华东师范大学数据科学与工程学院上机实践报告

课程名称：算法设计与分析	**年级：**19 级	**上机实践成绩：**
指导教师：金澈清	**姓名：**龚敬洋	
上机实践名称：矩阵乘法	**学号：**	**上机实践日期：**
	10195501436	2020/9/25
上机实践编号：No.2	**组号：**1-436	

一、目的

1. 熟悉算法设计的基本思想
2. 掌握 Strassen 算法的基本思想，并且能够分析算法性能

二、内容与设计思想

1. 设计一个随机数矩阵生成器，输入参数包括 N, s, t；可随机生成一个大小为 N*N、数值范围在[s, t]之间的矩阵。
2. 编程实现普通的矩阵乘法；
3. 编程实现 Strassen’s algorithm；
4. 在不同数据规模情况下（数据规模 N=2^4, 2^8, 2^9, 2^10, 2^11）下，两种算法的运行时间各是多少；
5. 思考题：修改 Strassen’s algorithm，使之适应矩阵规模 N 不是 2 的幂的情况；
6. 改进后的算法与 2 中的算法在相同数据规模下进行比较。

三、使用环境

推荐使用 C/C++集成编译环境。

四、实验过程

1. *写出矩阵生成器和算法的源代码：*

矩阵生成器

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <ctime>
using namespace std;
int main() {
    int n, s, t;
    srand(time(0));
    ofstream fout1("data1.txt");
    ofstream fout2("data2.txt");
    cin>>n>>s>>t;
    for(int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            fout1 << s + rand() % (t - s)<<" ";
            fout2 << s + rand() % (t - s)<<" ";
        }
```

```
        fout1<<endl;
        fout2<<endl;
    }
    fout1.close();
    fout2.close();
    return 0;
}
```

朴素矩阵乘法

```
#include <iostream>
#include <fstream>
#include <cmath>
using namespace std;
int m1[2048][2048], m2[2048][2048], tm1[2048 * 2048], tm2[2048 * 2048];
int main(){
    int cnt, n, tsum;
    ifstream fin1("data1.txt");
    ifstream fin2("data2.txt");
    cnt = 0;
    while (!fin1.eof()){
        fin1>>tm1[cnt];
        fin2>>tm2[cnt];
        cnt++;
    }
    cnt--;
    n = sqrt(cnt);
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            m1[i][j] = tm1[i * n + j];
            m2[i][j] = tm2[i * n + j];
        }
    }
    for(int i = 0; i < n; i++){
        for(int j = 0; j < n; j++){
            tsum = 0;
            for(int k = 0; k < n; k++){
                tsum += m1[i][k] * m2[k][j];
            }
            cout<<tsum<<" ";
        }
        cout<<endl;
    }
    fin1.close();
    fin2.close();
    return 0;
}
```

Strassen's Algorithm

```
#include<iostream>
#include<fstream>
#include<cmath>
#include<cstdlib>
using namespace std;
struct Matrix {
    int row, column;
    int** m;

    Matrix(int r, int c) {
        row = r;
```

```
        column = c;
        m = (int**) malloc(sizeof(int*)*r);
        for (int i = 0; i < r; i++)
            m[i] = (int*)malloc(sizeof(int) * c);
    }

    Matrix(const Matrix& mat) {
        row = mat.row;
        column = mat.column;
        m = (int**)malloc(sizeof(int*) * mat.row);
        for (int i = 0; i < mat.row; i++)
            m[i] = (int*)malloc(sizeof(int) * mat.column);
        for (int i = 0; i < row; i++)
            for (int j = 0; j < column; j++)
                m[i][j] = mat.m[i][j];
    }

    Matrix& operator = (const Matrix& mat) {
        if (this != &mat) {
            row = mat.row;
            column = mat.column;
            m = (int**)malloc(sizeof(int*) * mat.row);
            for (int i = 0; i < mat.row; i++)
                m[i] = (int*)malloc(sizeof(int) * mat.column);
            for (int i = 0; i < row; i++)
                for (int j = 0; j < column; j++)
                    m[i][j] = mat.m[i][j];
        }
        return *this;
    }

    ~Matrix() {
        if (m != NULL) {
            for (int i = 0; i < row; i++) {
                delete[] m[i];
            }
            delete[] m;
        }
    }

};

int tm1[2048 * 2048], tm2[2048 * 2048];

