

# Homework 3: Recursion

Bùi Khánh Duy - 20001898

## I. Lập và giải công thức đệ quy xác định độ phức tạp thuật toán

*Exercise 2.4, page 102-105, Anany's book*

### 1. Chọn 2 trong 5 ý của bài 1 để thực hiện (ví dụ 2.4.1.a, 2.4.1.c)

2.4.1 Solve the following recurrence relations.

a)  $x(n) = \begin{cases} 2x(n-3) & \text{for } n \geq 1 \\ 1 & \text{if } n=1 \end{cases}$

$$\begin{aligned} x(n) &= 2x(n-3) \quad (\text{then } x(n-3) = 2x(n-6)) \\ &= 2[2x(n-6)] = 4x(n-6) \quad (\text{then } x(n-6) = 2x(n-9)) \\ &= 8x(n-9) = 2^3 \cdot x(n-3 \cdot 3) \\ &= 2^k \cdot x(n-3 \cdot k) \\ &= 2^{\frac{n}{3}} \cdot x(n-3 \cdot \lfloor \frac{n}{3} \rfloor) \end{aligned}$$

$$\forall i \quad x(1) = 1 \quad (x(0) \text{ is not given}, x(2) = 2).$$

$$\Rightarrow x(n) = 2^{\frac{n}{3}} \cdot O(2^n)$$

(b)  $x(n) = \begin{cases} 2x(n-2) - n & \text{for } n \geq 0 \\ 0 & \text{for } n=0 \end{cases}$

$$\begin{aligned} x(n) &= 2x(n-2) - n \quad (\text{then } x(n-2) = 2x(n-4) - n) \\ &= 2(2x(n-4) - n) - n \\ &= 2 \cdot 2x(n-4) - 3n \quad (\text{then } x(n-4) = 2x(n-6) - n) \end{aligned}$$

$$= 2 \cdot 2 \cdot (2 \times (n-6) - n) - 3n$$

$$= 2 \cdot 2 \cdot 2 \cdot x(n-6) - 7n$$

$$= 2^k \cdot x(n-2^k) - (2^{k+1}-1) \cdot n$$

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

Vì  $x(0) = 0$

$$\Rightarrow x(n) \leq (2^{\frac{n}{2}}-1) \cdot n = O(2^n \cdot n)$$

d).  $x(n) = \begin{cases} x(n/2) - 2 & \text{for } n > 1, n \in 4^k \\ x(1) = 2 & \end{cases}$

$$x(n) = x(n/2) - 2. \quad (\text{thay } x(n/2) = x(n/4) - 2)$$

$$= x(n/4) - 2 - 2 = x(n/4) - 4. \quad (\text{thay } x(n/4) = x(n/8) - 2)$$

$$= x(n/8) - 3 \cdot 2$$

$$= x(n/16) - 4 \cdot 2 = x(n/16) - 8 \cdot 2$$

$$= x(1) - 4 \cdot k$$

$$\Rightarrow x(n) \approx O(\log_4(n))$$

## 2. Chọn 1 trong các bài còn lại (từ 2.4.2 đến 2.4.14)

### 2.4.3

a. Biểu thức độ phức tạp như sau:

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ T(n - 1) + 1 & \text{if } n > 1 \end{cases}$$

Giải:

$$\begin{aligned} T(n) &= T(n - 1) + 1 \\ &= T(n - 2) + 1 + 1 \\ &= T(n - 3) + 1 + 1 + 1 \\ &= \dots \\ &= T(1) + (n - 1) \\ &= 1 + n - 1 \\ &= n \end{aligned}$$

$$\Rightarrow T(n) \approx O(n)$$

b. Với phương pháp không đệ quy, chỉ cần 1 vòng for chạy từ 1 đến n để tính tổng này  
⇒ Độ phức tạp thời gian vẫn là  $O(n)$ , nhưng không cần gọi đến đệ quy.

