LAB 3: Phân tích thuật toán (Tiếp theo)

1 Master theorem

1.1

Sử dụng Master Theorem để giải các phương trình đệ quy sau:

a. $T(n) = 9T(\frac{n}{3}) + n$

$$
\begin{aligned}
& a = 9; \quad b = 3; \quad f(n) = n \\
& n^{\log_b a} = n^{\log_3 9} = n^2 \\
& f(n) = O(n^{\log_b a - 1}) = n^1, \qquad \epsilon = 1 \\
\Rightarrow & f(n) = \Theta(n^{\log_b a}) = \Theta(n^2)
\end{aligned}
$$

b. $T(n) = T(\frac{2n}{3}) + 1$

$$
\begin{aligned}
& a = 1; \quad b = 3/2; \quad f(n) = 1 \\
& n^{\log_b a} = n^{\log_{3/2} 1} = 1 \\
& f(n) = O(1) \\
\Rightarrow & f(n) = \Theta(\log n)
\end{aligned}
$$

c. $T(n) = 3T(\frac{n}{4}) + n \log n$

$$
\begin{aligned}
& a = 3; \quad b = 4; \quad f(n) = n \log n \\
& n^{\log_b a} = n^{\log_4 3} = n^{0.79} \\
& f(n) = n \log n = \Omega(n^{\log_4 3 + \epsilon}) \\
& af(\frac{n}{b}) = \frac{3n}{4} \log(\frac{n}{4}) = \frac{3n}{4} \log n + \frac{3n}{4} \log 4 \leq c \log n, \quad \exists c = \frac{3}{4} < 1 \\
\Rightarrow & f(n) = \Theta(n \log n)
\end{aligned}
$$

d. $T(n) = 2T(\frac{n}{3}) + n$

$$
\begin{aligned}
& a = 2; \quad b = 3; \quad f(n) = n \\
& n^{\log_b a} = n^{\log_3 2} = n^0.63 \\
& f(n) = n = \Omega(n^{0.63+0.37}), \qquad \epsilon = 0.37 > 0 \\
& af(\frac{n}{b}) = 2f(\frac{n}{3}) = 2\frac{n}{3} \leq cn, \qquad c = \frac{2}{3} < 1 \\
\Rightarrow & f(n) = \Theta(n)
\end{aligned}
$$

e. $T(n) = T(\frac{n}{2}) + n$

$$
\begin{aligned}
& a = 1; \quad b = 2; \quad f(n) = n \\
& n^{\log_2 1} = 1 \\
& f(n) = n = \Omega(n^{0+1}), \qquad \epsilon = 1 \\
& af(\frac{n}{b}) = \frac{n}{2} \leq cn, \qquad c = \frac{1}{2} < 1 \\
\Rightarrow & f(n) = \Theta(n)
\end{aligned}
$$

f. $3T(\frac{n}{2}) + n$

$$
\begin{aligned}
& a = 3; \quad b = 2; \quad f(n) = n \\
& n^{\log_2 3} = n^{1.58} \\
& f(n) = n = O(n^{1.58-0.58}), \qquad \epsilon = 0.58 \\
\Rightarrow & f(n) = \Theta(n^{\log_2 3})
\end{aligned}
$$

g. $2T(\frac{n}{2}) + n$

$$
\begin{aligned}
& a = 2; \quad b = 2; \quad f(n) = n \\
& n^{\log_2 2} = n \\
\Rightarrow & f(n) = \Theta(n \log n)
\end{aligned}
$$

