1. **Studi Kasus 5: Mencari Pasangan Titik Terdekat *(Closest Pait of Points)***
    a. Buatlah program untuk menyelesaikan problem closest pair of points menggunakan algoritma divide & conquer yang diberikan. Gunakan bahasa C++.

```
/*
NAMA    : SURIADI VAJRAKARNA
NPM     : 140810180038
KELAS   : B
TANGGAL : 30 MARET 2020
STUDI KASUS 5 - PRAKTIKUM DESAIN DAN ANALISIS ALGORITMA
*/


#include <iostream>
#include <cfloat>
#include <cstdlib>
#include <cmath>
using namespace std;
struct poi
{
    double poi1, poi2;
};
inline int Comp_poi1(const void *x, const void *b)
{
    poi *p1 = (poi *)x, *pnt2 = (poi *)b;
    return (p1->poi1 - pnt2->poi1);
}
inline int Comp_poi2(const void *x, const void *y)
{
    poi *pnt1 = (poi *)x, *pnt2 = (poi *)y;
    return (pnt1->poi2 - pnt2->poi2);
}
inline double Distance(poi pnt1, poi pnt2)
{
    return sqrt((pnt1.poi1 - pnt2.poi1) * (pnt1.poi1 - pnt2.poi1) +
                (pnt1.poi2 - pnt2.poi2) * (pnt1.poi2 - pnt2.poi2));
```

```
}
double S_Distance(poi P[], int n, poi &pnt1, poi &pnt2)
{
    double min = DBL_MAX;
    for (int i = 0; i < n; ++i)
        for (int j = i + 1; j < n; ++j)
            if (Distance(P[i], P[j]) < min)
            {
                min = Distance(P[i], P[j]);
                pnt1.poi1 = P[i].poi1, pnt1.poi2 = P[i].poi2;
                pnt2.poi1 = P[j].poi1, pnt2.poi2 = P[j].poi2;
            }
    return min;
}
inline double Minimum(double poi1, double poi2)
{
    return (poi1 < poi2) ? poi1 : poi2;
}
double Closest_dist_Spoint(poi stp[], int s, double dist, poi &pnt1, poi &pnt2)
{
    double Minimum = dist;
    qsort(stp, s, sizeof(poi), Comp_poi2);
    for (int i = 0; i < s; ++i)
        for (int j = i + 1; j < s && (stp[j].poi2 - stp[i].poi2) < Minimum; ++j)
            if (Distance(stp[i], stp[j]) < Minimum)
            {
                Minimum = Distance(stp[i], stp[j]);
                pnt1.poi1 = stp[i].poi1, pnt1.poi2 = stp[i].poi2;
                pnt2.poi1 = stp[j].poi1, pnt2.poi2 = stp[j].poi2;
            }
    return Minimum;
}
double Closest_dist(poi P[], poi stp[], int n, poi &pnt1, poi &pnt2)
{
    static poi pt1, pt2, pt3, pt4;
```

```
    if (n <= 3)
        return S_Distance(P, n, pt1, pt2);
    int medium = n / 2;
    poi mediumPoint = P[medium];
    double D_Left = Closest_dist(P, stp, medium, pt1, pt2);
    double D_Right = Closest_dist(P + medium, stp, n - medium, pt3, pt4);
    if (D_Left < D_Right)
    {
        pnt1.poi1 = pt1.poi1;
        pnt1.poi2 = pt1.poi2;
        pnt2.poi1 = pt2.poi1;
        pnt2.poi2 = pt2.poi2;
    }
    else
    {
        pnt1.poi1 = pt3.poi1;
        pnt1.poi2 = pt3.poi2;
        pnt2.poi1 = pt4.poi1;
        pnt2.poi2 = pt4.poi2;
    }
    double min_dist = Minimum(D_Left, D_Right);
    int j = 0;
    for (int i = 0; i < n; i++)
        if (abs(P[i].poi1 - mediumPoint.poi1) < min_dist)
            stp[j++] = P[i];
    double min_dist_strip = Closest_dist_Spoint(stp, j, min_dist, pt1, pt2);
    double F_Min = min_dist;
    if (min_dist_strip < min_dist)
    {
        pnt1.poi1 = pt1.poi1;
        pnt1.poi2 = pt1.poi2;
        pnt2.poi1 = pt2.poi1;
        pnt2.poi2 = pt2.poi2;
        F_Min = min_dist_strip;
    }
```

```
    return F_Min;
}
int main()
{
    poi P[] = {{4, 1}, {15, 20}, {30, 40}, {8, 4}, {13, 11}, {5, 6}};
    poi pnt1 = {DBL_MAX, DBL_MAX}, pnt2 = {DBL_MAX, DBL_MAX};
    int n = sizeof(P) / sizeof(P[0]);
    qsort(P, n, sizeof(poi), Comp_poi1);
    poi *stp = new poi[n];
    cout << "The closest distance of point in array is: " << Closest_dist(P, stp,
 n, pnt1, pnt2) << endl;
    cout << "The closest pair of point in array: (" << pnt1.poi1 << "," << pnt1.p
oi2 << ") and ("
         << pnt2.poi1 << "," << pnt2.poi2 << ")" << endl;
    delete[] stp;
    return 0;
}
```

```
PS D:\MEGA\SEMESTER 4\ANALGO\Praktikum\AnalgoKu\AnalgoKu5> g++ closestpair.cpp -o closestpa
ir
PS D:\MEGA\SEMESTER 4\ANALGO\Praktikum\AnalgoKu\AnalgoKu5> ./closestpair
The closest distance of point in array is: 3.60555
The closest pair of point in array: (13,11) and (15,20)
```

