算法基础 实验一 排序算法

PB18000334
徐家恒

一、实验内容

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。

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

pc

三、实验环境

visual studio 2017

四、实验方法

生成数据：

调用 rand()函数即可

插入排序：

```
template<typename T>
void InsertSort(T A[], int n) {
    for (int j = 1; j < n; ++j) {
        T key = A[j];
        int i = j - 1;
        while (i >= 0 && A[i] > key) {
            A[i + 1] = A[i];
            --i;
        }
        A[i + 1] = key;
    }
}
```

归并排序：

```cpp
template<typename T>
void merge(T A[], int p, int q, int r) {
    int n1 = q - p + 1;
    int n2 = r - q;
    int* L = new int[n1 + 1];
    int* R = new int[n2 + 1];
    for (int i = 0; i < n1; ++i)
        L[i] = A[p + i];
    for (int i = 0; i < n2; ++i)
        R[i] = A[q + i + 1];
    L[n1] = INF;
    R[n2] = INF;
    for (int i = 0, j = 0, k = p; k <= r; ++k) {
        if (L[i] < R[j])
            A[k] = L[i++];
        else
            A[k] = R[j++];
    }
    delete L;
    delete R;
}

template<typename T>
void MergeSort(T A[], int p, int r) {
    if (p < r) {
        int q = (p + r) / 2;
        MergeSort(A, p, q);
        MergeSort(A, q + 1, r);
        merge(A, p, q, r);
    }
}

template<typename T>
void MergeSort(T A[], int n) {
    MergeSort(A, 0, n - 1);
}
```

堆排序:

```cpp
template<typename T>
void MaxHeap(T A[], int n ,int i) {
    int l = LEFT(i), r = RIGHT(i), largest = 0;
    if (l < n && A[l] > A[i])
        largest = l;
    else
        largest = i;
    if (r < n && A[r] > A[largest])
        largest = r;
    if (largest != i) {
        exchange(A, i, largest);
        MaxHeap(A, n, largest);
    }
}

template<typename T>
void HeapSort(T A[], int n) {
    for (int i = n / 2 - 1; i >= 0; --i)
        MaxHeap(A, n, i);
    for (int i = n - 1; i > 0; --i) {
        exchange(A, 0, i);
        MaxHeap(A, i, 0);
    }
}
```

快速排序:

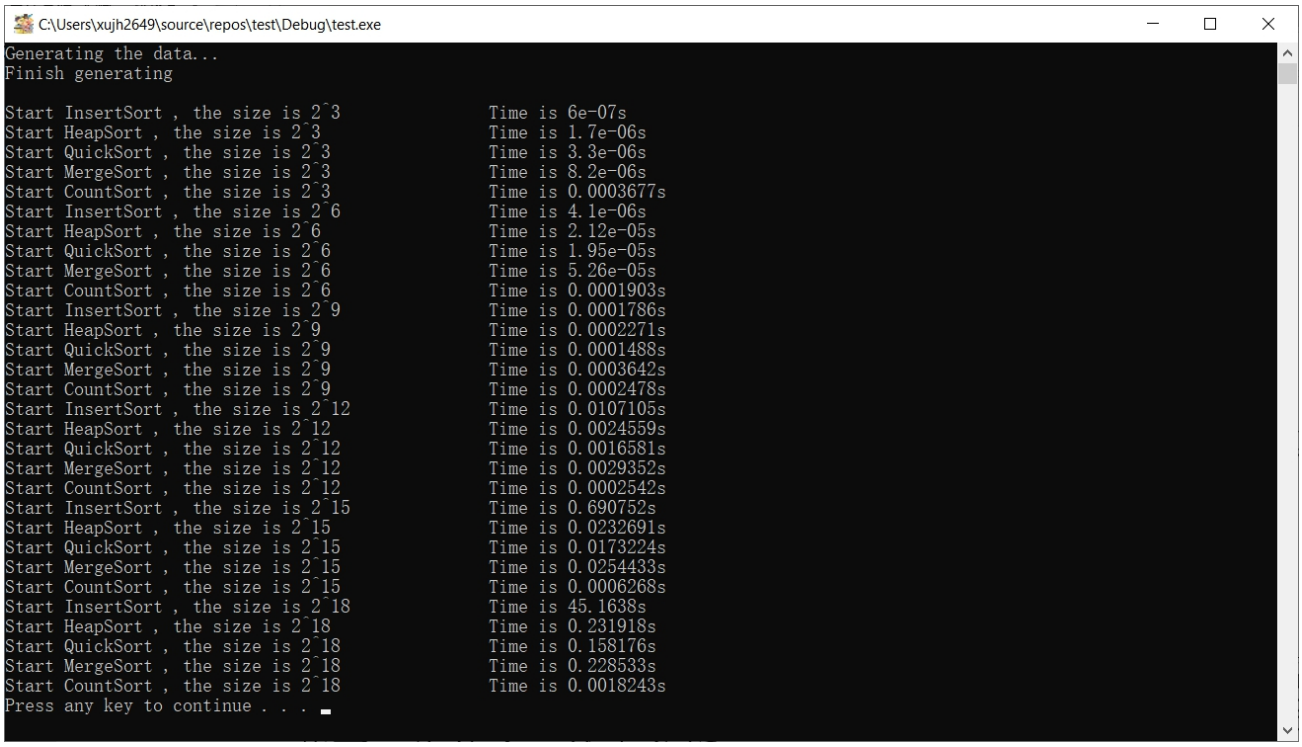
```cpp
template<typename T>
int Patition(T A[], int p, int r) {
    srand((unsigned)time(NULL));
    int random = rand() % (r - p) + p;
    exchange(A, random, r);
    int i = p - 1;
    for (int j = p; j < r; ++j)
        if (A[j] < A[r])
            exchange(A, ++i, j);
    exchange(A, r, ++i);
    return i;
}

template<typename T>
void QuickSort(T A[], int p, int r) {
    if (p < r) {
        int i = Patition(A, p, r);
        QuickSort(A, p, i - 1);
        QuickSort(A, i + 1, r);
    }
}

template<typename T>
void QuickSort(T A[], int n) {
    QuickSort(A, 0, n - 1);
}
```

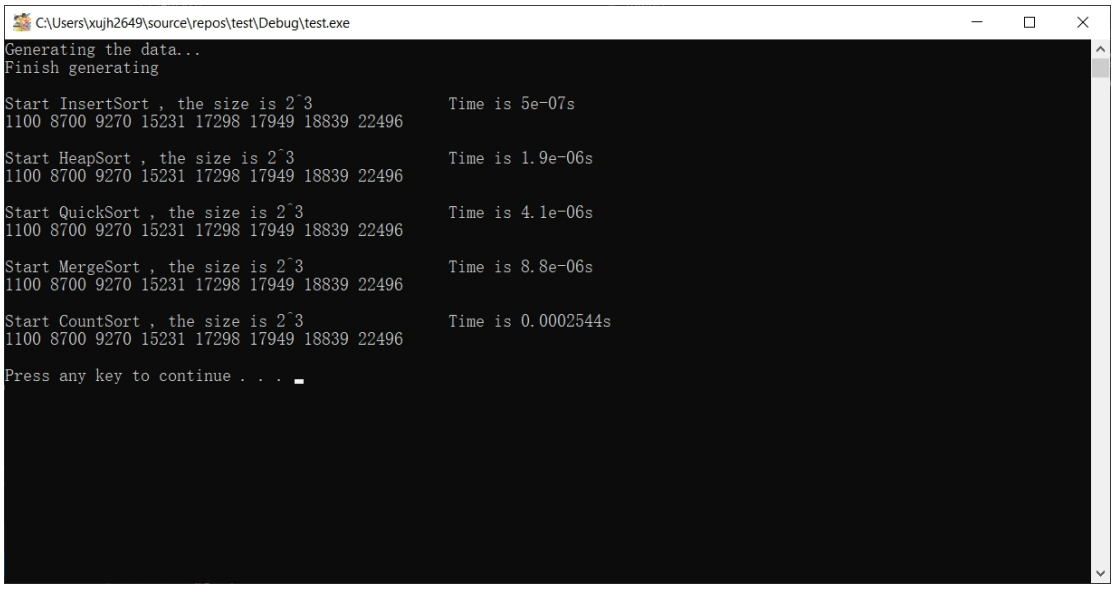
计数排序:

```
template<typename T>
void CountSort(T A[], int n, int length = RAND_MAX) {
    T *count = new int[length + 1];
    for (int i = 0; i <= length; ++i)
        count[i] = 0;
    for (int i = 0; i < n; ++i)
        count[A[i]] += 1;
    for (int i = 0, cur = 0; i <= length; ++i)
        if (count[i])
            for (int j = 0; j < count[i]; ++j)
                A[cur++] = i;
}
```

