# LAB 3: Phân tích thuật toán (tiếp theo)

**Nguyễn Tiến Dũng - 23001585**

## Bài 1.1: Áp dụng Master Theorem

a. $T(n) = 9T(n/3) + n$.

$a = 9,\ b = 3 \Rightarrow \log_b a = \log_3 9 = 2$. Ta có $f(n) = n = O(n^{2-\varepsilon})$ (ví dụ $\varepsilon = 1$), thuộc trường hợp 1.

$\Rightarrow T(n) = \Theta(n^2)$.

b. $T(n) = T(2n/3) + 1$.

Viết dưới dạng $T(n) = a\,T(n/b) + f(n)$, ta có $a = 1$ và $n/b = 2n/3 \Rightarrow b = 3/2$. Do đó $\log_b a = \log_{3/2} 1 = 0$. $f(n) = 1 = \Theta(n^0)$, bằng $n^{\log_b a}$, thuộc trường hợp 2.

$\Rightarrow T(n) = \Theta(\log n)$.

c. $T(n) = 3T(n/4) + n \log n$.

$a = 3,\ b = 4 \Rightarrow \log_b a = \log_4 3 \approx 0.79$. $f(n) = n \log n = n^1 \log n$, lớn hơn $n^{\log_4 3}$. Kiểm tra regularity:

$$a\,f(n/b) = 3 \cdot \frac{n}{4} \log\left(\frac{n}{4}\right) = \frac{3n}{4}(\log n - 2) \le c \cdot n \log n$$

với $c < 1$ khi $n$ đủ lớn.

$\Rightarrow T(n) = \Theta(n \log n)$.

d. $T(n) = 2T(n/3) + n$.

$a = 2,\ b = 3 \Rightarrow \log_b a = \log_3 2 \approx 0.6309$. $f(n) = n$ lớn hơn $n^{\log_3 2}$. Kiểm tra regularity: $a\,f(n/b) = 2 \cdot n/3 = 2n/3 \le cn$ với $c < 1$.

$\Rightarrow T(n) = \Theta(n)$.

e. $T(n) = T(n/2) + n$.

$a = 1,\ b = 2 \Rightarrow \log_b a = \log_2 1 = 0$. $f(n) = n$ lớn hơn $n^0$. Kiểm tra regularity: $f(n/2) = n/2 \le (1/2)n$.

$\Rightarrow T(n) = \Theta(n)$.

f. $T(n) = 3T(n/2) + n$.

$a = 3,\ b = 2 \Rightarrow \log_b a = \log_2 3 \approx 1.585$. $f(n) = n = n^1$, nhỏ hơn $n^{\log_2 3}$.

$\Rightarrow T(n) = \Theta(n^{\log_2 3})$.

g. $T(n) = 2T(n/2) + n$.
$a = 2,\ b = 2 \Rightarrow \log_b a = \log_2 2 = 1$. $f(n) = n = \Theta(n^1)$, bằng $n^{\log_2 a}$.
$\Rightarrow T(n) = \Theta(n \log n)$.

# Bài 1.2

a. $T_1(n) = 4T(n/2) + 1$.
$a = 4,\ b = 2 \Rightarrow \log_b a = \log_2 4 = 2$.
$f(n) = 1 = O(n^{2-\varepsilon})$, (với $\varepsilon = 2$) thuộc trường hợp 1 của Định lý Master Theorem.
$\Rightarrow T_1(n) = \Theta(n^2)$.

b. $T_2(n) = 4T(n/2) + \sqrt{n}$.
$a = 4,\ b = 2 \Rightarrow \log_b a = 2$.
$f(n) = n^{1/2} = O(n^{2-\varepsilon})$, (với $\varepsilon = 3/2$ ) thuộc trường hợp 1 của Định lý Master Theorem.
$\Rightarrow T_2(n) = \Theta(n^2)$.

c. $T_3(n) = 4T(n/2) + n$.
$a = 4,\ b = 2 \Rightarrow \log_b a = 2$.
$f(n) = n = O(n^{2-\varepsilon})$, (với $\varepsilon = 1$) thuộc trường hợp 1 của Định lý Master Theorem.
$\Rightarrow T_3(n) = \Theta(n^2)$.

d. $T_4(n) = 4T(n/2) + n^2$.
$a = 4,\ b = 2 \Rightarrow \log_b a = 2$.
$f(n) = \Theta(n^2) = \Theta(n^{\log_b a})$, thuộc trường hợp 2 của Định lý Master Theorem.
$\Rightarrow T_4(n) = \Theta(n^2 \log n)$.

e. $T_5(n) = 4T(n/2) + n^3$.
$a = 4,\ b = 2 \Rightarrow \log_b a = 2$.
$f(n) = \Omega(n^{2+\varepsilon})$ với $\varepsilon = 1$. Kiểm tra điều kiện:
$af(n/b) = 4(n/2)^3 = \frac{1}{2}n^3 \le cn^3$ với $c = \frac{1}{2} < 1$, thỏa điều kiện regularity.
$\Rightarrow T_5(n) = \Theta(n^3)$.

