Nama : Fauzan Akmal Hariz
NPM : 140810180005
Tugas-5

PRAKTIKUM ANALISIS ALGORITMA

PROBLEM-PROBLEM DENGAN PEMECAHAN MASALAH MENGGUNAKAN PARADIGMA DIVIDE & CONQUER

PENDAHULUAN

PARADIGMA DIVIDE & CONQUER

Divide & Conquer merupakan teknik algoritmik dengan cara memecah input menjadi beberapa bagian, memecahkan masalah di setiap bagian secara rekursif, dan kemudian menggabungkan solusi untuk subproblem ini menjadi solusi keseluruhan. Algoritma yang menggunakan divide and conquer adalah algoritma merge-sort.

PENERAPANNYA DALAM STUDI KASUS

1. Mencari Pasangan Tititk Terdekat (Closest Pair of Points)
2. Algoritma Karatsuba untuk Perkalian Cepat
3. Permasalahan Tata Letak Keramik Lantai (Tilling Problem)

Studi Kasus (Lanjutan)

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

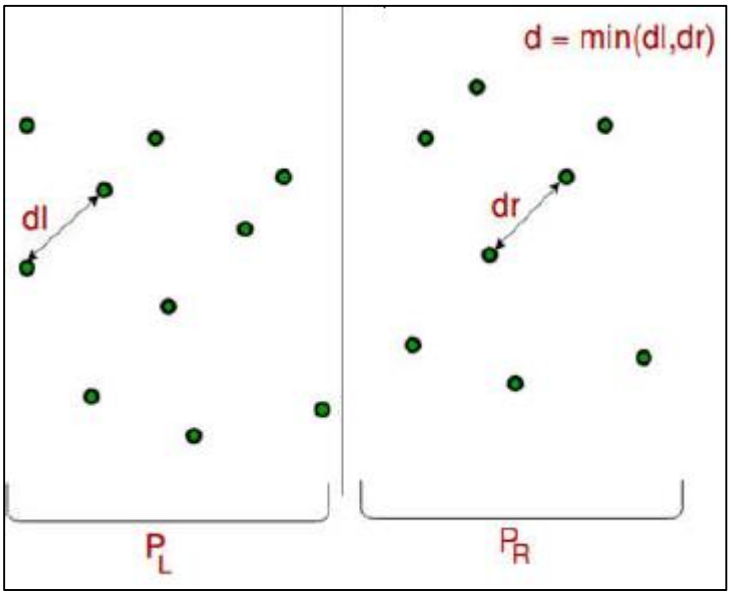
Identifikasi Problem:
Diberikan array n poin dalam bidang kartesius, dan problemnya adalah mencari tahu pasangan poin terdekat dalam bidang tersebut dengan merepresentasikannya ke dalam array. Masalah ini muncul di sejumlah aplikasi. Misalnya, dalam kontrol lalu lintas udara, kita mungkin ingin memantau 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 (n log n) menggunakan strategi Divide and Conquer. Ikuti algoritma berikut:

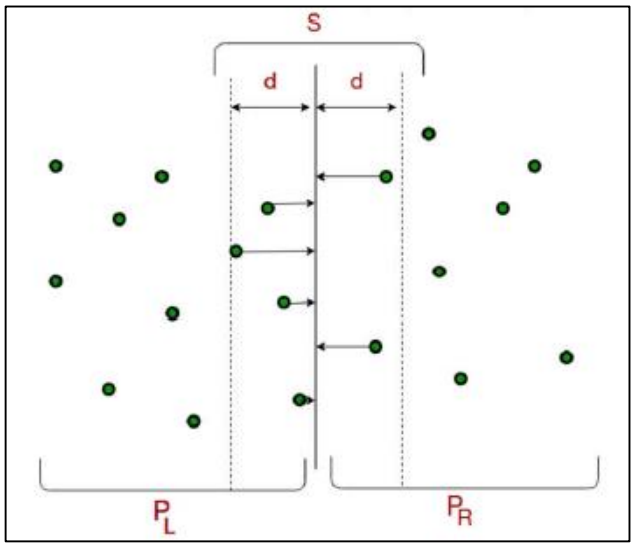
Input : An array of n points P[]
Output : The smallest distance between two points in the given array.
As a pre-processing step, input array is sorted according to x coordinates.

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 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(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++.