1.2

Tìm độ phức tạp của từng phương trình và sắp xếp theo thứ tự tăng dần:

a. $T_1(n) = 4T(\frac{n}{2} + 1)$

$$
\begin{aligned}
& a = 4; \quad b = 2; \quad f(n) = 1 \\
& n^{\log_b a} = n^{\log_2 4} = n^2 \\
& f(n) = O(n^{2-2}), \qquad \epsilon = 2 \\
\Rightarrow & f(n) = \Theta(n^2)
\end{aligned}
$$

b. $T_2(n) = 4T(\frac{n}{2} + \sqrt{n})$

$$
\begin{aligned}
& a = 4; \quad b = 2; \quad f(n) = n^{\frac{1}{2}} \\
& n^{\log_b a} = n^{\log_2 4} = n^2 \\
& f(n) = O(n^{2-3/2}), \qquad \epsilon = \frac{3}{2} \\
\Rightarrow & f(n) = \Theta(n^2)
\end{aligned}
$$

c. $T_3(n) = 4T(\frac{n}{2} + n)$

$$
\begin{aligned}
& a = 4; \quad b = 2; \quad f(n) = n \\
& n^{\log_b a} = n^{\log_2 4} = n^2 \\
& f(n) = O(n^{2-1}), \qquad \epsilon = 1 \\
\Rightarrow & f(n) = \Theta(n^2)
\end{aligned}
$$

d. $T_4(n) = 4T(\frac{n}{2} + n^2)$

$$
\begin{aligned}
& a = 4; \quad b = 2; \quad f(n) = n^2 \\
& n^{\log_b a} = n^{\log_2 4} = n^2 \\
\Rightarrow & f(n) = \Theta(n^2 \log n)
\end{aligned}
$$

e. $T_5(n) = 4T(\frac{n}{2} + n^3)$

$$
\begin{aligned}
& a = 4; \quad b = 2; \quad f(n) = n^{\frac{1}{2}} \\
& n^{\log_b a} = n^3 \\
& f(n) = \Omega(n^{2+1}), \qquad \epsilon = 1 \\
& af(\frac{n}{b}) = \frac{4n^3}{8} = \frac{n^3}{2} \leq cn^3, \qquad c = \frac{1}{2} \\
\Rightarrow & f(n) = \Theta(n^3)
\end{aligned}
$$

$\Rightarrow T_1 = T_2 = T_3 < T_4 < T_5$

1.3

Có thể áp dụng Master Theorem để giải phương trình đệ quy dưới đây không? Giải thích lý do.

a. $T(n) = 2T(\frac{n}{2}) + n \log n$

$$
\begin{aligned}
& n^{\log_b a} = n^{\log_2 2} = n \\
& \text{Xét:} \quad \frac{n \log n}{n} = \log n \neq n^\epsilon \quad \text{với } \epsilon > 0 \\
& \Rightarrow \text{Không áp dụng được}
\end{aligned}
$$

b. $T(n) = 4T(\frac{n}{2}) + n^2 \log n$

$$
\begin{aligned}
& n^{\log_b a} = n^{\log_2 4} = n^2 \\
& \text{Xét:} \quad \frac{n^2 \log n}{n^2} = \log n \neq n^\epsilon \quad \text{với } \epsilon > 0 \\
& \Rightarrow \text{Không áp dụng được}
\end{aligned}
$$

2 Chia để trị

Chia để trị (Divide and Conquer) là một kỹ thuật thiết kế thuật toán, trong đó ta giải quyết một bài toán đệ quy bằng các áp dụng 3 bước:

1. **Chia (Divide)**: Chia bài toán thành các bài toán con nhỏ hơn cùng loại.
2. **Trị (Conquer)**: Giải quyết các bài toán con đệ quy. Nếu bài toán con đủ nhỏ, giải quyết trực tiếp.
3. **Hợp nhất (Combine)**: Kết hợp các lời giải của các bài toán con để tạo thành lời giải cho bài toán gốc.