**Kết luận:**

$$T_1(n),\, T_2(n),\, T_3(n) = \Theta(n^2) \quad < \quad T_4(n) = \Theta(n^2 \log n) \quad < \quad T_5(n) = \Theta(n^3)$$

Thứ tự tăng dần: $T_1,\, T_2,\, T_3 < T_4 < T_5$

# Bài 1.3

a. $T(n) = 2T(n/2) + n \log n$.
   $a = 2,\ b = 2 \Rightarrow n^{\log_b a} = n^{\log_2 2} = n$.
   $f(n) = n \log n$. So sánh với $n^{\log_b a} = n$:

- Không thuộc trường hợp 1, vì $f(n)$ không là $O(n^{1-\varepsilon})$ cho bất kỳ $\varepsilon > 0$ (thực tế $f(n) = n \log n$ lớn hơn $n$).
- Không thuộc trường hợp 2 theo định nghĩa (vì $f(n) \neq \Theta(n)$).
- Không thuộc trường hợp 3, vì $f(n)$ không đạt $\Omega(n^{1+\varepsilon})$ cho bất kỳ $\varepsilon > 0$ (chỉ lớn hơn $n$ bởi một nhân tử $\log n$, không phải bởi một luỹ thừa $n^{\varepsilon}$).

Ngoài ra kiểm tra điều kiện regularity của trường hợp 3:

$$a\, f\left(\frac{n}{b}\right) = 2 \cdot \frac{n}{2} \log \frac{n}{2} = n(\log n - 1).$$

Ta có $af(n/2) \approx n \log n - n \sim f(n)$ (tỉ lệ tiến tới 1), nên không tồn tại hằng số $c < 1$ sao cho $af(n/2) \leq cf(n)$ cho $n$ đủ lớn. Vì vậy theo đúng định nghĩa, không thể áp dụng Master Theorem cho trường hợp (a).

b. $T(n) = 4T(n/2) + n^2 \log n$.
   $a = 4,\ b = 2 \Rightarrow n^{\log_b a} = n^{\log_2 4} = n^2$.
   $f(n) = n^2 \log n$. So sánh với $n^{\log_b a} = n^2$:

- Không thuộc trường hợp 1, vì $f(n)$ không là $O(n^{2-\varepsilon})$.
- Không thuộc trường hợp 2 theo định nghĩa (vì $f(n) \neq \Theta(n^2)$).
- Không thuộc trường hợp 3, vì $f(n)$ không là $\Omega(n^{2+\varepsilon})$ cho bất kỳ $\varepsilon > 0$ (chỉ lớn hơn $n^2$ bởi một nhân tử $\log n$).

Kiểm tra điều kiện regularity:

$$a\, f\left(\frac{n}{b}\right) = 4 \cdot \left(\frac{n}{2}\right)^2 \log \frac{n}{2} = n^2(\log n - 1).$$

Tương tự, $af(n/2) \approx n^2 \log n - n^2 \sim f(n)$ và không tồn tại $c < 1$ thỏa $af(n/2) \leq cf(n)$ cho mọi $n$ lớn. Do đó theo đúng định nghĩa , không thể áp dụng Master Theorem cho trường hợp (b).

**Kết luận :**
Cả hai recurrence

$$\text{(a) } T(n) = 2T(n/2) + n \log n \quad \text{và} \quad \text{(b) } T(n) = 4T(n/2) + n^2 \log n$$

*không thỏa bất kỳ một trong ba trường hợp* của định nghĩa Master Theorem (vì $f(n)$ chỉ lớn hơn $n^{\log_b a}$ bởi một nhân tử $\log n$, tức không "polynomially larger"), và do đó không thể áp dụng Master Theorem theo định nghĩa.

# Bài 2.1

1. Mô tả các bước:
   Bước 1: Divide
   - Nếu mảng đang xét chỉ có 1 phần tử ($left == right$)
     - Nếu phần tử đó $= X$ thì return 1.
     - Ngược lại thì return 0.
   - Nếu mảng có nhiều hơn 1 phần tử:
     - Tính $mid = (left + right)/2$.
     - Chia mảng thành 2 nửa:
       * Bên trái: $a[left \dots mid]$
       * Bên phải: $a[mid + 1 \dots right]$

   Bước 2: Conquer
   - Gọi đệ quy hàm tìm kiếm trên 2 nửa mảng:
     - $CountX(a, left, mid, X)$
     - $CountX(a, mid + 1, right, X)$

   Bước 3: Combine
   - Kết quả trả về:
     - return $CountX(a, left, mid, X) + CountX(a, mid + 1, right, X)$

2. Phân tích độ phức tạp của thuật toán:
   - Phân tích: Mỗi lần thuật toán chia mảng thành 2 nửa, gọi đệ quy trên mỗi nửa (kích thước là $n/2$) và cộng kết quả lại.
   - Nên ta có phương trình hồi quy : $T(n) = 2T(n/2) + O(1)$
   - Áp dụng Master Theorem với $a = 2$, $b = 2$, $f(n) = O(1)$ ta có $\log_b a = \log_2 2 = 1$.
     Do đó $f(n) = O(n^{\log_b a - \varepsilon})$ với $\varepsilon = 1$.
     $\Rightarrow$ Thuộc trường hợp 1 của Master Theorem.
   - Kết luận: $T(n) = \Theta(n^{\log_2 2}) = \Theta(n)$.

