**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：lab1\_排序算法

学生姓名：胡毅翔

学生学号：PB18000290

完成日期：2020年11月8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:Insertion sort, Merge Sort, Heap Sort, Quick Sort, Counting Sort。

2.对不同规模的序列使用上述算法进行排序，统计排序时间。

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**

本次实验所用的排序算法有:Insertion sort, Merge Sort, Heap Sort, Quick Sort, Counting Sort。其正确性已在《算法导论》一书中得到证明。

具体实现(实现中的调用函数未包含于实验报告中，详见[sort.h](https://github.com/Alpha-Girl/algorithm2020_labs/blob/main/project1/ex1/src/sort.h))如下：

1.插入排序

void InsertionSort(int \*a, int n)

{

    for (int j = 1; j < n; j++)

    {

        int key = a[j];

        int i = j - 1;

        while (i >= 0 && a[i] > key)

        {

            a[i + 1] = a[i];

            i = i - 1;

        }

        a[i + 1] = key;

    }

}

2.归并排序

void Merge(int \*a, int start, int q, int end)

{

    int m = q - start + 1;

    int n = end - q;

    int \*L, \*R, i, j;

    L = (int \*)calloc(m + 1, sizeof(int));

    R = (int \*)calloc(n + 1, sizeof(int));

    for (i = 0; i < m; i++)

        L[i] = a[start + i];

    for (j = 1; j <= n; j++)

        R[j - 1] = a[q + j];

    L[m] = INT\_MAX;

    R[n] = INT\_MAX;

    i = 0;

    j = 0;

    for (int k = start; k <= end; k++)

    {

        if (L[i] <= R[j])

        {

            a[k] = L[i];

            i++;

        }

        else

        {

            a[k] = R[j];

            j++;

        }

    }

    free(L);

    free(R);

}

3.堆排序

void HeapSort(int \*a, int n)

{

    int size = n;

    BuildMaxHeap(a, n);

    for (int i = n; i > 1; i--)

    {

        exchange(&a[1], &a[i]);

        size--;

        MaxHeapify(a, 1, size);

    }

}

4.快速排序

void QuickSort(int \*a, int start, int end)

{

    if (start < end)

    {

        int q = Partition(a, start, end);

        QuickSort(a, start, q - 1);

        QuickSort(a, q + 1, end);

    }

}

5.计数排序

void CountingSort(int \*a, int \*b, int length, int n)

{

    int i, j;

    int \*c = (int \*)calloc(length + 1, sizeof(int));

    for (i = 0; i <= length; i++)

        c[i] = 0;

    for (j = 0; j < n; j++)

        c[a[j]] = c[a[j]] + 1;

    for (i = 1; i <= length; i++)

        c[i] = c[i] + c[i - 1];

    for (j = n - 1; j >= 0; j--)

    {

        b[c[a[j]] - 1] = a[j];

        c[a[j]] = c[a[j]] - 1;

    }

    free(c);