2.1

Cho một số nguyên X và một mảng số nguyên gồm m phần tử. Hãy thiết kế một thuật toán chia để trị để đếm số lần xuất hiện của X trong mảng.
Mô tả các bước và phân tích độ phức tạp thời gian của thuật toán

Bài làm

```
int cnt_search(int *a, int x, int l, int r) {
    if (l >= r) {
        if (a[l] == x) return 1;
        else return 0;
    }
    int m = (l + r) / 2;
    if (a[m] == x) return 1;
    return cnt_search(a, x, l, m) + cnt_search(a, x, m + 1, r);
}
```

Thời gian chạy:

$$
\begin{aligned}
& T(n) = 2T(\frac{n}{2}) + 1 \\
& a = 2, \quad b = 2; \quad f(n) = 1 \\
& f(n) = 1 = O(n^{1-1}), \qquad \epsilon = 1 \\
\Rightarrow & T(n) = \Theta(n)
\end{aligned}
$$

2.2

Cho một số nguyên S và một mảng số nguyên gồm m phần tử. Tìm hai phần tử trong mảng có tổng bằng S. Hãy thiết kế một thuật toán chia để trị để thực hiện bài toán này (giả sử mảng đã được sắp xếp).
Mô tả các bước và phân tích độ phức tạp thời gian của thuật toán.

Bài làm

```
pair<int, int> find_pair(int *a, int s, int l, int r) {
    if (l >= r) return {-1, -1};
    int m = (l + r) / 2;

    auto result = find_pair(a, s, l, m);
    if (result.first != -1) return result;

    result = find_pair(a, s, m + 1, r);
```

```
        if (result.first != -1)  return result;

        int i = m;
        int j = m + 1;
        while (i >= l and j <= r) {
            int sum = a[i] + a[j];
            if (sum == s)
                return {i, j};
            else if (sum > s)
                i--;
            else
                j++;
        }
        return {-1, -1};
    }
```

Thời gian chạy:

$$
\begin{aligned}
&T(n) = 2T(\frac{n}{2}) + n \\
&a = 2, \quad b = 2; \quad f(n) = n \\
&f(n) = n = O(n) \\
\Rightarrow &T(n) = \Theta(n \log n)
\end{aligned}
$$

2.3

Cho n điểm trên mặt phẳng có tọa độ (x1, y1),(x2, y2), ...,(xn, yn). Hãy thiết kế một thuật toán chia để trị để tìm cặp điểm gần nhau nhất (có khoảng cách Euclidean nhỏ nhất). Mô tả các bước và phân tích độ phức tạp thời gian của thuật toán.

Bài làm

```
    double distance(Point p1, Point p2) {
        return sqrt(pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2));
    }

    bool compareX(Point a, Point b) {
            return a.x < b.x;
    }

    bool compareY(Point a, Point b) {
            return a.y < b.y;
```

```
}

double bruteForce(vector<Point>& P) {
    double d = 1e18;
    for (int i = 0; i < (int)P.size(); ++i)
        for (int j = i + 1; j < (int)P.size(); ++j)
            d = min(d, distance(P[i], P[j]));

    return d;
}

double closestPair(vector<Point>& Px, vector<Point>& Py) {
    int n = Px.size();
    if (n <= 3) return bruteForce(Px);

    int mid = n / 2;
    double midX = Px[mid].x;

    vector<Point> Pxl(Px.begin(), Px.begin() + mid);
    vector<Point> Pxr(Px.begin() + mid, Px.end());

    vector<Point> Pyl, Pyr;
    for (auto &p : Py) {
        if (p.x <= midX)
            Pyl.push_back(p);
        else
            Pyr.push_back(p);
    }

    double d1 = closestPair(Pxl, Pyl);
    double d2 = closestPair(Pxr, Pyr);
    double d = min(d1, d2);

    vector<Point> Sy;
    for (auto &p : Py)
        if (fabs(p.x - midX) < d)
            Sy.push_back(p);

    for (int i = 0; i < (int)Sy.size(); ++i) {
        for (int j = i + 1; j < (int)Sy.size() &&
            (Sy[j].y - Sy[i].y) < d && j - i <= 6; ++j)
        {
```

```
                d = min(d, distance(Sy[i], Sy[j]));
            }
        }
        return d;
    }
```

Thời gian chạy:

$$
\begin{aligned}
&T(n) = 2T(\frac{n}{2}) + n \\
&a = 2, \quad b = 2; \quad f(n) = n \\
&f(n) = n = O(n) \\
\Rightarrow &T(n) = \Theta(n \log n)
\end{aligned}
$$