**Code C++: Đếm số lần xuất hiện của X trong mảng**

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

int CountX(int a[], int left, int right, int X){
    if (left == right) {
        if (a[left] == X){
            return 1;
        }
        else return 0;
    }
    int mid = (left + right) / 2;
    return CountX(a, left, mid, X) + CountX(a, mid + 1,
       right, X);
}
int main() {
    int n;
    cout << "Nhap so phan tu mang: ";
    cin >> n;
```

```
    int a[n];
    cout << "Nhap cac phan tu mang: ";
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    int X;
    cout << "Nhap gia tri X can dem: ";
    cin >> X;
    int count = CountX(a, 0, n - 1, X);
    cout << "So lan xuat hien cua " << X << " trong mang la: " << count << endl;
    return 0;
}
```

# Bài 2.2

1. Mô tả các bước:

   Bước 1 : Nhập dữ liệu
   - Hàm `nhapMang(int a[], int m)` nhận vào mảng $a$ và số phần tử $m$ của mảng.
   - Nhập sum (tổng cần tìm).

   Bước 2 : Sắp xếp mảng
   - Dùng `sort(a,a+m)` để sắp xếp mảng $a$ theo thứ tự tăng dần.
   - Việc sắp xếp giúp thuật toán Binary Search hoạt động chính xác và hiệu quả.

   Bước 3: In ra mảng đã sắp xếp
   - Dùng hàm `inMang(int a[], int m)` để in mảng đã sắp xếp.

   Bước 4: Tìm cặp phần tử có tổng bằng sum
   - Duyệt từng phần tử $a[i]$ trong mảng từ $i = 0$ đến $m - 1$.
   - Với mỗi phần tử $a[i]$, tính $target = sum - a[i]$.
   - Sử dụng hàm `binarySearch(int a[], int left, int right, int target)` để tìm $target$ trong mảng đã sắp xếp.
   - Nếu tìm thấy $target$ tại vị trí $j$ khác $i$, in ra cặp $(a[i], a[j])$ và kết thúc.
   - Nếu không tìm thấy cặp nào, in ra thông báo không tìm thấy.

2. Phân tích độ phức tạp của thuật toán:

   a. Xác định công thức hồi quy:
   - Chia mảng thành 2 nửa $\rightarrow$ mỗi nửa được gọi với kích thước $n/2 \Rightarrow 2T(n/2)$.
   - Duyệt $n/2$ phần tử, mỗi phần tử thực hiện Binary Search với độ phức tạp $O(\log n) \Rightarrow f(n) = \Theta(n/2 \log n) = \Theta(n \log n)$.
   - Recurrence : $T(n) = 2T(n/2) + \Theta(n \log n)$.

   b. Áp dụng Master Theorem:
   - Với $a = 2$, $b = 2$, ta có $\log_b a = \log_2 2 = 1$.
   - So sánh $f(n)$ với $n^{\log_b a}$: $f(n) = \Theta(n \log n)$ và $n^{\log_b a} = n^1 = n$.
   - Ta thấy $f(n)$ lớn hơn $n^{\log_b a}$ theo đa thức (vì có thêm nhân $\log n$).
   - Ta có $f(n) = \Theta(n \log n) = \Theta(n \log^1 n)$.
   - Nghĩa là $f(n) = \Theta(n^{\log_b a} \log^k n)$.

– Với $k = 1 \geq 0$, nên thuộc trường hợp 2 của Master Theorem.

c. Kết luận:

– Theo Master Theorem, ta có: $T(n) = \Theta(n^{\log_b a} \log^{k+1} n) = \Theta(n^1 \log^{1+1} n) = \Theta(n \log^2 n)$.

– Vậy độ phức tạp thời gian của thuật toán là $T(n) = O(n \log^2 n)$.

**Code C++: Tìm cặp phần tử có tổng bằng sum trong mảng**

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

int binarySearch(int a[], int n, int x) {
    int left = 0, right = n - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (a[mid] == x){
            return mid;
        }
        else if (a[mid] < x) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
    }
    return -1;
}

int findPairWithSum(int a[], int n, int sum) {
    for (int i = 0; i < n; i++) {
        int x = sum - a[i];
        int index = binarySearch(a, n, x);
        if (index != -1 && index != i) {
            cout << "Pair : (" << a[i] << ", " << a[index
               ] << ")" << endl;
            return 1;
        }
    }
    cout << "khong thoa man" << endl;
    return 0;
}

void nhapMang(int a[], int m) {
    for (int i = 0; i < m; i++) {
        cout << "Nhap phan tu thu " << i + 1 << ": ";
        cin >> a[i];
    }
}

```

```cpp
void xuatMang(int a[], int m) {
    cout << "Mang vua nhap: ";
    for (int i = 0; i < m; i++) {
        cout << a[i] << " ";
    }
    cout << endl;
}

int main() {
    int a[100] ,m, sum;
    cout << "Nhap so luong phan tu trong mang: ";
    cin >> m;
    cout << "Nhap gia tri sum: ";
    cin >> sum;

    nhapMang(a, m);
    sort(a, a + m);
    xuatMang(a, m);
    findPairWithSum(a, m, sum);

    return 0;
}
```

# Bài 2.3

1. Mô tả các bước:
   Bước 1 : Sắp xếp
   – Sắp xếp tất cả các điểm theo tọa độ x tăng dần.
   Bước 2 : Chia đôi(Divide)
   – Chia đôi mảng thành 2 nửa.
   Bước 3: Gọi đệ quy (Conquer)
   – Gọi đệ quy tìm dmin.
   – Tính dmin = $\min(dleft, dright)$.
   Bước 4: Kết hợp (Combine)
   – Tạo mảng các điểm nằm gần đường chia (trong khoảng dmin).
   – Duyệt qua các điểm trong strip, tính khoảng cách giữa các điểm trong strip với nhau và cập nhật dmin nếu tìm được khoảng cách nhỏ hơn.
   – Trả về dmin và 2 điểm tương ứng.

