

LAPORAN PRAKTIKUM 5
ANALISIS ALGORITMA



Disusun oleh :
Muhammad Zulfikar Ali
140810180064

PROGRAM STUDI S-1 TEKNIK INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS PADJADJARAN
2020

Studi Kasus 5

```
1. /*
Nama      : Muhammad Zulfikar Ali
NPM       : 140810180064
Kelas    : B
*/

#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; // Initialize the minimum distance as 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 there are 2 or 3 points, then use brute force
    if (n <= 3)
        return bruteForce(P, n);

    // Find the middle point
    int mid = n/2;
    Point midPoint = P[mid];

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

    // Find the smaller of two distances
    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);
}

// Driver code
int main(){
    Point P[] = {{4, 1}, {5, 10}, {44, 69}};
    int n = sizeof(P) / sizeof(P[0]);
    cout << "The smallest distance is " << closest(P, n);
    return 0;
}

```

```
"C:\Users\asus\Documents\College\Sem 4\prak analgo\AnalgoKu\AnalgoKu5\closest pair.exe"
The smallest distance is 9.05539
Process returned 0 (0x0)   execution time : 0.424 s
Press any key to continue.
-
```

2. Kita asumsikan bahwa kita menggunakan algoritma pengurutan $O(n \log n)$. Algoritma di atas membagi semua titik dalam dua set dan secara rekursif memanggil dua set. Setelah membelah, ia menemukan strip dalam waktu $O(n)$, mengurutkan strip dalam waktu $O(n \log n)$ dan akhirnya menemukan titik terdekat dalam strip dalam waktu $O(n)$.

Jadi $T(n)$ dapat dinyatakan sebagai berikut :

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

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

$$T(n) = T(n \times \log n \times \log n)$$

Catatan

- Kompleksitas waktu dapat ditingkatkan menjadi $O(n \log n)$ dengan mengoptimalkan langkah 5 dari algoritma di atas.
- Kode menemukan jarak terkecil. Dapat dengan mudah dimodifikasi untuk menemukan titik dengan jarak terkecil.
- Kode ini menggunakan pengurutan cepat yang bisa $O(n^2)$ dalam kasus terburuk. Untuk memiliki batas atas sebagai $O(n (\log n)^2)$, algoritma pengurutan $O(n \log n)$ seperti pengurutan gabungan atau pengurutan tumpukan dapat digunakan

Studi Kasus 6

1.

```
/*
Nama      : Muhammad Zulfikar Ali
NPM       : 140810180064
Kelas    : B
*/

#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;

        // boolean expression for 3-bit addition
        carry = (firstBit&secondBit) | (secondBit&carry) | (firstBit&carry);
    }

    // if overflow, then add a leading 1
    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);

```

```

//kasus dasar
if (n == 0) return 0;
if (n == 1) return multiplySingleBit(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("0011", "1110"));
    printf ("%ld\n", multiply("1011", "0111"));
    printf ("%ld\n", multiply("0010", "1010"));
    printf ("%ld\n", multiply("1111", "1010"));
    printf ("%ld\n", multiply("0001", "1010"));
    printf ("%ld\n", multiply("1100", "1010"));
    printf ("%ld\n", multiply("1100", "1101"));
}

```

```

"C:\Users\asus\Documents\College\Sem 4\prak analgo\AnalgoKu\AnalgoKu5\karatsuba.exe"
42
77
20
150
10
120
156

Process returned 0 (0x0)   execution time : 1.085 s
Press any key to continue.

