

数据结构上机实验题实验报告（四）

题目: 内部排序算法比较

姓名: 王磊

班级: 2020211311

学号: 2020211538

提交日期: 2021年12月14日

一 题目描述

1. 输入形式: 长度不小于100的伪随机数排序表。
2. 输出形式: 每种排序方式的关键字的比较次数和关键字的移动次数。。
3. 程序功能: 读入待排序表, 分别使用六种排序方法进行排序, 给出每种排序方法关键字参加的比较次数和关键字的移动次数并作出分析。

二 程序设计

1. 定义结构变量SqList代表待排序表

```
typedef struct {  
    int l[MAXSIZE + 1];  
    int length;  
}SqList;
```

2. 概要设计

主程序中定义了一个长为201（下标为0的位置留作哨兵）的顺序表结构存放需要排序的数据, 主程序运行之后, 先由随机数生成函数初始化顺序表T, 再为T1--T5赋给T完全相同的值, 对T和T1--T5分别使用六种排序方法进行排序操作, 输出每种排序过程中关键字的比较次数和移动次数。该过程循环5次, 即用5个不同的待排序表进行实验。

1. BubbleSort模块调用了swap函数
2. SelectSort模块调用了swap函数
3. QuickSort模块调用了Qsort函数, 同时Qsort模块调用了Partition函数, Partition调用了swap函数
4. HeapSort模块调用了HeapAdjust函数, HeapAdjust函数中又调用了swap函数

以下为函数声明

```
void swap(SqList* L, int m, int n);  
    //交换L中下标为m和n的元素的位置  
void BubbleSort(SqList* L);  
    //冒泡排序  
void InsertSort(SqList* L);  
    //直接插入排序  
void SelectSort(SqList* L);  
    //简单选择排序
```

```

void QuickSort(SqList* L);
    //快速排序
int Partition(SqList* L, int low, int high, int& CompareCount, int& MoveCount);
    //快速排序中用于确定枢轴位置
void QSort(SqList* L, int low, int high, int& CompareCount, int& MoveCount);
    //对子表进行快速排序的递归程序
void ShellSort(SqList* L);
    //希尔排序
void HeapSort(SqList* L);
    //堆排序
void HeapAdjust(SqList* L, int s, int m, int& CompareCount, int& MoveCount);
    //将表调整为大根堆

```

3. 详细设计

1. 头文件

```

#include<stdio.h>
#include<stdlib.h>

```

2. 外部变量定义

```

#define MAXSIZE 200

```

3. 结构体定义

```

typedef struct {
    int l[MAXSIZE + 1];
    int length;
}SqList;

```

4. swap交换函数

```

void swap(SqList* L, int m, int n)
{
    int temp = L->l[m];
    L->l[m] = L->l[n];
    L->l[n] = temp;
}

```

5. 冒泡排序

```

void BubbleSort(SqList* L)
{
    int MoveCount = 0;
    int CompareCount = 0;
    for (int i = 1; i < L->length; i++) {
        for (int k = i + 1; k <= L->length; k++) {
            CompareCount++;
            if (L->l[i] > L->l[k]) {
                swap(L, i, k);
                MoveCount = MoveCount + 3;
            }
        }
    }
    printf("The times of comparision of BubbleSort is %d\n", CompareCount);
    printf("The times of movement of BubbleSort is %d\n", MoveCount);
}

```

```
}
```

6. 直接插入排序

```
void InsertSort(SqList* L)
{
    int j;
    int MoveCount = 0;
    int CompareCount = 0;
    for (int i = 2; i < L->length; i++) {
        CompareCount++;
        if (L->l[i] < L->l[i - 1]) {
            L->l[0] = L->l[i];
            MoveCount++;
            for (j = i - 1; L->l[j] > L->l[0]; j--) {
                L->l[j + 1] = L->l[j];
                CompareCount++;
                MoveCount++;
            }
            CompareCount++;
            L->l[j + 1] = L->l[0];
            MoveCount++;
        }
    }
    printf("The times of comparision of InsertSort is %d\n", CompareCount);
    printf("The times of movement of InsertSort is %d\n", MoveCount);
}
```

7. 简单选择排序

```
void SelectSort(SqList* L)
{
    int min;
    int MoveCount = 0;
    int CompareCount = 0;
    for (int i = 1; i < L->length; i++) {
        min = i;
        for (int j = i + 1; j <= L->length; j++) {
            CompareCount++;
            if (L->l[min] > L->l[j]) {
                min = j;
            }
        }
        if (i != min) {
            swap(L, i, min);
            MoveCount += 3;
        }
    }
    printf("The times of comparision of SelectSort is %d\n", CompareCount);
    printf("The times of movement of SelectSort is %d\n", MoveCount);
}
```