}

## **实验环境**

1.PC一台

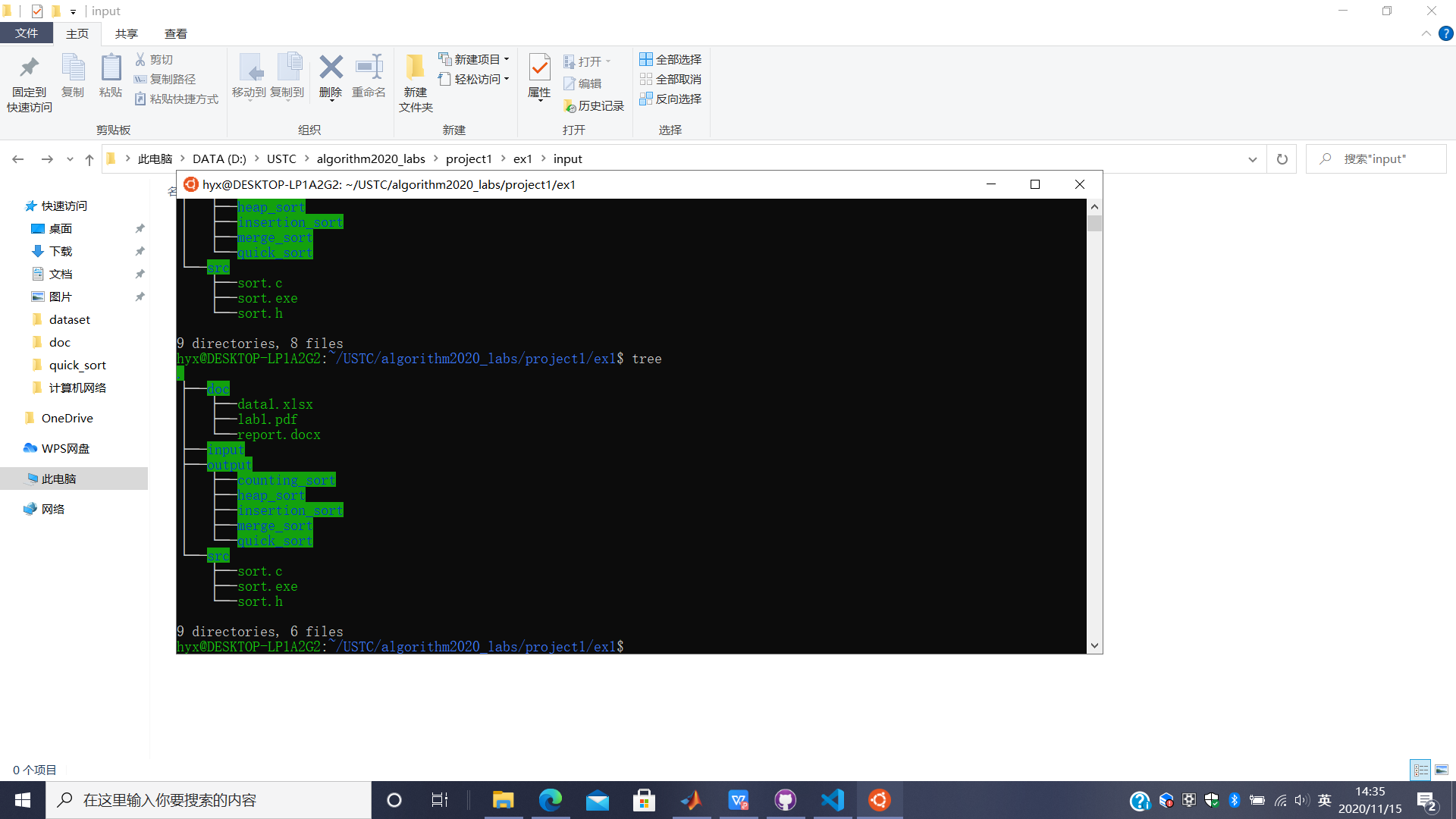
2.Windows系统

3.gcc编译器

## **实验过程**

### **目录框架**