Matrix matAdd(Matrix* matA, Matrix* matB) {
    Matrix matR = Matrix((*matA).row, (*matA).column);
    for (int i = 0; i < (*matA).row; i++)
        for (int j = 0; j < (*matA).column; j++) {
            matR.m[i][j] = (*matA).m[i][j] + (*matB).m[i][j];
        }
    return matR;
}
Matrix matSub(Matrix* matA, Matrix* matB) {
    Matrix matR = Matrix((*matA).row, (*matA).column);
    for (int i = 0; i < (*matA).row; i++)
        for (int j = 0; j < (*matA).column; j++) {
            matR.m[i][j] = (*matA).m[i][j] - (*matB).m[i][j];
        }
    return matR;
}
Matrix matSplit(Matrix mat, int rowStart, int columnStart, int rowEnd, int columnEnd) {
    Matrix matR = Matrix(rowEnd - rowStart + 1, columnEnd - columnStart + 1);
    for (int i = 0; i <= rowEnd - rowStart; i++)
        for (int j = 0; j <= columnEnd - columnStart; j++) {
            matR.m[i][j] = mat.m[rowStart + i][columnStart + j];
        }
```

```
78.     return matR;
79. }
80. Matrix matCombine(Matrix mat1, Matrix mat2, Matrix mat3, Matrix mat4) {
81.     Matrix matR = Matrix(mat1.row + mat3.row, mat1.column + mat2.column);
82.     for (int i = 0; i < mat1.row; i++)
83.         for (int j = 0; j < mat1.column; j++)
84.             matR.m[i][j] = mat1.m[i][j];
85.     for (int i = 0; i < mat2.row; i++)
86.         for (int j = 0; j < mat2.column; j++)
87.             matR.m[i][mat1.column + j] = mat2.m[i][j];
88.     for (int i = 0; i < mat3.row; i++)
89.         for (int j = 0; j < mat3.column; j++)
90.             matR.m[mat1.row + i][j] = mat3.m[i][j];
91.     for (int i = 0; i < mat4.row; i++)
92.         for (int j = 0; j < mat4.column; j++)
93.             matR.m[mat1.row + i][mat1.column + j] = mat4.m[i][j];
94.     return matR;
95. }
96. Matrix matProduct(Matrix* matA, Matrix* matB) {
97.     if ((*matA).row == 1 && (*matA).column == 1 && (*matB).row == 1 && (*matB).column == 1
    ) {
98.         Matrix matR = Matrix(1, 1);
99.         matR.m[0][0] = (*matA).m[0][0] * (*matB).m[0][0];
100.            return matR;
101.        }
102.        int midR = (*matA).row / 2 - 1;
103.        int midC = (*matA).column / 2 - 1;
104.        Matrix a11 = matSplit((*matA), 0, 0, midR, midC);
105.        Matrix a12 = matSplit((*matA), 0, midC + 1, midR, (*matA).column - 1);
106.        Matrix a21 = matSplit((*matA), midR + 1, 0, (*matA).row - 1, midC);
107.        Matrix a22 = matSplit((*matA), midR + 1, midC + 1, (*matA).row - 1, (*matA).col
    umn - 1);
108.        Matrix b11 = matSplit((*matB), 0, 0, midR, midC);
109.        Matrix b12 = matSplit((*matB), 0, midC + 1, midR, (*matA).column - 1);
110.        Matrix b21 = matSplit((*matB), midR + 1, 0, (*matA).row - 1, midC);
111.        Matrix b22 = matSplit((*matB), midR + 1, midC + 1, (*matA).row - 1, (*matA).col
    umn - 1);
112.        Matrix tmp1 = matAdd(&a11, &a22);
113.        Matrix tmp2 = matAdd(&b11, &b22);
114.        Matrix tmp3 = matAdd(&a21, &a22);
115.        Matrix tmp4 = matSub(&b12, &b22);
116.        Matrix tmp5 = matSub(&b21, &b11);
117.        Matrix tmp6 = matAdd(&a11, &a12);
118.        Matrix tmp7 = matSub(&a21, &a11);
119.        Matrix tmp8 = matAdd(&b11, &b12);
120.        Matrix tmp9 = matSub(&a12, &a22);
121.        Matrix tmp10 = matAdd(&b21, &b22);
122.        Matrix m1 = matProduct(&tmp1, &tmp2);
123.        Matrix m2 = matProduct(&tmp3, &b11);
124.        Matrix m3 = matProduct(&a11, &tmp4);
125.        Matrix m4 = matProduct(&a22, &tmp5);
126.        Matrix m5 = matProduct(&tmp6, &b22);
127.        Matrix m6 = matProduct(&tmp7, &tmp8);
128.        Matrix m7 = matProduct(&tmp9, &tmp10);
129.        Matrix mtmp1 = matAdd(&m1, &m4);
130.        Matrix mtmp2 = matSub(&mtmp1, &m5);
131.        Matrix mtmp3 = matSub(&m1, &m2);
132.        Matrix mtmp4 = matAdd(&mtmp3, &m3);
133.        Matrix c11 = matAdd(&mtmp2, &m7);
134.        Matrix c12 = matAdd(&m3, &m5);
135.        Matrix c21 = matAdd(&m2, &m4);
136.        Matrix c22 = matAdd(&mtmp4, &m6);
137.        return matCombine(c11, c12, c21, c22);
138.    }
139.    int main() {
140.        int cnt = 0, n;
```

