit untuk dua bagian tengah:

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

Jadi nilai akhir XY menjadi:

XY = 2n XlYl + 2n / 2 \* [(Xl + Xr) (Yl + Yr) - XlYl - XrYr] + XrYr

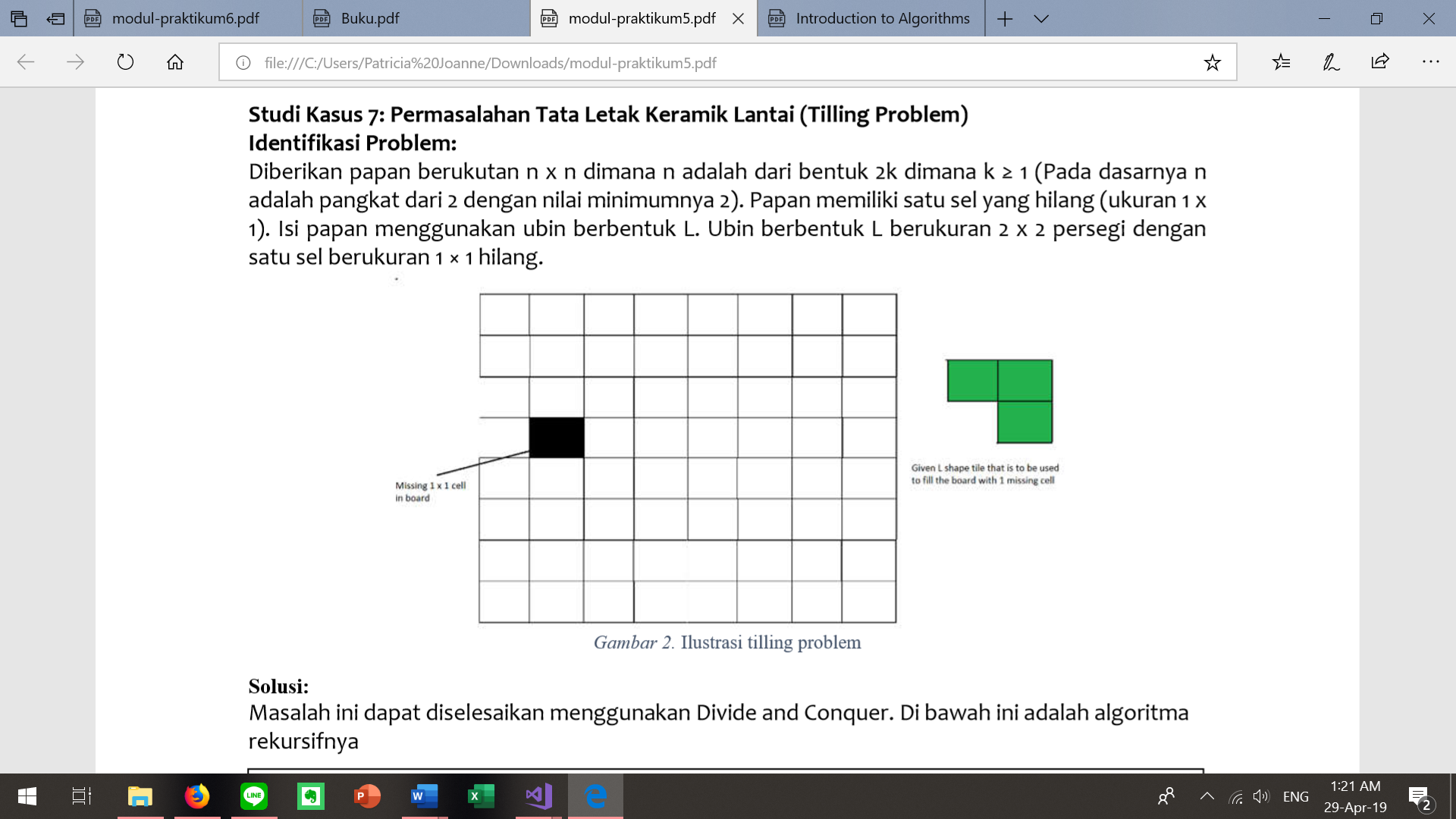
Dengan trik di atas, perulangan menjadi T (n) = 3T (n / 2) + O (n) dan solusi dari perulangan ini adalah O (n1.59).