2. Phân tích độ phức tạp của thuật toán:
   a. Xác định công thức hồi quy:
   – Nên ta có phương trình hồi quy : $T(n) = 2T(n/2) + \theta(n)$
   – $2T(n/2)$ : chia đôi và xử lý 2 nửa.
   – $O(n)$ : duyệt qua các điểm trong strip.
   b. Áp dụng Master Theorem:
   – Ta có : a = 2, b = 2, f(n) = $\theta(n)$

– $n^{log_b a} = n^1 \Rightarrow$ f(n) $= n^{log_b a} = n^1 \Rightarrow$ Thuộc trường hợp 2 của Master Theorem.
– $T(n) = \theta(n^{log_b a} log(n)) = \theta(nlog(n))$.

c. Kết luận:

– Vậy độ phức tạp thời gian của thuật toán là $T(n) = O(n \log n)$.

**Code C++: Tìm cặp điểm gần nhau nhất.**

```cpp
#include <iostream>
#include <cmath>
#include <algorithm>
using namespace std;

struct Point {
    int x, y;
};
double dist(Point a, Point b) {
    return sqrt((a.x - b.x)*(a.x - b.x) + (a.y - b.y)*(a.y - b.y));
}

double bruteForce(Point P[], int n, Point &p1, Point &p2) {
    double min_dist = 1e9;
    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            double d = dist(P[i], P[j]);
            if (d < min_dist) {
                min_dist = d;
                p1 = P[i];
                p2 = P[j];
            }
        }
    }
    return min_dist;
}

double kcmin(Point P[], int n, Point &p1, Point &p2) {
    if (n <= 3)
        return bruteForce(P, n, p1, p2);

    int mid = n / 2;
    Point midPoint = P[mid];

    Point p1_left, p2_left, p1_right, p2_right;
    double dl = kcmin(P, mid, p1_left, p2_left);
    double dr = kcmin(P + mid, n - mid, p1_right, p2_right);

    double d;
    if (dl < dr) {
```

```
            d = dl;
            p1 = p1_left;
            p2 = p2_left;
        } else {
            d = dr;
            p1 = p1_right;
            p2 = p2_right;
        }
        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];

        sort(strip, strip + j, [](Point a, Point b){ return a
           .y < b.y; });
        for (int i = 0; i < j; i++) {
            for (int k = i + 1; k < j && (strip[k].y - strip[
               i].y) < d; k++) {
                double dist_now = dist(strip[i], strip[k]);
                if (dist_now < d) {
                    d = dist_now;
                    p1 = strip[i];
                    p2 = strip[k];
                }
            }
        }

        return d;
}
double closesUtil(Point P[], int n, Point &A, Point &B) {
        sort(P, P + n, [](Point a, Point b){ return a.x < b.x
           ; });
        return kcmin(P, n, A, B);
}

int main() {
        int n;
        cout << "Nhap so diem: ";
        cin >> n;
        Point P[n];
        cout << "Nhap toa do cac diem:\n";
        for (int i = 0; i < n; i++) {
            cout << "Diem " << i + 1 << ": ";
            cin >> P[i].x >> P[i].y;
        }

        Point A, B;
        double min_dist = closesUtil(P, n, A, B);

```

```cpp
    cout << "\nHai diem gan nhau nhat la:\n";
    cout << "A(" << A.x << ", " << A.y << ")\n";
    cout << "B(" << B.x << ", " << B.y << ")\n";
    cout << "Khoang cach nho nhat la: " << min_dist <<
       endl;

    return 0;
}
```

# Bài 2.4

1. Chứng minh tính đúng đắn của thuật toán:
   Phân tích độ phức tạp của thuật toán :
   a. Xác định công thức hồi quy:
   - Chia mảng thành 2 nửa → mỗi nửa được gọi với kích thước $n/2 \Rightarrow 2T(n/2)$.
   - So sánh 2 giá trị lớn nhất từ 2 nửa để tìm giá trị lớn thứ 2 của toàn mảng $\Rightarrow O(1)$.
   - Recurrence : $T(n) = 2T(n/2) + O(1)$.

   b. Áp dụng Master Theorem:
   - Với $a = 2$, $b = 2$, ta có $\log_b a = \log_2 2 = 1$.
   - So sánh $f(n)$ với $n^{\log_b a}$: $f(n) = O(1)$ và $n^{\log_b a} = n^1 = n$.
   - Ta thấy $f(n) < n^{\log_b a}$ theo đa thức (vì mũ $0 < 1$).
   - Nghĩa là $f(n) = O(n^{\log_b a - \varepsilon})$ với $\varepsilon = 1$.
   - Nên thuộc trường hợp 1 của Master Theorem.

   c. Kết luận:
   - Theo Master Theorem, ta có: $T(n) = \Theta(n^{\log_b a}) = \Theta(n^1) = \Theta(n)$.
   - Vậy độ phức tạp thời gian của thuật toán là $T(n) = O(n) < O(2n) \Rightarrow$(Đpcm).