```
            ifstream fin1("data1.txt");
            ifstream fin2("data2.txt");
            while (!fin1.eof() && !fin2.eof()) {
                fin1 >> tm1[cnt];
                fin2 >> tm2[cnt];
                cnt++;
            }
            cnt--;
            n = sqrt(cnt);
            Matrix m1 = Matrix(n, n);
            Matrix m2 = Matrix(n, n);
            Matrix mr = Matrix(n, n);
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    m1.m[i][j] = tm1[i * n + j];
                    m2.m[i][j] = tm2[i * n + j];
                }
            }
            mr = matProduct(&m1, &m2);
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    cout << mr.m[i][j] << " ";
                }
                cout << endl;
            }
            fin1.close();
            fin2.close();
            return 0;
        }
```

扩展的 Strassen's Algorithm

```
#include<iostream>
#include<fstream>
#include<cmath>
#include<cstdlib>
#include<ctime>
using namespace std;
struct Matrix {
    int row, column;
    int** m;

    Matrix(int r, int c) {
        row = r;
        column = c;
        m = (int**) malloc(sizeof(int*)*r);
        for (int i = 0; i < r; i++)
            m[i] = (int*)malloc(sizeof(int) * c);
    }

    Matrix(const Matrix& mat) {
        row = mat.row;
        column = mat.column;
        m = (int**)malloc(sizeof(int*) * mat.row);
        for (int i = 0; i < mat.row; i++)
            m[i] = (int*)malloc(sizeof(int) * mat.column);
        for (int i = 0; i < row; i++)
            for (int j = 0; j < column; j++)
                m[i][j] = mat.m[i][j];
    }

    Matrix& operator = (const Matrix& mat) {
        if (this != &mat) {
```

```
            row = mat.row;
            column = mat.column;
            m = (int**)malloc(sizeof(int*) * mat.row);
            for (int i = 0; i < mat.row; i++)
                m[i] = (int*)malloc(sizeof(int) * mat.column);
            for (int i = 0; i < row; i++)
                for (int j = 0; j < column; j++)
                    m[i][j] = mat.m[i][j];
        }
        return *this;
    }

    ~Matrix() {
        if (m != NULL) {
            for (int i = 0; i < row; i++) {
                delete[] m[i];
            }
            delete[] m;
        }
    }

};

int tm1[2048 * 2048], tm2[2048 * 2048];

Matrix matExtend(Matrix* mat){
    int pos = 0, exn;
    int n = (*mat).row;
    while(n){
        pos++;
        n = n>>1;
    }
    if(((*mat).row & ((*mat).row - 1)) == 0) exn = (*mat).row;
    else exn = 1<<pos;
    Matrix matR = Matrix(exn, exn);
    for (int i = 0; i < (*mat).row; i++)
        for (int j = 0; j < (*mat).column; j++)
            matR.m[i][j] = (*mat).m[i][j];
    for(int i = (*mat).row; i < exn; i++)
        for(int j = (*mat).column; j < exn; j++)
            matR.m[i][j] = 0;
    return matR;
}
Matrix matAdd(Matrix* matA, Matrix* matB) {
    Matrix matR = Matrix((*matA).row, (*matA).column);
    for (int i = 0; i < (*matA).row; i++)
        for (int j = 0; j < (*matA).column; j++) {
            matR.m[i][j] = (*matA).m[i][j] + (*matB).m[i][j];
        }
    return matR;
}
Matrix matSub(Matrix* matA, Matrix* matB) {
    Matrix matR = Matrix((*matA).row, (*matA).column);
    for (int i = 0; i < (*matA).row; i++)
        for (int j = 0; j < (*matA).column; j++) {
            matR.m[i][j] = (*matA).m[i][j] - (*matB).m[i][j];
        }
    return matR;
}
Matrix matSplit(Matrix mat, int rowStart, int columnStart, int rowEnd, int columnEnd) {
    Matrix matR = Matrix(rowEnd - rowStart + 1, columnEnd - columnStart + 1);
    for (int i = 0; i <= rowEnd - rowStart; i++)
        for (int j = 0; j <= columnEnd - columnStart; j++) {
            matR.m[i][j] = mat.m[rowStart + i][columnStart + j];
        }
    return matR;
```