2.4

Giả sử bạn có một mảng gồm n số nguyên dương. Bạn muốn tìm số lớn thứ hai trong mảng. Bạn có thể làm điều này bằng cách sắp xếp mảng và chọn phần tử ở vị trí thứ hai từ cuối. Tuy nhiên, bạn có thể làm tốt hơn thế. Hãy thiết kế một thuật toán sử dụng ít hơn 2n phép so sánh để tìm số lớn thứ hai trong mảng. Chứng minh tính đúng của thuật toán và phân tích độ phức tạp thời gian của nó.

Bài làm

```
int cmp = 0;

int second_max(int *a, int n) {
    int max = a[0];
    int sec_max = INT_MIN;

    for (int i = 1; i < n; i++) {
        if (a[i] > max) {
            sec_max = max;
            max = a[i];
            cmp++;
        }
        else {
            if (a[i] > sec_max) {
                sec_max = a[i];
                cmp++;
            }
        }
```

```
        }
        return sec_max;
    }
```

Thời gian chạy:

$$T(n) = O(n)$$

2.5

Bài toán nhân ma trận Cho hai ma trận vuông A và B có kích thước $n \times n$. Hãy thiết kế thuật toán để tính tích ma trận $C = A \times B$.

a. Viết pseudocode cho thuật toán dựa trên định nghĩa của phép nhân ma trận. Xác định độ phức tạp thời gian của thuật toán.

b. Đưa ra ý tưởng của thuật toán chia để trị để tính tích ma trận. Xác định độ phức tạp thời gian của thuật toán.

c. Tìm hiểu về phương pháp Strassen để tính tích ma trận. Đưa ra ý tưởng chính của phương pháp này để giảm số phép nhân ma trận.

d. Viết pseudocode cho phương pháp Strassen và xác định độ phức tạp thời gian của nó.

e. Cài đặt ba phương pháp nêu trên và thử nghiệm với các ma trận có kích thước khác nhau. So sánh thời gian chạy của ba thuật toán.

Bài làm

a. Viết pseudocode cho thuật toán dựa trên định nghĩa của phép nhân ma trận. Xác định độ phức tạp thời gian của thuật toán.

SQUARE-MATRIX-MULTIPLY(A, B)

```
1  n = A.rows
2  let C be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          c_ij = 0
6          for k = 1 to n
7              c_ij = c_ij + a_ik · b_kj
8  return C
```

Độ phức tạp thuật toán: $\Theta(n^3)$

b. Dưa ra ý tưởng của thuật toán chia để trị để tính tích ma trận. Xác định độ phức tạp thời gian của thuật toán.

Thuật toán *chia để trị* để tính tích ma trận $C = A \cdot B$, ta giả sử rằng n là một lũy thừa của 2. Chúng ta sẽ chia ma trận $n \times n$ thành 4 ma trận $\frac{n}{2} \times \frac{n}{2}$.

Giả sử rằng chúng ta phân hoạch (partition) mỗi ma trận A, B, và C thành bốn ma trận con kích thước $\frac{n}{2} \times \frac{n}{2}$ như sau:

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}, \quad C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \tag{$*$}$$

So that $C = A \cdot B$ as:

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \tag{$**$}$$

Equation $(**)$ corresponds to the four equations:

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

MULT-REC(A, B)

```
1  n = A.rows
2  let C be a new n × n matrix
3  if n = 1
4  else
5      partition A, B, and C as in equations (*)
6      C11 = MULT-REC(A11, B11) + MULT-REC(A12, B21)
7      C12 = MULT-REC(A11, B12) + MULT-REC(A12, B22)
8      C21 = MULT-REC(A21, B11) + MULT-REC(A22, B21)
9      C22 = MULT-REC(A21, B12) + MULT-REC(A22, B22)
10 return C
```

Xác định một ma trận con bằng phạm vi các chỉ số hàng và phạm vi các chỉ số cột của ma trận gốc.