**Code C++: Tìm giá trị lớn thứ hai trong mảng**

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

pair<int , int> Max2(const vector<int>& a, int left, int
   right) {
    if (left == right) {
        return {a[left], -1};
    }
    int mid = left + (right - left) / 2;
    pair<int, int> leftMax = Max2(a, left, mid);
    pair<int, int> rightMax = Max2(a, mid + 1, right);

    int max1, max2;
    if (leftMax.first > rightMax.first) {
```

```
        max1 = leftMax.first;
        max2 = max(leftMax.second, rightMax.first);
    } else {
        max1 = rightMax.first;
        max2 = max(rightMax.second, leftMax.first);
    }
    return {max1, max2};
}
int main(){
    int n;
    cout << "Nhap so luong phan tu: ";
    cin >> n;
    vector<int> a(n);
    cout << "Nhap cac phan tu: ";
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    pair<int, int> result = Max2(a, 0, n - 1);
    cout << "Phan tu lon thu hai la: " << result.second
       << endl;
}
```

# Bài 2.5

1. Phân tích định độ phức tạp của thuật toán (phương pháp truyền thống).
   – Phương pháp truyền thống nhân hai ma trận kích thước $n \times n$ có độ phức tạp thời gian là $O(n^3)$ (có 3 vòng for lồng nhau, mỗi vòng chạy từ 1 đến n).

2. Phân tích độ phức tạp sử dụng phương pháp chia để trị.
   – Phương pháp chia để trị chia mỗi ma trận thành 4 ma trận con kích thước $n/2 \times n/2$ và sử dụng đệ quy để nhân các ma trận con này.

   a. Xác định công thức truy hồi:
   – Ta có công thức truy hồi : $T(n) = 8T(n/2) + O(n^2)$.
      – Chia ma trận thành 4 ma trận con $(n/2)$ x $(n/2) \Rightarrow$ 8 phép nhân.
      – Cộng các ma trận con lại để tạo thành ma trận kết quả $\Rightarrow O(n^2)$.

   b. Áp dụng Master Theorem:
   – Với $a = 8$, $b = 2$, ta có $\log_b a = \log_2 8 = 3$.
   – So sánh $f(n)$ với $n^{\log_b a}$: $f(n) = O(n^2)$ và $n^{\log_b a} = n^3$.
   – Ta thấy $f(n) < n^{\log_b a}$ theo đa thức (vì mũ $2 < 3$).Nghĩa là $f(n) = O(n^{\log_b a - \varepsilon})$ với $\varepsilon = 1 \Rightarrow$ Nên thuộc trường hợp 1 của Master Theorem.
   – Theo Master Theorem, ta có: $T(n) = \Theta(n^{\log_b a}) = \Theta(n^3)$.
   – Vậy độ phức tạp thời gian của thuật toán là $T(n) = O(n^3)$ (bằng với phương pháp truyền thống).

3. Phương pháp Strassen.
   – Phương pháp Strassen cải tiến phương pháp chia để trị bằng cách giảm số phép nhân ma trận con từ 8 xuống còn 7.

- Strassen định nghĩa 7 ma trận con mới (P1 đến P7) dựa trên các ma trận con của A và B.

P1 = A11 * (B12 - B22)

P2 = (A11 + A12) * B22

P3 = (A21 + A22) * B11

P4 = A22 * (B21 - B11)

P5 = (A11 + A22) * (B11 + B22)

P6 = (A12 - A22) * (B21 + B22)

P7 = (A11 - A21) * (B11 + B12)

- Sau đó, các ma trận con của kết quả C được tính như sau:

C11 = P5 + P4 - P2 + P6

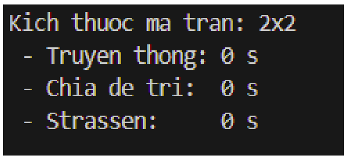
C12 = P1 + P2

C21 = P3 + P4

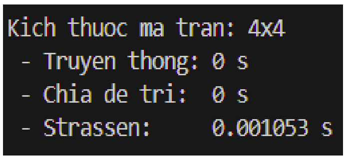
C22 = P5 + P1 - P3 - P7

4. Phân tích độ phức tạp của thuật toán sử dụng phương pháp Strassen.
   a. Xác định công thức truy hồi:
      - Công thức truy hồi của thuật toán Strassen là : $T(n) = 7T(n/2) + O(n^2)$.
        - 7 phép nhân ma trận con $(n/2)$ x $(n/2)$ $\Rightarrow$ 7 phép nhân.
        - Cộng các ma trận con lại để tạo thành ma trận kết quả $\Rightarrow O(n^2)$.
   b. Áp dụng Master Theorem:
      - Với $a = 7$, $b = 2$, ta có $\log_b a = \log_2 7 \approx 2.81 \Rightarrow$ So sánh $f(n)$ với $n^{\log_b a}$: $f(n) = O(n^2)$ và $n^{\log_b a} = n^{2.81}$.
      - Ta thấy $f(n) < n^{\log_b a}$ theo đa thức (vì mũ $2 < 2.81$).Nghĩa là $f(n) = O(n^{\log_b a - \varepsilon})$ với $\varepsilon \approx 0.81 \Rightarrow$ Nên thuộc trường hợp 1 của Master Theorem.
      - Theo Master Theorem, ta có: $T(n) = \Theta(n^{\log_b a}) = \Theta(n^{2.81})$.
      - Vậy độ phức tạp thời gian của thuật toán là $T(n) = O(n^{2.81}) < O(n^3)$ (hiệu quả hơn phương pháp truyền thống và chia để trị).

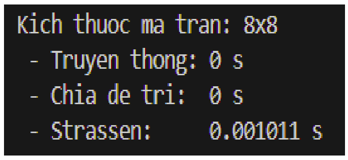
5. So sánh thời gian chạy của 3 thuật toán.
   - Phương pháp truyền thống và phương pháp chia để trị có độ phức tạp thời gian là $O(n^3)$, trong khi phương pháp Strassen có độ phức tạp thời gian là $O(n^{2.81})$.
   - Do đó, phương pháp Strassen nhanh hơn so với hai phương pháp còn lại, đặc biệt là với các ma trận lớn.
   - Tuy nhiên, phương pháp Strassen có thể không hiệu quả hơn đối với các ma trận nhỏ do chi phí cố định của việc chia và kết hợp ma trận con.




(a) Ảnh 1

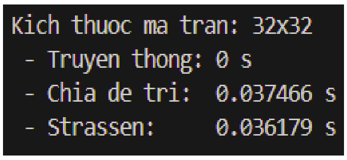



(b) Ảnh 2

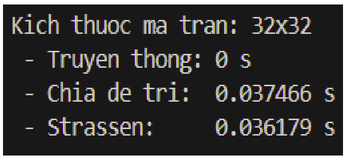



(c) Ảnh 3

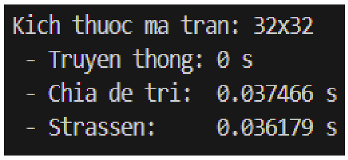
Hình 1: Ba ảnh minh họa phương pháp Strassen kém hiệu quả với các ma trận nhỏ.




(a) Ảnh 4




(b) Ảnh 5




(c) Ảnh 6

Hình 2: Ba ảnh minh họa phương pháp Strassen hiệu quả với các ma trận lớn hơn.

**Code C++: Thiết kế thuật toán để tìm tích ma trận C = A x B**