8. 快速排序

```
void QuickSort(SqList* L)
{
    int MoveCount = 0;
    int CompareCount = 0;
    QSort(L, MoveCount, L->length, CompareCount, MoveCount);
}
```

```

printf("The times of comparision of QuickSort is %d\n", CompareCount);
printf("The times of movement of QuickSort is %d\n", MoveCount);
}
void QSort(SqList* L, int low, int high, int& CompareCount, int& MoveCount)
{
    int pivotloc;
    if (low < high) {
        pivotloc = Partition(L, low, high, CompareCount, MoveCount);
        QSort(L, low, pivotloc - 1, CompareCount, MoveCount);
        QSort(L, pivotloc + 1, high, CompareCount, MoveCount);
    }
}
int Partition(SqList* L, int low, int high, int& CompareCount, int& MoveCount)
{
    int pivotkey;
    L->l[0] = L->l[low];
    pivotkey = L->l[low];
    while (low < high) {
        while (low < high && L->l[high] >= pivotkey) {
            CompareCount++;
            --high;
        }
        CompareCount++;
        swap(L, low, high);
        MoveCount += 3;
        while (low < high && L->l[low] <= pivotkey) {
            CompareCount++;
            low++;
        }
        CompareCount++;
        swap(L, low, high);
        MoveCount += 3;
    }
    return low;
}
}

```

9. 希尔排序

```

void ShellSort(SqList* L)
{
    int j;
    int MoveCount = 0;
    int CompareCount = 0;
    int increase = L->length;
    do {
        increase = increase / 5 + 1;
        for (int i = increase + 1; i < L->length; i++) {
            CompareCount++;
            if (L->l[i] < L->l[i - increase]) {
                L->l[0] = L->l[i];
                MoveCount++;
                for (j = i - increase; L->l[0] < L->l[j] && j>0; j = j - increase) {
                    CompareCount++;
                    L->l[j + increase] = L->l[j];
                    MoveCount++;
                }
                CompareCount++;
                L->l[j + increase] = L->l[0];
                MoveCount++;
            }
        }
    }
}

```

```

    }
} while (increase > 1);
printf("The times of comparision of ShellSort is %d\n", CompareCount);
printf("The times of movement of ShellSort is %d\n", MoveCount);
}

```

10. 堆排序

```

void HeapSort(Sqlist* L)
{
    int i;
    int MoveCount = 0;
    int CompareCount = 0;
    for (i = L->length / 2; i > 0; i--) {
        HeapAdjust(L, i, L->length, CompareCount, MoveCount);
    }
    for (i = L->length; i > 1; i--) {
        swap(L, 1, i);
        MoveCount += 3;
        HeapAdjust(L, 1, i - 1, CompareCount, MoveCount);
    }
    printf("The times of comparision of HeapSort is %d\n", CompareCount);
    printf("The times of movement of HeapSort is %d\n", MoveCount);
}

void HeapAdjust(Sqlist* L, int s, int m, int& CompareCount, int& MoveCount)
{
    int rc = L->l[s];
    MoveCount++;
    for (int i = 2 * s; i <= m; i *= 2) {
        CompareCount++;
        if (i < m && L->l[i] < L->l[i + 1]) {
            i++;
        }
        CompareCount++;
        if (rc >= L->l[i])
            break;
        L->l[s] = L->l[i];
        MoveCount++;
        s = i;
    }
    L->l[s] = rc;
    MoveCount++;
}