### 2.4.4

a. Tính giá trị:

$$Q(n) = \begin{cases} 1 & \text{if } n = 1 \\ Q(n - 1) + n/(n * n) & \text{if } n > 1 \end{cases}$$

Giải:

$$\begin{aligned}
Q(n) &= Q(n-1) + n/(n * n) \\
&= Q(n-2) + (n-1)/((n-1) * (n-1)) \\
&= \dots \\
&= Q(1) + \sum_{i=2}^n i/(i * i) \\
&= 1 + \sum_{i=2}^n i/(i * i)
\end{aligned}$$

Vì chuỗi  $\sum_{i=2}^n i/(i * i)$  là chuỗi hội tụ nên khi  $n$  đủ lớn, nên  $Q(n)$  sẽ hội tụ về 1 giá trị nhất định.

b. Đặt số phép nhân là  $M(n)$

$$M(n) = \begin{cases} 0 & \text{if } n = 1 \\ M(n-1) + 1 & \text{if } n > 1 \end{cases}$$

Giải:

$$\begin{aligned}
M(n) &= M(n-1) + 1 \\
&= M(n-2) + 1 + 1 \\
&= M(n-3) + 1 + 1 + 1 \\
&= \dots \\
&= M(1) + (n-1) \\
&= n - 1
\end{aligned}$$

$$\Rightarrow M(n) \approx O(n)$$

c. Vì mỗi lần gọi đệ quy cần dùng ít nhất 1 phép cộng (`else return Q(n - 1) + n/(n * n)`) nên ta có biểu thức về số lượng phép cộng/trừ như sau:

$$A(n) = \begin{cases} 0 & \text{if } n = 1 \\ A(n-1) + 1 & \text{if } n > 1 \end{cases}$$

Giải

$$\begin{aligned}
A(n) &= A(n-1) + 1 \\
&= A(n-2) + 1 + 1 \\
&= A(n-3) + 1 + 1 + 1 \\
&= \dots \\
&= A(1) + (n-1) \\
&= n - 1
\end{aligned}$$

$\Rightarrow A(n) \approx O(n)$

## 2.4.6

Pseudo code:

```

function RestrictedHanoi(n, source, intermediate, destination):
    if n == 1:
        moveDisk(source, intermediate)
        moveDisk(intermediate, destination)
    else:
        RestrictedHanoi(n-1, source, destination, intermediate)
        moveDisk(source, intermediate)
        RestrictedHanoi(n-1, destination, source, intermediate)
        moveDisk(intermediate, destination)
        RestrictedHanoi(n-1, source, intermediate ,destination)

function moveDisk(fromPeg,toPeg):
    print("Move disk from " + fromPeg + " to " + toPeg)

```

Gọi  $M(n)$  là biểu thức tính số lượng các bước, ta có công thức sau:

$$M(n) = \begin{cases} 0 & \text{if } n = 0 \\ 2 & \text{if } n = 1 \\ M(n) = 3 * M(n - 1) + 2 & \text{if } n > 1 \end{cases}$$