Source Code:

```
/*
Nama            :   Fauzan Akmal Hariz
NPM             :   140810180005
Kelas Kuliah    :   A
Kelas Praktikum :   B
Tanggal Buat    :   30 Maret 2020
Praktikum       :   Analisis Algoritma
Program         :   Closest Pair of Points
Deskripsi       :   Studi Kasus 5 - Mencari Pasangan Titik Terdekat (Clos
est Pair of Points)
*/

#include <bits/stdc++.h>
#include <chrono>

using namespace std;
using namespace std::chrono;
```

```
class Point{
    public:
    int x;
    int 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(){
    int n;

    cout << "\n==============================\n";
    cout << "Program Closest Pair of Points\n";
    cout << "==============================\n";

    high_resolution_clock::time_point t1 = high_resolution_clock::now();
    Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};
    n = sizeof(P) / sizeof(P[0]);

    cout << "\nP[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4
}}\n\n";
    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 << "\n\nDurasi Waktu: " << duration << " microseconds\n";
}
```

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

```
==============================
Program Closest Pair of Points
==============================

P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}}

Jarak Terkecil: 1.41421

Durasi Waktu: 1000 microseconds
```

Durasi waktu yang dibutuhkan untuk 6 titik input: 1000 microseconds

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

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).

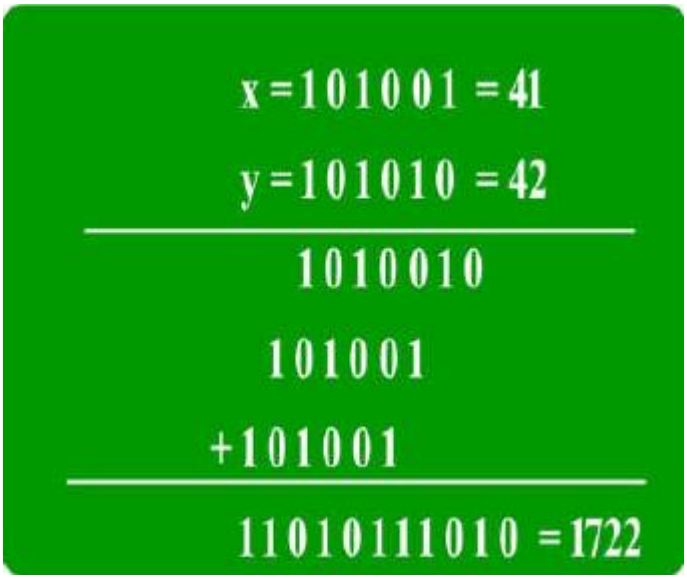
Jadi, T(n) dapat dinyatakan sebagai berikut:

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

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++.

Source Code:

```
/*
Nama            :   Fauzan Akmal Hariz
NPM             :   140810180005
Kelas Kuliah    :   A
Kelas Praktikum :   B
Tanggal Buat    :   30 Maret 2020
Praktikum       :   Analisis Algoritma
Program         :   Algoritma Karatsuba
Deskripsi       :   Studi Kasus 5 - Algoritma Karatsuba untuk Menyelesaik
an Problem Fast Multiplication
*/

#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&carr
y);
    }

    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(){
    string string1;
    string string2;

    cout << "\n=======================================\n";
    cout << "    Program Algoritma Karatsuba untuk\n";
    cout << "Menyelesaikan Problem Fast Multiplication\n";
    cout << "=======================================\n";

    cout << "\nMasukkan String 1 (Contoh: 1100): "; cin >> string1;
    cout << "Masukkan String 2 (Contoh: 1010): "; cin >> string2;

    high_resolution_clock::time_point t1 = high_resolution_clock::now();
    cout << "\nHasil Kali: " << multiply(string1, string2);


    high_resolution_clock::time_point t2 = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>( t2 - t1 ).count();
    cout << "\n\nDurasi Waktu: " << duration << " microseconds\n";
}
```

Output (input 1100 dan 1010):

```
=======================================
    Program Algoritma Karatsuba untuk
Menyelesaikan Problem Fast Multiplication
=======================================

Masukkan String 1 (Contoh: 1100): 1100
Masukkan String 2 (Contoh: 1010): 1010

Hasil Kali: 120

Durasi Waktu: 854 microseconds
```

Durasi waktu yang dibutuhkan: 854 microseconds

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).

