**一，实验内容**

用C/C++语言实现插入排序算法、归并排序算法快速排序算法和堆排序算法（假定按递增排序方式）。随机生成具有以下规模n=100, 1000, 5000, 10000, 50000, 100000, 500000, 1000000等的数组A, 对数组A运行以上算法排序并对比相应的时间开销。

**二，实验目的**

通过实验，更好地认识插入排序、归并排序、快速排序和堆排序等排序算法随着输入实例规模不断增大时其时间复杂性变化情况。

**三，实验过程**

#include<iostream>

#include<cstdlib>

#include<ctime>

#include<algorithm>

using namespace std;

#define N 1000000

//这个地方的N可以进行更换，来达到不同数量的计算

//插入函数

void insertsort(int arr[],int n){

    for(int i=1;i<n;i++){

        int temp=arr[i];

        int j=i-1;

        while (j>=0&&arr[j]>temp){

            arr[j+1]=arr[j];

            j--;

        }

        arr[j+1]=temp;

    }

}

//快速排序

int partition (int arr[],int low,int high){

    int pivot = arr[(low+high)/2];

    int i=low-1;

    int j=high+1;

    while(true){

        do{

            i++;

        }while(arr[i]<pivot);

        do{

            j--;

        }while(arr[j]>pivot);

        if(i>=j) return j;

        swap(arr[i],arr[j]);

    }

}

void quicksort(int arr[],int low,int high){

    if(low>=high) return ;

    int p= partition(arr,low,high);

    quicksort(arr,low,p);

    quicksort(arr,p+1,high);

}

//归并排序

void merge(int arr[],int low,int mid,int high,int temp[]){

    int i=low,j=mid+1,k=0;

    while(i<=mid&&j<=high){

        if(arr[i]<=arr[j]) temp[k++]=arr[i++];

        else temp[k++]=arr[j++];

    }while (i <= mid) temp[k++] = arr[i++];

    while (j <= high) temp[k++] = arr[j++];

    for (i = low, k = 0; i <= high; i++, k++) arr[i] = temp[k];

}

void merge\_sort(int arr[], int low, int high, int temp[]) {

    if (low >= high) return;

    int mid = low + (high - low) / 2;

    merge\_sort(arr, low, mid, temp);

    merge\_sort(arr, mid + 1, high, temp);

    merge(arr, low, mid, high, temp);

}

//自底向上归并排序

void BottomUpSort(int arr[], int n) {

    int\* temp = new int[n];

    for (int width = 1; width < n; width \*= 2) {

        for (int i = 0; i < n; i += 2 \* width) {

            int left = i;

            int mid = min(left + width - 1, n - 1);

            int right = min(left + 2 \* width - 1, n - 1);

            merge(arr, left, mid, right, temp);

        }

    }

    delete[] temp;

}

//主函数

int main() {

    int size = N;

    int\* R = new int[size];

    clock\_t time\_start, time\_end;

    // 插入排序

    srand(time(NULL));

    for (int i = 0; i < N; i++) R[i] = rand();

    time\_start = clock();

    insertsort(R, N);

    time\_end = clock();

    cout << "Insertion Sort: " << time\_end - time\_start << "ms" << endl;

    // 快速排序

    for (int i = 0; i < N; i++) R[i] = rand();

    time\_start = clock();

    quicksort(R, 0, N - 1);

    time\_end = clock();

    cout << "Quick Sort: " << time\_end - time\_start << "ms" << endl;

    // 归并排序

    int\* temp = new int[N];

    for (int i = 0; i < N; i++) R[i] = rand();

    time\_start = clock();

    merge\_sort(R, 0, N - 1, temp);

    time\_end = clock();

    cout << "TopDown Merge: " << time\_end - time\_start << "ms" << endl;

    delete[] temp;

    //自底向上归并排序

    for (int i = 0; i < N; i++) R[i] = rand();

    time\_start = clock();

    BottomUpSort(R, N);

    time\_end = clock();

    cout << "BottomUp Merge: " << time\_end - time\_start << "ms" << endl;