```cpp
#include<iostream>
#include<vector>
#include<ctime>
#include<chrono>
using namespace std;
using Matrix = vector<vector<int>>;

Matrix congmatran(const Matrix& A, const Matrix& B) {
    int n = A.size(), m = A[0].size();
    Matrix C(n, vector<int>(m));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            C[i][j] = A[i][j] + B[i][j];
    return C;
}
Matrix trumatran(const Matrix& A, const Matrix& B) {
    int n = A.size(), m = A[0].size();
    Matrix C(n, vector<int>(m));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            C[i][j] = A[i][j] - B[i][j];
    return C;
}
Matrix nhanmatran(const Matrix& A, const Matrix& B) {
    int n = A.size(), m = A[0].size(), p = B[0].size();
    Matrix C(n, vector<int>(p, 0));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < p; j++)
            for (int k = 0; k < m; k++)
                C[i][j] += A[i][k] * B[k][j];
```

```
    return C;
}

Matrix chiadetri(const Matrix& A, const Matrix& B) {
    int n = A.size();
    if (n == 1) return {{A[0][0] * B[0][0]}};
    int k = n / 2;
    Matrix A11(k, vector<int>(k)), A12(k, vector<int>(k));
    Matrix A21(k, vector<int>(k)), A22(k, vector<int>(k));
    Matrix B11(k, vector<int>(k)), B12(k, vector<int>(k));
    Matrix B21(k, vector<int>(k)), B22(k, vector<int>(k));
    for (int i = 0; i < k; i++)
        for (int j = 0; j < k; j++) {
            A11[i][j] = A[i][j];A12[i][j] = A[i][j + k];
            A21[i][j] = A[i + k][j];A22[i][j] = A[i + k][j + k];
            B11[i][j] = B[i][j];B12[i][j] = B[i][j + k];
            B21[i][j] = B[i + k][j];B22[i][j] = B[i + k][j + k];
        }

    Matrix C11 = congmatran(chiadetri(A11, B11), chiadetri(A12, B21));
    Matrix C12 = congmatran(chiadetri(A11, B12), chiadetri(A12, B22));
    Matrix C21 = congmatran(chiadetri(A21, B11), chiadetri(A22, B21));
    Matrix C22 = congmatran(chiadetri(A21, B12), chiadetri(A22, B22));

    Matrix C(n, vector<int>(n));
    for (int i = 0; i < k; i++)
        for (int j = 0; j < k; j++) {
            C[i][j] = C11[i][j];
            C[i][j + k] = C12[i][j];
            C[i + k][j] = C21[i][j];
            C[i + k][j + k] = C22[i][j];
        }
    return C;
}

Matrix Strassen(const Matrix& A, const Matrix& B) {
    int n = A.size();
    if (n == 1) return {{A[0][0] * B[0][0]}};
    int k = n / 2;
    Matrix A11(k, vector<int>(k)), A12(k, vector<int>(k))
```

```
        ;
    Matrix A21(k, vector<int>(k)), A22(k, vector<int>(k));
    Matrix B11(k, vector<int>(k)), B12(k, vector<int>(k));
    Matrix B21(k, vector<int>(k)), B22(k, vector<int>(k));
    for (int i = 0; i < k; i++)
        for (int j = 0; j < k; j++) {
            A11[i][j] = A[i][j];A12[i][j] = A[i][j + k];
            A21[i][j] = A[i + k][j];A22[i][j] = A[i + k][j + k];
            B11[i][j] = B[i][j];B12[i][j] = B[i][j + k];
            B21[i][j] = B[i + k][j];B22[i][j] = B[i + k][j + k];
        }