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 = $Xl*2^{n/2}+Xr$ [Xl dan Xr mengandung n/2 bit paling kiri dan paling kanan X]
Y = $Yl*2^{n/2}+Yr$ [Yl dan Yr mengandung n/2 bit paling kiri dan paling kanan Y]

Hasilnya seperti ini:
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 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

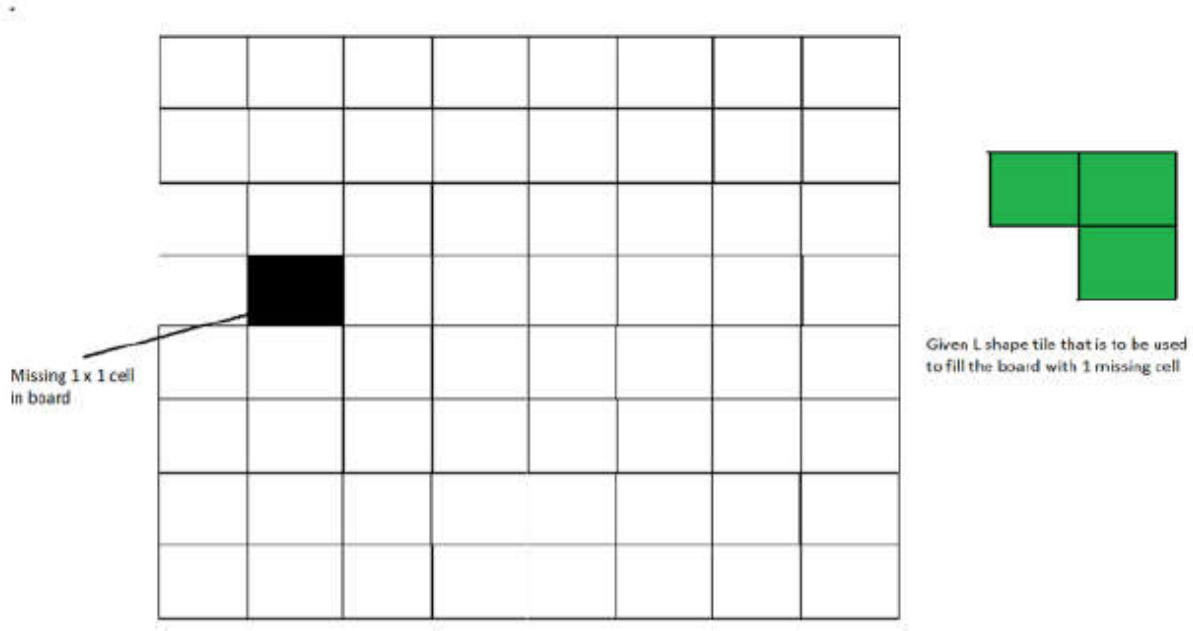
Jadi nilai akhir XY menjadi:
XY = 2n XlYl + 2n / 2 * [(Xl + Xr) (Yl + Yr) - XlYl - XrYr] + XrYr
Dengan trik di atas, perulangan menjadi T (n) = 3T (n / 2) + O (n) dan solusi dari perulangan ini adalah O ($n^{1.59}$).

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

Jadi, T(n) = O ($n^{1.59}$)

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

Identifikasi Problem:
Diberikan papan berukuran 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.



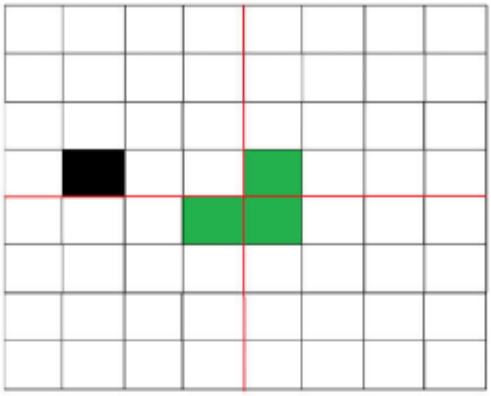
Gambar 2. Ilustrasi tilling problem

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

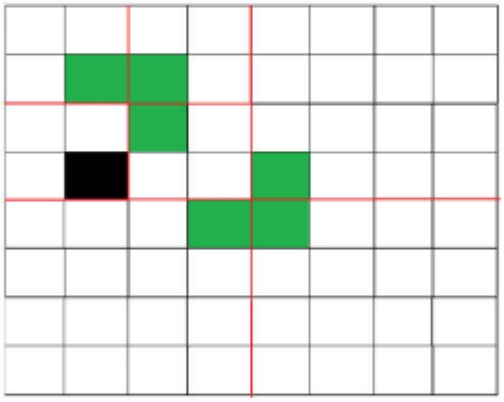
```
// n adalah ukuran kotak yang diberikan, p adalah lokasi sel yang hilang
Ubin (int n, Point p)

1) Kasus dasar: n = 2, A 2 x 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 x 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)
```