## II. Lập trình đệ quy

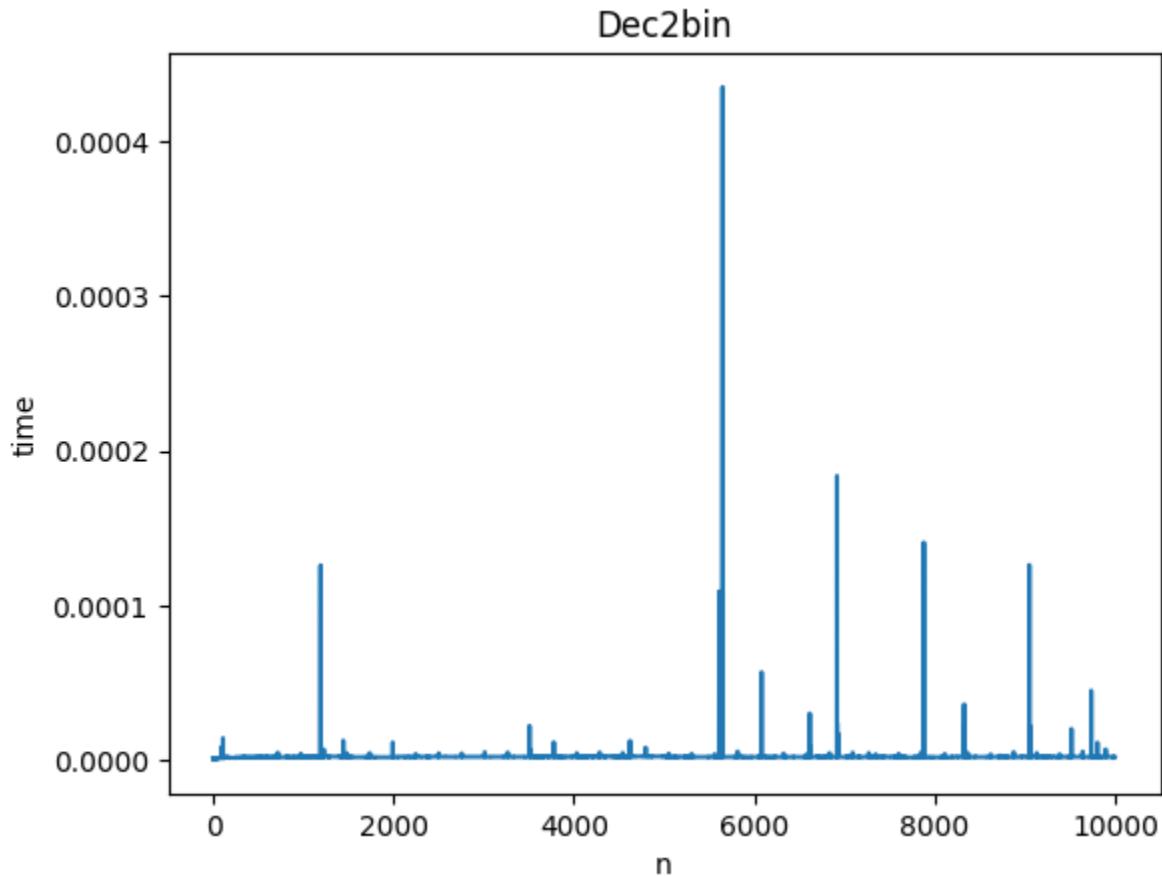
Viết chương trình cho ít nhất 2 trong số 4 bài toán sau:

1. In biểu diễn nhị phân của số nguyên.

```

def dec2bin(x):
    if x == 0:
        return str(0)
    if x == 1:
        return str(1)
    ans = dec2bin(x//2)
    return ans + str(1) if x % 2 == 1 else ans + str(0)
print(dec2bin(20))