```

三 调试分析

- 编译环境：Visual Studio2019
- 运行环境：WIN10

1. 第一组数据运行结果

```
Microsoft Visual Studio 调试控制台
Table NO.0 to be sorted is:
41 467 334 500 169 724 478 358 962 464 705 145 281 827 961 4
91 995 942 436 391 604 902 153 292 382 421 716 718 895 447 7
26 771 538 869 912 667 299 35 894 703 811 322 333 673 664 1
41 711 253 868 547 644 662 757 37 859 723 741 529 778 316 1
90 842 288 106 40 264 648 446 805 890 729 370 350 6 101 3
93 548 629 623 84 954 756 840 966 376 931 308 944 439 626 3
23 537 118 82 929 541 833 115 639 658 704 930 977 306 386 2
1 745 924 72 270 829 777 573 97 512 986 290 161 636 355 7
67 655 574 31 52 150 941 430 107 191 7 337 457 287 753 3
83 945 909 209 758 221 588 422 946 506 30 413 168 900 591 7
62 410 359 624 483 595 602 291 836 374 20 596 348 199 668 4
84 734 53 999 418 938 788 127 728 893 807 310 617 813 514 3
09 616 935 451 600 249 519 556 798 303 224 8 844 609 989 7
02 195 485 93 343
Table NO.0 after BubbleSort is:
The times of comparision of BubbleSort is 19900
The times of movement of BubbleSort is 30939
Table NO.0 after InsertSort is:
The times of comparision of InsertSort is 10564
The times of movement of InsertSort is 10558
Table NO.0 after SelectSort is:
The times of comparision of SelectSort is 19900
The times of movement of SelectSort is 591
Table NO.0 after QuickSort is:
The times of comparision of QuickSort is 2561
The times of movement of QuickSort is 2328
Table NO.0 after ShellSort is:
The times of comparision of ShellSort is 2410
The times of movement of ShellSort is 2130
Table NO.0 after HeapSort is:
The times of comparision of HeapSort is 2452
The times of movement of HeapSort is 2343
```

2. 第二组数据运行结果

```
Microsoft Visual Studio 调试控制台
Table NO.1 to be sorted is:
523 587 314 503 448 200 458 618 580 796 798 281 589 9 157 4
72 622 538 292 38 179 190 657 958 191 815 888 156 511 202 6
34 272 55 328 646 362 886 875 433 869 142 844 416 881 998 3
22 651 21 699 557 476 892 389 75 712 600 510 3 861 688 4
01 789 255 423 2 585 182 285 88 426 617 757 832 932 169 1
54 721 189 976 329 368 692 425 555 434 549 441 512 145 60 7
18 753 139 279 996 687 529 437 866 949 193 195 297 286 105 4
88 282 455 734 114 701 316 671 786 263 313 355 185 53 912 8
08 945 756 321 558 982 481 144 196 222 129 161 535 450 173 4
66 44 659 439 253 24 745 649 186 474 22 168 18 787 905 3
91 625 477 414 824 334 874 372 159 833 70 487 518 177 773 2
70 763 668 192 985 102 480 213 627 802 99 527 543 924 23 9
72 61 181 432 505 593 725 31 492 64 900 187 360 413 974 1
70 235 711 760 896
Table NO.1 after BubbleSort is:
The times of comparision of BubbleSort is 19900
The times of movement of BubbleSort is 30354
Table NO.1 after InsertSort is:
The times of comparision of InsertSort is 10493
The times of movement of InsertSort is 10487
Table NO.1 after SelectSort is:
The times of comparision of SelectSort is 19900
The times of movement of SelectSort is 582
Table NO.1 after QuickSort is:
The times of comparision of QuickSort is 2441
The times of movement of QuickSort is 2298
Table NO.1 after ShellSort is:
The times of comparision of ShellSort is 2606
The times of movement of ShellSort is 2339
Table NO.1 after HeapSort is:
The times of comparision of HeapSort is 2472
The times of movement of HeapSort is 2360
```

3. 第三组数据运行结果

```
Microsoft Visual Studio 调试控制台
Table NO.2 to be sorted is:
667 285 550 140 694 695 624 19 125 576 658 302 371 466 678 5
93 851 484 18 464 119 152 800 87 60 926 10 757 170 315 2
27 43 758 164 109 882 86 565 487 577 474 625 627 629 928 4
23 520 902 962 123 596 737 261 195 525 264 260 202 116 30 3
26 11 771 411 547 153 790 924 188 763 940 662 829 900 713 9
58 578 365 7 477 200 58 439 303 760 357 324 108 113 887 8
01 850 460 428 993 384 405 540 111 704 835 356 72 350 823 4
85 556 216 626 526 337 271 869 361 896 22 617 112 717 696 5
85 41 129 229 559 932 296 855 53 584 734 654 972 457 369 5
32 963 607 483 911 635 67 848 675 938 223 142 754 511 741 1
75 459 825 221 870 934 205 783 398 279 701 193 637 534 176 7
05 548 881 300 413 641 462 611 877 424 752 443 673 40 313 8
75 818 610 17 169 831 488 685 90 497 589 990 145 353 314 6
51 740 44 258 335

Table NO.2 after BubbleSort is:
The times of comparision of BubbleSort is 19900
The times of movement of BubbleSort is 28074
Table NO.2 after InsertSort is:
The times of comparision of InsertSort is 9615
The times of movement of InsertSort is 9608
Table NO.2 after SelectSort is:
The times of comparision of SelectSort is 19900
The times of movement of SelectSort is 588
Table NO.2 after QuickSort is:
The times of comparision of QuickSort is 2358
The times of movement of QuickSort is 2160
Table NO.2 after ShellSort is:
The times of comparision of ShellSort is 2566
The times of movement of ShellSort is 2280
Table NO.2 after HeapSort is:
The times of comparision of HeapSort is 2496
The times of movement of HeapSort is 2366
```

