

中国科学技术大学计算机学院

算法基础实验报告

实验一

排序算法

学 号： PB18000203

姓 名： 汪洪韬

专 业： 计算机科学与技术

指导老师： 顾乃杰

中国科学技术大学计算机学院

2020 年 11 月 14 日

一、 实验内容

1. 排序 n 个元素，元素为随机生成的 0 到 $2^{15} - 1$ 之间的整数， n 的取值为： 2^3 ， 2^6 ， 2^9 ， 2^{12} ， 2^{15} ， 2^{18} 。

2. 实现以下算法：直接插入排序，堆排序，快速排序，归并排序，计数排序。

二、 实验设备和环境

1. 实验设备：PC 机；

2. 实验环境：Visual Studio 2019

三、 实验方法和步骤

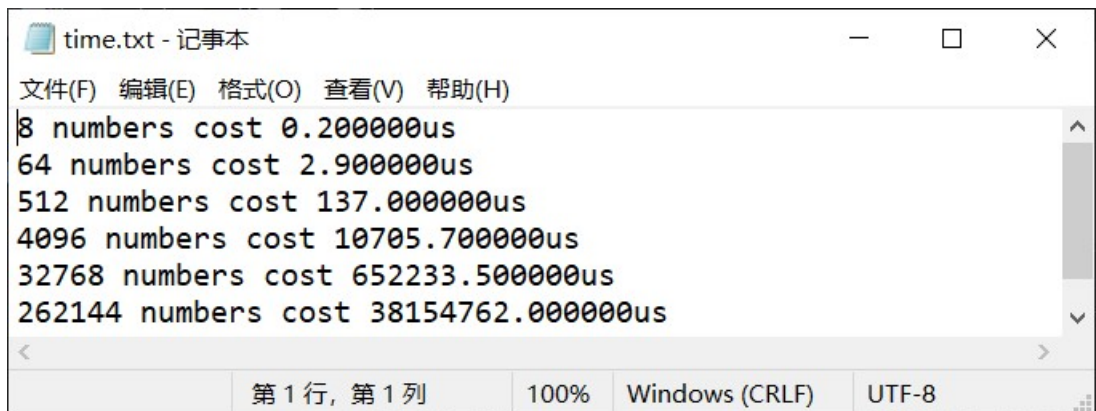
1. 插入排序：

排序代码：

```
void insertsort(int *a, int n) {
    int i, j, key;
    for (i = 1; i < n; i++) {
        key = a[i];
        j = i - 1;
        while ((j ≥ 0) && (a[j] > key)) {
            a[j + 1] = a[j];
            j--;
        }
        a[j + 1] = key;
    }
}
```

排序结果与用时：





```
time.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
8 numbers cost 0.200000us
64 numbers cost 2.900000us
512 numbers cost 137.000000us
4096 numbers cost 10705.700000us
32768 numbers cost 652233.500000us
262144 numbers cost 38154762.000000us
第 1 行, 第 1 列 100% Windows (CRLF) UTF-8
```

2.堆排序:

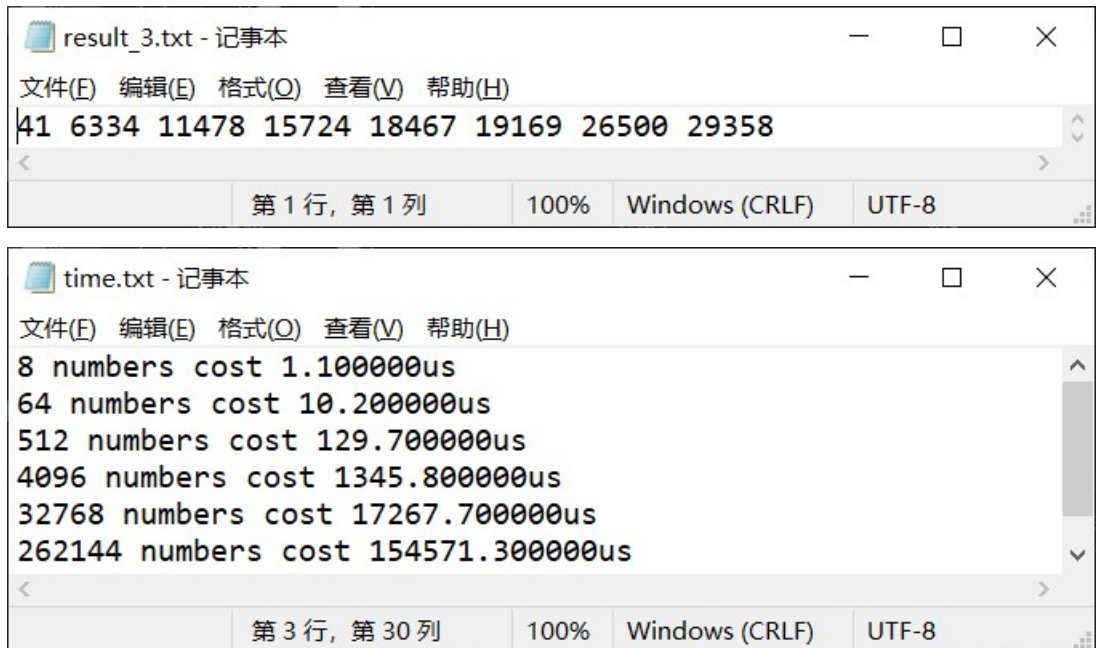
排序代码:

```
void max_heapify(int* a, int x, int size) { //调整为大根堆堆
    int l, r, max;
    l = 2 * x; //左儿子
    r = 2 * x + 1; //右儿子
    if (l ≤ size && a[l] > a[x])
        max = l;
    else max = x;
    if (r ≤ size && a[r] > a[max])
        max = r;
    if (max ≠ x) {
        swap(a[x], a[max]); //与左右儿子较大的一个交换
        max_heapify(a, max, size); //继续调整子树为大根堆堆
    }
}

void build_max_heap(int* a, int n, int size) { //建堆
    int i;
    for (i = n / 2; i > 0; i--)
        max_heapify(a, i, size); //对每个子树都调整为大根堆堆
}

void heapify(int* a, int n) { //堆排序
    int i, size = n;
    build_max_heap(a, n, size); //建堆
    for (i = n; i > 1; i--) {
        swap(a[i], a[1]);
        size--; //每次排序后取出堆顶元素, 把a[1]与之调换
        max_heapify(a, 1, size);
    }
}
```

排序结果与用时：



The first screenshot shows a Notepad window titled 'result_3.txt - 记事本'. The text content is '41 6334 11478 15724 18467 19169 26500 29358'. The status bar at the bottom indicates '第 1 行, 第 1 列' (Line 1, Column 1), '100%', 'Windows (CRLF)', and 'UTF-8'.

The second screenshot shows a Notepad window titled 'time.txt - 记事本'. The text content is:
8 numbers cost 1.100000us
64 numbers cost 10.200000us
512 numbers cost 129.700000us
4096 numbers cost 1345.800000us
32768 numbers cost 17267.700000us
262144 numbers cost 154571.300000us
The status bar at the bottom indicates '第 3 行, 第 30 列' (Line 3, Column 30), '100%', 'Windows (CRLF)', and 'UTF-8'.