```



## 2. Phân tích số nguyên thành tích các thừa số nguyên tố.

```

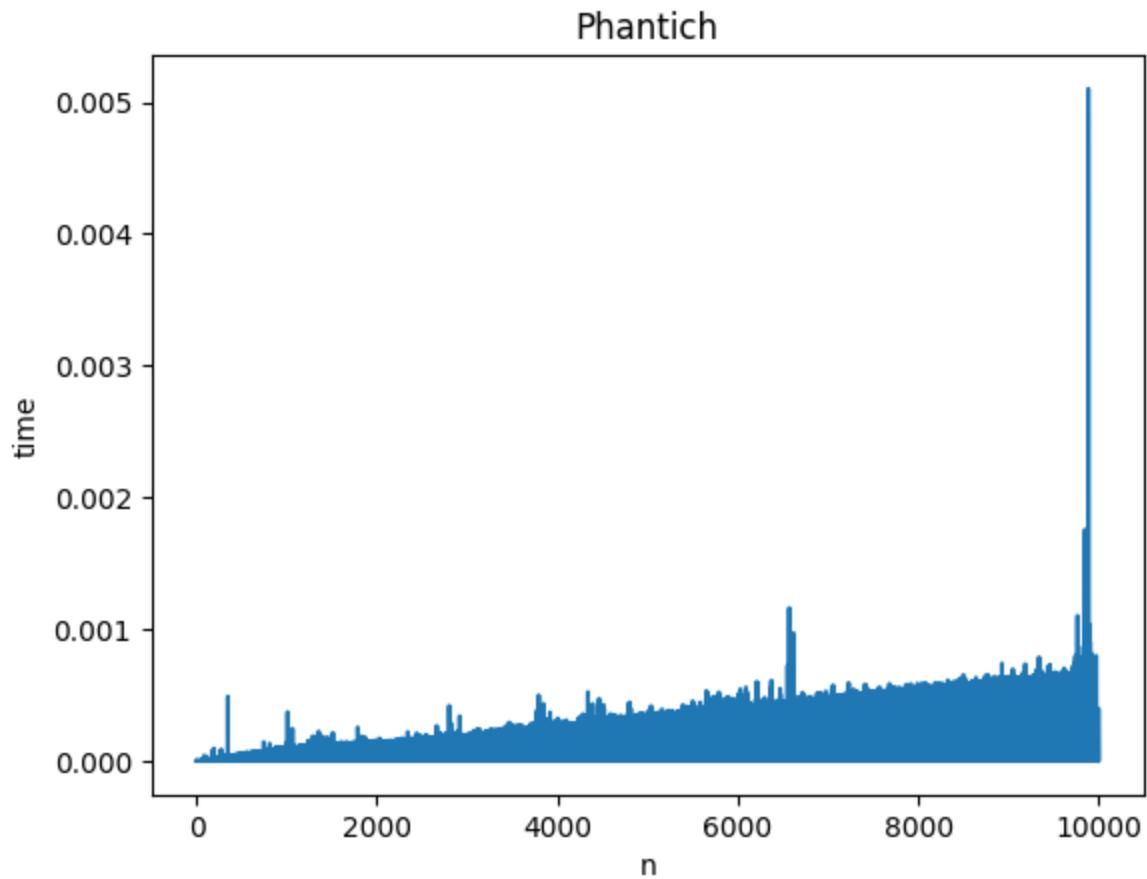
def phantich(x, n):
    if x > n:
        if n != 1: return [n]
        else: return []
    else:
        if n % x == 0:
            while n % x == 0:

```

```

        n //>= x
        return [x] + phantich(x+1, n)
    else:
        return phantich(x+1, n)
print(phantich(2, 20794))