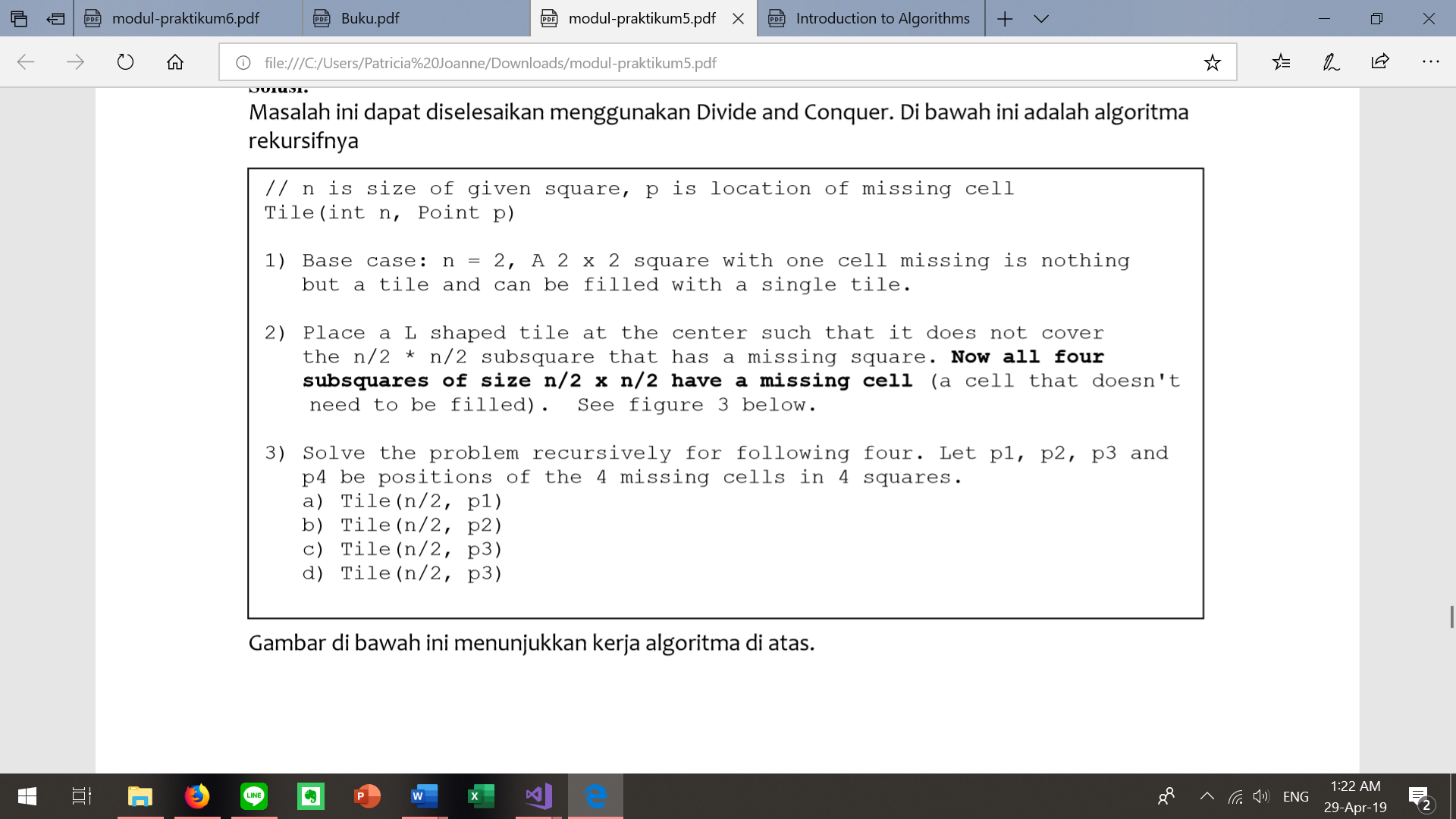
Untuk menangani kasus panjang yang berbeda, tambahkan 0 di awal. Untuk menangani panjang ganjil, tempatkan bit bawah (n / 2) di setengah kiri dan atas (n / 2) bit di setengah kanan. Jadi ekspresi untuk XY berubah menjadi berikut:

XY = 22bawah(n/2) XlYl + 2bawah(n/2) \* [(Xl + Xr)(Yl + Yr) - XlYl - XrYr] + XrYr

T(n) = O (n1.59)

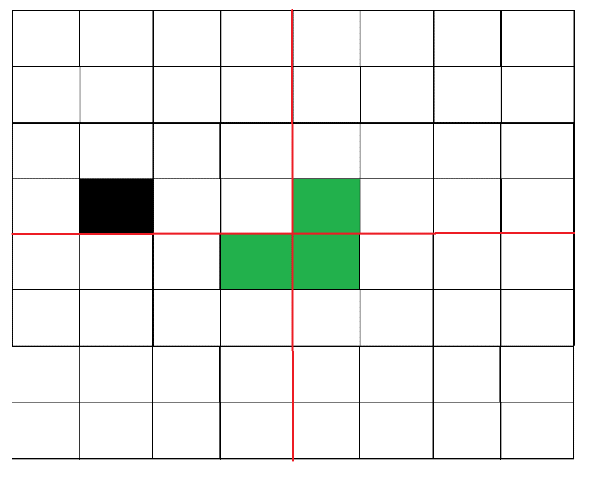
**Nomor 3**

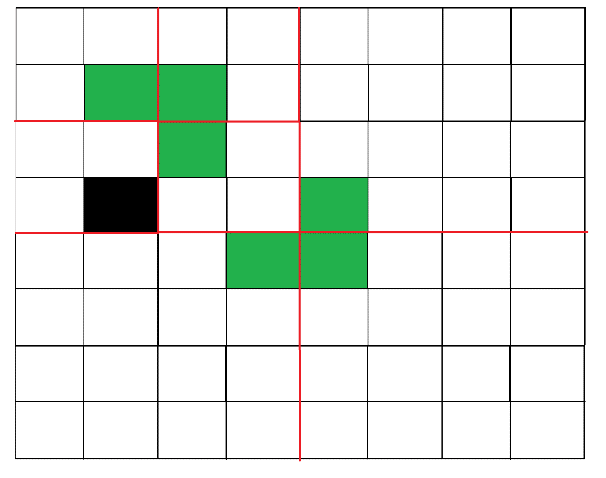


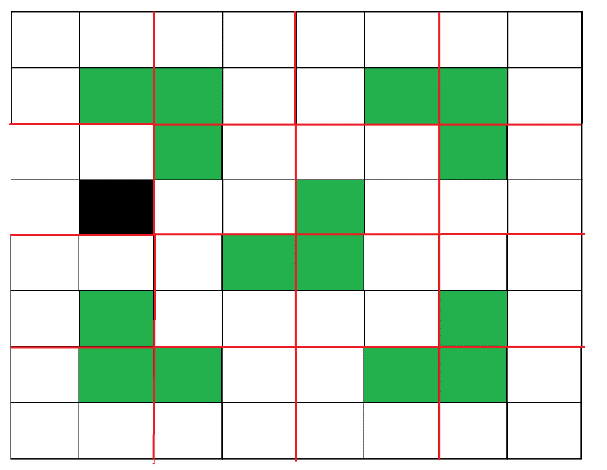


Jawab:

* Tahapan algoritma:







Pembuktian dari algoritma:

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

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

Pengulangan tersebut dapat diselesaikan dengan kompleksitas waktu O(n2).