LAPORAN PRAKTIKUM

ANALISIS ALGORITMA

Alfari Sidnan Ghilmana

140810180011

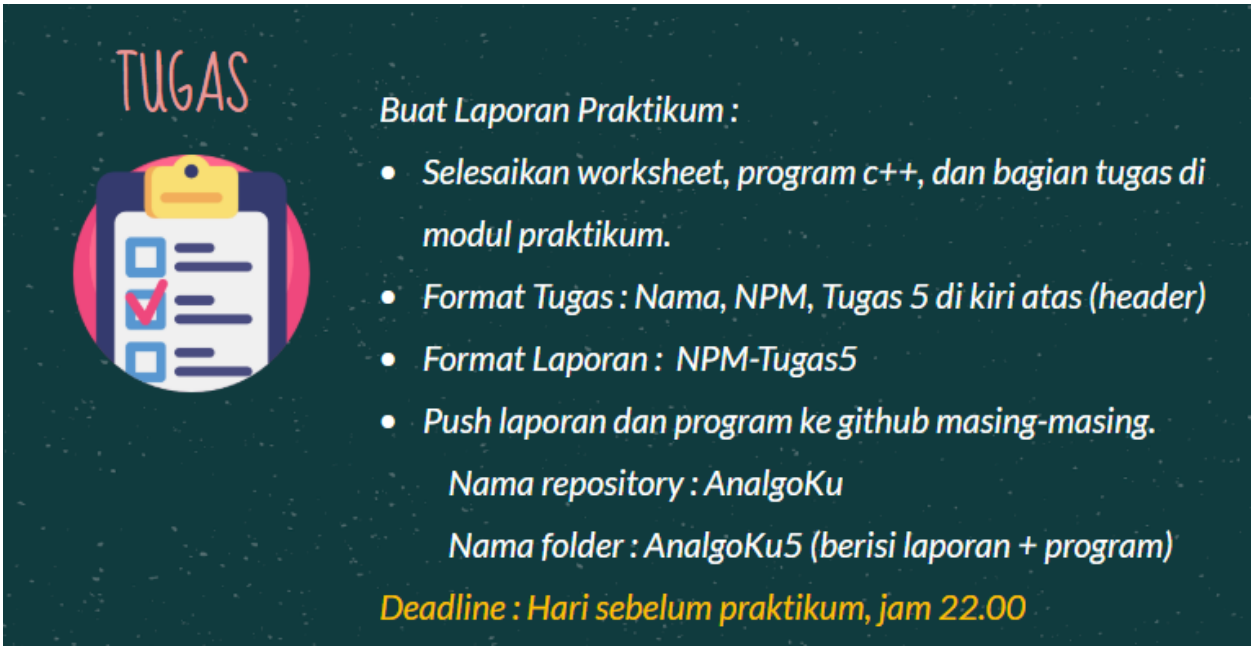
Kelas A

Program Studi S-1 Teknik Informatika

Departemen Ilmu Komputer

Fakultas Matematika dan Ilmu Pengetahuan Alam

Universitas Padjadjaran



Worksheet 5

1.

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)

a.