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

   Jawab:

   - Rekurensi algoritma tersebut adalah $T(n)=2T(n/2)+O(n)$.
   - Hasil tersebut didapat menggunakan metode master karena $a=b^k$ didapat $O(n \log n)$.

2. **Studi Kasus 6: Algoritma Karatsuba untuk Perkalian Cepat**
    a. Buatlah program untuk menyelesaikan problem fast multiplication menggunakan algoritma divide & conquer yang diberikan (Algoritma Karatsuba). Gunakan bahasa C++.

```
/*
NAMA    : SURIADI VAJRAKARNA
NPM     : 140810180038
KELAS   : B
TANGGAL : 30 MARET 2020
STUDI KASUS 6 - PRAKTIKUM DESAIN DAN ANALISIS ALGORITMA
*/

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

#define max(a, b) ((a) > (b) ? (a) : (b))

string add(string lhs, string rhs)
{
    int length = max(lhs.size(), rhs.size());
    int carry = 0;
    int sum_col;
    string result;

    while (lhs.size() < length)
        lhs.insert(0, "0");

    while (rhs.size() < length)
        rhs.insert(0, "0");

    for (int i = length - 1; i >= 0; i--)
    {
        sum_col = (lhs[i] - '0') + (rhs[i] - '0') + carry;
```

```
        carry = sum_col / 10;
        result.insert(0, to_string(sum_col % 10));
    }

    if (carry)
        result.insert(0, to_string(carry));

    return result.erase(0, min(result.find_first_not_of('0'), result.size() - 1))
;
}

string subtract(string lhs, string rhs)
{
    int length = max(lhs.size(), rhs.size());
    int diff;
    string result;

    while (lhs.size() < length)
        lhs.insert(0, "0");

    while (rhs.size() < length)
        rhs.insert(0, "0");

    for (int i = length - 1; i >= 0; i--)
    {
        diff = (lhs[i] - '0') - (rhs[i] - '0');
        if (diff >= 0)
            result.insert(0, to_string(diff));
        else
        {

            int j = i - 1;
            while (j >= 0)
            {
                lhs[j] = ((lhs[j] - '0') - 1) % 10 + '0';
```

```
                if (lhs[j] != '9')
                    break;
                else
                    j--;
            }
            result.insert(0, to_string(diff + 10));
        }
    }

    return result.erase(0, min(result.find_first_not_of('0'), result.size() - 1))
;
}

string multiply(string lhs, string rhs)
{
    int length = max(lhs.size(), rhs.size());

    while (lhs.size() < length)
        lhs.insert(0, "0");

    while (rhs.size() < length)
        rhs.insert(0, "0");

    if (length == 1)
        return to_string((lhs[0] - '0') * (rhs[0] - '0'));

    string lhs0 = lhs.substr(0, length / 2);
    string lhs1 = lhs.substr(length / 2, length - length / 2);