```



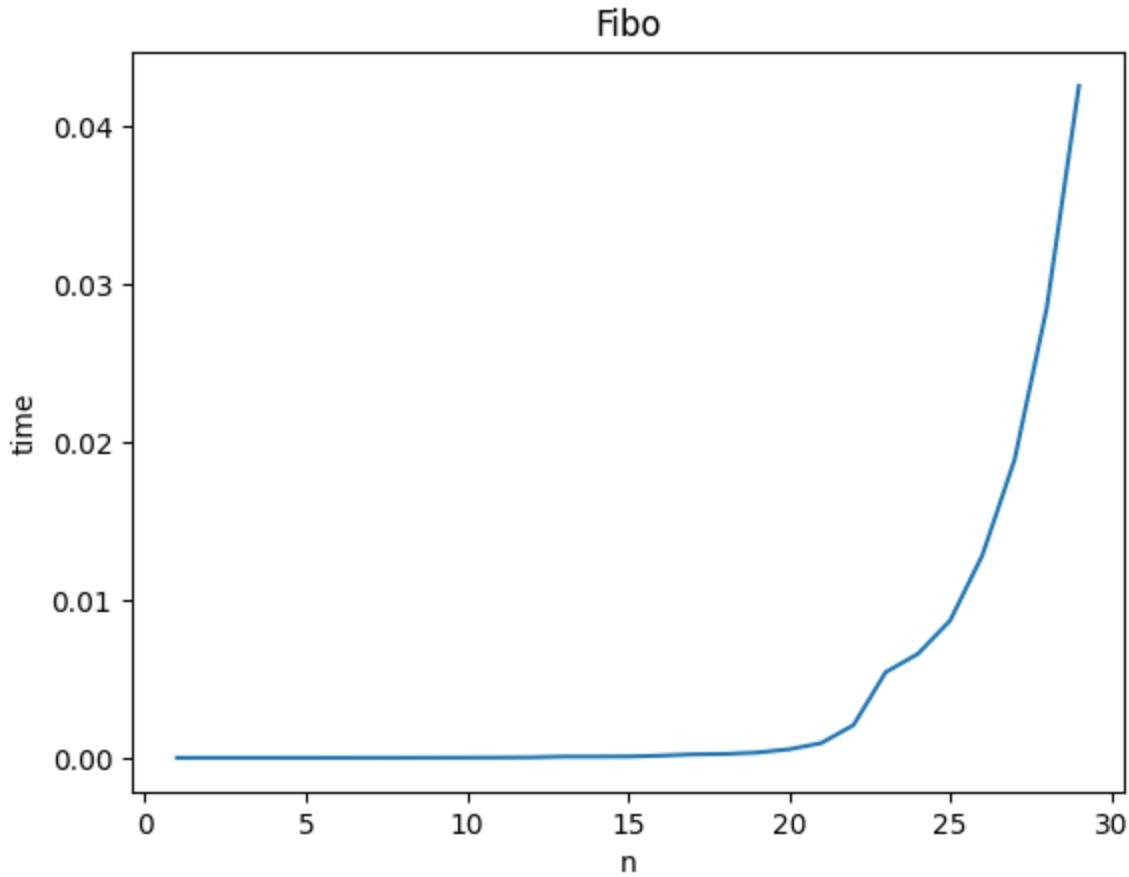
### 3. Tìm số Fibonacci thứ $n$ .

```

def fibo(n):
    if n == 1 or n == 2: return 1
    return fibo(n-1) + fibo(n-2)

print(fibo(6))

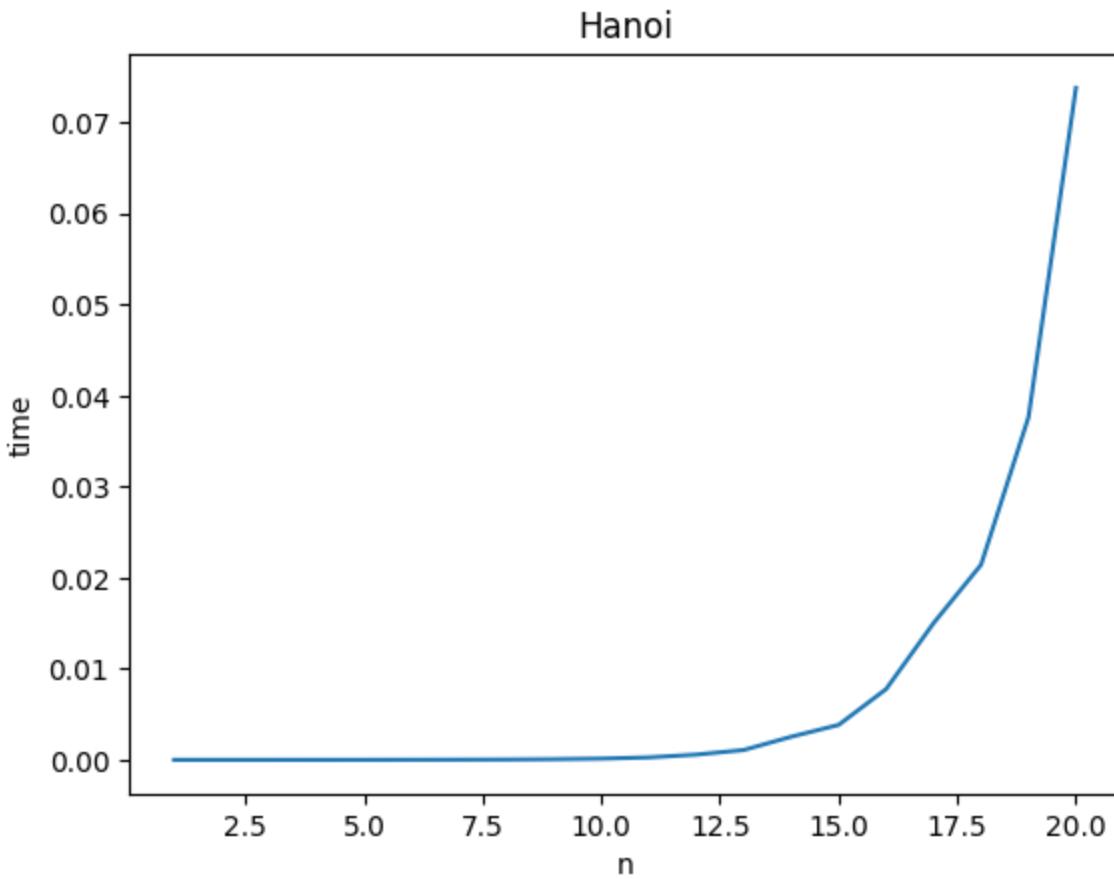
```