本次实验的目录框架(执行sort.exe前)如下图所示：



### **程序执行**

执行sort.exe：

1.根据时间播种随机数，将218个随机数写入input目录下的input.txt文件保存。

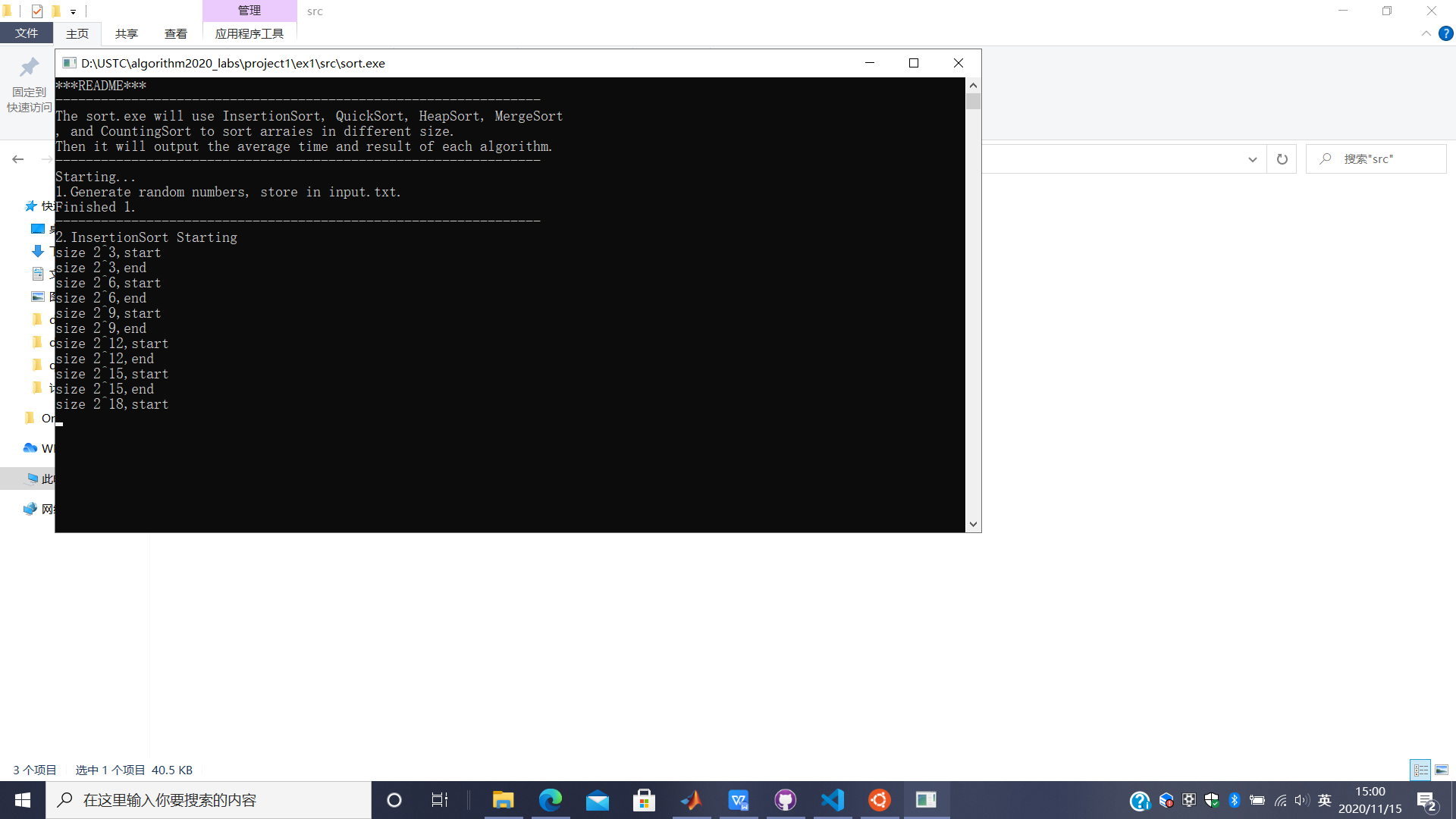
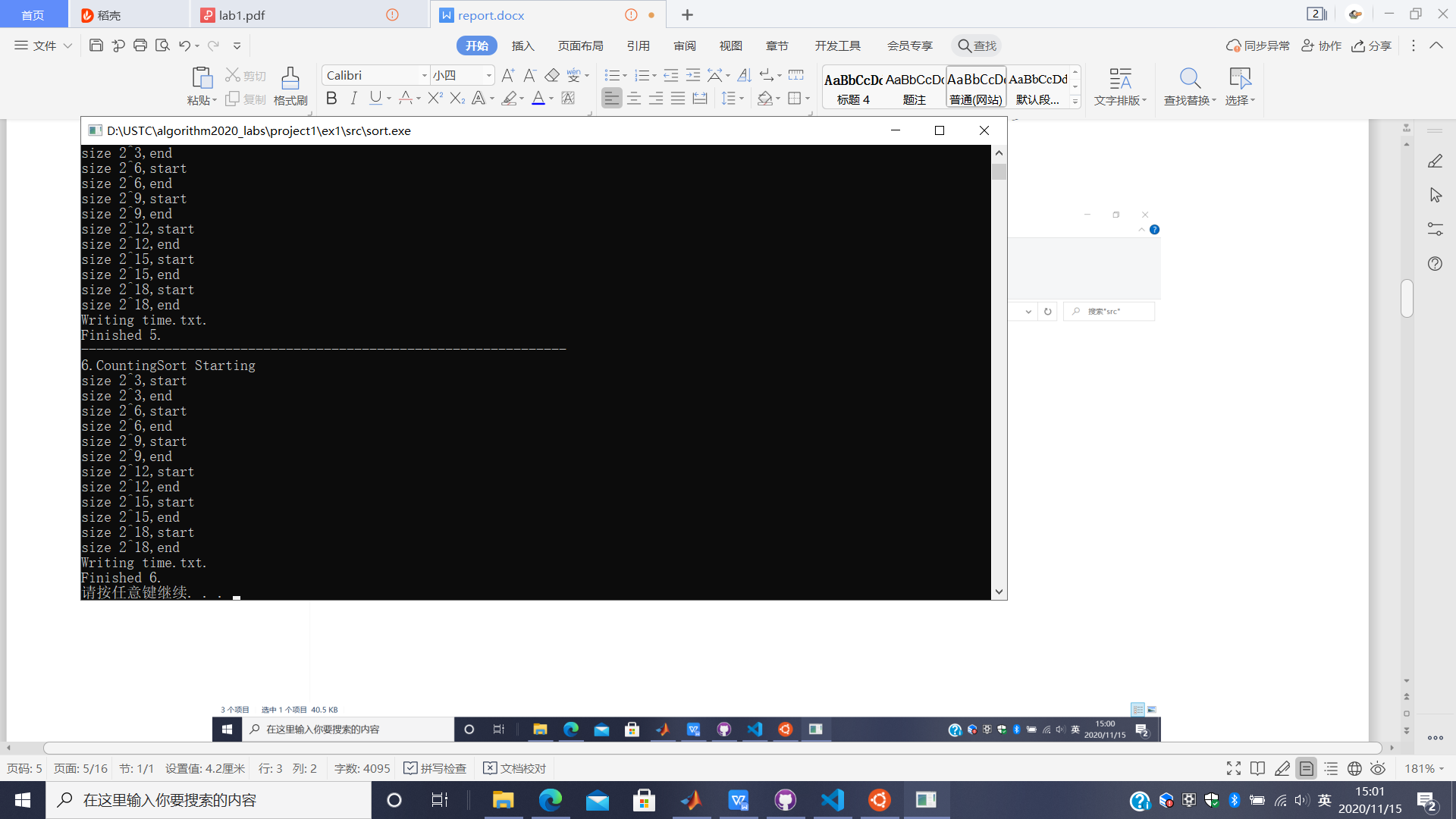
2.从中读取前23个数，放入待排序数组，开始计时，调用InsertionSort()，结束计时。

