**中国科学技术大学计算机学院**

**《算法基础》实验报告**



实验题目：lab2\_动态规划和FFT

学生姓名：胡毅翔

学生学号：PB18000290

完成日期：2020年12月9日

计算机实验教学中心制

2019年09月

## **实验目的**

1.实现5种排序算法:矩阵链乘和FFT算法。

2.对给定的输入得到正确结果。

3.对获得的实验数据进行分析，并与理论进行比较。

## **实验原理**

本次实验所实现的算法有:martix chain order和FFT。其正确性已在《算法导论》一书中得到证明。

1.martix chain order

void martix\_chain\_order(int \*array, int length)

{

    int n = length;

    int i, j, k, l;

    long long tmp;

    for (i = 1; i <= n; i++)

    {

        m[i][i] = 0;

    }

    for (l = 2; l <= n; l++)

    {

        for (i = 1; i <= n - l + 1; i++)

        {

            j = i + l - 1;

            m[i][j] = LLONG\_MAX;

            for (k = i; k <= j - 1; k++)

            {

                tmp = m[i][k] + m[k + 1][j] + ((long long)array[i - 1] \* (long long)array[k] \* (long long)array[j]);

                if (tmp < m[i][j])

                {

                    m[i][j] = tmp;

                    s[i][j] = k;

                }

            }

        }

    }

}

2.FFT

在复现算法过程中，为使代码结构更加紧凑，在[complex.h](https://github.com/Alpha-Girl/algorithm2020_labs/blob/main/project2/ex2/src/complex.h)中实现复数类及相关操作的定义。

void recursive\_fft(complex \*array, complex \*result, int length)

{

    int n = length;

    if (n == 1)

    {

        equal(array, result);

        return;

    }

    complex omiga\_n, tmp;

    exp\_i(2 / (float)n, &omiga\_n);

    complex omiga = {1, 0};

    complex \*a\_0, \*a\_1, \*y\_0, \*y\_1;

    a\_0 = (complex \*)calloc(n / 2, sizeof(complex));

    a\_1 = (complex \*)calloc(n / 2, sizeof(complex));

    y\_0 = (complex \*)calloc(n / 2, sizeof(complex));

    y\_1 = (complex \*)calloc(n / 2, sizeof(complex));

    for (int i = 0; i < n / 2; i++)

    {

        equal(&array[2 \* i], &a\_0[i]);

        equal(&array[2 \* i + 1], &a\_1[i]);

    }

    recursive\_fft(a\_0, y\_0, n / 2);

    recursive\_fft(a\_1, y\_1, n / 2);

    for (int k = 0; k < n / 2; k++)

    {

        multiply(&omiga, &y\_1[k], &tmp);

        plus(&y\_0[k], &tmp, &result[k]);

        miuns(&y\_0[k], &tmp, &result[k + n / 2]);

        multiply(&omiga\_n, &omiga, &tmp);

        equal(&tmp, &omiga);

    }

    free(a\_0);

    free(a\_1);

    free(y\_0);

    free(y\_1);

    return;