```

2.
 - Let's try divide and conquer.
 - Divide each number into two halves.
 - $x = x_H r^{n/2} + x_L$
 - $y = y_H r^{n/2} + y_L$
 - Then:

$$\begin{aligned}
 xy &= (x_H r^{n/2} + x_L) (y_H r^{n/2} + y_L) \\
 &= x_H y_H r^n + (x_H y_L + x_L y_H) r^{n/2} + x_L y_L
 \end{aligned}$$
 - Runtime?
 - $T(n) = 4 T(n/2) + O(n)$
 - $T(n) = O(n^2)$
 - Instead of 4 subproblems, we only need 3 (with the help of clever insight).
 - Three subproblems:
 - $a = x_H y_H$
 - $d = x_L y_L$
 - $e = (x_H + x_L) (y_H + y_L) - a - d$
 - Then $xy = a r^n + e r^{n/2} + d$
 - $T(n) = 3 T(n/2) + O(n)$
 - $T(n) = O(n^{\log_2 3}) = O(n^{1.584...})$

Studi Kasus 7

1.

```

/*
Nama      : Muhammad Zulfikar Ali
NPM       : 140810180064
Kelas    : B
*/

#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];

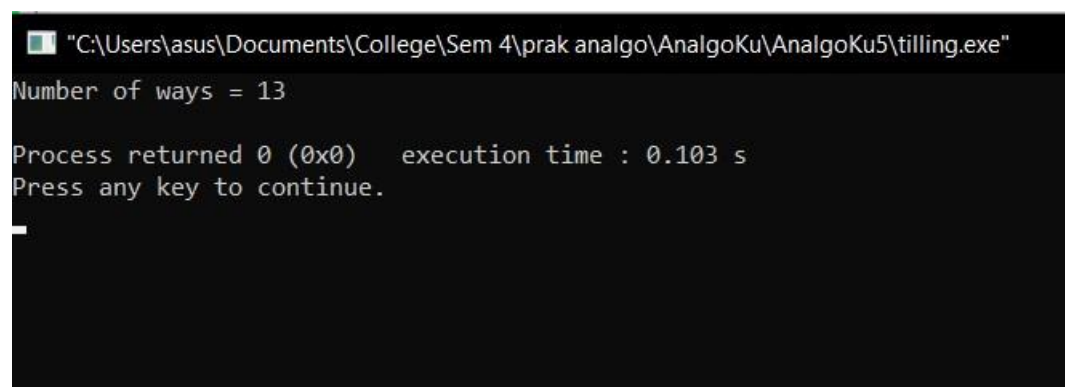
        // base cases
        else if (i < m)
            count[i] = 1;

        // i == m
        else
            count[i] = 2;
    }

    // required number of ways
    return count[n];
}

// Driver program to test above
int main()
{
    int n = 8, m = 3;
    cout << "Number of ways = " << countWays(n, m);
    cout << "\n";
    return 0;
}

```



```

C:\Users\asus\Documents\College\Sem 4\prak analgo\AnalgoKu\AnalgoKu5\tilling.exe
Number of ways = 13

Process returned 0 (0x0)   execution time : 0.103 s
Press any key to continue.

```

// n adalah ukuran kotak yang diberikan, p adalah lokasi sel yang hilang

Tile (int n, Point p)

1) Kasus dasar: $n = 2$, A 2×2 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 $n/2 \times n/2$ memiliki sel yang hilang (sel yang tidak perlu diisi). Lihat gambar 2 di bawah ini.

3) Memecahkan masalah secara rekursif untuk mengikuti empat. Biarkan p_1 , p_2 , p_3 dan p_4 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 ($n / 2, p3$)

2. 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 $O(n^2)$

Bagaimana cara kerjanya?

Pengerjaan algoritma Divide and Conquer dapat dibuktikan menggunakan Mathematical Induction. Biarkan kuadrat input berukuran $2k \times 2k$ di mana $k \geq 1$.

Kasus Dasar: Kita tahu bahwa masalahnya dapat diselesaikan untuk $k = 1$. Kami memiliki 2×2 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 subsquare dengan dimensi $2^{k-1} \times 2^{k-1}$ seperti yang ditunjukkan pada gambar 2 di atas. Jadi jika dapat menyelesaikan 4 subkuarses, dapat menyelesaikan kuadrat lengkap.