Thời gian chạy:

$$\begin{aligned} T(n) &= \Theta(1) + 8T(\frac{n}{2}) + \Theta(n^2) \\ &= 8T(\frac{n}{2}) + \Theta(n^2) \Rightarrow T(n) &= \Theta(n^3) \end{aligned}$$

c. Tìm hiểu về phương pháp Strassen để tính tích ma trận. Đưa ra ý tưởng chính của phương pháp này để giảm số phép nhân ma trận.

Thay vì thực hiện **8 phép nhân đệ quy** giữa các ma trận kích thước $\frac{n}{2} \times \frac{n}{2}$, thuật toán chỉ thực hiện **7 phép nhân**.

Cái giá phải trả cho việc **loại bỏ một phép nhân ma trận** là phải **thêm một số phép cộng ma trận** kích thước $\frac{n}{2} \times \frac{n}{2}$ nhưng **số lượng phép cộng này vẫn chỉ là hằng số**.

Phương pháp của Strassen hoàn toàn không hề hiển nhiên. Thuật toán gồm 4 bước như sau:

1. Chia các ma trận đầu vào A và B, cùng ma trận đầu ra C, thành các ma trận con kích thước $\frac{n}{2} \times \frac{n}{2}$, như $(*)$. Bước này tốn thời gian $\Theta(1)$ nhờ phép tính chỉ số, giống như trong .
2. Tạo 10 ma trận $S_1, S_2, \ldots, S_{10}$, mỗi ma trận có kích thước $\frac{n}{2} \times \frac{n}{2}$ và được hình thành bằng tổng hoặc hiệu của hai ma trận từ bước 1.

$$\begin{aligned} S_1 &= B_{12} - B_{22}, \\ S_2 &= A_{11} + A_{12}, \\ S_3 &= A_{21} + A_{22}, \\ S_4 &= B_{21} - B_{11}, \\ S_5 &= A_{11} + A_{22}, \\ S_6 &= B_{11} + B_{22}, \\ S_7 &= A_{12} - A_{22}, \\ S_8 &= B_{21} + B_{22}, \\ S_9 &= A_{11} - A_{21}, \\ S_{10} &= B_{11} + B_{12}. \end{aligned}$$

Việc tạo tất cả 10 ma trận này tốn thời gian $\Theta(n^2)$.

3. Sử dụng các ma trận con từ bước 1 và 10 ma trận từ bước 2, tính đệ quy 7 phép nhân ma trận $P_1, P_2, \ldots, P_7$.

$$
\begin{aligned}
P_1 &= A_{11} \cdot S_1 = A_{11} \cdot (B_{12} - B_{22}),\\
P_2 &= S_2 \cdot B_{22} = (A_{11} + A_{12}) \cdot B_{22},\\
P_3 &= S_3 \cdot B_{11} = (A_{21} + A_{22}) \cdot B_{11},\\
P_4 &= A_{22} \cdot S_4 = A_{22} \cdot (B_{21} - B_{11}),\\
P_5 &= S_5 \cdot S_6 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22}),\\
P_6 &= S_7 \cdot S_8 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22}),\\
P_7 &= S_9 \cdot S_{10} = (A_{11} - A_{21}) \cdot (B_{11} + B_{12}).
\end{aligned}
$$

Mỗi ma trận P_i đều có kích thước $\frac{n}{2} \times \frac{n}{2}$.

4. Tính các ma trận con $C_{11}, C_{12}, C_{21}, C_{22}$ của ma trận kết quả C bằng cách cộng và trừ các tổ hợp khác nhau của các ma trận P_i.

$$
\begin{aligned}
C_{11} &= P_5 + P_4 - P_2 + P_6,\\
C_{12} &= P_1 + P_2,\\
C_{21} &= P_3 + P_4,\\
C_{22} &= P_5 + P_1 - P_3 - P_7.
\end{aligned}
$$

Toàn bộ bốn ma trận con này có thể được tính trong thời gian $\Theta(n^2)$.

d. Viết pseudocode cho phương pháp Strassen và xác định độ phức tạp thời gian của nó.