```
98. }
99. Matrix matCombine(Matrix mat1, Matrix mat2, Matrix mat3, Matrix mat4) {
100.            Matrix matR = Matrix(mat1.row + mat3.row, mat1.column + mat2.column);
101.            for (int i = 0; i < mat1.row; i++)
102.                for (int j = 0; j < mat1.column; j++)
103.                    matR.m[i][j] = mat1.m[i][j];
104.            for (int i = 0; i < mat2.row; i++)
105.                for (int j = 0; j < mat2.column; j++)
106.                    matR.m[i][mat1.column + j] = mat2.m[i][j];
107.            for (int i = 0; i < mat3.row; i++)
108.                for (int j = 0; j < mat3.column; j++)
109.                    matR.m[mat1.row + i][j] = mat3.m[i][j];
110.            for (int i = 0; i < mat4.row; i++)
111.                for (int j = 0; j < mat4.column; j++)
112.                    matR.m[mat1.row + i][mat1.column + j] = mat4.m[i][j];
113.            return matR;
114.        }
115.        Matrix matProduct(Matrix* matA, Matrix* matB) {
116.            if ((*matA).row == 1 && (*matA).column == 1 && (*matB).row == 1 && (*matB).colu
    mn == 1) {
117.                Matrix matR = Matrix(1, 1);
118.                matR.m[0][0] = (*matA).m[0][0] * (*matB).m[0][0];
119.                return matR;
120.            }
121.            int midR = (*matA).row / 2 - 1;
122.            int midC = (*matA).column / 2 - 1;
123.            Matrix a11 = matSplit((*matA), 0, 0, midR, midC);
124.            Matrix a12 = matSplit((*matA), 0, midC + 1, midR, (*matA).column - 1);
125.            Matrix a21 = matSplit((*matA), midR + 1, 0, (*matA).row - 1, midC);
126.            Matrix a22 = matSplit((*matA), midR + 1, midC + 1, (*matA).row - 1, (*matA).col
    umn - 1);
127.            Matrix b11 = matSplit((*matB), 0, 0, midR, midC);
128.            Matrix b12 = matSplit((*matB), 0, midC + 1, midR, (*matA).column - 1);
129.            Matrix b21 = matSplit((*matB), midR + 1, 0, (*matA).row - 1, midC);
130.            Matrix b22 = matSplit((*matB), midR + 1, midC + 1, (*matA).row - 1, (*matA).col
    umn - 1);
131.            Matrix tmp1 = matAdd(&a11, &a22);
132.            Matrix tmp2 = matAdd(&b11, &b22);
133.            Matrix tmp3 = matAdd(&a21, &a22);
134.            Matrix tmp4 = matSub(&b12, &b22);
135.            Matrix tmp5 = matSub(&b21, &b11);
136.            Matrix tmp6 = matAdd(&a11, &a12);
137.            Matrix tmp7 = matSub(&a21, &a11);
138.            Matrix tmp8 = matAdd(&b11, &b12);
139.            Matrix tmp9 = matSub(&a12, &a22);
140.            Matrix tmp10 = matAdd(&b21, &b22);
141.            Matrix m1 = matProduct(&tmp1, &tmp2);
142.            Matrix m2 = matProduct(&tmp3, &b11);
143.            Matrix m3 = matProduct(&a11, &tmp4);
144.            Matrix m4 = matProduct(&a22, &tmp5);
145.            Matrix m5 = matProduct(&tmp6, &b22);
146.            Matrix m6 = matProduct(&tmp7, &tmp8);
147.            Matrix m7 = matProduct(&tmp9, &tmp10);
148.            Matrix mtmp1 = matAdd(&m1, &m4);
149.            Matrix mtmp2 = matSub(&mtmp1, &m5);
150.            Matrix mtmp3 = matSub(&m1, &m2);
151.            Matrix mtmp4 = matAdd(&mtmp3, &m3);
152.            Matrix c11 = matAdd(&mtmp2, &m7);
153.            Matrix c12 = matAdd(&m3, &m5);
154.            Matrix c21 = matAdd(&m2, &m4);
155.            Matrix c22 = matAdd(&mtmp4, &m6);
156.            return matCombine(c11, c12, c21, c22);
157.        }
158.        int main() {
159.            int cnt = 0, n;
160.            ifstream fin1("data1.txt");
```

```
        ifstream fin2("data2.txt");
        while (!fin1.eof() && !fin2.eof()) {
            fin1 >> tm1[cnt];
            fin2 >> tm2[cnt];
            cnt++;
        }
        cnt--;
        n = sqrt(cnt);
        Matrix m1 = Matrix(n, n);
        Matrix m2 = Matrix(n, n);
        Matrix mr = Matrix(n, n);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                m1.m[i][j] = tm1[i * n + j];
                m2.m[i][j] = tm2[i * n + j];
            }
        }
        cntp = 0;
        Matrix m1e = matExtend(&m1);
        Matrix m2e = matExtend(&m2);
        mr = matProduct(&m1e, &m2e);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                cout << mr.m[i][j] << " ";
            }
            cout << endl;
        }
        fin1.close();
        fin2.close();
        return 0;
    }
```

Strassen's Algorithm（剪枝优化）

```
#include<iostream>
#include<fstream>
#include<cmath>
#include<cstdlib>
using namespace std;
struct Matrix {
    int row, column;
    int** m;