Sourch code :

```
// A divide and conquer program in C++
// to find the smallest distance from a
// given set of points.

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

// A structure to represent a Point in 2D plane
class Point
{
    public:
    int x, y;
};
```

```
/* Following two functions are needed for library function qsort().
Refer: http://www.cplusplus.com/reference/clibrary/cstdlib/qsort/ */

// Needed to sort array of points
// according to X coordinate
int compareX(const void* a, const void* b)
{
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->x - p2->x);
}

// Needed to sort array of points according to Y coordinate
int compareY(const void* a, const void* b)
{
    Point *p1 = (Point *)a, *p2 = (Point *)b;
    return (p1->y - p2->y);
}

// A utility function to find the
// distance between two points
float dist(Point p1, Point p2)
{
    return sqrt( (p1.x - p2.x)*(p1.x - p2.x) +
                 (p1.y - p2.y)*(p1.y - p2.y)
            );
}

// A Brute Force method to return the
// smallest distance between two points
// in P[] of size n
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;
}

// A utility function to find
// minimum of two float values
float min(float x, float y)
{
    return (x < y)? x : y;
```

```
}


// A utility function to find the
// distance beween the closest points of
// strip of given size. All points in
// strip[] are sorted accordint to
// y coordinate. They all have an upper
// bound on minimum distance as d.
// Note that this method seems to be
// a O(n^2) method, but it's a O(n)
// method as the inner loop runs at most 6 times
float stripClosest(Point strip[], int size, float d)
{
    float min = d; // Initialize the minimum distance as d

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

    // Pick all points one by one and try the next points till the difference
    // between y coordinates is smaller than d.
    // This is a proven fact that this loop runs at most 6 times
    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;
}

// A recursive function to find the
// smallest distance. The array P contains
// all points sorted according to x coordinate
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];

    // Consider the vertical line passing
    // through the middle point calculate
    // the smallest distance dl on left
```

```
    // of middle point and dr on right side
    float dl = closestUtil(P, mid);
    float dr = closestUtil(P + mid, n - mid);

    // Find the smaller of two distances
    float d = min(dl, dr);

    // Build an array strip[] that contains
    // points close (closer than d)
    // to the line passing through the middle point
    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++;

    // Find the closest points in strip.
    // Return the minimum of d and closest
    // distance is strip[]
    return min(d, stripClosest(strip, j, d) );
}

// The main function that finds the smallest distance
// This method mainly uses closestUtil()
float closest(Point P[], int n)
{
    qsort(P, n, sizeof(Point), compareX);

    // Use recursive function closestUtil()
    // to find the smallest distance
    return closestUtil(P, n);
}

// Driver code
int main()
{
    Point P[] = {{2, 3}, {12, 30}, {40, 50}, {5, 1}, {12, 10}, {3, 4}};
    int n = sizeof(P) / sizeof(P[0]);
    cout << "The smallest distance is " << closest(P, n);
    return 0;
}

// This is code is contributed by rathbhupendra
```

Screenshot :

```
The smallest distance is 1.41421
PS D:\Kuliah\SMT 4\Analisis Algoritma\Praktikum Analisis Algoritma\AnalgoKu\AnalgoKu5>
```

b. Kompleksitas Waktu

Biarkan kompleksitas waktu dari algoritma di atas menjadi T (n). Mari kita asumsikan bahwa kita menggunakan algoritma pengurutan O (nLogn). 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 (nLogn) 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 (nLogn) + O (n)

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

T (n) = T (n x Logn x Logn)

2.

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)

a.

Sourch code :

```
// C++ implementation of Karatsuba algorithm for bit string multiplication.
#include<iostream>
#include<stdio.h>

using namespace std;

// FOLLOWING TWO FUNCTIONS ARE COPIED FROM http://goo.gl/q0OhZ
// Helper method: given two unequal sized bit strings, converts them to
// same length by adding leading 0s in the smaller string. Returns the
// the new length
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
}

// The main function that adds two bit sequences and returns the addition
string addBitStrings( string first, string second )
{
    string result;  // To store the sum bits

    // make the lengths same before adding
    int length = makeEqualLength(first, second);
    int carry = 0;  // Initialize carry

    // Add all bits one by one
    for (int i = length-1 ; i >= 0 ; i--)
    {
        int firstBit = first.at(i) - '0';
        int secondBit = second.at(i) - '0';

        // boolean expression for sum of 3 bits
        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;
}

// A utility function to multiply single bits of strings a and b
int multiplyiSingleBit(string a, string b)
{  return (a[0] - '0')*(b[0] - '0');  }
```

```
// The main function that multiplies two bit strings X and Y and returns
// result as long integer
long int multiply(string X, string Y)
{
    // Find the maximum of lengths of x and Y and make length
    // of smaller string same as that of larger string
    int n = makeEqualLength(X, Y);

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

    int fh = n/2;   // First half of string, floor(n/2)
    int sh = (n-fh); // Second half of string, ceil(n/2)

    // Find the first half and second half of first string.
    // Refer http://goo.gl/lLmgn for substr method
    string Xl = X.substr(0, fh);
    string Xr = X.substr(fh, sh);

    // Find the first half and second half of second string
    string Yl = Y.substr(0, fh);
    string Yr = Y.substr(fh, sh);

    // Recursively calculate the three products of inputs of size n/2
    long int P1 = multiply(Xl, Yl);
    long int P2 = multiply(Xr, Yr);
    long int P3 = multiply(addBitStrings(Xl, Xr), addBitStrings(Yl, Yr));

    // Combine the three products to get the final result.
    return P1*(1<<(2*sh)) + (P3 - P1 - P2)*(1<<sh) + P2;
}

// Driver program to test above functions
int main()
{
    printf ("%ld\n", multiply("1100", "1010"));
    printf ("%ld\n", multiply("110", "1010"));
    printf ("%ld\n", multiply("11", "1010"));
    printf ("%ld\n", multiply("1", "1010"));
    printf ("%ld\n", multiply("0", "1010"));
    printf ("%ld\n", multiply("111", "111"));
    printf ("%ld\n", multiply("11", "11"));
}
```

