LAPORAN PRAKTIKUM 5
ANALISIS ALGORITMA

DISUSUN OLEH:
NAMA :Putri Nabila
NPM :140810180007

Program Studi S-1 Teknik Informatika
Departemen Ilmu Komputer
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Padjadjaran
2020

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

Tugas:

1) Buatlah program untuk menyelesaikan problem closest pair of points menggunakan algoritma divide & conquer yang diberikan. Gunakan bahasa C++

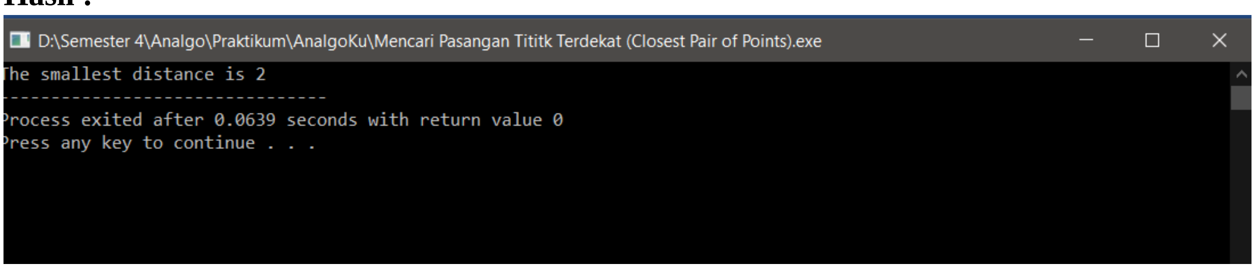
   Program :

```
/*
Nama    :Putri Nabila
NPM     : 140810180007
Kelas   : A
eskripsi : Mencari Pasangan Tititk Terdekat (Closest Pair of Points)
*/


#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;

	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);
}


int main()
{
	Point P[] = {{5, 6}, {22, 30}, {45, 55}, {1, 5}, {3, 6}};
	int n = sizeof(P) / sizeof(P[0]);
	cout << "The smallest distance is " << closest(P, n);
	return 0;
}
```

Hasil :



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

Jawab :

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)

Catatan :

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

Studi Kasus 6: Algoritma Karatsuba untuk Perkalian Cepat

Tugas:

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

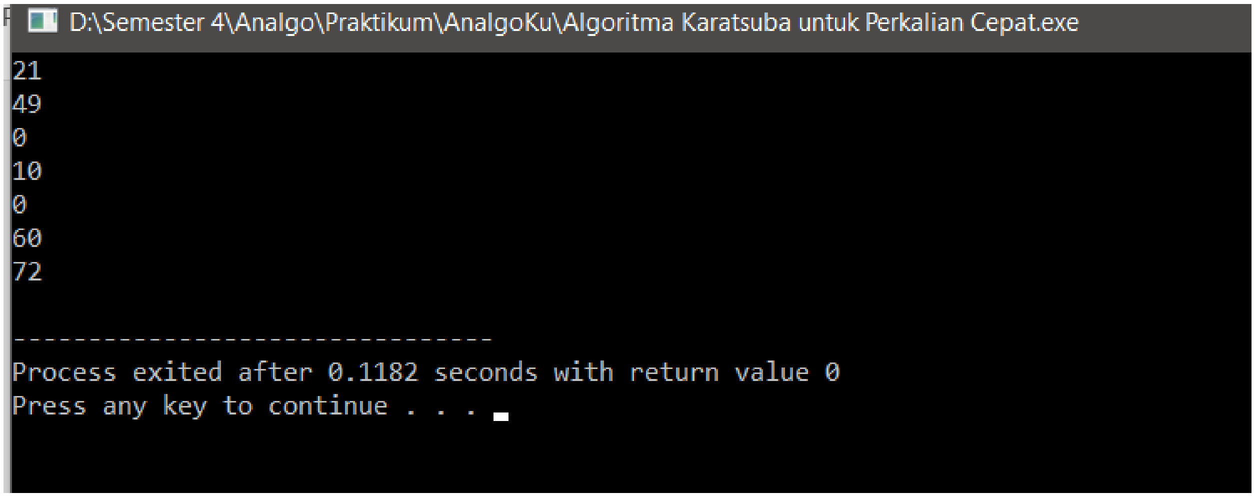
   Program :

```
/*
Nama    :Putri Nabila
NPM     : 140810180007
Kelas   : A
Deskripsi :Algoritma Karatsuba untuk Perkalian Cepat
*/

#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 multiplyiSingleBit(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 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()
{
	printf ("%ld\n", multiply("11", "111"));
	printf ("%ld\n", multiply("111", "111"));
	printf ("%ld\n", multiply("0", "1010"));
	printf ("%ld\n", multiply("1", "1010"));
	printf ("%ld\n", multiply("0", "1010"));
	printf ("%ld\n", multiply("110", "1010"));
	printf ("%ld\n", multiply("1100", "110"));
}
```