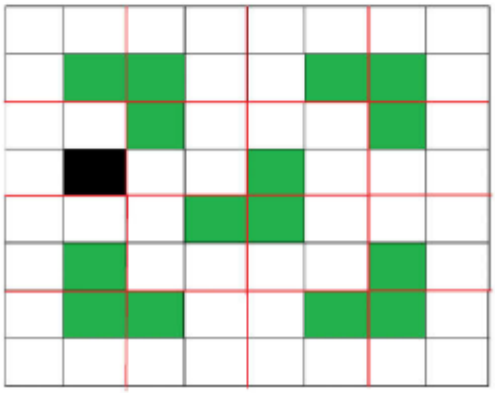
Gambar di bawah ini menunjukkan kerja algoritma di atas.



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++.

Source Code:

```
/*
Nama            :   Fauzan Akmal Hariz
NPM             :   140810180005
Kelas Kuliah    :   A
```

```
Kelas Praktikum :   B
Tanggal Buat    :   30 Maret 2020
Praktikum       :   Analisis Algoritma
Program         :   Tilling Problem
Deskripsi       :   Studi Kasus 5 - Tilling Problem - Permasalahan Tata L
etak Keramik Lantai
*/

// C++ implementation to count number of ways to
// tile a floor of size n x m using 1 x m tiles
#include <bits/stdc++.h>

using namespace std;

// function to count the total number of ways
int countWays(int n, int m){
    // 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 > m){
            count[i] = count[i - 1] + count[i - m];
        }
        // base cases
        else if (i < m){
            count[i] = 1;
        }
        // i = = m
        else{
            count[i] = 2;
        }
    }
    // required number of ways
    return count[n];
}

// Driver program to test above
int main(){
    int n;
    int m;

    cout << "\n=======================\n";
    cout << "Program Tilling Problem\n";
    cout << "=======================\n\n";
```

```
    cout << "Masukkan Angka 1 (Contoh: 2): "; cin >> n;
    cout << "Masukkan Angka 2 (Contoh: 2): "; cin >> m;

    cout << "\nNumber of Ways = " << countWays(n, m) << endl;
    return 0;
}
```

Output:

```
======================
Program Tilling Problem
======================

Masukkan Angka 1 (Contoh: 2): 2
Masukkan Angka 2 (Contoh: 2): 2

Number of Ways = 2
```

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.

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 $O(n^2)$.

Cara kerjanya:
Pengerjaan algoritma Divide and Conquer dapat dibuktikan menggunakan Mathematical Induction. Biarkan kuadrat input berukuran 2k * 2k di mana k>=1.

Kasus Dasar:
Kita tahu bahwa masalahnya dapat diselesaikan untuk k = 1. Kami memiliki 2x2 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 subsqure dengan dimensi 2k-1 * 2k-1 seperti yang ditunjukkan pada gambar 2 di atas. Jadi jika dapat menyelesaikan 4 subskuares, maka dapat menyelesaikan kuadrat lengkap.

TEKNIK PENGUMPULAN

- Lakukan push ke github/gitlab untuk semua program dan laporan hasil analisa yang berisi jawaban dari pertanyaan-pertanyaan yang diajukan. Silahkan sepakati dengan asisten praktikum.

PENUTUP

- Ingat, berdasarkan Peraturan Rektor No 46 Tahun 2016 tentang Penyelenggaraan Pendidikan, mahasiswa wajib mengikuti praktikum 100%.
- Apabila tidak hadir pada salah satu kegiatan praktikum segeralah minta tugas pengganti ke asisten praktikum.
- Kurangnya kehadiran Anda di praktikum, memungkinkan nilai praktikum Anda tidak akan dimasukkan ke nilai mata kuliah.

REFERENSI

PPT Praktikum Analisis Algortima (Pertemuan 5)

Modul Praktikum 5 Analisis Algoritma