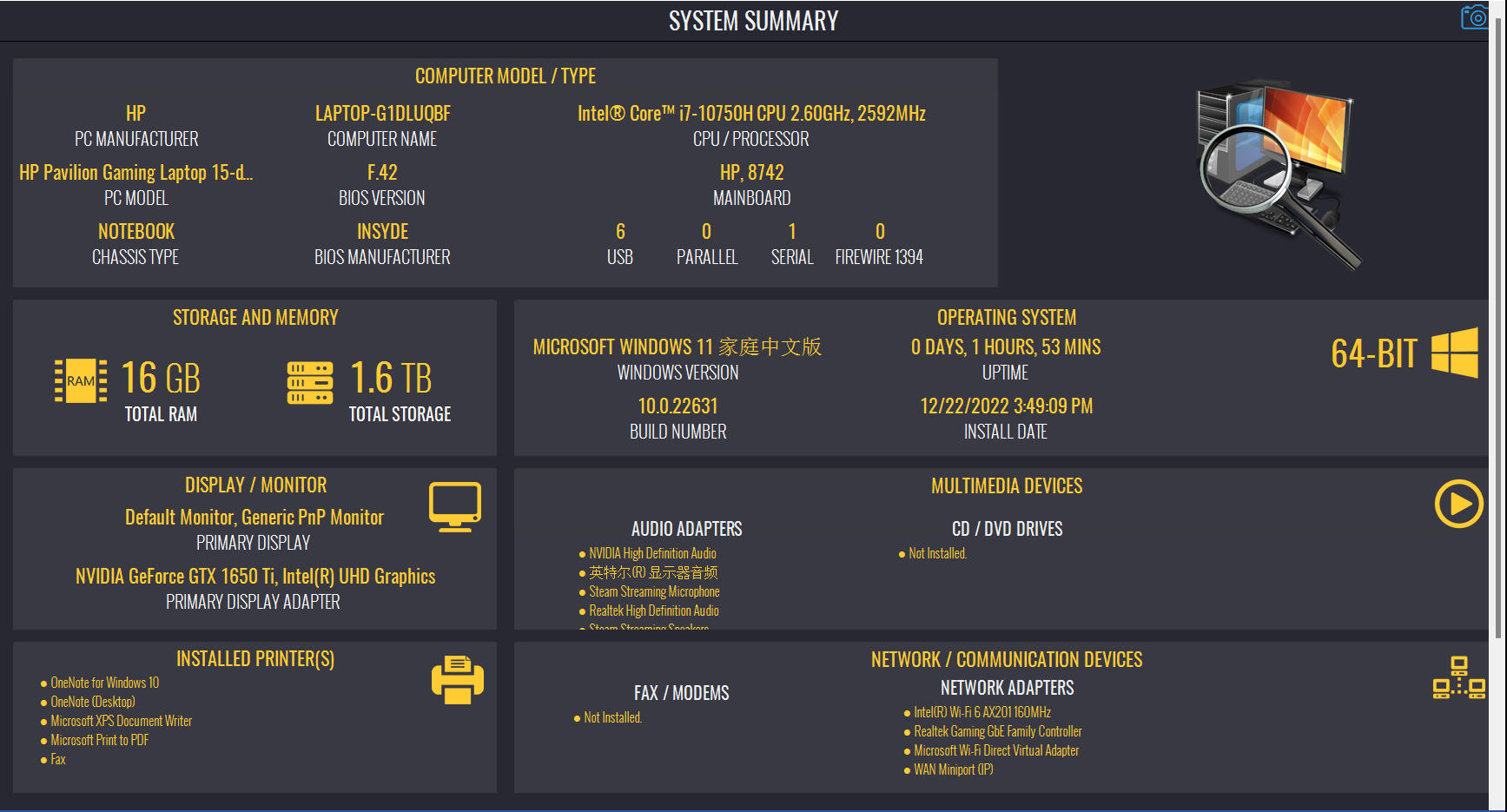
    delete[] R;

    return 0;

