南京邮电大学

实 验 报 告

（2017 / 2018 学年 第 一 学期）

课程名称	数据结构 A
实验名称	内排序算法的实现及性能比较
实验时间	2017 年 12 月 14 日
指导单位	计算机软件与工程学院
指导教师	邹志强

学生姓名	柏超宇	班级学号	Q15010125
学院(系)	贝尔英才学院	专 业	信息安全

实 验 报 告

实验名称	内排序算法的实现及性能比较	指导教师	邹志强

一、 实验目的和要求

掌握各种内排序算法的实现方法

学会分析各种内排序算法的时间复杂度

二、实验环境(实验设备)

sublime+clang

实验内容

```
#include<stdio.h>
#include <sys/timeb.h>
#include<time.h>
#include<algorithm>
#include<iostream>
using namespace std;
const int MaxSize = 100000;
int original[MaxSize];//记录原始数据
struct timeb t1, t2; //计算程序运行时间
typedef struct entry {
    int key;
    int data;
} Entry;
typedef struct list {
    int n;
    Entry D[MaxSize + 20];
} List;
List list;

void random1() { //产生 n 个随机数
    srand((unsigned)time(NULL));//初始化随机数种子
    for(int i = 0; i < MaxSize; i++) {
        int k = rand();
        original[i] = k;
    }
}

void initial(List *list) {
    for(int i = 0; i < MaxSize; i++) {
        list->D[i].key = original[i];
    }
}

int FindMin(List list, int startIndex) {
    int i, minIndex = startIndex;
    for(i = startIndex + 1; i < list.n; i++) {
        if(list.D[i].key < list.D[minIndex].key) {
            minIndex = i;
        }
    }
```

```
41.     return minIndex;
42. }
43. 
44. void swap(Entry *D, int i, int j) {
45.     if(i == j) return;
46.     Entry temp = *(D + i);
47.     *(D + i) = *(D + j);
48.     *(D + j) = temp;
49. }
50. 
51. void SelectSort(List *list) {
52.     int minIndex, startIndex = 0;
53.     initial(list);
54.     ftime(&t1);
55.     while(startIndex < list->n - 1) {
56.         minIndex = FindMin(*list, startIndex);
57.         swap(list->D, startIndex, minIndex);
58.         startIndex++;
59.     }
60.     ftime(&t2);
61.     printf("SelectSort:");
62.     cout<<(t2.time-t1.time)*1000+(t2.millitm-t1.millitm)<<"ms"<<endl;
63. }
64. 
65. void InsertSort(List *list) {
66.     int i, j;
67.     initial(list);
68.     ftime(&t1);
69.     for(i = 1; i < list->n; i++) {
70.         Entry insertItem = list->D[i];
71.         for(j = i - 1; j >= 0; j--) {
72.             if(insertItem.key < list->D[j].key) {
73.                 list->D[j + 1] = list->D[j];
74.             } else break;
75.         }
76.         list->D[j + 1] = insertItem;
77.     }
78.     ftime(&t2);
79.     printf("InsertSort:");
80.     cout<<(t2.time-t1.time)*1000+(t2.millitm-t1.millitm)<<"ms"<<endl;
81. }
82. 
83. void BubbleSort(List *list) {
84.     int i, j;
```

```
85.  initial(list);
86.  ftime(&t1);
87.  bool isSwap = false;
88.  for(i = list->n - 1; i > 0; i--) {
89.     for(j = 0; j < i; j++) {
90.        if(list->D[j].key > list->D[j + 1].key) {
91.           swap(list->D, j, j + 1);
92.           isSwap = true;
93.        }
94.     }
95.     if(!isSwap) break;
96.  }
97.  ftime(&t2);
98.  printf("BubbleSort:");
99.  cout<<(t2.time-t1.time)*1000+(t2.millitm-t1.millitm)<<"ms"<<endl;
100. }
101.
102. int Partition(List *list, int low, int high) {
103.    int i = low;
104.    int j = high + 1;
105.    Entry pivot = list->D[low];
106.    do {
107.       do {
108.          i++;
109.       } while (list->D[i].key < pivot.key);
110.       do {
111.          j--;
112.       } while (list->D[j].key > pivot.key);
113.       if(i < j) {
114.          swap(list->D, i, j);
115.       }
116.    } while(i < j);
117.    swap(list->D, low, j);
118.    return j;
119. }
120.
121. void QuickSort(List *list, int low, int high) {
122.    int k;
123.    if(low < high) {
124.       k = Partition(list, low, high);
125.       QuickSort(list, low, k - 1);
126.       QuickSort(list, k + 1, high);
127.    }
128. }
```

```
129.
130. void QuickSort(List *list) {
131.     initial(list);
132.     ftime(&t1);
133.     QuickSort(list, 0, list->n - 1);
134.     ftime(&t2);
135.     printf("QuickSort:");
136.     cout<<(t2.time-t1.time)*1000+(t2.millitm-t1.millitm)<<"ms"<<endl;
137. }
138.
139. void MergeArray(List *list, int first, int mid, int last) { //合并两个序列
140.     int i = first, j = mid + 1;
141.     int m = mid, n = last;
142.     int temp[MaxSize], k = 0;
143.     while(i <= m && j <= n) {
144.         if(list->D[i].key < list->D[j].key)
145.             temp[k++] = list->D[i++].key;
146.         else
147.             temp[k++] = list->D[j++].key;
148.     }
149.     while(i <= m)
150.         temp[k++] = list->D[i++].key;
151.     while(j <= n)
152.         temp[k++] = list->D[j++].key;
153.     for(i = 0; i < k; i++)
154.         list->D[first + i].key = temp[i];
155. }
156.
157. void MergeSort(List *list, int first, int last) {
158.     int mid;
159.     if(first < last) {
160.         mid = (first + last) / 2;
161.         MergeSort(list, first, mid);
162.         MergeSort(list, mid + 1, last);
163.         MergeArray(list, first, mid, last);
164.     }
165. }
166.
167. void mergesort(List *list) { //归并排序 1
168.         initial(list);
169.     ftime(&t1);
170.     MergeSort(list, 0, list->n - 1);
171.     ftime(&t2);
172.     printf("MergeSort:");
```

```
173.     cout<<(t2.time-t1.time)*1000+(t2.millitm-t1.millitm)<<"ms"<<endl;
174. }
175. 
176. void AdjustHeap(List *list,int s,int m)//调整堆
177. {
178.     int j,temp=list->D[s].key;
179.     for(j=2*s;j<=m;j*=2)
180.     {
181.         if(j<m&&list->D[j].key<list->D[j+1].key)
182.             j++;
183.         if(temp>=list->D[j].key)
184.             break;
185.         list->D[s].key=list->D[j].key;
186.         s=j;
187.     }
188.     list->D[s].key=temp;
189. }
190. 
191. void HeapSort(List *list)//堆排序
192. {
193.     int i;
194.     initial(list);
195.     ftime(&t1);
196.     for(i=MaxSize/2-1;i>=0;i--)//建立堆
197.         AdjustHeap(list,i,MaxSize);
198.     for(i=MaxSize-1;i>0;i--)
199.     {
200.         swap(list->D,0,i);//交换堆顶和栈底元素
201.         AdjustHeap(list,0,i-1);//重新调整堆顶节点成为大顶堆
202.     }
203.     ftime(&t2);
204.     printf("HeapSort:");
205.     cout<<(t2.time-t1.time)*1000+(t2.millitm-t1.millitm)<<"ms"<<endl;
206. }
207. 
208. int main() {
209.     random1();
210.     list.n = MaxSize;
211.     SelectSort(&list);
212.     InsertSort(&list);
213.     BubbleSort(&list);
214.     QuickSort(&list);
215.     mergesort(&list);
216.     HeapSort(&list);
```

```
217.
218.     return 0;
219. }
```