3.将排序结果写入/output/insertion\_sort/result\_3.txt，计时求差并暂存于计时数组。

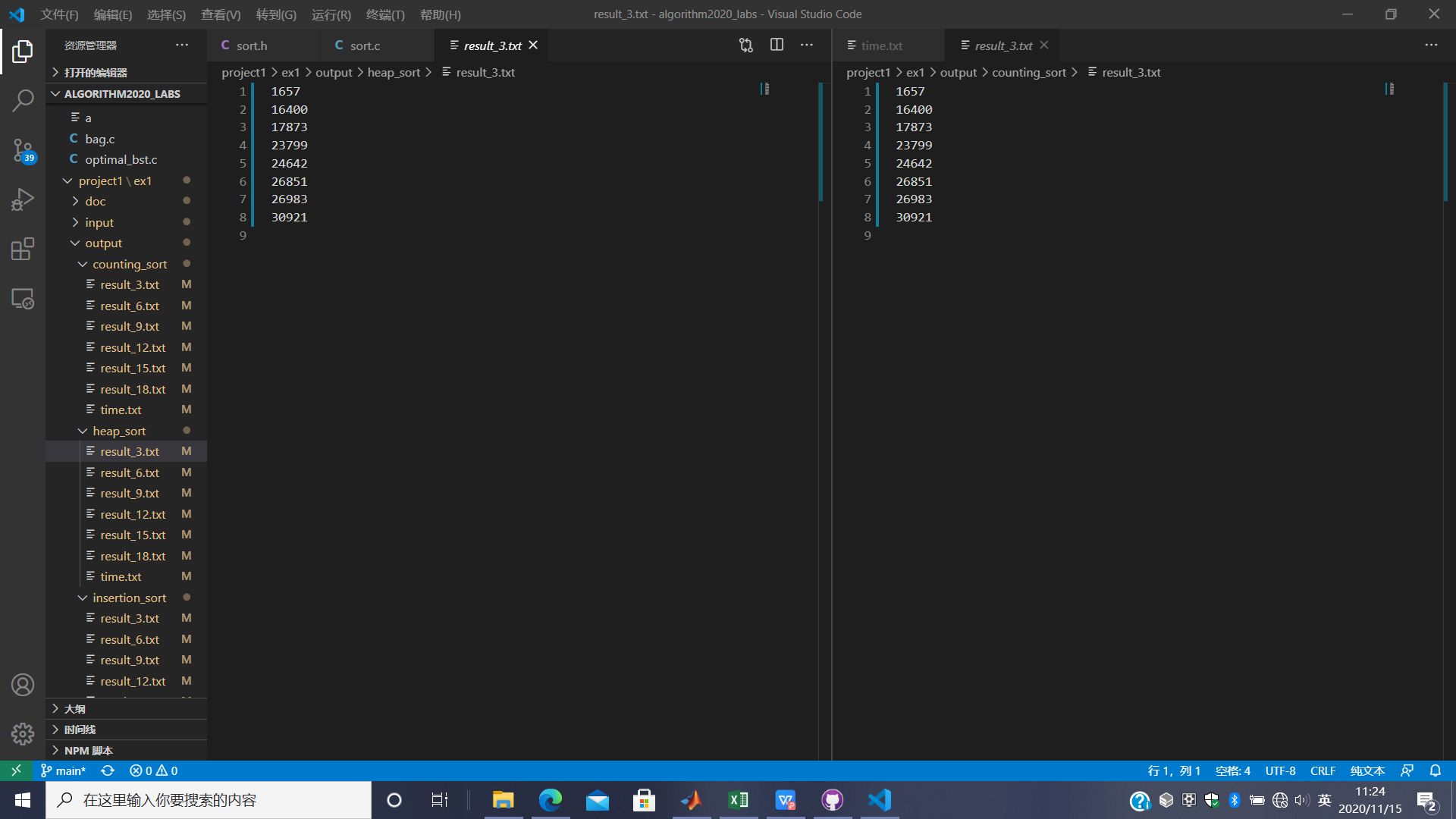
4.修改规模为26，29，212，215，218重复2-3步。