Thời gian chạy:

$$
\begin{aligned}
T(n) &= 7T(\frac{n}{2}) + \Theta(n^2)\\
\Rightarrow T(n) &= \Theta(n^{\log 7}) \approx \Theta(n^{2.81})
\end{aligned}
$$

```
STRASSEN-MULTIPLY(A, B)
 1  n = A.rows
 2  let C be a new n × n matrix
 3  if n = 1
 4  else
 5      Partition A, B, and C into submatrices
        Compute the 10 auxiliary matrices:
 6      S_1 = B_12 − B_22
 7      S_2 = A_11 + A_12
 8      S_3 = A_21 + A_22
 9      S_4 = B_21 − B_11
10      S_5 = A_11 + A_22
11      S_6 = B_11 + B_22
12      S_7 = A_12 − A_22
13      S_8 = B_21 + B_22
14      S_9 = A_11 − A_21
15      S_10 = B_11 + B_12
        Compute the 7 products recursively:
16      P_1 = STRASSEN-MULTIPLY(A_11, S_1)
17      P_2 = STRASSEN-MULTIPLY(S_2, B_22)
18      P_3 = STRASSEN-MULTIPLY(S_3, B_11)
19      P_4 = STRASSEN-MULTIPLY(A_22, S_4)
20      P_5 = STRASSEN-MULTIPLY(S_5, S_6)
21      P_6 = STRASSEN-MULTIPLY(S_7, S_8)
22      P_7 = STRASSEN-MULTIPLY(S_9, S_10)
        Combine results:
23      C_11 = P_5 + P_4 − P_2 + P_6
24      C_12 = P_1 + P_2
25      C_21 = P_3 + P_4
26      C_22 = P_5 + P_1 − P_3 − P_7
27  return C
```

e. Cài đặt ba phương pháp nêu trên và thử nghiệm với các ma trận có kích thước khác nhau. So sánh thời gian chạy của ba thuật toán.

Nhân theo định nghĩa:

```
int **multiply(int **a, int **b, int n) {
    int **c = create_matrix(n, n);
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
            c[i][j] = 0;
            for (int k = 0; k < n; k++){
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }
    return c;
}
```

Chia để chị:

```
using Matrix = vector<vector<double>>;

Matrix add(Matrix &A, Matrix &B) {
    int n = A.size();
    Matrix C(n, vector<double>(n));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            C[i][j] = A[i][j] + B[i][j];
    return C;
}

Matrix sub(Matrix &A, Matrix &B) {
    int n = A.size();
    Matrix C(n, vector<double>(n));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            C[i][j] = A[i][j] - B[i][j];
    return C;
}

Matrix multRec(Matrix &A, Matrix &B) {
    int n = A.size();
    Matrix C(n, vector<double>(n, 0.0));

    if (n == 1) {
        C[0][0] = A[0][0] * B[0][0];
        return C;
```

```
        }

        int m = n / 2;

        Matrix A11(m, vector<double>(m)), A12(m, vector<double>(m)),
               A21(m, vector<double>(m)), A22(m, vector<double>(m));
        Matrix B11(m, vector<double>(m)), B12(m, vector<double>(m)),
               B21(m, vector<double>(m)), B22(m, vector<double>(m));

        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < m; ++j) {
                A11[i][j] = A[i][j];
                A12[i][j] = A[i][j + m];
                A21[i][j] = A[i + m][j];
                A22[i][j] = A[i + m][j + m];

                B11[i][j] = B[i][j];
                B12[i][j] = B[i][j + m];
                B21[i][j] = B[i + m][j];
                B22[i][j] = B[i + m][j + m];
            }
        }

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

        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < m; ++j) {
                C[i][j] = C11[i][j];
                C[i][j + m] = C12[i][j];
                C[i + m][j] = C21[i][j];
                C[i + m][j + m] = C22[i][j];
            }
        }

        return C;
    }
```