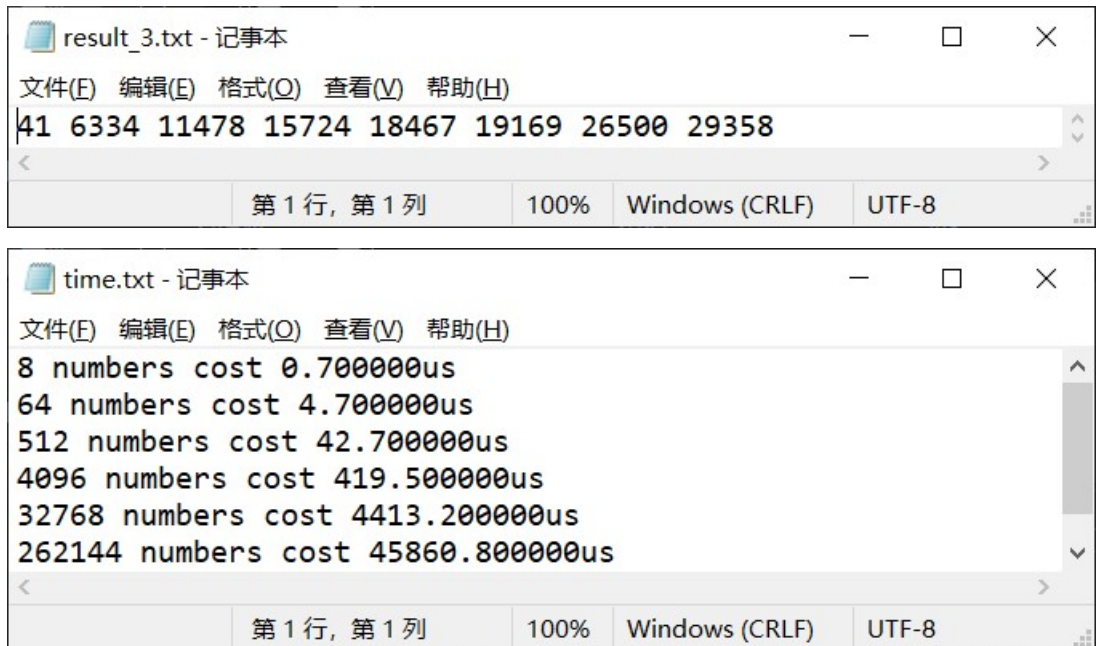
3.快速排序：

排序代码：

```
int Partition(int *a, int p, int r) {
    int x = a[p]; // 设置比较基准为x
    while (p < r) {
        while (p < r && a[r] ≥ x) {
            --r;
        }
        a[p] = a[r];
        while (p < r && a[p] ≤ x) {
            ++p;
        }
        a[r] = a[p]; // 比较当前p (r) 指向的值与x的大小, 若大于 (小于) x, 则调换到r (p) 处
    }
    a[p] = x;
    return p;
}

void quicksort(int *a, int p, int r) // 快排母函数
{
    if (p < r) {
        int q = Partition(a, p, r); // 分治
        quicksort(a, p, q - 1);
        quicksort(a, q + 1, r);
    }
}
```

排序结果与用时：



The first screenshot shows a Notepad++ window titled 'result_3.txt - 记事本'. The text content is a single line of numbers: '41 6334 11478 15724 18467 19169 26500 29358'. The status bar at the bottom indicates '第 1 行, 第 1 列', '100%', 'Windows (CRLF)', and 'UTF-8'.

The second screenshot shows a Notepad++ window titled 'time.txt - 记事本'. The text content lists the cost of sorting different numbers of elements: '8 numbers cost 0.700000us', '64 numbers cost 4.700000us', '512 numbers cost 42.700000us', '4096 numbers cost 419.500000us', '32768 numbers cost 4413.200000us', and '262144 numbers cost 45860.800000us'. The status bar at the bottom indicates '第 1 行, 第 1 列', '100%', 'Windows (CRLF)', and 'UTF-8'.