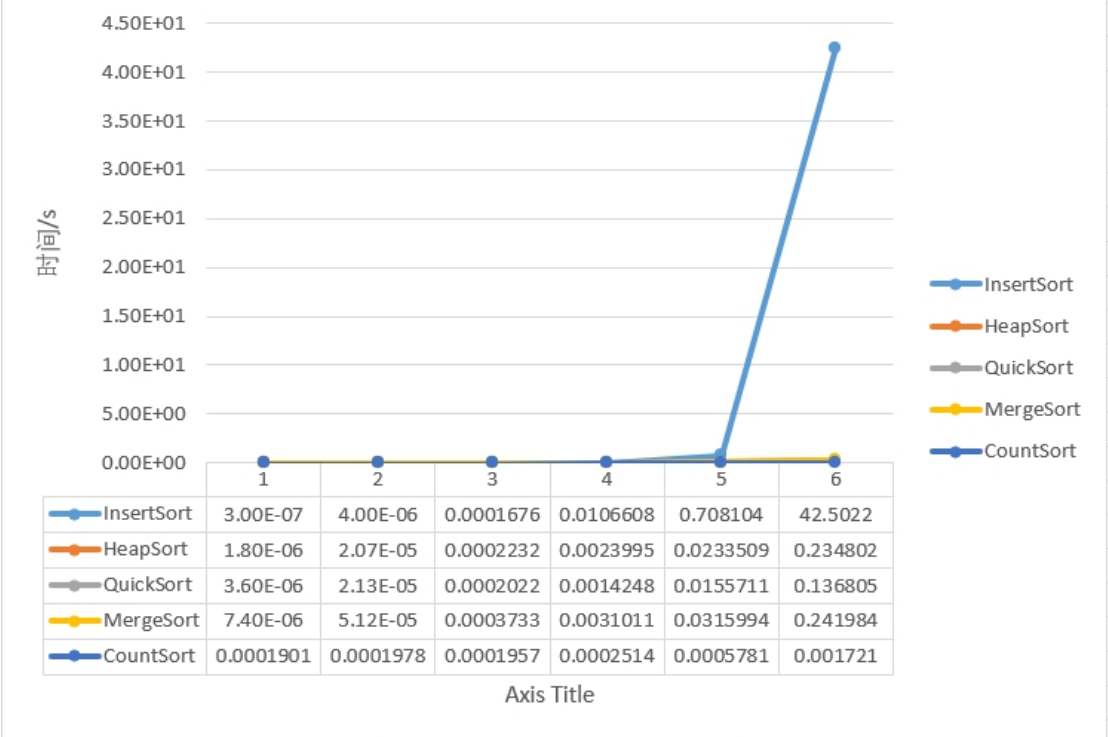
五、实验步骤、实验结果

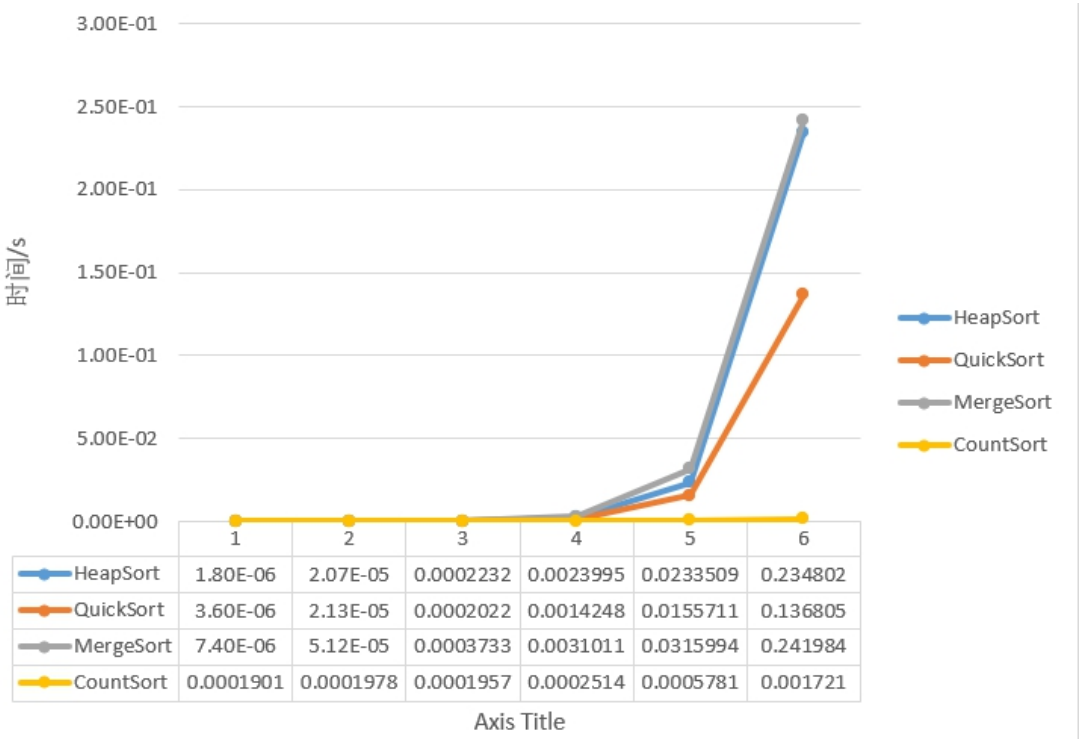
crtl+f5 开始运行，结果如下:




C:\Users\xujh2649\source\repos\test\Debug\test.exe
Generating the data...
Finish generating
Start InsertSort , the size is 2^3 Time is 5e-07s
1100 8700 9270 15231 17298 17949 18839 22496
Start HeapSort , the size is 2^3 Time is 1.9e-06s
1100 8700 9270 15231 17298 17949 18839 22496
Start QuickSort , the size is 2^3 Time is 4.1e-06s
1100 8700 9270 15231 17298 17949 18839 22496
Start MergeSort , the size is 2^3 Time is 8.8e-06s
1100 8700 9270 15231 17298 17949 18839 22496
Start CountSort , the size is 2^3 Time is 0.0002544s
1100 8700 9270 15231 17298 17949 18839 22496
Press any key to continue . . .

运行时间比较:


4.50E+01
4.00E+01
3.50E+01
3.00E+01
2.50E+01