Strassen:

```
Matrix naiveMult(const Matrix &A, const Matrix &B) {
    int n = A.size();
    Matrix C(n, vector<double>(n, 0));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            for (int k = 0; k < n; ++k)
                C[i][j] += A[i][k] * B[k][j];
    return C;
}

Matrix strassen(const Matrix &A, const Matrix &B) {
    int n = A.size();
    if (n <= 2) return naiveMult(A, B);

    int k = n / 2;

    Matrix A11(k, vector<double>(k)), A12(k, vector<double>(k)),
           A21(k, vector<double>(k)), A22(k, vector<double>(k));
    Matrix B11(k, vector<double>(k)), B12(k, vector<double>(k)),
           B21(k, vector<double>(k)), B22(k, vector<double>(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];
        }

    // S1..S10
    Matrix S1  = sub(B12, B22);
    Matrix S2  = add(A11, A12);
    Matrix S3  = add(A21, A22);
    Matrix S4  = sub(B21, B11);
    Matrix S5  = add(A11, A22);
    Matrix S6  = add(B11, B22);
```

```
        Matrix S7  = sub(A12, A22);
        Matrix S8  = add(B21, B22);
        Matrix S9  = sub(A11, A21);
        Matrix S10 = add(B11, B12);

        // P1..P7
        Matrix P1 = strassen(A11, S1);
        Matrix P2 = strassen(S2, B22);
        Matrix P3 = strassen(S3, B11);
        Matrix P4 = strassen(A22, S4);
        Matrix P5 = strassen(S5, S6);
        Matrix P6 = strassen(S7, S8);
        Matrix P7 = strassen(S9, S10);

        // C11..C22
        Matrix C11 = add(sub(add(P5, P4), P2), P6);
        Matrix C12 = add(P1, P2);
        Matrix C21 = add(P3, P4);
        Matrix C22 = sub(sub(add(P5, P1), P3), P7);

        Matrix C(n, vector<double>(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;
    }
```

2.6 Bài toán Maximum Subarray

Xét bài toán sau:

Một nhà đầu tư muốn mua và bán cổ phiếu của một công ty trong khoảng thời gian nhất định để tối đa hóa lợi nhuận. Mỗi ngày, nhà đầu tư được biết giá cổ phiếu và chỉ được mua một cổ phiếu trong một ngày. Nhà đầu tư chỉ được bán cổ phiếu đã mua sau một ngày với mục tiêu tối đa hóa lợi nhuận (giá bán - giá mua).

Dựa vào dữ liệu giá cổ phiếu trong 10 ngày liên tiếp được minh họa ở hình dưới, hãy

xác định: **Ngày mua và ngày bán** để đạt được lợi nhuận tối đa. **Lợi nhuận tối đa** có thể đạt được.

a. Viết pseudocode cho thuật toán vét cạn (brute-force) để giải quyết bài toán này theo thời gian $O(n^2)$.

BRUTE-FORCE(A, n)

```
1  max_value = 0
2  for i = 0 to n − 2
3      for j = i + 1 to n − 1
4          value = A[j] − A[i]
5          if value > max_value
6              min = i
7              max = j
8              max_value = value
9  return [min, max, max_value]
```

b. Cài đặt thuật toán vét cạn (brute-force) để giải quyết bài toán.

```
vector<int> burte_force(vector<int> a, int n) {
    int min, max;
    int max_value = 0;
    for (int i = 0; i < n-1; i++) {
        for (int j = i + 1; j < n; j++) {
            int value = a[j] - a[i];
            if (value > max_value) {
                min = i;
                max = j;
                max_value = value;
            }
        }
    }
    vector<int> ans;
    ans.push_back(min);
    ans.push_back(max);
    ans.push_back(max_value);

    return ans;
}
```

c. Giải thích ý tưởng của thuật toán chia để trị (divide and conquer) để giải quyết bài toán. Xác định độ phức tạp thời gian của thuật toán.