Screenshot :

```
120
60
30
10
0
49
9
PS D:\Kuliah\SMT 4\Analisis Algoritma\Praktikum Analisis Algoritma\AnalgoKu\AnalgoKu5>
```

b.

- 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:

    $xy = (x_H r^{n/2} + x_L)\, y_H r^{n/2} + y_L$

    $= x_H y_H r^n + (x_H y_L + x_L y_H) r^{n/2} + x_L y_L$
  - 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 $xy = a\, r^n + e\, r^{n/2} + d$
- $T(n) = 3\, T(n/2) + O(n)$
- $T(n) = O(n^{\log 3}) = O(n^{1.584\ldots})$

3.

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

a.

Sourch code :

```
#include <iostream>
using namespace std;

int countWays(int n, int m)  {
     int count[n + 1];
     count[0] = 0;

     for (int i = 1; i <= n; i++) {
         // recurrence relation
          if (i > m) {
               count[i] = count[i - 1] + count[i - m];
          }
          else if (i < m){
               count[i] = 1;
          }
          else{
               count[i] = 2;
          }
     }

    return count[n];
}

int main()  {
    int n = 4, m = 2;
    cout << "Number of ways = " << countWays(n, m);
    return 0;
}
```

Screenshot :

```
Number of ways = 5
PS D:\Kuliah\SMT 4\Analisis Algoritma\Praktikum Analisis Algoritma\AnalgoKu>
```

b. 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 (n2)

   Bagaimana cara kerjanya?

Pengerjaan algoritma Divide and Conquer dapat dibuktikan menggunakan Mathematical Induction. Biarkan kuadrat input berukuran 2k x 2k di mana k> = 1.
Kasus Dasar: Kita tahu bahwa masalahnya dapat diselesaikan untuk k = 1. Kami memiliki 2 x 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 subsqure dengan dimensi 2k-1 x 2k-1 seperti yang ditunjukkan pada gambar 2 di atas. Jadi jika dapat menyelesaikan 4 subskuares, dapat menyelesaikan kuadrat lengkap.

Nama : Alfari Sidnan G
NPM : 1408101800111
Kelas : A

Worksheet 3

1.) $T(n) = 2+4+8+16+\dots+2^n$

Deret Geometri $= \frac{a(r^n-1)}{r-1} = \frac{2(2^n-1)}{2-1}$

$= 2(2^n-1)$

$= 2^{n+1}-2$

Notasi big $O(n) \rightarrow O(2^n)$

$T(n) \leq c.f(n)$

$2^{n+1}-2 \leq c.2^n$ misal $n_0 = 1$

$\frac{2^{n+1}}{2^n} - \frac{2}{2^n} \leq c$ $2 - \frac{2}{2} \leq c$

$2 - \frac{2}{2^n} \leq c$ $c \geq 1$

2.) Buktikan bahwa utk konstanta positif p, q, dan r : $T(n) = pn^2 + qn + r$ adalah $O(n^2)$, $\Omega(n^2)$, $\Theta(n^2)$

→ Big $O(n^2)$

$T(n) \leq c.f(n)$

$pn^2 + qn + r \leq c.n^2$

$\frac{pn^2}{n^2} + \frac{qn}{n^2} + \frac{r}{n^2} \leq \frac{c.n^2}{n^2}$

$p + \frac{q}{n} + \frac{r}{n^2} \leq c$

misalkan $n_0 = 1$

misalkan $p = q = r = 1$

$1 + \frac{1}{1} + \frac{1}{1} \leq c$

$c \geq 3$ .. Terbukti / benar

→ Big $\Omega(\Omega(n^2))$

$T(n) \geq c.f(n)$

$pn^2 + qn + r \geq c.n^2$

$\frac{pn^2}{n^2} + \frac{qn}{n^2} + \frac{r}{n^2} \geq c$

$p + \frac{q}{n} + \frac{r}{n^2} \geq c$

misalkan $n_0 = 1$

$p + q + r \geq c$

misalkan $p = 1, q = 1$, $r = 1$

$1+1+1 \geq c$

$c \leq 3$ .. terbukti / benar

→ Big Θ

karena $O(n^2)$ dan $\Omega(n^2)$ terbukti dan berderajat sama maka $\Theta(n^2)$ terbukti benar