    Matrix(int r, int c) {
        row = r;
        column = c;
        m = (int**) malloc(sizeof(int*)*r);
        for (int i = 0; i < r; i++)
            m[i] = (int*)malloc(sizeof(int) * c);
    }

    Matrix(const Matrix& mat) {
        row = mat.row;
        column = mat.column;
        m = (int**)malloc(sizeof(int*) * mat.row);
        for (int i = 0; i < mat.row; i++)
            m[i] = (int*)malloc(sizeof(int) * mat.column);
        for (int i = 0; i < row; i++)
            for (int j = 0; j < column; j++)
                m[i][j] = mat.m[i][j];
    }

    Matrix& operator = (const Matrix& mat) {
```

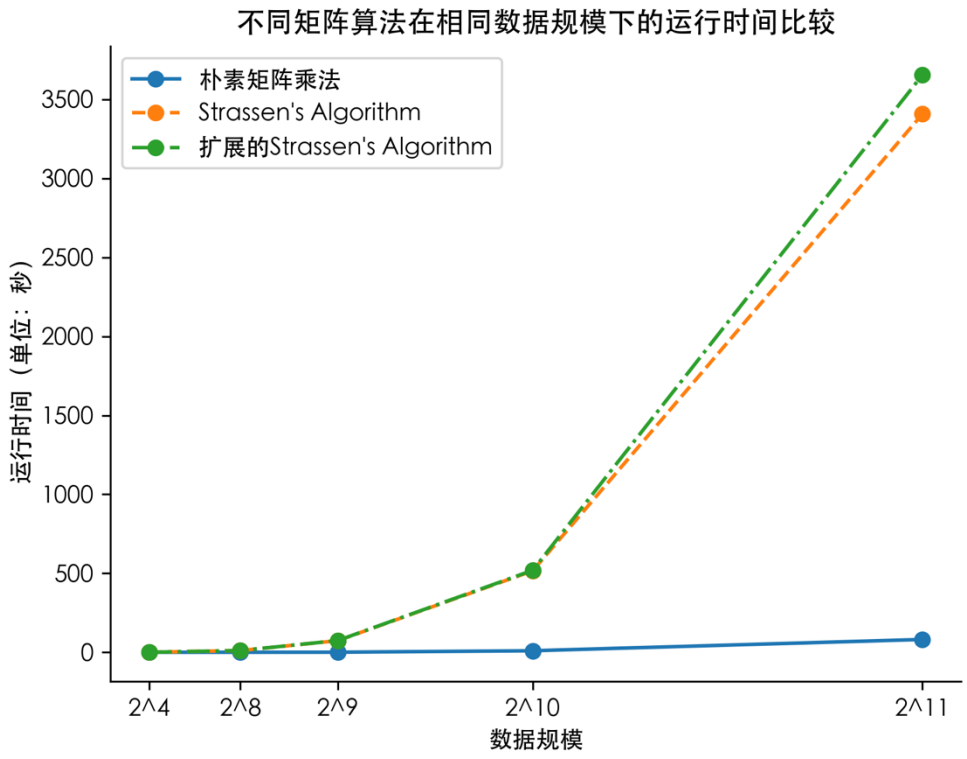
```
        if (this != &mat) {
            row = mat.row;
            column = mat.column;
            m = (int**)malloc(sizeof(int*) * mat.row);
            for (int i = 0; i < mat.row; i++)
                m[i] = (int*)malloc(sizeof(int) * mat.column);
            for (int i = 0; i < row; i++)
                for (int j = 0; j < column; j++)
                    m[i][j] = mat.m[i][j];
        }
        return *this;
    }

    ~Matrix() {
        if (m != NULL) {
            for (int i = 0; i < row; i++) {
                delete[] m[i];
            }
            delete[] m;
        }
    }

};

int tm1[2048 * 2048], tm2[2048 * 2048];