}

## **实验环境**

1.PC一台

2.Windows系统

3.gcc编译器

## **实验过程**

### **目录框架**

本次实验的目录框架如下图所示：

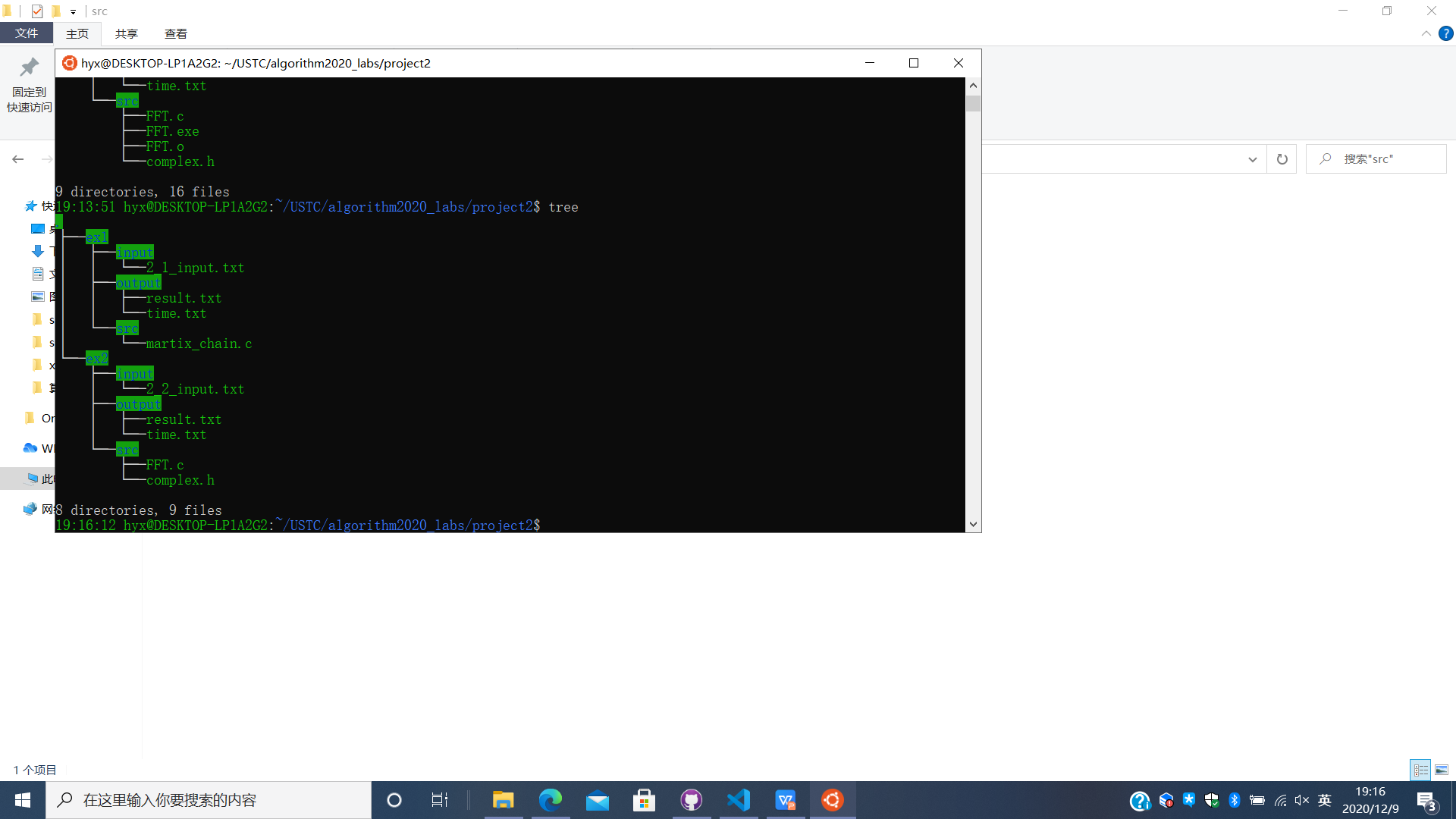


图 1

### **程序执行**

执行martix\_chain.exe：

1.读入一组输入。

2.开始计时。

3.调用martix\_chain\_order()进行求解。

