LAPORAN PRAKTIKUM 5
ANALISIS ALGORITMA

NAMA	:	Nadhifal Abdurrahman Rendusara
NPM	:	140810180048
KELAS	:	B

PROGRAM STUDI S-1 TEKNIK INFORMATIKA
DEPARTEMEN ILMU KOMPUTER

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN

Tugas 5

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.

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

Tugas:

1) Buatlah program untuk menyelesaikan problem closest pair of points menggunakan algoritma divide & conquer yang diberikan. Gunakan bahasa C++
2) Tentukan rekurensi dari algoritma tersebut, dan selesaikan rekurensinya menggunakan metode recursion tree untuk membuktikan bahwa algoritma tersebut memiliki Big-O (n lg n)

Jawaban Studi Kasus 5.1

```
1.  /*
2.  Nama    = Nadhifal A. Rendusara
3.  NPM     = 140810180048
4.  Kelas   = B
5.  */
6.  #include <iostream>
7.  #include <float.h>
8.  #include <math.h>
9.  using namespace std;
10.
11. struct point {
12.     int x, y;
13. };
14.
15. void input(int &n)
16. {
17.     cout << "Banyak titik dalam array : ";
18.     cin >> n;
19. }
20.
21. void input(point P[], int n)
22. {
23.     cout << endl;
24.     for (int i = 0; i < n; i++)
25.     {
26.         cout << "Masukkan nilai x titik ke-" << i + 1 << " : ";
27.         cin >> P[i].x;
28.         cout << "Masukkan nilai y titik ke-" << i + 1 << " : ";
29.         cin >> P[i].y;
30.     }
31. }
32.
33. int compareX(const void *a, const void *b)
34. {
35.     point *p1 = (point *)a, *p2 = (point *)b;
36.     return (p1->x - p2->x);
37. }
38.
39. int compareY(const void *a, const void *b)
40. {
```

```
    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);
```

```
107.        }
108.
109.        int main()
110.        {
111.            int n = 0;
112.            input(n);
113.
114.            point P[n];
115.            input(P, n);
116.
117.            cout << "\nJarak pasangan titik terdekat adalah " << closest(P, n) << en
    dl;
118.        }
```

Jawaban Studi Kasus 5.2

Rekurensi algoritma Closest Pair of Points menggunakan metode recursion tree:

- Algoritma ini menerapkan algoritma divide and conquer dimana algoritma ini membagi array titik menjadi dua bagian dan secara rekursif memanggil dua bagian tersebut. Ketika suatu bagian hanya memiliki 2 titik maka jarak kedua titik tersebut dibandingkan dengan jarak titik lainnya sampai ditemukan jarak minimum. Waktu yang dibutuhkan dari langkah ini adalah 2T(n/2).

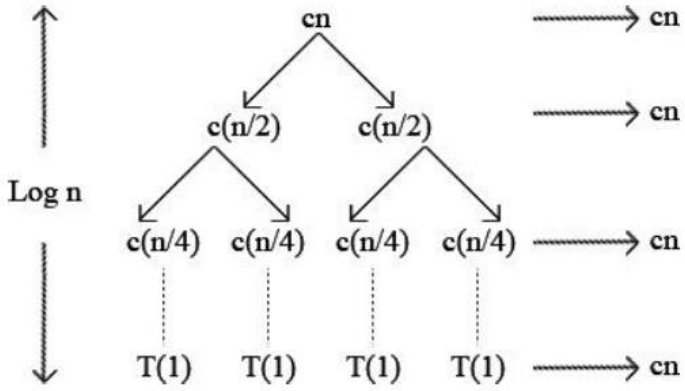
- Setelah didapatkan jarak minimum dibuatlah array strip ditengah dengan jarak minimum tadi ke kanan dan ke kiri. Pembuatan array strip ini membutuhkan waktu O(n)

- Array strip ini lalu diurutkan sehingga membutuhkan waktu O(n).

- Setelah array strip terurut, dicarilah jarak pasangan titik terdekat dari dalam array strip. Langkah ini membutuhkan waktu O(n).