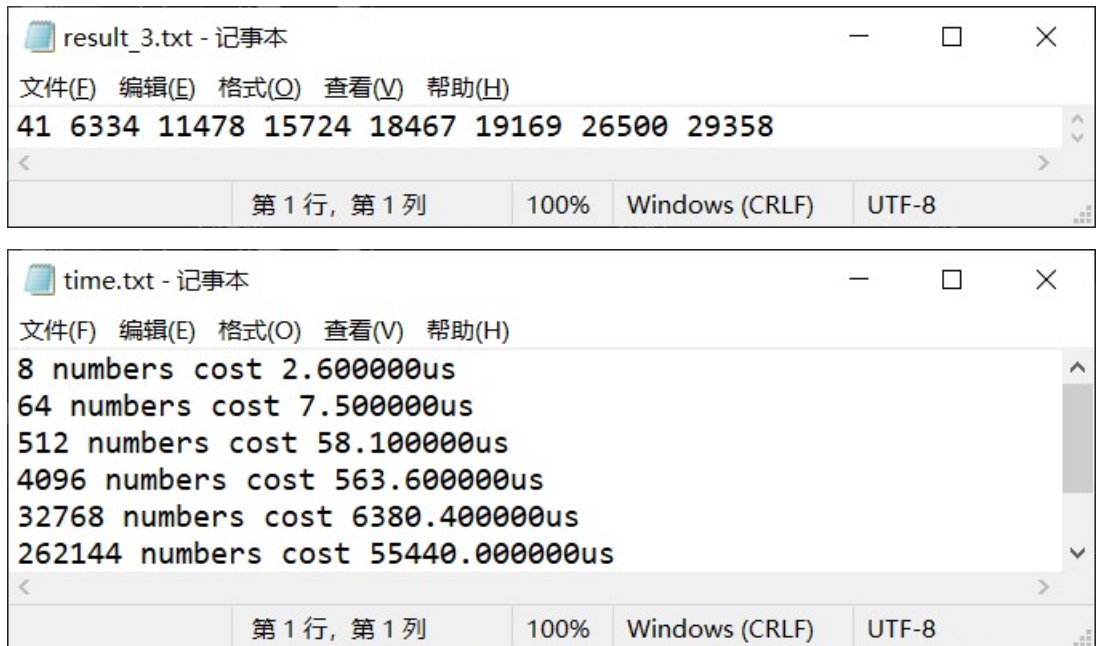
4.归并排序：

排序代码：

```
void mergesort_recursive(int *a, int *reg, int p, int r) {
    if (p ≥ r)
        return;
    int len = r - p, mid = len / 2 + p;
    int p1 = p, r1 = mid;
    int p2 = mid + 1, r2 = r; //分为一半
    mergesort_recursive(a, reg, p1, r1); //左边归并
    mergesort_recursive(a, reg, p2, r2); //右边归并
    int k = p;
    while (p1 ≤ r1 && p2 ≤ r2)
        reg[k++] = a[p1] < a[p2] ? a[p1++] : a[p2++]; //左右两部分归并
    while (p1 ≤ r1)
        reg[k++] = a[p1++];
    while (p2 ≤ r2)
        reg[k++] = a[p2++];
    for (k = p; k ≤ r; k++) //复制回原数组
        a[k] = reg[k];
}

void mergesort(int *a, int len) {
    int* r;
    r = (int*)malloc(len * sizeof(int));
    mergesort_recursive(a, r, 0, len - 1);
    free(r);
}
```