}

**四，实验环境**

计算机硬件情况：



使用的软件：

VS Code 1.97.2 版本。

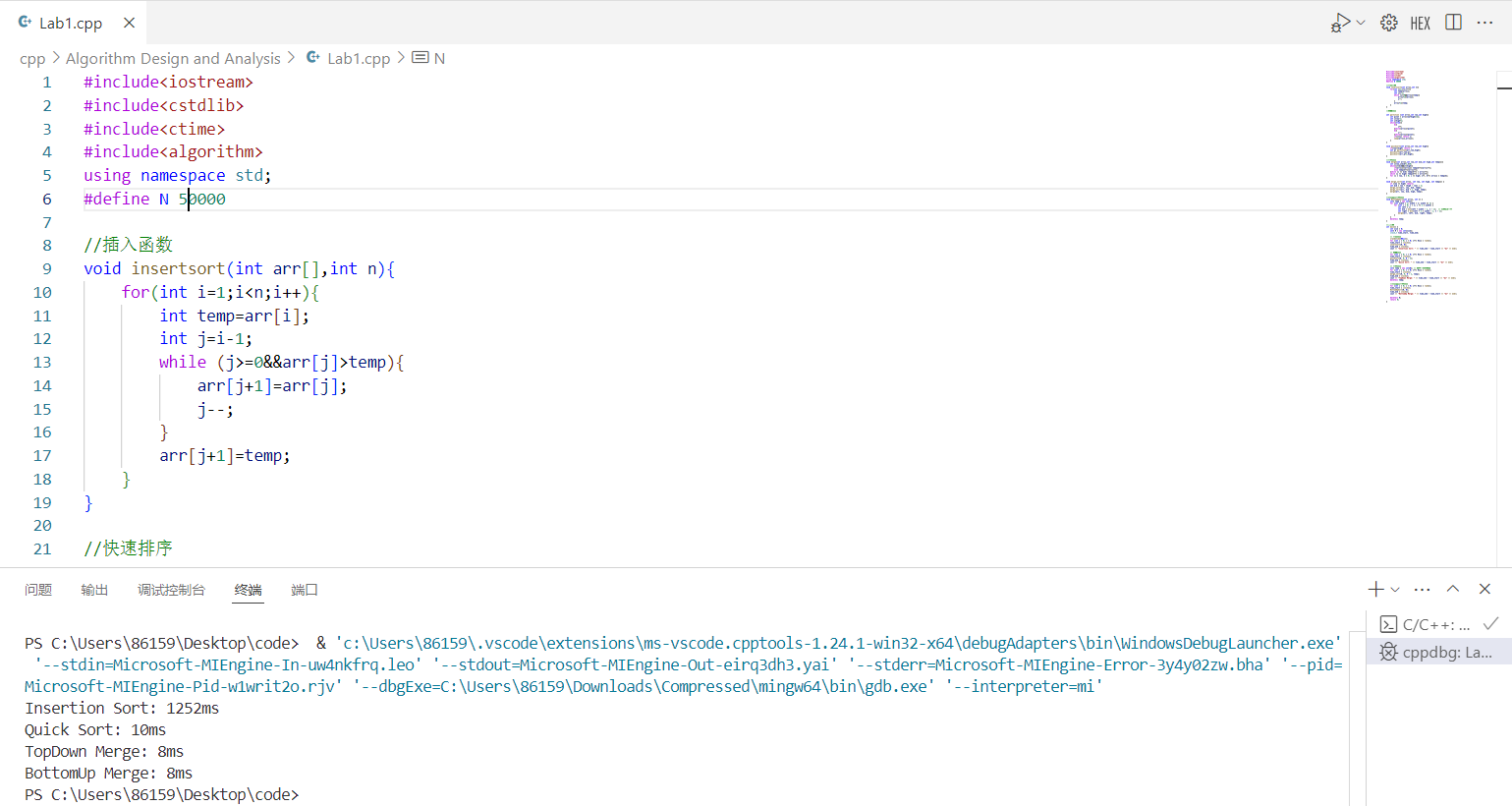
**五，实验结果**



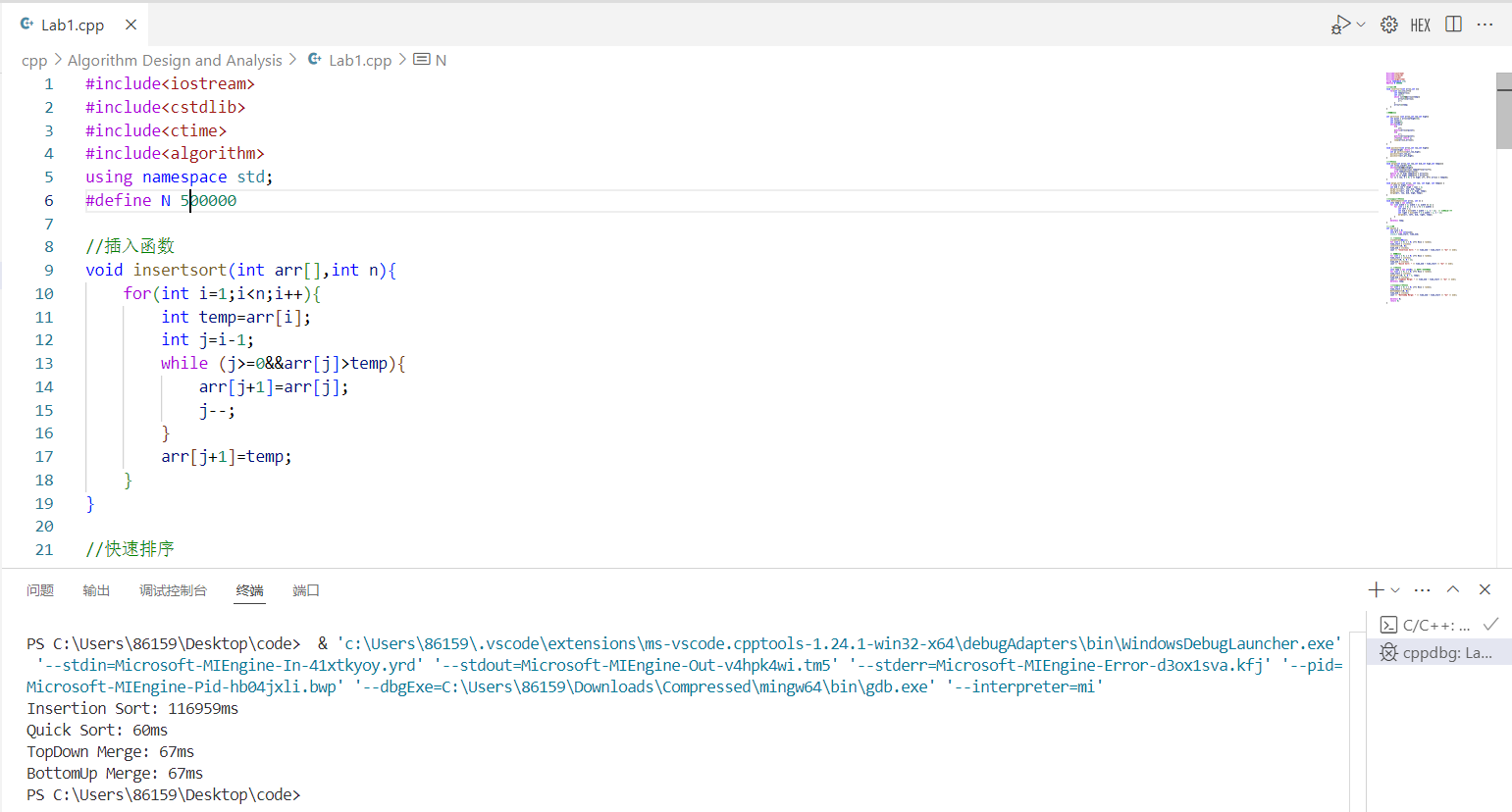


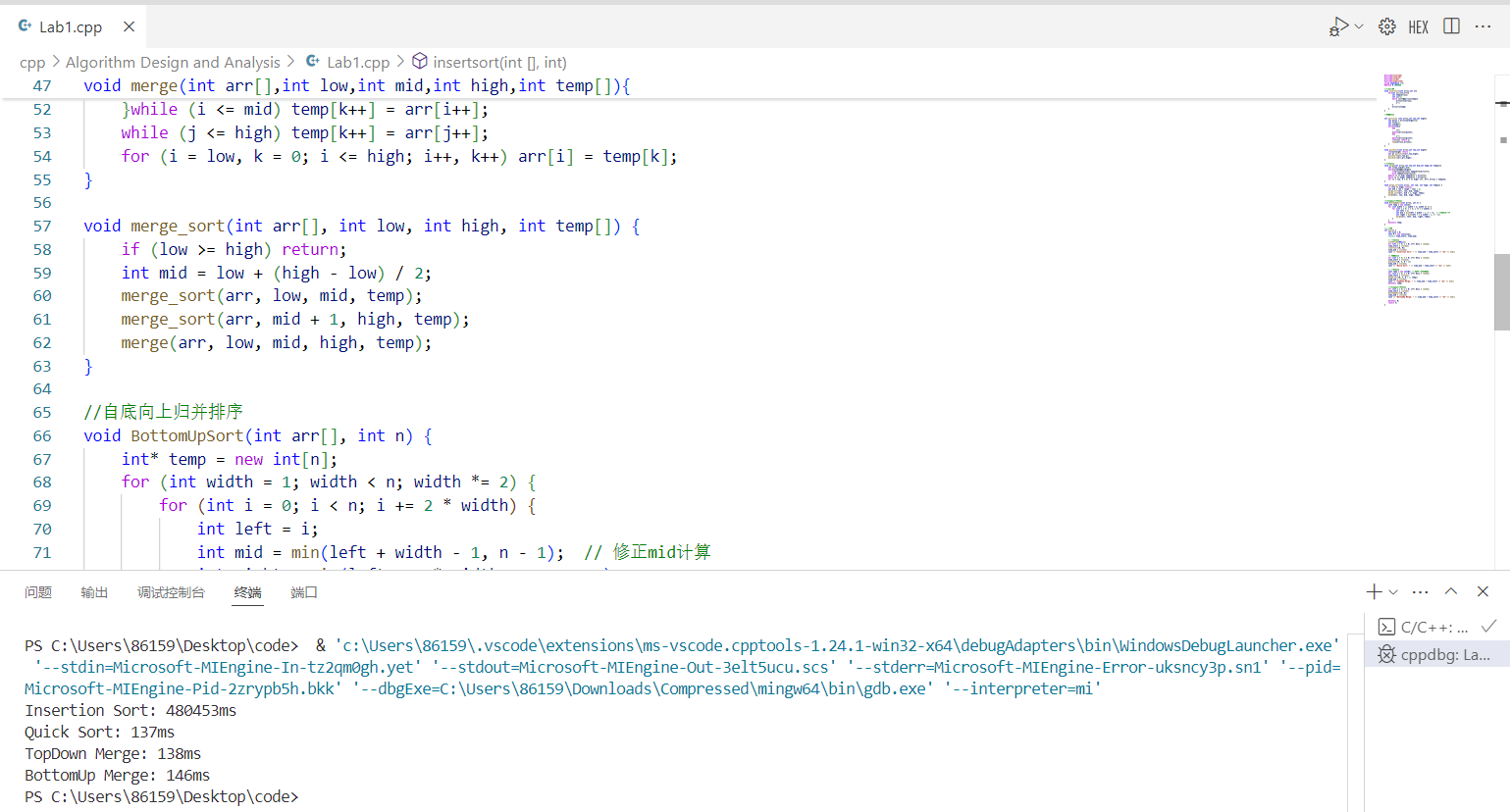