Hasil :



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)

Jawab :

- 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?
    - $T(n) = 4\,T(n/2) + O(n)$
    - $T(n) = O(n\text{^}2)$
- 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 $xy = a\, r^n + e\, r^{n/2} + d$
- $T(n) = 3\,T(n/2) + O(n)$
- $T(n) = O(n^{\log 3}) = O(n^{1.584\ldots})$

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

Tugas:

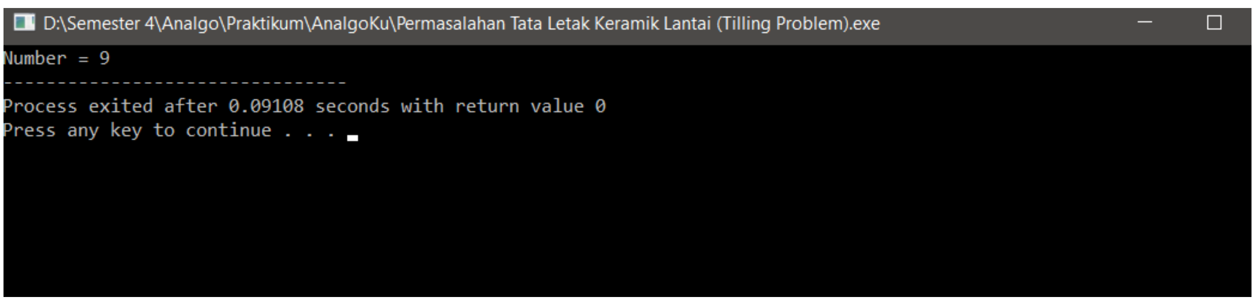
1) Buatlah program untuk menyelesaikan problem tilling menggunakan algoritma divide & conquer yang diberikan. Gunakan bahasa C++

   Program :

```
/*
Nama    :Putri Nabila
NPM     : 140810180007
Kelas   : A
Deskripsi :Permasalahan Tata Letak Keramik Lantai (Tilling Problem)
*/
#include <bits/stdc++.h>

using namespace std;



int countWays(int n, int m)
{



    int count[n + 1];
    count[0] = 0;
    for (int i = 1; i <= n; i++) {
        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 = 7, m = 3;
    cout << "Number = "
         << countWays(n, m);
    return 0;
}
```

Hasil :



1) 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

   Jawab :

   Kompleksitas Waktu:

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

   Berdasarkan rumus dasar yaitu T (n) = 4T (n / 2) + C , menjadikan sebuah relasi perulangan untuk algoritma yang bisa diselesaikan dengan metode Master dan Kompleksitas waktunya adalah O(n2). Dan 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.

Apabila 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, apabila kita dapat menyelesaikan 4 subskuares Maka dapat menyelesaikan kuadrat lengkap.