2.00E+01
1.50E+01
1.00E+01
5.00E+00
0.00E+00
时间/s
1 2 3 4 5 6
InsertSort 3.00E-07 4.00E-06 0.0001676 0.0106608 0.708104 42.5022
HeapSort 1.80E-06 2.07E-05 0.0002232 0.0023995 0.0233509 0.234802
QuickSort 3.60E-06 2.13E-05 0.0002022 0.0014248 0.0155711 0.136805
MergeSort 7.40E-06 5.12E-05 0.0003733 0.0031011 0.0315994 0.241984
CountSort 0.0001901 0.0001978 0.0001957 0.0002514 0.0005781 0.001721
InsertSort
HeapSort
QuickSort
MergeSort
CountSort
Axis Title



	1	2	3	4	5	6
HeapSort	1.80E-06	2.07E-05	0.0002232	0.0023995	0.0233509	0.234802
QuickSort	3.60E-06	2.13E-05	0.0002022	0.0014248	0.0155711	0.136805
MergeSort	7.40E-06	5.12E-05	0.0003733	0.0031011	0.0315994	0.241984
CountSort	0.0001901	0.0001978	0.0001957	0.0002514	0.0005781	0.001721

基本渐进情况和教材一样，计数排序在这种小规模数据情况下表现最优，其次是快速排序。