Didapat rekurensi:
$T(n) = 2T(n/2) + O(n) + O(n) + O(n)$
$T(n) = 2T(n/2) + O(n)$
$T(n) = 2T(n/2) + cn$

Pembuktian menggunakan recursion tree:



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.

Tugas:

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

Jawaban Studi Kasus 6.1

```
/*
Nama    = Nadhifal A. Rendusara
NPM     = 140810180048
Kelas   = B
*/
#include <iostream>
using namespace std;

void input(string &X, string &Y)
{
    cout << "String pertama\t: ";
    getline(cin, X);
    cout << "String kedua\t: ";
    getline(cin, Y);
}

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';
```

```
48.
49.         result = (char)sum + result;
50.
51.         carry = (firstBit & secondBit) | (secondBit & carry) | (firstBit & carry);

52.     }
53.
54.     if (carry)
55.         result = '1' + result;
56.
57.     return result;
58. }
59.
60. int multiplyBit(string a, string b)
61. {
62.     return (a[0] - '0') * (b[0] - '0');
63. }
64.
65. long int multiplyBits(string X, string Y)
66. {
67.     for (int i = 0; i < X.length(); i++)
68.         if (X[i] != '0' && X[i] != '1') {
69.             cout << "String pertama bukan kode biner!" << endl;
70.             return 0;
71.         }
72.     for (int i = 0; i < Y.length(); i++)
73.         if (Y[i] != '0' && Y[i] != '1') {
74.             cout << "String pertama bukan kode biner!" << endl;
75.             return 0;
76.         }
77.
78.     int n = makeEqualLength(X, Y);
79.
80.     if (n == 0)
81.         return 0;
82.     else if (n == 1)
83.         return multiplyBit(X, Y);
84.
85.     int fh = n / 2;
86.     int sh = (n - fh);
87.
88.     string Xl = X.substr(0, fh);
89.     string Xr = X.substr(fh, sh);
90.
91.     string Yl = Y.substr(0, fh);
92.     string Yr = Y.substr(fh, sh);
93.
94.     long int P1 = multiplyBits(Xl, Yl);
95.     long int P2 = multiplyBits(Xr, Yr);
96.     long int P3 = multiplyBits(addBitStrings(Xl, Xr), addBitStrings(Yl, Yr));
97.
98.     return P1 * (1 << (2 * sh)) + (P3 - P1 - P2) * (1 << sh) + P2;
99. }
100.
101.        int main()
102.        {
103.            string X = "\0", Y = "\0";
104.            input(X, Y);
105.
106.            cout << "\nHasil perkalian kedua bit adalah " << multiplyBits(X, Y);
107.        }
```

Jawaban Studi Kasus 6.2

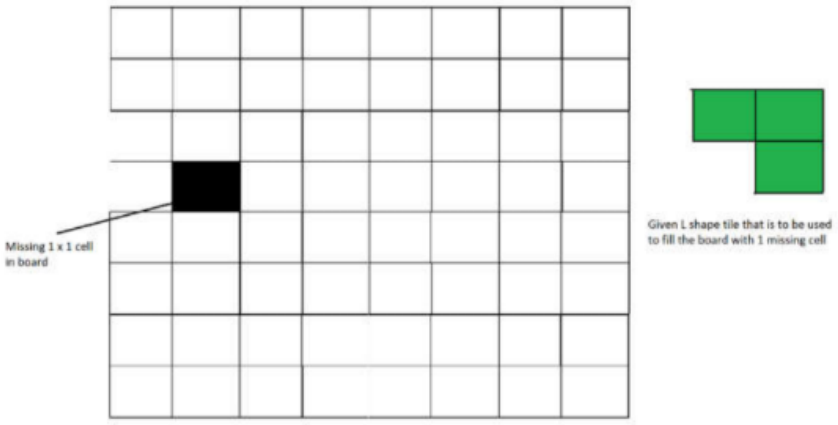
Rekurensi algoritma Karatsuba menggunakan metode substitusi

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

$T(n) \leq 3(c(n/2) \log(n/2)) + cn$
$T(n) \leq (3/2)(n \log(n/2)) + cn$
$T(n) = cn \log n – cn \log 2 + cn$
$T(n) = cn \log n – cn + cn$
$T(n) = cn \log n$
$T(n) = O(n \log n)$