    Matrix M1 = Strassen(congmatran(A11, A22), congmatran(B11, B22));
    Matrix M2 = Strassen(congmatran(A21, A22), B11);
    Matrix M3 = Strassen(A11, trumatran(B12, B22));
    Matrix M4 = Strassen(A22, trumatran(B21, B11));
    Matrix M5 = Strassen(congmatran(A11, A12), B22);
    Matrix M6 = Strassen(trumatran(A21, A11), congmatran(B11, B12));
    Matrix M7 = Strassen(trumatran(A12, A22), congmatran(B21, B22));

    Matrix C11 = congmatran(trumatran(congmatran(M1, M4), M5), M7);
    Matrix C12 = congmatran(M3, M5);
    Matrix C21 = congmatran(M2, M4);
    Matrix C22 = congmatran(congmatran(trumatran(M1, M2), M3), M6);

    Matrix C(n, vector<int>(n));
    for (int i = 0; i < k; i++)
        for (int j = 0; j < k; j++) {
            C[i][j] = C11[i][j];
            C[i][j + k] = C12[i][j];
            C[i + k][j] = C21[i][j];
            C[i + k][j + k] = C22[i][j];
        }
    return C;
}

void thoigianchay(int n) {
    Matrix A(n, vector<int>(n)), B(n, vector<int>(n));
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++) {
```

```
                A[i][j] = rand() % 10;
                B[i][j] = rand() % 10;
            }

    auto start = chrono::high_resolution_clock::now();
    Matrix C1 = nhanmatran(A, B);
    auto end = chrono::high_resolution_clock::now();
    double elapsed_basic = chrono::duration<double>(end -
        start).count();

    start = chrono::high_resolution_clock::now();
    Matrix C2 = chiadetri(A, B);
    end = chrono::high_resolution_clock::now();
    double elapsed_divide = chrono::duration<double>(end
       - start).count();

    start = chrono::high_resolution_clock::now();
    Matrix C3 = Strassen(A, B);
    end = chrono::high_resolution_clock::now();
    double elapsed_strassen = chrono::duration<double>(
       end - start).count();

    cout << "Kich thuoc ma tran: " << n << "x" << n <<
       endl;
    cout << " - Truyen thong: " << elapsed_basic << " s"
       << endl;
    cout << " - Chia de tri:  " << elapsed_divide << " s"
        << endl;
    cout << " - Strassen:     " << elapsed_strassen << "
       s" << endl;
    cout << "" << endl;
}

int main() {
    srand(time(0));
    vector<int> sizes = {2, 4, 8, 16, 32, 64, 128};
    for (int n : sizes)
        thoigianchay(n);
}
```

# Bài 2.6

1. Viết pseudocode cho thuật toán vét cạn (brute-force).

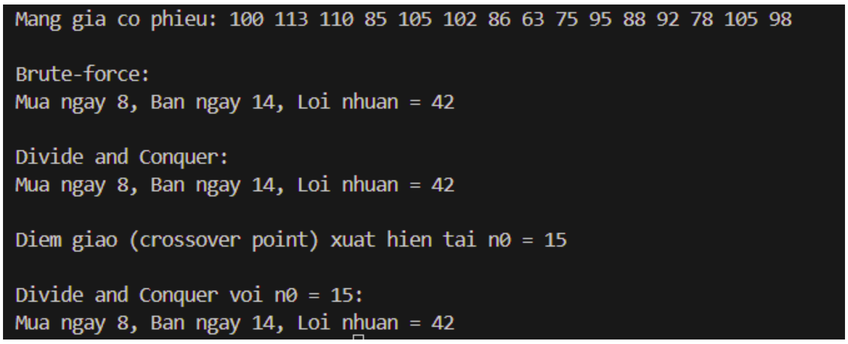
```
Input: Mảng giá P[1..n]
Output: Ngày mua bestBuy, ngày bán bestSell,
lợi nhuận tối đa maxProfit.
```

```
maxProfit ← vô cùng
bestBuy ← 0
bestSell ← 0

for i ← 1 to n-1 do
    for j ← i+1 to n do
        profit ← P[j] - P[i]
        if profit > maxProfit then
            maxProfit ← profit
            bestBuy ← i
            bestSell ← j
        end if
    end for
end for