Matrix matAdd(Matrix* matA, Matrix* matB) {
    Matrix matR = Matrix((*matA).row, (*matA).column);
    for (int i = 0; i < (*matA).row; i++)
        for (int j = 0; j < (*matA).column; j++) {
            matR.m[i][j] = (*matA).m[i][j] + (*matB).m[i][j];
        }
    return matR;
}
Matrix matSub(Matrix* matA, Matrix* matB) {
    Matrix matR = Matrix((*matA).row, (*matA).column);
    for (int i = 0; i < (*matA).row; i++)
        for (int j = 0; j < (*matA).column; j++) {
            matR.m[i][j] = (*matA).m[i][j] - (*matB).m[i][j];
        }
    return matR;
}
Matrix matSplit(Matrix mat, int rowStart, int columnStart, int rowEnd, int columnEnd) {
    Matrix matR = Matrix(rowEnd - rowStart + 1, columnEnd - columnStart + 1);
    for (int i = 0; i <= rowEnd - rowStart; i++)
        for (int j = 0; j <= columnEnd - columnStart; j++) {
            matR.m[i][j] = mat.m[rowStart + i][columnStart + j];
        }
    return matR;
}
Matrix matCombine(Matrix mat1, Matrix mat2, Matrix mat3, Matrix mat4) {
    Matrix matR = Matrix(mat1.row + mat3.row, mat1.column + mat2.column);
    for (int i = 0; i < mat1.row; i++)
        for (int j = 0; j < mat1.column; j++)
            matR.m[i][j] = mat1.m[i][j];
    for (int i = 0; i < mat2.row; i++)
        for (int j = 0; j < mat2.column; j++)
            matR.m[i][mat1.column + j] = mat2.m[i][j];
    for (int i = 0; i < mat3.row; i++)
        for (int j = 0; j < mat3.column; j++)
            matR.m[mat1.row + i][j] = mat3.m[i][j];
    for (int i = 0; i < mat4.row; i++)
        for (int j = 0; j < mat4.column; j++)
            matR.m[mat1.row + i][mat1.column + j] = mat4.m[i][j];
    return matR;
}
```

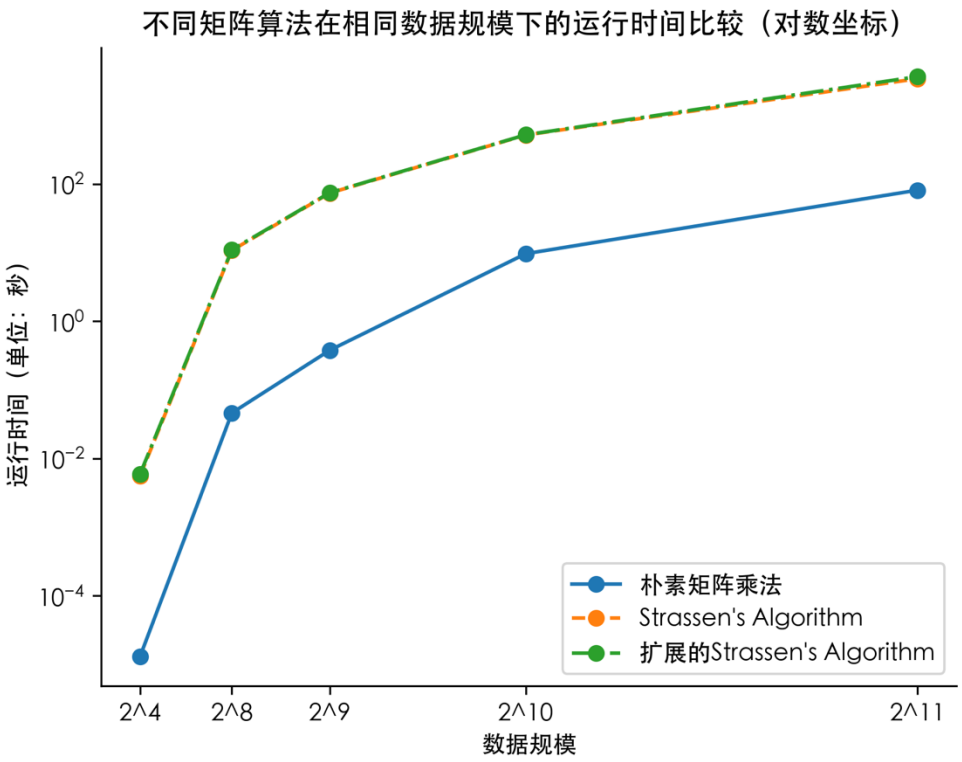
```
Matrix matProduct(Matrix* matA, Matrix* matB, int leafcut) {
    if ((*matA).row <= leafcut && (*matA).column <= leafcut && (*matB).row <= leafcut && (*matB).column <= leafcut) {
        Matrix matR = Matrix(leafcut, leafcut);
            for(int i = 0; i < leafcut; i++) {
                for (int j = 0; j < leafcut; j++) {
                    matR.m[i][j] = 0;
                    for (int k = 0; k < leafcut; k++) {
                        matR.m[i][j] += (*matA).m[i][k] * (*matB).m[k][j];
                    }
                }
            }
                return matR;
            }
            int midR = (*matA).row / 2 - 1;
            int midC = (*matA).column / 2 - 1;
            Matrix a11 = matSplit((*matA), 0, 0, midR, midC);
            Matrix a12 = matSplit((*matA), 0, midC + 1, midR, (*matA).column - 1);