4.结束计时。

5.记录运行时间，保存运行结果。

6.若仍有输入转1，否则结束程序。

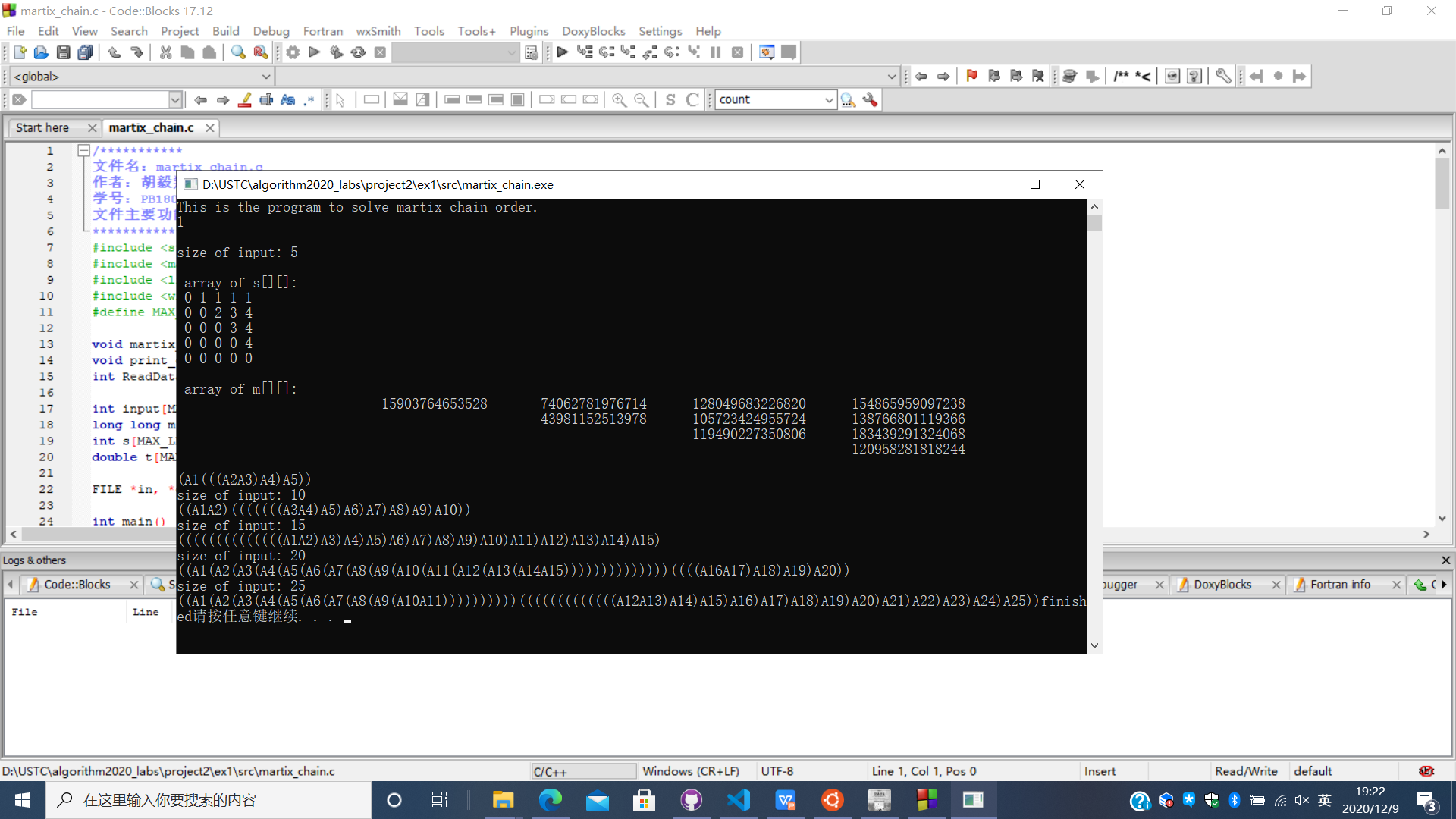


图 2

运行过程截图

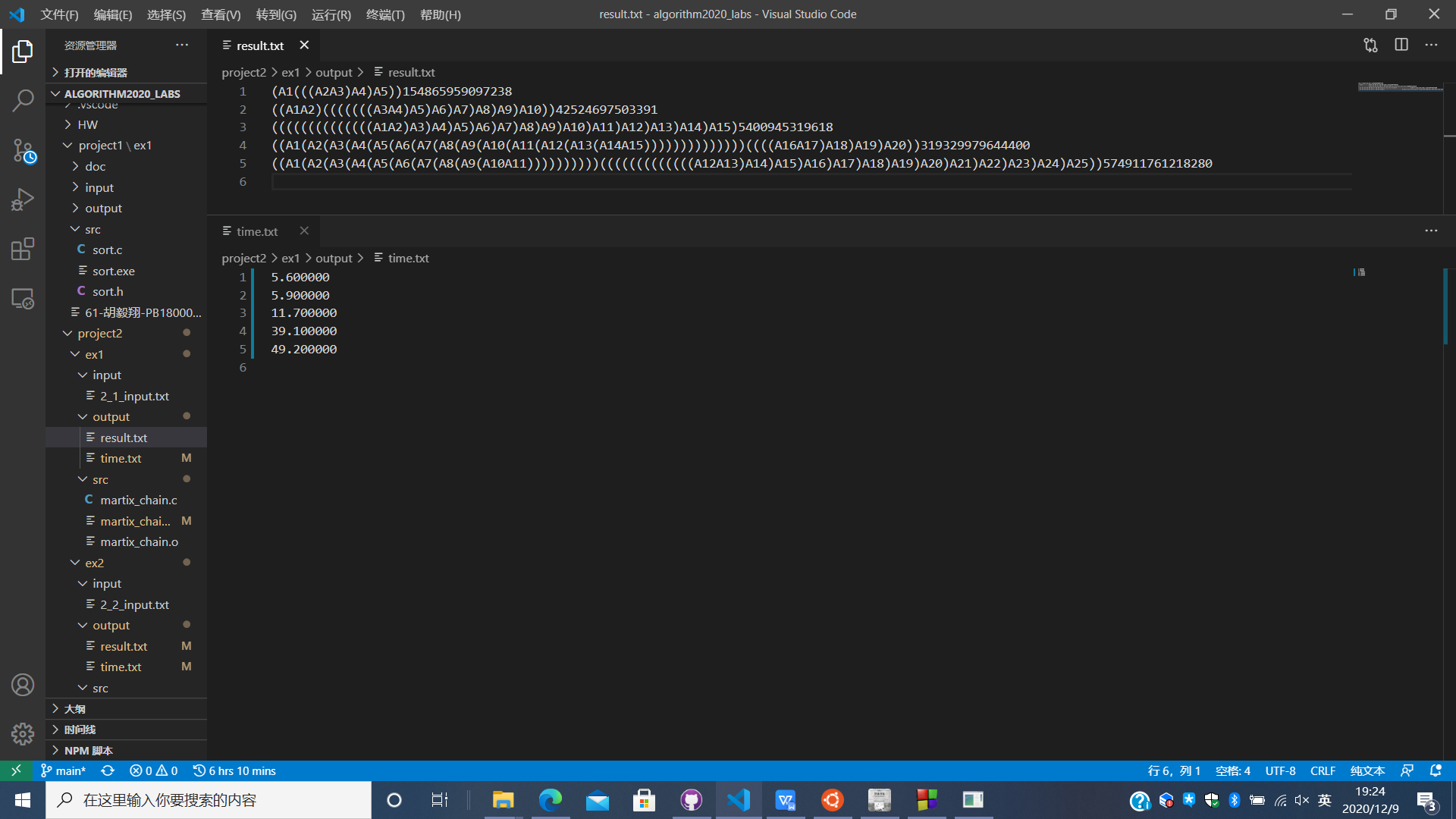


图 3

输出文件结果截图

执行FFT.exe：

1.读入一组输入。

2.开始计时。

3.调用recursive\_fft()进行求解。