    string rhs0 = rhs.substr(0, length / 2);
    string rhs1 = rhs.substr(length / 2, length - length / 2);

    string p0 = multiply(lhs0, rhs0);
    string p1 = multiply(lhs1, rhs1);
    string p2 = multiply(add(lhs0, lhs1), add(rhs0, rhs1));
    string p3 = subtract(p2, add(p0, p1));
```

```
    for (int i = 0; i < 2 * (length - length / 2); i++)
        p0.append("0");
    for (int i = 0; i < length - length / 2; i++)
        p3.append("0");

    string result = add(add(p0, p1), p3);

    return result.erase(0, min(result.find_first_not_of('0'), result.size() - 1))
;
}

int main()
{
    string s1, s2;
    cout << "Masukkan Angka Pertama\t\t: "; cin >> s1;
    cout << "Masukkan Angka Kedua\t\t: "; cin >> s2;
    cout << "Hasil Perkalian Kedua Bilangan\t: " << multiply(s1, s2) << endl;
    return 0;
}
```

```
PS D:\MEGA\SEMESTER 4\ANALGO\Praktikum\AnalgoKu\AnalgoKu5> g++ karatsuba.cpp -o karatsuba
PS D:\MEGA\SEMESTER 4\ANALGO\Praktikum\AnalgoKu\AnalgoKu5> ./karatsuba
Masukkan Angka Pertama          : 123456789
Masukkan Angka Kedua            : 987654321
Hasil Perkalian Kedua Bilangan  : 121932631112635269
```

b. 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:

   - Dari persamaan T (n) = 3T (n / 2) + O (n), dengan menggunakan metode master didapat

$$O(n^{\log(3)}) \approx O(n^{1.59})$$

3. **Studi Kasus 7: Permasalahan Tata Letak Keramik Lantai *(Tilling Problem)***
    a. Buatlah program untuk menyelesaikan problem tilling menggunakan algoritma divide & conquer yang diberikan. Gunakan bahasa C++.

```
/*
NAMA    : SURIADI VAJRAKARNA
NPM     : 140810180038
KELAS   : B
TANGGAL : 30 MARET 2020
STUDI KASUS 7 - PRAKTIKUM DESAIN DAN ANALISIS ALGORITMA
*/

#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,m;
    cout << "Masukkan Panjang Kotak\t\t: "; cin >> n;
    cout << "Masukkan Lebar Kotak\t\t: "; cin >> m;
    cout << "Banyak Cara yang dibutuhkan\t: " << countWays(n, m);
    return 0;
}
```

```
PS D:\MEGA\SEMESTER 4\ANALGO\Praktikum\AnalgoKu\AnalgoKu5> g++ tilling.cpp -o tilling
PS D:\MEGA\SEMESTER 4\ANALGO\Praktikum\AnalgoKu\AnalgoKu5> ./tilling
Masukkan Panjang Kotak          : 7
Masukkan Lebar Kotak            : 4
Banyak Cara yang dibutuhkan     : 5
```

b. 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:

- Dengan melihat persamaan T (n) = 4T (n / 2) + C, kita memperoleh a=4, b=2, dan k=1.
- Teorema master menyatakan

$$T(n) \in \Theta(n^{\log_b a}) \text{ jika } a > b^k, \text{ maka } T(n) \in \Theta(n^2).$$

- Maka persamaan tersebut sesuai dengan syarat metode master.