#### 4. Bài toán tháp Hà Nội.

```
def Hanoi(n, source, intermediate, destination):
    if n == 1:
        moveDisk(source, destination)
    else:
        Hanoi(n-1, source, destination ,intermediate)
        moveDisk(source ,destination)
        Hanoi(n-1 ,intermediate ,source ,destination)

def moveDisk(fromPeg,toPeg):
    print(f"Move disk from {fromPeg} to {toPeg}")
```

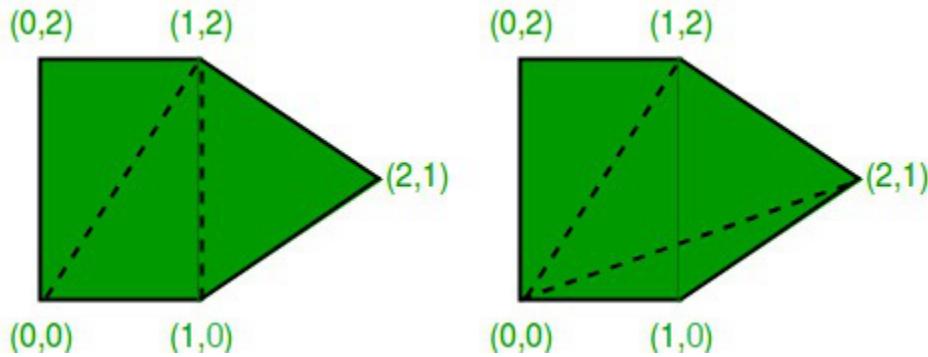


### III. Đặt bài toán, thiết kế, phân tích và triển khai thuật toán

#### Bài toán chia đa giác

Cho một đa giác lồi  $n$  đỉnh, người ta muốn chia đa giác thành các tam giác bởi các đường chéo không giao nhau, hãy tìm tổng nhỏ nhất của các đường chéo trong các phương án chia.

**Ví dụ:**



Ở 2 hình này, kết quả của hình bên trái là  $1 + \sqrt{5} \approx 3.326$  còn hình bên phải là  $2\sqrt{5} \approx 4.472$

### Ý tưởng:

Chia hình đa giác thành 3 phần: Phần tam giác, phần bên trái và phần bên phải. Bằng cách thử hết tất cả các cách chia, ta sẽ thu được kết quả là tổng độ dài đường cắt nhỏ nhất để chia đa giác thành các tam giác.

### Pseudo-code:

```

Gọi Chi phí nhỏ nhất của các đỉnh từ i đến j là minCost(i, j)
If j < i + 2 Then
    minCost(i, j) = 0
Else
    minCost(i, j) = Min { minCost(i, k) + minCost(k, j) + dist(i, k) + dist(k, j) }
                           // k thuộc khoảng [i+1, j-1], cần kiểm tra điều kiện của để thực hiện tính khoảng cách giữa (i, k) và (k, j)

Chi phí để cắt thành tam giác từ 3 đỉnh i, j, k là khoảng cách của đỉnh i và j
dist(i, j) = sqrt((x_i-x_j)^2 + (y_i-y_j)^2)