排序结果与用时：



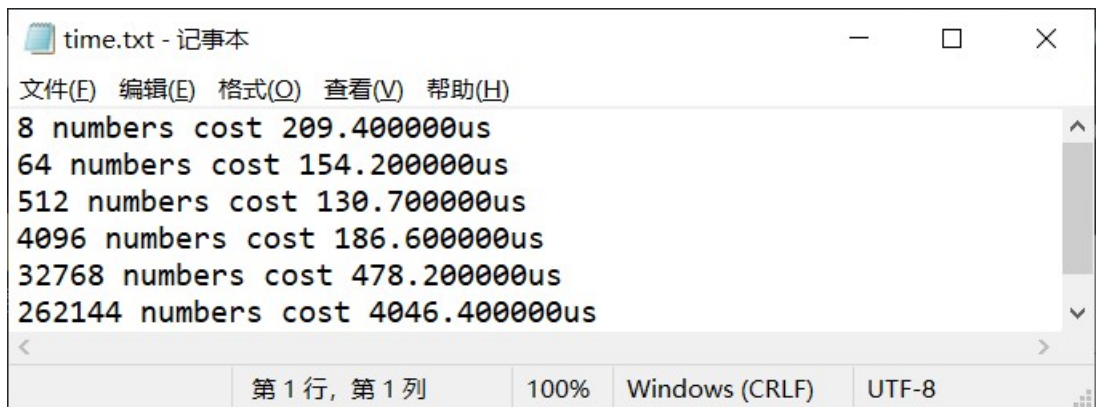
5.计数排序：

排序代码：

```
void countingsort(int* a, int* b, int n) {
    int* c = (int*)malloc(sizeof(int) * 35000);
    int i, j, k;
    for (k = 0; k < 35000; k++)
        c[k] = 0; // c 置零
    for (i = 0; i < n; i++)
        c[a[i]]++; // 循环后 c 数组元素 i 表示数据中 i 的个数
    for (k = 1; k < 35000; k++)
        c[k] += c[k - 1]; // 循环后 c 数组元素 i 表示数据中小于等于 i 的个数
    for (j = n; j > 0; j--)