            Matrix a21 = matSplit((*matA), midR + 1, 0, (*matA).row - 1, midC);
            Matrix a22 = matSplit((*matA), midR + 1, midC + 1, (*matA).row - 1, (*matA).column - 1);
            Matrix b11 = matSplit((*matB), 0, 0, midR, midC);
            Matrix b12 = matSplit((*matB), 0, midC + 1, midR, (*matA).column - 1);
            Matrix b21 = matSplit((*matB), midR + 1, 0, (*matA).row - 1, midC);
            Matrix b22 = matSplit((*matB), midR + 1, midC + 1, (*matA).row - 1, (*matA).column - 1);
            Matrix tmp1 = matAdd(&a11, &a22);
            Matrix tmp2 = matAdd(&b11, &b22);
            Matrix tmp3 = matAdd(&a21, &a22);
            Matrix tmp4 = matSub(&b12, &b22);
            Matrix tmp5 = matSub(&b21, &b11);
            Matrix tmp6 = matAdd(&a11, &a12);
            Matrix tmp7 = matSub(&a21, &a11);
            Matrix tmp8 = matAdd(&b11, &b12);
            Matrix tmp9 = matSub(&a12, &a22);
            Matrix tmp10 = matAdd(&b21, &b22);
            Matrix m1 = matProduct(&tmp1, &tmp2, leafcut);
            Matrix m2 = matProduct(&tmp3, &b11, leafcut);
            Matrix m3 = matProduct(&a11, &tmp4, leafcut);
            Matrix m4 = matProduct(&a22, &tmp5, leafcut);
            Matrix m5 = matProduct(&tmp6, &b22, leafcut);
            Matrix m6 = matProduct(&tmp7, &tmp8, leafcut);
            Matrix m7 = matProduct(&tmp9, &tmp10, leafcut);
            Matrix mtmp1 = matAdd(&m1, &m4);
            Matrix mtmp2 = matSub(&mtmp1, &m5);
            Matrix mtmp3 = matSub(&m1, &m2);
            Matrix mtmp4 = matAdd(&mtmp3, &m3);
            Matrix c11 = matAdd(&mtmp2, &m7);
            Matrix c12 = matAdd(&m3, &m5);
            Matrix c21 = matAdd(&m2, &m4);
            Matrix c22 = matAdd(&mtmp4, &m6);
            return matCombine(c11, c12, c21, c22);
        }
        int main() {
            int cnt = 0, n;
            ifstream fin1("data1.txt");
            ifstream fin2("data2.txt");
            while (!fin1.eof() && !fin2.eof()) {
                fin1 >> tm1[cnt];
                fin2 >> tm2[cnt];
                cnt++;
            }
            cnt--;
            n = sqrt(cnt);
            Matrix m1 = Matrix(n, n);
            Matrix m2 = Matrix(n, n);
```

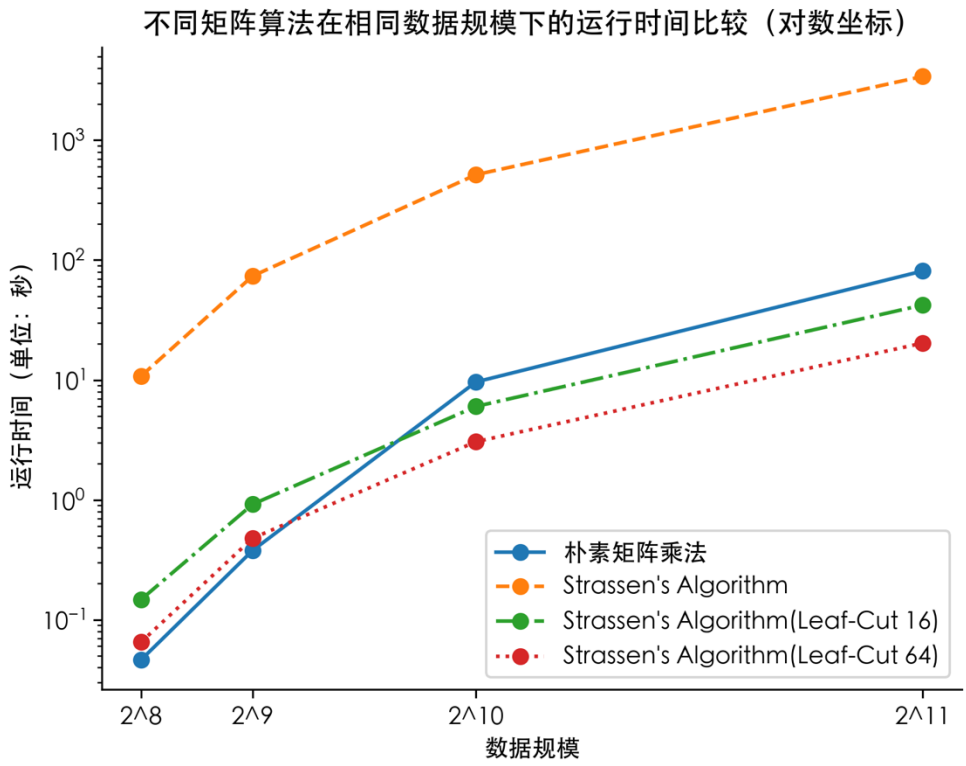
```
159.            Matrix mr = Matrix(n, n);
160.            for (int i = 0; i < n; i++) {
161.                for (int j = 0; j < n; j++) {
162.                    m1.m[i][j] = tm1[i * n + j];