4.结束计时。

5.记录运行时间，保存运行结果。

6.若仍有输入转1，否则结束程序。

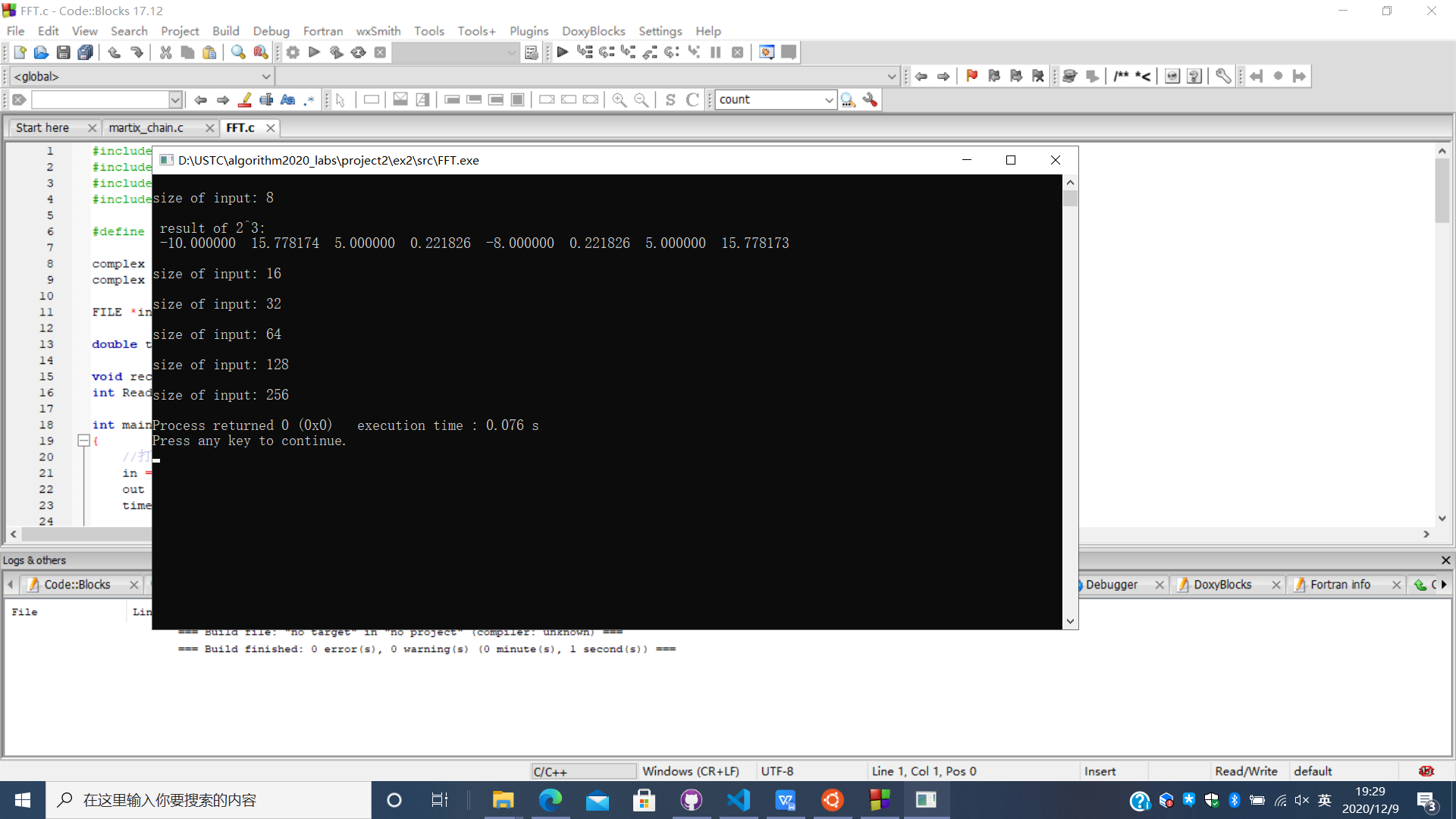


图 4

运行过程截图

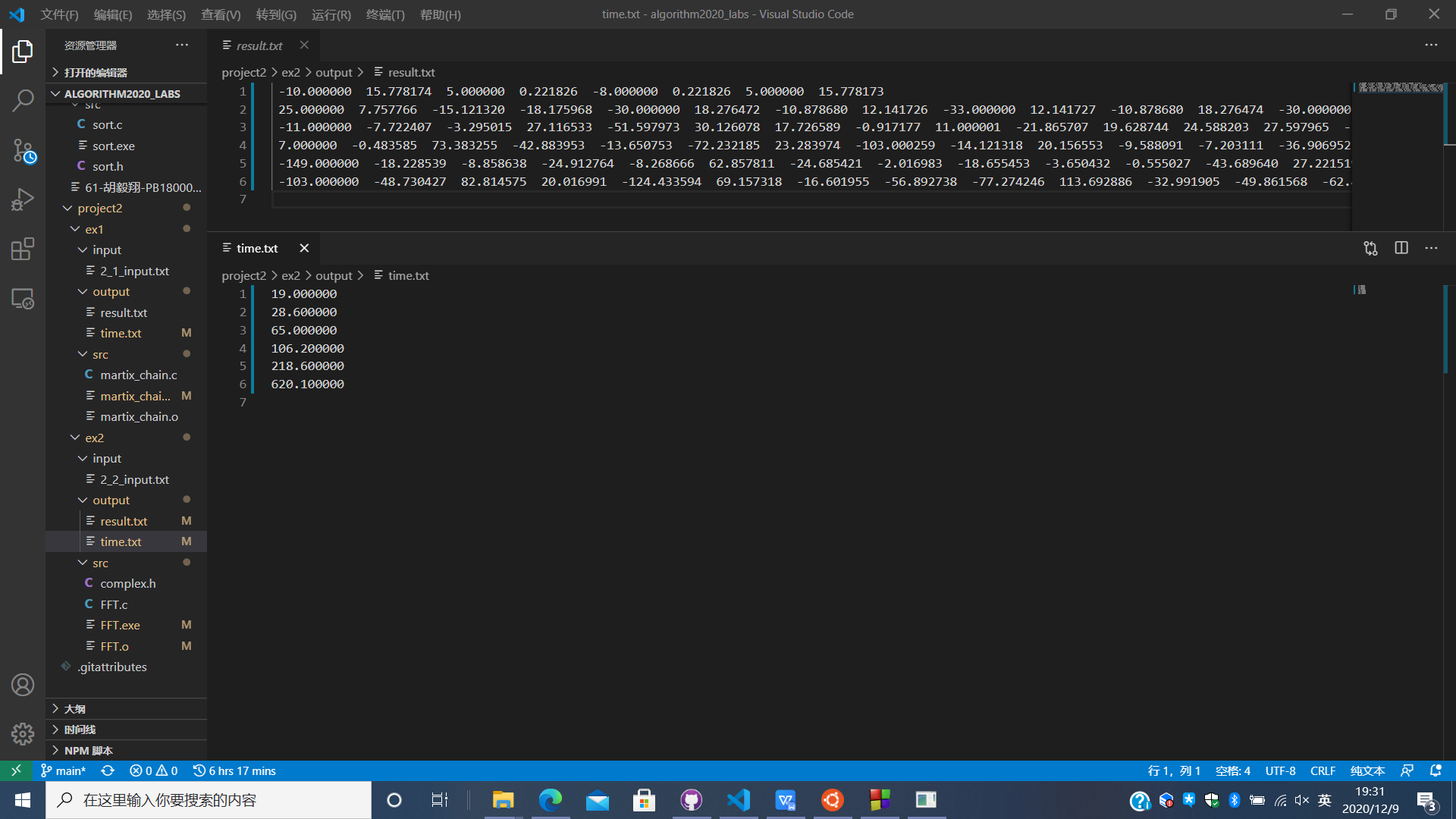


图 5

输出文件结果截图

## **结果分析**

#### **运行时间统计分析**

矩阵链乘运行时间统计：

|  |  |
| --- | --- |
| 输入规模 | T/μs |
| 5 | 5.600000 |
| 10 | 5.900000 |
| 15 | 11.700000 |
| 20 | 39.100000 |
| 25 | 49.200000 |

表 1

图 6

与理论的拟合程度较好，可以认为满足输入规模的三次方与运行时间成正比。

FFT运行时间统计：

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| 8 | 19 | 24 | 1.263157895 |
| 16 | 28.6 | 64 | 2.237762238 |
| 32 | 65 | 160 | 2.461538462 |
| 64 | 106.2 | 384 | 3.615819209 |
| 128 | 218.6 | 896 | 4.098810613 |

表 2

图 7

结合图象及数据可知，本次实验中FFT运行时间与理论一致。