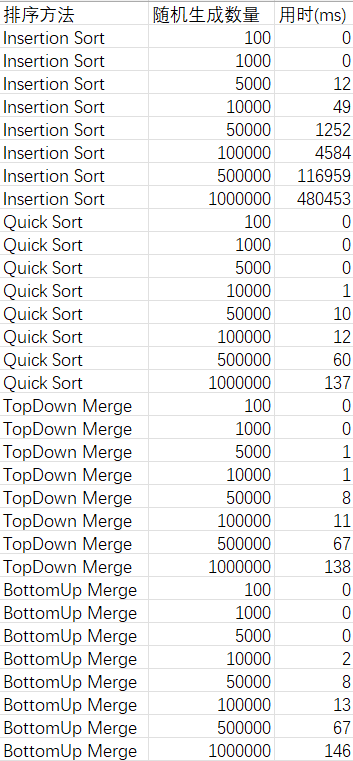






**六，实验结果说明**

通过实验结果以及统计图表我们不难看出，在处理数据比较少的排序中，这几种排序所用的时间其实并没有明显的差别，但是在数量级往上走之后，我们可以非常明显的发现，快速排序的巨大优点就是排序时间非常的短，插入排序耗时会愈发的大，而其

他的两种排序，在百万级别及以下看不出明显的差别。

**七，实验总结**

本次实验不仅在代码上，要求我们掌握四种代码的编码方法，同时要求我们学会用生成随机数字代码，以及记录时间，是一个相当综合性质的代码编写实验。通过这次实验，让我掌握了以上基本的实验操作方法，同时学习了如何利用统计，表格的手段，来辅助处理数据，得出严谨的实验结果。