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

Tugas:

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

Jawaban Studi Kasus 7.1

```
/*
Nama    = Nadhifal A. Rendusara
NPM     = 140810180048
Kelas   = B
*/
#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;
        }
```

```
94.     } else {
95.         if (hole_row < x + half && hole_col < y + half) {
96.             i = rand() % 10;
97.             tromino_tile[x + half][y + half - 1] = i;
98.             tromino_tile[x + half][y + half] = i;
99.             tromino_tile[x + half - 1][y + half] = i;
100.
101.                    trominoTile(half, hole_row, hole_col, x, y);
102.                    trominoTile(half, x + half, y + half - 1, x + half, y);
103.                    trominoTile(half, x + half, y + half, x + half, y + half);
104.                    trominoTile(half, x + half - 1, y + half, x, y + half);
105.                } else if (hole_row < x + half && hole_col >= y + half) {
106.                    i = rand() % 10;
107.                    tromino_tile[x + half][y + half - 1] = i;
108.                    tromino_tile[x + half][y + half] = i;
109.                    tromino_tile[x + half - 1][y + half - 1] = i;
110.
111.                    trominoTile(half, hole_row, hole_col, x, y + half);
112.                    trominoTile(half, x + half, y + half - 1, x + half, y);
113.                    trominoTile(half, x + half, y + half, x + half, y + half);
114.                    trominoTile(half, x + half - 1, y + half - 1, x, y);
115.                } else if(hole_row >= x + half && hole_col < y + half) {
116.                    i = rand() % 10;
117.                    tromino_tile[x + half - 1][y + half] = i;
118.                    tromino_tile[x + half][y + half] = i;
119.                    tromino_tile[x + half - 1][y + half - 1] = i;
120.
121.                    trominoTile(half, hole_row, hole_col, x + half, y);
122.                    trominoTile(half, x + half - 1, y + half, x, y + half);
123.                    trominoTile(half, x + half, y + half, x + half, y + half);
124.                    trominoTile(half, x + half - 1, y + half - 1, x, y);
125.                } else {
126.                    i = rand() % 10;
127.                    tromino_tile[x + half - 1][y + half] = i;
128.                    tromino_tile[x + half][y + half - 1] = i;
129.                    tromino_tile[x + half - 1][y + half - 1] = i;
130.
131.                    trominoTile(half, hole_row, hole_col, x + half, y + half);
132.                    trominoTile(half, x + half - 1, y + half, x, y + half);
133.                    trominoTile(half, x + half, y + half - 1, x + half, y);
134.                    trominoTile(half, x + half - 1, y + half - 1, x, y);
135.                }
136.            }
137.        }
138.
139.        int main()
140.        {
141.            int k = 2, hole_row = 2, hole_col = 2;
142.            srand(time(NULL));
143.            if (k < 1)
144.                cout<<"\n<nilai k> harus positive\n";
145.            else {
146.                if(hole_row == 0 || hole_col == 0)
147.                    cout<<"\nThe <jumlah baris> dan <jumlah kolom> harus positive da
    n integer\n";
148.                else {
149.                    int n = power_2(k);
150.
151.                    if (n >= hole_row && n >= hole_col) {
152.                    hole_row--;
153.                    hole_col--;
154.
155.                    allocate(n, hole_row, hole_col);
156.
157.                    trominoTile(n, hole_row, hole_col, 0, 0);
158.                    print_tromino(n);
```

```
                    free_memory(n);
                    }
                    else
                        cout<<"\nBaris dan Kolom tidak boleh lebih dari 2^k\n";
                }
            }
        }
```

Jawaban Studi Kasus 7.2

Rekurensi algoritma Tilling Problem menggunakan metode master
$T(n) = 4T(n/2) + c$
$T(n) = 4T(n/2) + 1$

$a = 4$, $b = 2$, $f(n) = 1$
$n^{\log_b a} = n^{\log_2 4}$
$f(n) = 1 = O(n^{\log_2 4-\varepsilon})$ untuk $\varepsilon = 2$

Case 1 applies, $T(n) = O(n^2)$