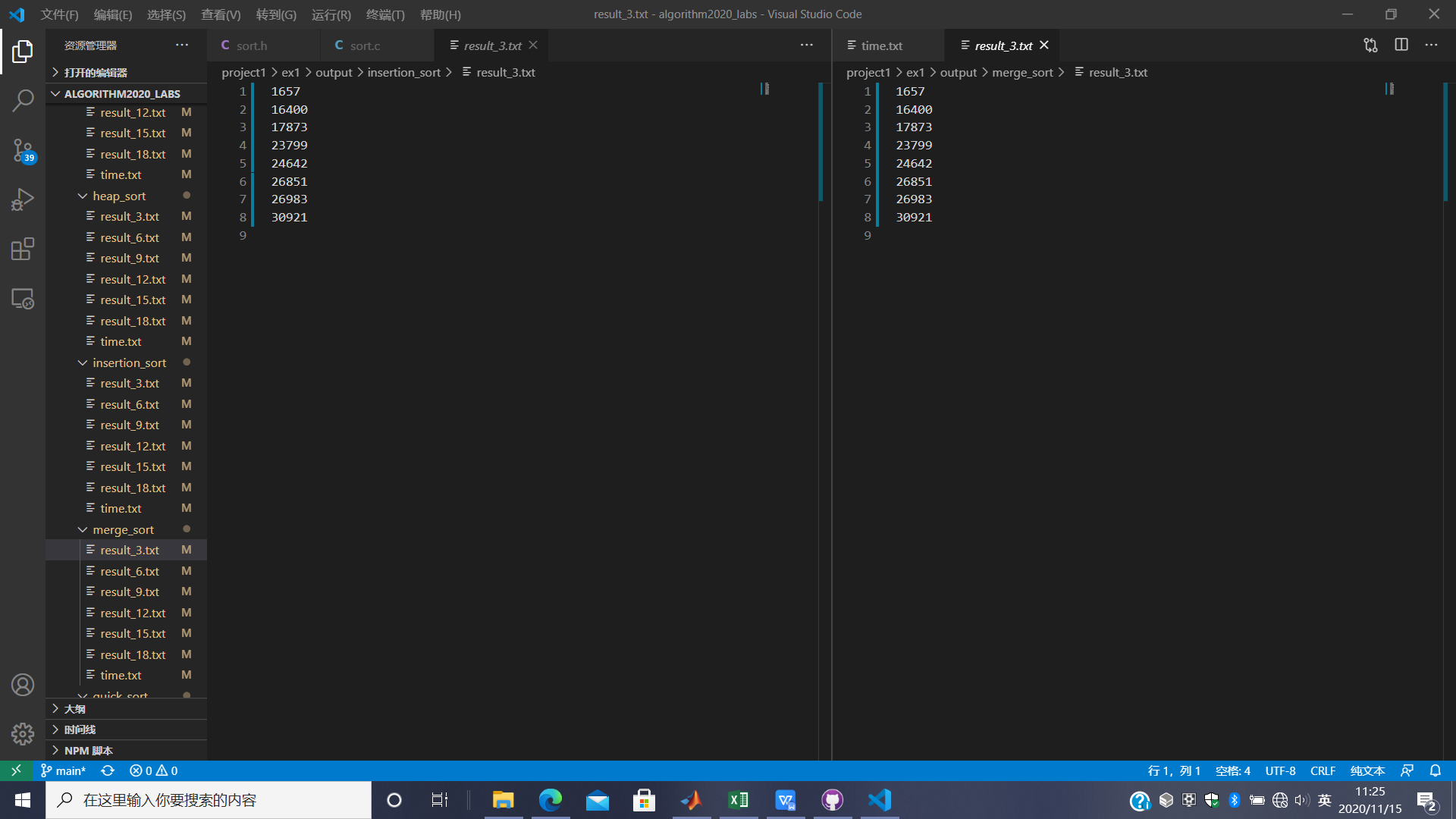
5.修改排序算法为MergeSort()，Heap Sort()， Quick Sort()， Counting Sort()，重复执行2-4步。