- Chia thành 2 nửa
- Tìm hiệu lớn nhất của 2 bên
- Tìm Min bên trái và Max bên phải
- Tính giá trị hiệu lớn nhất giữa 2 bên
- Lấy giá trị Max của hiệu lớn nhất trong 3 phần: bên trái, bên phải và ở giữa

Thời gian chạy:

$$T(n) = 2T(\frac{n}{2}) + n$$
$$\Rightarrow T(n) = \Theta(n \log n)$$

d. Cài đặt thuật toán chia để trị để giải quyết bài toán.

```
vector<int> divide_conquer(vector<int> &A, int low, int high) {
    if (low == high) {
        return {low, high, 0};
    }

    int mid = (low + high) / 2;

    vector<int> left = divide_conquer(A, low, mid);
    vector<int> right = divide_conquer(A, mid + 1, high);

    int min_left_idx = low;
    for (int i = low + 1; i <= mid; i++) {
        if (A[i] < A[min_left_idx]) min_left_idx = i;
    }

    int max_right_idx = mid + 1;
    for (int j = mid + 2; j <= high; j++) {
        if (A[j] > A[max_right_idx]) max_right_idx = j;
    }

    int cross_value = A[max_right_idx] - A[min_left_idx];
    vector<int> cross = {min_left_idx, max_right_idx, cross_value};

    if (left[2] >= right[2] && left[2] >= cross[2])
        return left;
    else if (right[2] >= left[2] && right[2] >= cross[2])
        return right;
```

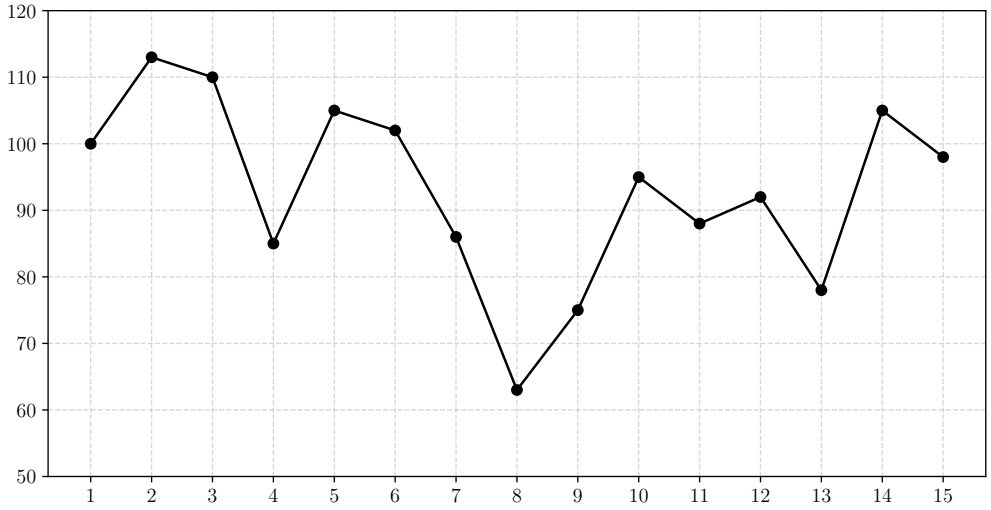
```
        else
            return cross;
    }
```

e. Xác định kích thước n_0 để xuất hiện điểm giao (crossover point) mà tại đó thuật toán chia để trị bắt đầu chạy nhanh hơn thuật toán vét cạn.

$$\begin{aligned} & n \log n < n^2 \\ \Leftrightarrow & \log n < n \\ \Rightarrow & n_0 = 1 \end{aligned}$$

f. Thay đổi trường hợp cơ sở của thuật toán chia để trị để sử dụng thuật toán vét cạn khi kích thước mảng nhỏ hơn hoặc bằng n_0. Điểm giao mới là bao nhiêu?

$$\begin{aligned} & n \log n < n^2, \qquad \forall n \\ \Rightarrow & n > 0 \text{ là được.} \end{aligned}$$



Ngày	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Giá	100	113	110	85	105	102	86	63	75	95	88	92	78	105	98