实验结果：

```
MaxSize = 500
SelectSort:1ms
InsertSort:0ms
BubbleSort:1ms
QuickSort:0ms
MergeSort:0ms
HeapSort:0ms


MaxSize = 10000
SelectSort:168ms
InsertSort:77ms
BubbleSort:391ms
QuickSort:1ms
MergeSort:1ms
HeapSort:1ms


MaxSize = 50000
SelectSort:4412ms
InsertSort:1939ms
BubbleSort:9700ms
QuickSort:6ms
MergeSort:8ms
HeapSort:8ms


MaxSize = 100000
SelectSort:17118ms
InsertSort:7431ms
BubbleSort:36997ms
QuickSort:18ms
MergeSort:21ms
HeapSort:19ms
```

根据实验数据可得，随着数据量的增加，平方复杂度的排序算法耗费时间飞速上升，而 O(nlogn)复杂度的算法表现优秀。

四、实验小结（包括问题和解决方法、心得体会、意见与建议等）

快速排序是书上的，数据量大的时候偶尔会触发段错误或者总线错误，后面时间允许的话会做进一步调试。分析比较了书上归并写法，和利用递归进行归并的时间，发现相差 2 倍左右，推测是频繁的函数调用中效率变低。

通过本次实验，熟悉了常用的排序操作，提高了自己的代码能力。

五、指导教师评语

成　绩		**批阅人**		**日　期**	