3.) Kompleksitas waktu

Operasi Assigment

$w_{ij} \leftarrow w_{ik}$ or w_{ik} and w_{kj} berulang sebanyak

n kali di loop "for j ← i to n do" serta

n kali di loop "for i ← 1 to n do" dan

n kali di loop "for k ← 1 to n do" maka

$T(n) = n.n.n = n^3$

→ Big $O()$ $O(n^3)$

$n^3 \leq c.n^3$

$\frac{n^3}{n^3} \leq c$

$1 \leq c$

$c \geq 1$

→ Big Ω

$n^3 \geq c\, n^3$

$1 \geq c$

$c \leq 1$

→ Big Θ : $\Theta(n^3)$

karena $O(n^3)$ dan $\Omega(n^3)$ berderajat sama maka $\Theta(n^3)$

4.) Algoritma menjumlahkan dua matriks

```
for i ← 1 to n do
  for j ← 1 for n do
    mij ← aij + bij
  end for
endfor
```

$T(n) = n^2$

→ $O(n^2)$: $n^2 \leq cn^2$, $c \geq 1$

→ $\Omega(n^2)$: $n^2 \geq cn^2$, $c \leq 1$

→ $\Theta(n^2)$: $O(n^2)$ dan $\Omega(n^2)$ berderajat sama maka $\Theta(n^2)$

5.) Algoritma menyalons larik

```
for i ← 1 to n do
  ai ← bi
endfor
```

$T(n) = n$

→ $O(n)$: $n \leq c.n$, $c \geq 1$

→ $\Omega(n)$: $n \geq c.n$, $c \leq 1$

→ $\Theta(n)$: $O(n)$ dan $\Omega(n)$ derajat sama maka $\Theta(n)$

a. Jumlah operasi perbandingan

$T(n) = (n-1) + (n-2) + (n-3) + \dots + 1$

$= \frac{n(n-1)}{2} = \frac{n^2-n}{2}$

b.) maksimum perluasan terjadi ketika $\frac{n(n-1)}{2}$

c.) kompleksitas waktu

-> Best case

Perbandingan $\rightarrow \frac{n(n-1)}{2}$ kali

$T_{min}(n) = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$

-> Worst case

Perbandingan $\rightarrow \frac{n(n-1)}{2}$ kali

Assigment $\rightarrow \frac{3n(n-1)}{2}$ kali

$T_{max}(n) = \frac{n(n-1)}{2} + \frac{3n(n-1)}{2} = \frac{4n(n-1)}{2} = 2n^2 - 2$

-> $O(n^2)$

$2n^2 - 2 \leq c.n^2$

$2 - \frac{2}{n^2} \leq c, n_0 = 1$

$c \geq 0$

$\Omega(n^2)$
$2n^2 - 2 \geq c.n^2$
$2 - \frac{2}{n^2} \geq c, n_0 = 1$
$c \leq 0$

-> $\Theta(n^2)$

$O(n^2)$ dan $\Omega(n^2)$ berderajat sama

7.) a.) Algoritma A -> $O(\log N)$

b.) Algoritma B -> $O(\log N)$

c.) Algoritma C -> $O(n^2)$

$N = 8$ maka

algoritma A -> $O(\log 8) = O(3 \log 2)$

algoritma B -> $O(8 \log 8) = O(24 \cdot \log 2)$

algoritma C -> $O(8^2) = O(64)$

Algoritma A lebih cepat dan no 1 yg lainya

8.) Operasi Assigment

-> $bn \leftarrow an$ - 1 kali

-> $hu \leftarrow ak + bu \times$: n kali

$T(n) = n+1$

$O(n)$ untuk P'

Algoritma P

Pertambahan : n kali

Perkalian : n kali

$T(n) = 2n$

maka algoritma P' lebih baik dari pada P