163.                    m2.m[i][j] = tm2[i * n + j];
164.                }
165.            }
166.            mr = matProduct(&m1, &m2, 16);  //可调整剪枝层数
167.            for (int i = 0; i < n; i++) {
168.                for (int j = 0; j < n; j++) {
169.                    cout << mr.m[i][j] << " ";
170.                }
171.                cout << endl;
172.            }
173.            fin1.close();
174.            fin2.close();
175.            return 0;
176.        }
```

2. *分别画出各个实验结果的折线图*

时间记录使用了 C++自带的 clock()函数，通过在程序开头和结尾分别调用 clock()函数并将两值相减，即可得到程序运行时间。结果如下：







五、总结

对上机实践结果进行分析，问题回答，上机的心得体会及改进意见。

从理论上看，Strassen 算法的时间复杂度为$O(n^{log7})$，相比朴素矩阵乘法$O(n^3)$的复杂度略快，但由于其在规模较小时的常数很大，相比朴素矩阵乘法优势并不明显。此外，由于使用 Strassen 算法需要动态申请大量的临时空间，而申请、访问和删除这些辅助空间的代价远大于运算本身的代价（使用 Visual Studio 性能检测工具分析程序可以发现仅创建和删除这些临时空间的耗时就占了整个程序运行时间的 70%左右），因此从实际测试结果来看 Strassen 算法甚至还远慢于朴素矩阵乘法。

函数名	调用数	已用非独占时间百分比	已用独占时间百分比	平均已用非独占时间	平均已用独占时间	模块名
operator delete	10,424,609	21.77%	21.77%	0.00	0.00	Strassen.exe
matProduct	402,653	98.80%	14.57%	30,480.92	0.01	Strassen.exe
_CheckForDebugge...	11,192,209	11.49%	11.49%	0.00	0.00	Strassen.exe
operator delete[]	10,424,609	32.72%	10.95%	0.00	0.00	Strassen.exe
malloc	10,430,579	9.80%	9.80%	0.00	0.00	ucrtbased.dll
Matrix::~Matrix	4,486,575	42.00%	8.72%	0.00	0.00	Strassen.exe
Matrix::Matrix	2,588,512	15.98%	8.66%	0.00	0.00	Strassen.exe
Matrix::Matrix	1,898,243	8.76%	4.49%	0.00	0.00	Strassen.exe
matAdd	690,258	17.47%	2.92%	0.01	0.00	Strassen.exe
__security_check_co...	1,955,900	1.45%	1.45%	0.00	0.00	Strassen.exe
_RTC_CheckStackVars	1,955,899	1.35%	1.35%	0.00	0.00	Strassen.exe
matSplit	460,200	17.22%	1.31%	0.01	0.00	Strassen.exe
matSub	345,136	21.55%	1.24%	0.02	0.00	Strassen.exe
??5?$basic_istream...	131,074	1.12%	0.86%	0.00	0.00	MSVCP140D.dll
std::basic_filebuf<c...	131,074	0.20%	0.18%	0.00	0.00	Strassen.exe
invoke_main	1	100.00%	0.06%	30,852.14	19.41	Strassen.exe
matCombine	57,518	1.40%	0.06%	0.01	0.00	Strassen.exe
std::basic_filebuf<c...	131,074	0.05%	0.02%	0.00	0.00	Strassen.exe
_unlock_file	131,074	0.02%	0.02%	0.00	0.00	ucrtbased.dll
_lock_file	131,074	0.02%	0.02%	0.00	0.00	ucrtbased.dll
?eof@ios_base@std...	131,075	0.01%	0.01%	0.00	0.00	MSVCP140D.dll
main	1	99.94%	0.01%	30,832.72	2.71	Strassen.exe

为了避免常数和内存交互时间对测试结果造成影响，我们可以对原算法进行适当的剪枝（Leaf-Cut）优化，即当矩阵规模缩小到一个固定值后改用朴素矩阵乘法继续处理该矩阵。经实测表明，当剪枝范围为 16～64 之间时，性能瓶颈得到了极大的缓解，且随着数据规模的增大，改进后的算法相比纯朴素矩阵乘法逐渐出现显著的性能优势，与理论计算值基本吻合。

对于矩阵大小不为 2 的幂次时，可以先将原矩阵扩展至最邻近的 2 的幂次大小（扩展部分全部用 0 补齐），随后即可使用 Strassen 算法进行分治计算。