4. 第四组数据运行结果

```
Microsoft Visual Studio 调试控制台
Table NO.3 to be sorted is:
759 192 605 264 181 503 829 775 608 292 997 549 556 561 627 4
67 541 129 240 813 174 601 77 215 683 213 992 824 392 670 4
28 27 84 75 786 498 970 287 847 604 221 663 706 363 10 1
71 489 164 542 619 913 591 704 818 232 750 205 975 539 303 4
22 98 247 584 648 971 864 545 712 546 678 769 262 519 985 2
89 944 865 540 245 508 318 870 323 132 472 152 87 570 763 9
01 103 423 527 600 969 15 565 28 543 347 88 943 637 409 4
63 49 681 588 342 60 758 954 888 146 690 949 843 430 620 7
48 67 536 783 35 226 185 38 853 629 224 923 359 257 766 9
55 726 411 25 355 1 496 515 964 142 196 948 72 426 606 1
73 429 404 705 626 812 375 93 36 736 141 814 994 256 652 9
36 838 482 131 230 841 625 11 186 650 662 634 893 353 416 4
52 8 233 454 148 124 317 109 200 80 858 50 155 361 903 6
76 643 909 902 282

Table NO.3 after BubbleSort is:
The times of comparision of BubbleSort is 19900
The times of movement of BubbleSort is 31134
Table NO.3 after InsertSort is:
The times of comparision of InsertSort is 10635
The times of movement of InsertSort is 10633
Table NO.3 after SelectSort is:
The times of comparision of SelectSort is 19900
The times of movement of SelectSort is 576
Table NO.3 after QuickSort is:
The times of comparision of QuickSort is 2423
The times of movement of QuickSort is 2286
Table NO.3 after ShellSort is:
The times of comparision of ShellSort is 2599
The times of movement of ShellSort is 2305
Table NO.3 after HeapSort is:
The times of comparision of HeapSort is 2458
The times of movement of HeapSort is 2350
```

5. 第五组数据运行结果

```
Microsoft Visual Studio 调试控制台
Table NO.4 to be sorted is:
653 674 220 402 923 831 369 878 259 8 619 971 3 945 781 5
04 392 685 313 698 589 722 938 37 410 461 234 508 961 959 4
93 515 269 937 869 58 700 264 117 215 555 815 330 39 212 2
88 82 954 85 710 484 774 380 951 541 115 679 110 898 73 7
88 977 132 956 689 113 941 790 723 363 28 184 778 200 71 8
85 974 333 867 153 295 168 825 676 629 650 598 309 693 686 8
0 116 249 667 528 864 421 405 826 816 516 726 666 87 681 9
64 340 21 662 721 64 415 902 873 124 745 762 423 531 806 2
68 318 602 907 307 481 12 136 630 114 809 84 556 290 293 9
96 152 54 345 708 248 491 712 131 439 958 704 995 52 479 2
38 918 866 659 498 486 196 462 633 158 22 146 925 647 458 8
07 98 830 292 600 278 799 352 448 882 540 315 575 567 336 3
97 418 897 828 851 230 449 658 229 520 940 560 147 162 655 6
75 792 361 754 398
Table NO.4 after BubbleSort is:
The times of comparision of BubbleSort is 19900
The times of movement of BubbleSort is 30315
Table NO.4 after InsertSort is:
The times of comparision of InsertSort is 10373
The times of movement of InsertSort is 10368
Table NO.4 after SelectSort is:
The times of comparision of SelectSort is 19900
The times of movement of SelectSort is 573
Table NO.4 after QuickSort is:
The times of comparision of QuickSort is 2334
The times of movement of QuickSort is 2322
Table NO.4 after ShellSort is:
The times of comparision of ShellSort is 2418
The times of movement of ShellSort is 2124
Table NO.4 after HeapSort is:
The times of comparision of HeapSort is 2462
The times of movement of HeapSort is 2355
```