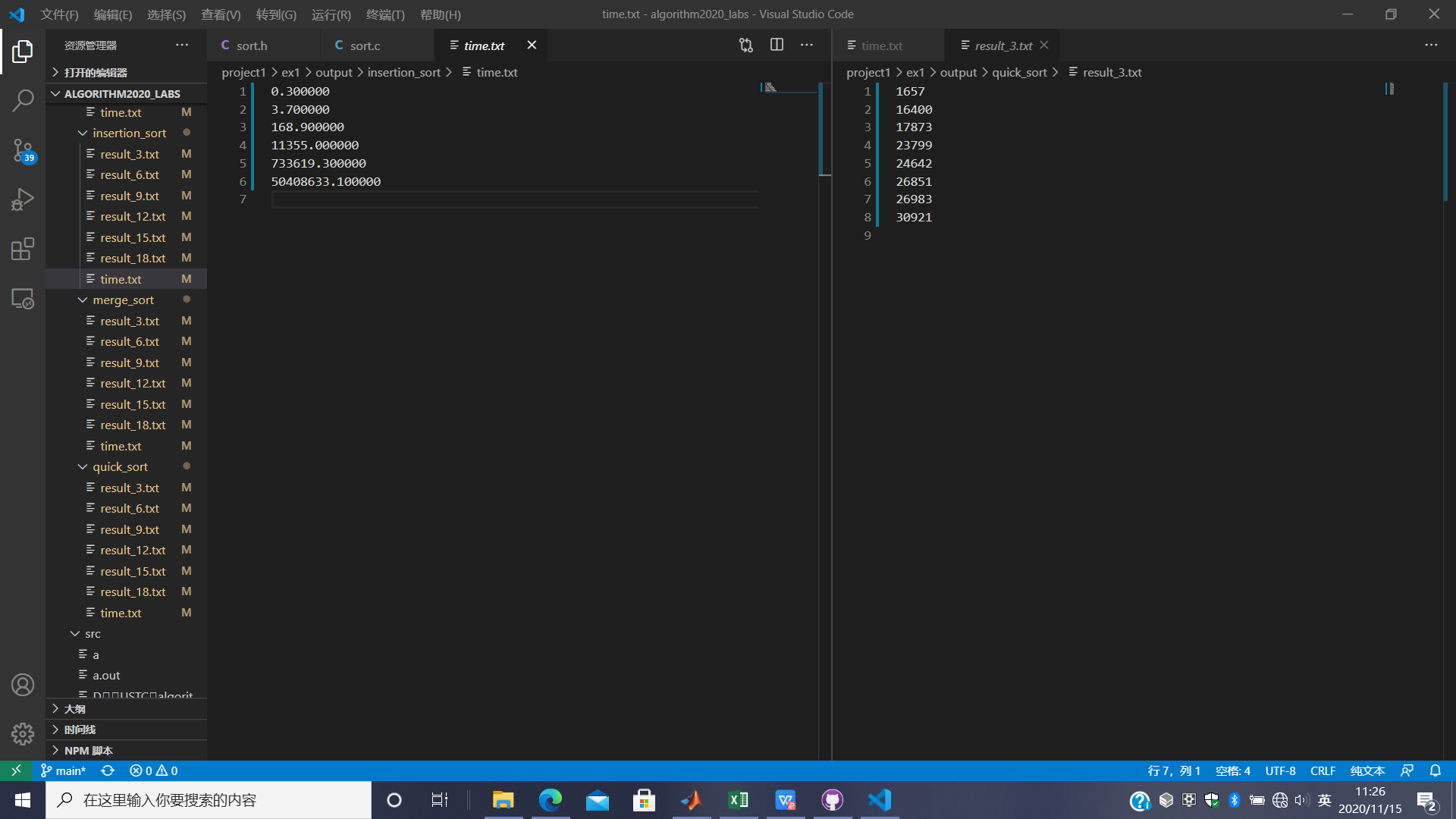
6.程序结束。

运行过程截图







排序结果及计时结果截图

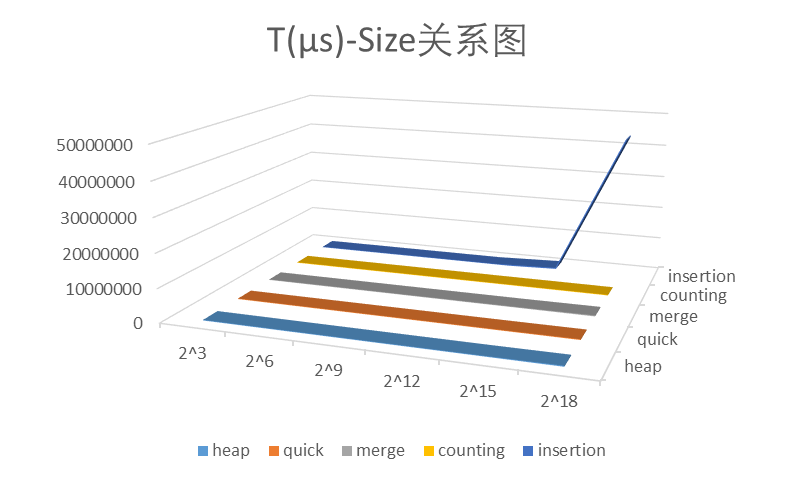
## **结果分析**

#### **运行时间统计分析**

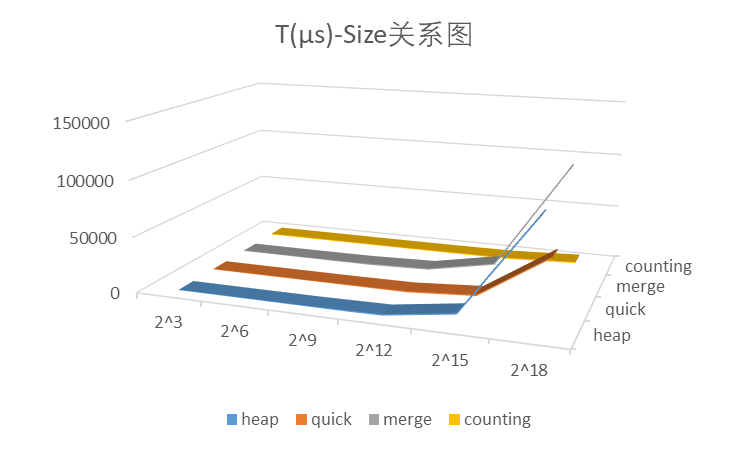
运行程序5次，计算求得各种算法在不同输入规模下的平均运行时间(如表1)。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| t(μs) | heap | quick | merge | counting | insertion |
| 23 | 1.08 | 0.52 | 4.34 | 298.32 | 0.56 |
| 26 | 7.78 | 4.14 | 30.14 | 270.38 | 3.58 |
| 29 | 109.44 | 57.12 | 310.32 | 190.72 | 181.92 |
| 212 | 876.78 | 439.84 | 1992.84 | 246.06 | 11084.94 |
| 215 | 9670.5 | 4472.8 | 13535.5 | 624.82 | 660223.86 |
| 218 | 94451.12 | 44662.26 | 104914.4 | 3314.78 | 43422953.1 |