        b[--c[a[j - 1]]] = a[j - 1]; // 将 c[a[j]] 个 a[j] 复制到 b 中
    for (i = 0; i < n; i++)
        a[i] = b[i];
    free(c);
}
```

排序结果与用时：





```
time.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
8 numbers cost 209.400000us
64 numbers cost 154.200000us
512 numbers cost 130.700000us
4096 numbers cost 186.600000us
32768 numbers cost 478.200000us
262144 numbers cost 4046.400000us
第 1 行, 第 1 列 100% Windows (CRLF) UTF-8
```

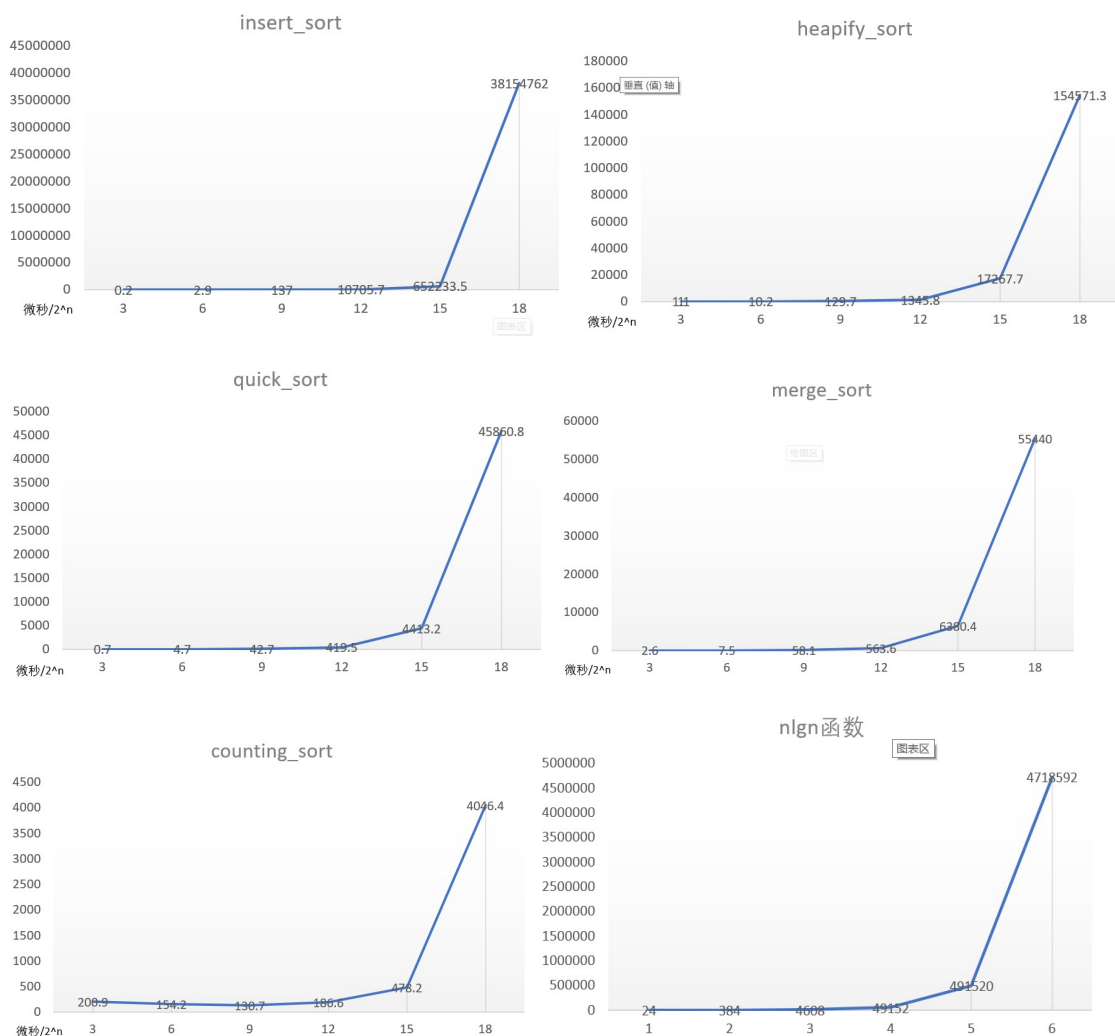
5.main 函数:

采用文件指针输入输出数据, 利用计时器计算不同算法在不同输入规模下的时间。

```
int main() {
    double run_time;
    _LARGE_INTEGER time_start; //开始时间
    _LARGE_INTEGER time_over; //结束时间
    double dqFreq; //计时器频率
    LARGE_INTEGER f; //计时器频率
    QueryPerformanceFrequency(&f);
    dqFreq = (double)f.QuadPart;
    FILE* fp1,*fp2;
    int i, j;
    int* a;
    a = (int*)malloc(MAX_SIZE * sizeof(int));
    fp1 = fopen("input/input.txt", "r");//输入文件
    fp2 = fopen("output/insertsort/time.txt", "w+");//时间输出
    FILE* fp[6];
    fp[0] = fopen("output/insertsort/result_3.txt", "w+");
    fp[1] = fopen("output/insertsort/result_6.txt", "w+");
    fp[2] = fopen("output/insertsort/result_9.txt", "w+");
    fp[3] = fopen("output/insertsort/result_12.txt", "w+");
    fp[4] = fopen("output/insertsort/result_15.txt", "w+");
    fp[5] = fopen("output/insertsort/result_18.txt", "w+");//不同数据规模输出
    for (i = 0; i < 6; i++) {
        fseek(fp1, 0, SEEK_SET);
        for (j = 0; j < test[i]; j++)
            fscanf(fp1, "%d", &a[j]);//按不同数据规模读入数据
        QueryPerformanceCounter(&time_start);//开始计时
        insertsort(a, test[i]);//开始排序
        QueryPerformanceCounter(&time_over);//结束计时
        for (j = 0; j < test[i]; j++)
            fprintf(fp[i], "%d ", a[j]);
        run_time = 1000000 * (time_over.QuadPart - time_start.QuadPart) / dqFreq;//计算所用时间
        fprintf(fp2, "%d numbers cost %lfus\n", test[i], run_time);
    }
    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

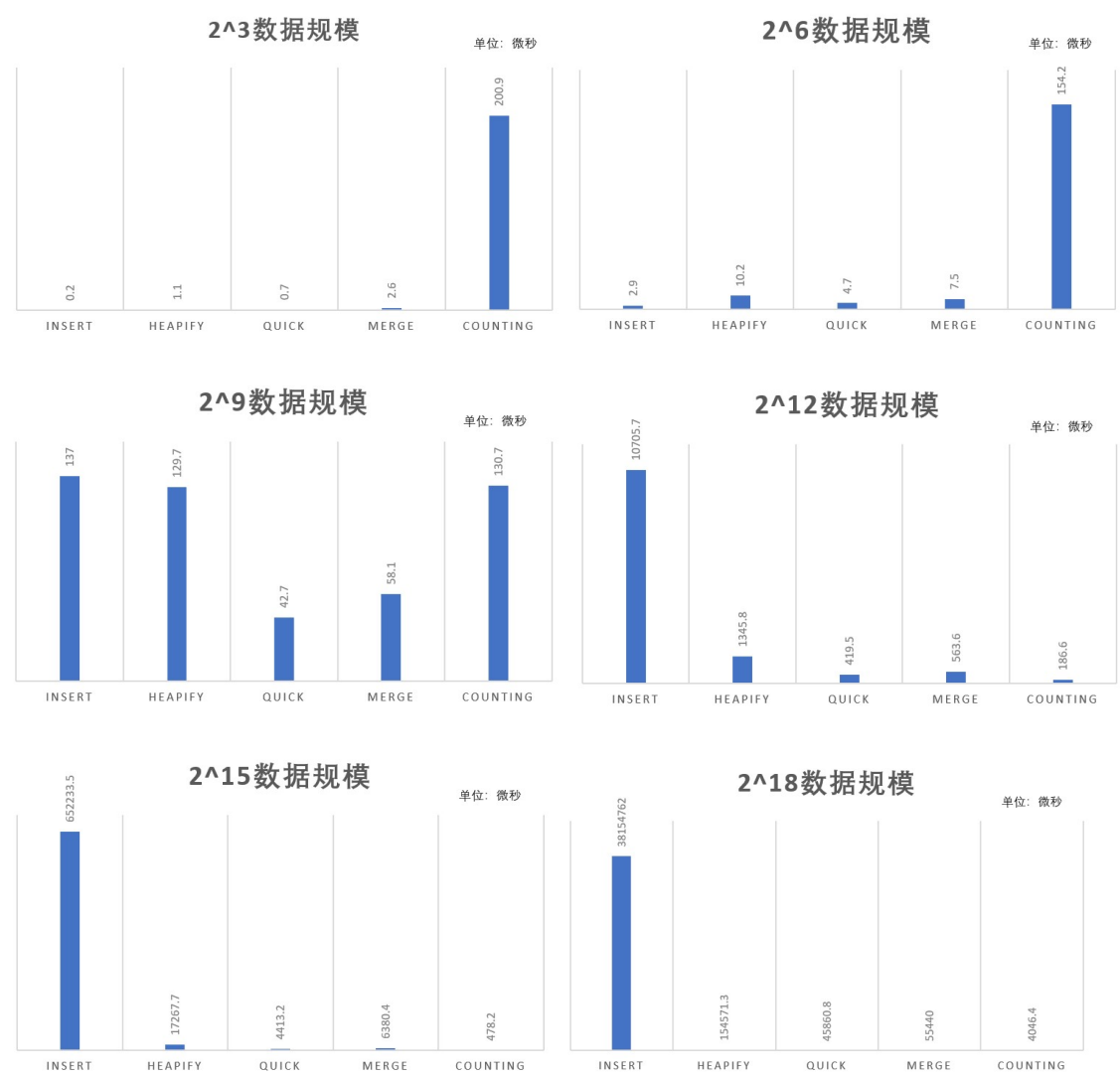
四、 实验结果与分析

1. 五种排序算法在不同数据输入规模的排序时间



可以看出除了插入排序，其余四种算法与 $n \lg n$ 函数的图形基本一致，渐进性能相同。

2. 六个数据规模使用不同算法的排序时间



在较小的数据规模（ $<2^9$ ）下，除了计数排序性能较差，其余排序运行时间差别不大；在中等数据规模（ $<2^{12}$ ）下，快速排序和归并排序性能较好；在较大数据规模（ $>2^{15}$ ）下，计数排序性能最好，插入排序性能极差。