print "Ngày mua:", bestBuy
print "Ngày bán:", bestSell
print "Lợi nhuận tối đa:", maxProfit
```

2. Ý tưởng của thuật toán chia để trị.
   a. Ý tưởng
      – Chia mảng giá cổ phiếu thành 2 nửa.
      – Tìm lợi nhuận max ở nửa trái và phải.
      – Tìm lợi nhuận qua đường chia : Mua trái, bán phải.
      – Lợi nhuận max = max(lợi nhuận trái, phải và qua đường chia).
   b. Phân tích độ phức tạp của thuật toán
      – Chia đôi mảng $\Rightarrow O(logn)$
      – Tìm max qua mid $\Rightarrow O(n)$
      – $\Rightarrow T(n) = O(nlogn)$




Hình 3: Mô tả kết quả.

**Code C++: Bài toán Maximum Subarray**

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
```

```
using namespace std;
using namespace std::chrono;

struct Result {
    int buyDay;
    int sellDay;
    int profit;
};
Result brute_force_max_profit(const vector<int>& prices)
    {
    int n = prices.size();
    Result res = {0, 0, 0};

    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            int p = prices[j] - prices[i];
            if (p > res.profit) {
                res.profit = p;
                res.buyDay = i;
                res.sellDay = j;
            }
        }
    }
    return res;
}
Result divide_and_conquer(const vector<int>& prices, int
    left, int right) {
    if (left >= right) return {left, right, 0};

    int mid = (left + right) / 2;

    Result leftRes = divide_and_conquer(prices, left, mid
        );
    Result rightRes = divide_and_conquer(prices, mid + 1,
        right);

    int minLeftPrice = prices[left], minLeftDay = left;
    for (int i = left; i <= mid; i++) {
        if (prices[i] < minLeftPrice) {
            minLeftPrice = prices[i];
            minLeftDay = i;
        }
    }

    int maxRightPrice = prices[mid + 1], maxRightDay =
        mid + 1;
    for (int i = mid + 1; i <= right; i++) {
        if (prices[i] > maxRightPrice) {
            maxRightPrice = prices[i];
            maxRightDay = i;
```

```
        }
    }

    int crossProfit = maxRightPrice - minLeftPrice;
    Result crossRes = {minLeftDay, maxRightDay, crossProfit};

    if (leftRes.profit >= rightRes.profit && leftRes.profit >= crossRes.profit)
        return leftRes;
    else if (rightRes.profit >= leftRes.profit && rightRes.profit >= crossRes.profit)
        return rightRes;
    else
        return crossRes;
}
Result divide_and_conquer_with_n0(const vector<int>& prices, int left, int right, int n0) {
    if (right - left + 1 <= n0) {
        vector<int> sub(prices.begin() + left, prices.begin() + right + 1);
        Result res = brute_force_max_profit(sub);
        res.buyDay += left;
        res.sellDay += left;
        return res;
    }

    int mid = (left + right) / 2;
    Result leftRes = divide_and_conquer_with_n0(prices, left, mid, n0);
    Result rightRes = divide_and_conquer_with_n0(prices, mid + 1, right, n0);

    int minLeftPrice = prices[left], minLeftDay = left;
    for (int i = left; i <= mid; i++) {
        if (prices[i] < minLeftPrice) {
            minLeftPrice = prices[i];
            minLeftDay = i;
        }
    }

    int maxRightPrice = prices[mid + 1], maxRightDay = mid + 1;
    for (int i = mid + 1; i <= right; i++) {
        if (prices[i] > maxRightPrice) {
            maxRightPrice = prices[i];
            maxRightDay = i;
        }
    }

```

```
    int crossProfit = maxRightPrice - minLeftPrice;
    Result crossRes = {minLeftDay, maxRightDay,
       crossProfit};

    if (leftRes.profit >= rightRes.profit && leftRes.
       profit >= crossRes.profit)
        return leftRes;
    else if (rightRes.profit >= leftRes.profit &&
       rightRes.profit >= crossRes.profit)
        return rightRes;
    else
        return crossRes;
}
int find_crossover_point(const vector<int>& prices) {
    int n0 = 2;
    while (n0 < (int)prices.size()) {
        auto start1 = high_resolution_clock::now();
        brute_force_max_profit(vector<int>(prices.begin()
           , prices.begin() + n0));
        auto end1 = high_resolution_clock::now();
        double t1 = duration<double, milli>(end1 - start1
           ).count();

        auto start2 = high_resolution_clock::now();
        divide_and_conquer(prices, 0, n0 - 1);
        auto end2 = high_resolution_clock::now();
        double t2 = duration<double, milli>(end2 - start2
           ).count();

        if (t2 < t1) return n0;
        n0++;
    }
    return n0;
}
int main() {
    vector<int> prices = {100, 113, 110, 85, 105, 102,
       86, 63, 75, 95, 88, 92, 78, 105, 98};

    cout << "Mang gia co phieu: ";
    for (int p : prices) cout << p << " ";
    cout << "\n\n";

    Result brute = brute_force_max_profit(prices);
    cout << "Brute-force:\n";
    cout << "Mua ngay " << brute.buyDay + 1 << ", Ban
       ngay " << brute.sellDay + 1
          << ", Loi nhuan = " << brute.profit << "\n\n";

    Result div = divide_and_conquer(prices, 0, prices.
       size() - 1);
```

```
    cout << "Divide and Conquer:\n";
    cout << "Mua ngay " << div.buyDay + 1 << ", Ban ngay
       " << div.sellDay + 1
         << ", Loi nhuan = " << div.profit << "\n\n";

    int n0 = find_crossover_point(prices);
    cout << "Diem giao (crossover point) xuat hien tai n0
        = " << n0 << "\n\n";

    Result divn0 = divide_and_conquer_with_n0(prices, 0,
       prices.size() - 1, n0);
    cout << "Divide and Conquer voi n0 = " << n0 << ":\n"
       ;
    cout << "Mua ngay " << divn0.buyDay + 1 << ", Ban
       ngay " << divn0.sellDay + 1
         << ", Loi nhuan = " << divn0.profit << "\n";

    return 0;
}
```