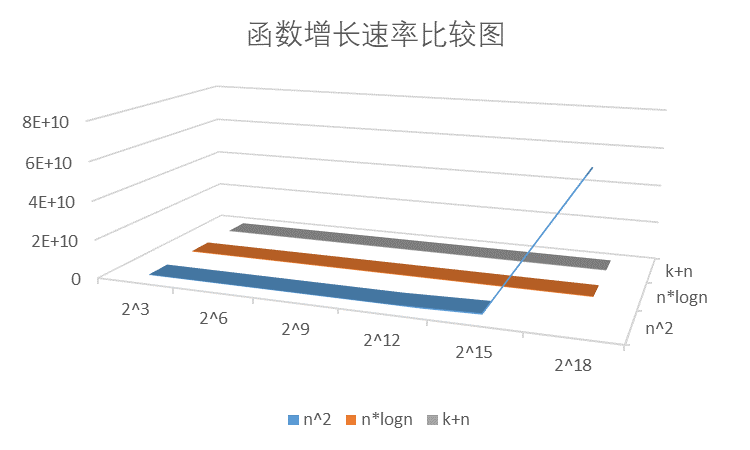
表 1

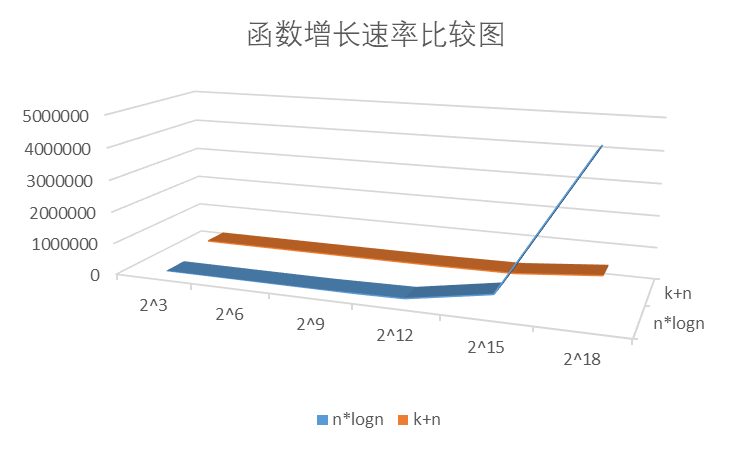


因insertion sort的运行时间数量级较大，无法明显区分其余四个排序算法的差异，故其余四个算法另外作图。



此外做出n2，nlogn，k+n的函数增长速率关系图，与实验结果加以比较。





通过观察比较可知，本次实验实现的算法的运行时间，与理论估计基本相同：Insertion sort的时间复杂度为；Merge Sort，Heap Sort，Quick Sort的时间复杂度为；Counting Sort的时间复杂度为。

在数据范围固定，数据规模较大时，Counting Sort优势明显，但在数据规模较小时，则不适用。

在时间复杂度同为的三个算法中，Quick Sort的运行时间与其余两个算法相比较低，可能的原因是：相较Merge Sort，不需要内存的分配和释放；相较Heap Sort，访问低一级别存储空间的次数较少。故三种算法虽然时间复杂度相同，但考虑机器本身存储空间的特性，Quick Sort的性能更优。

插入排序在数据规模较小时表现较好，但随着规模增大，性能急剧下降。