```

### Input:

```

5
0 0
0 2
1 2
2 1
1 0

```

---

## Code 1

```
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

const double INF = 1e18;

// Structure of a point in 2D plane
struct Point
{
    int x, y;
};

vector<vector<bool>> check;

bool check_points(int i, int j) {
    return i + 1 == j || check[i][j] || check[j][i];
}

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

// Recursive function, with check_points function to avoid duplicate diagonal.
double triangulate_polygon(Point points[], int n, int i, int j) {
    if (j <= i + 2) {
        return 0;
    }

    double min_cost = INF;
    for (int k = i + 2; k < j; k++) {
        double cur_cost = 0.0;
        bool mod1, mod2;
        mod1 = mod2 = false;
        if (!check_points(i, k)) {
            mod1 = true;
            check[i][k] = 1;
            cur_cost += dist(points[i], points[k]);
        }
        if (!check_points(k, j)) {
            mod2 = true;
            check[k][j] = 1;
            cur_cost += dist(points[k], points[j]);
        }
    }
}
```

```

        double left = triangulate_polygon(points, n, i, k);
        double right = triangulate_polygon(points, n, k, j);
        double cost = left + right + cur_cost;
        // Return the before state.
        if (mod1) check[i][k] = 0;
        if (mod2) check[k][j] = 0;

        min_cost = min(min_cost, cost);
    }
    return min_cost;
}

// Driver program to test above functions
int main()
{
    // freopen("input.txt", "r", stdin);
    int n; cin >> n;
    check.resize(n, vector<bool>(n, 0));

    Point points[n];
    for(int i = 0; i < n; ++i) {
        int x, y; cin >> x >> y;
        points[i] = {x, y};
    }
    double ans = triangulate_polygon(points, n, 0, n-1);
    cout << ans;
    return 0;
}

```

Đây là code với ý tưởng đơn giản như trên, có độ phức tạp là  $O(2^n)$  vì gọi 2 lần đệ quy. Tuy nhiên cách này sinh ra nhiều lần duyệt bị lặp (overlap) nên có thể sửa đổi bằng Kỹ thuật Memoization để giảm độ phức tạp.

## Code 2

```

#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

const double INF = 1e18;

// Structure of a point in 2D plane

```

```

struct Point
{
    int x, y;
};

vector<vector<bool>> check;
vector<vector<double>> dp;

bool check_points(int i, int j) {
    return i + 1 == j || check[i][j] || check[j][i];
}

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

// Recursive function, with check_points function to avoid duplicate diagonal.
double triangulate_polygon(Point points[], int n, int i, int j) {
    if (j <= i + 2) {
        return 0;
    }
    if (dp[i][j] != -1.0) return dp[i][j];
    double min_cost = INF;
    for (int k = i + 2; k < j; k++) {
        double cur_cost = 0.0;
        bool mod1, mod2;
        mod1 = mod2 = false;
        if (!check_points(i, k)) {
            mod1 = true;
            check[i][k] = 1;
            cur_cost += dist(points[i], points[k]);
        }
        if (!check_points(k, j)) {
            mod2 = true;
            check[k][j] = 1;
            cur_cost += dist(points[k], points[j]);
        }
        double left = triangulate_polygon(points, n, i, k);
        double right = triangulate_polygon(points, n, k, j);
        double cost = left + right + cur_cost;
        // Return the before state.
        if (mod1) check[i][k] = 0;
        if (mod2) check[k][j] = 0;

        min_cost = min(min_cost, cost);
    }
    return dp[i][j] = min_cost;
}

// Driver program to test above functions

```

```
int main()
{
    freopen("input.txt", "r", stdin);
    int n; cin >> n;
    check.resize(n, vector<bool>(n, 0));
    dp.resize(n, vector<double>(n, -1.0));

    Point points[n];
    for(int i = 0; i < n; ++i) {
        int x, y; cin >> x >> y;
        points[i] = {x, y};
    }
    double ans = triangulate_polygon(points, n, 0, n-1);
    cout << ans;
    return 0;
}
```

Với phương pháp tối ưu này, độ phức tạp thời gian chỉ còn  $O(n^3)$ .