四 工作总结

1. 现有的排序算法的程序实现都比较成熟，该实验的重点和难点在于找到合适的地方插入对关键字移动次数和比较次数的计数变量，大致的规律是在if语句之前加入一次比较，在swap之后加入三次移动。还有一些会出现比较和移动的地方，不同的排序方式都有差别。
2. 又遇到了传参的问题，由于C语言里没有传引用调用，所有只能通过指针的使用来模拟传引用调用，在这个过程中就特别考验对指针概念和符号的理解和掌握，例如在什么地方用*、在什么地方用&，都是需要考虑的问题。
3. 最后的结果基本符合预期，快速排序和堆排序都展现出了在时间复杂度上的优越性。

五 源代码

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE 200
typedef struct {
    int l[MAXSIZE + 1];
    int length;
}SqList;
void swap(SqList* L, int m, int n);
void BubbleSort(SqList* L);
void InsertSort(SqList* L);
void SelectSort(SqList* L);
void QuickSort(SqList* L);
int Partition(SqList* L, int low, int high, int& CompareCount, int& MoveCount);
void QSort(SqList* L, int low, int high, int& CompareCount, int& MoveCount);
void ShellSort(SqList* L);
void HeapAdjust(SqList* L, int s, int m, int& CompareCount, int& MoveCount);
void HeapSort(SqList* L);
```



```

int main()
{
    int s;
    SqList T, T1, T2, T3, T4, T5;
    for (int i = 0; i < 5; i++) {
        bool check[1000] = { 0 };
        printf("Table NO.%d to be sorted is:\n", i);
        for (int j = 1; j < (MAXSIZE + 1); j++) {
            s = rand() % 1000;
            while (check[s] == 1) {
                s = rand() % 1000;
            }
            check[s] = 1;
            T.l[j] = s;
            printf("%d\t", s);
        }
        T.length = MAXSIZE;
        printf("\n");
        T1 = T;
        T2 = T;
        T3 = T;
        T4 = T;
        T5 = T;
        printf("Table NO.%d after BubbleSort is:\n", i);
        BubbleSort(&T);
        printf("Table NO.%d after InsertSort is:\n", i);
        InsertSort(&T1);
        printf("Table NO.%d after SelectSort is:\n", i);
        SelectSort(&T2);
        printf("Table NO.%d after QuickSort is:\n", i);
        QuickSort(&T3);
        printf("Table NO.%d after ShellSort is:\n", i);
        ShellSort(&T4);
        printf("Table NO.%d after HeapSort is:\n", i);
        HeapSort(&T5);
    }
    return 0;
}

void swap(SqList* L, int m, int n)
{
    int temp = L->l[m];
    L->l[m] = L->l[n];
    L->l[n] = temp;
}

void BubbleSort(SqList* L)
{
    int MoveCount = 0;
    int CompareCount = 0;
    for (int i = 1; i < L->length; i++) {
        for (int k = i + 1; k <= L->length; k++) {
            CompareCount++;
            if (L->l[i] > L->l[k]) {
                swap(L, i, k);
                MoveCount = MoveCount + 3;
            }
        }
    }
    printf("The times of comparision of BubbleSort is %d\n", CompareCount);
    printf("The times of movement of BubbleSort is %d\n", MoveCount);
}

void InsertSort(SqList* L)

```

```

{
    int j;
    int MoveCount = 0;
    int CompareCount = 0;
    for (int i = 2; i < L->length; i++) {
        CompareCount++;
        if (L->l[i] < L->l[i - 1]) {
            L->l[0] = L->l[i];
            MoveCount++;
            for (j = i - 1; L->l[j] > L->l[0]; j--) {
                L->l[j + 1] = L->l[j];
                CompareCount++;
                MoveCount++;
            }
            CompareCount++;
            L->l[j + 1] = L->l[0];
            MoveCount++;
        }
    }
    printf("The times of comparision of InsertSort is %d\n", CompareCount);
    printf("The times of movement of InsertSort is %d\n", MoveCount);
}

void SelectSort(SqList* L)
{
    int min;
    int MoveCount = 0;
    int CompareCount = 0;
    for (int i = 1; i < L->length; i++) {
        min = i;
        for (int j = i + 1; j <= L->length; j++) {
            CompareCount++;
            if (L->l[min] > L->l[j]) {
                min = j;
            }
        }
        if (i != min) {
            swap(L, i, min);
            MoveCount += 3;
        }
    }
    printf("The times of comparision of SelectSort is %d\n", CompareCount);
    printf("The times of movement of SelectSort is %d\n", MoveCount);
}

int Partition(SqList* L, int low, int high, int& CompareCount, int& MoveCount)
{
    int pivotkey;
    L->l[0] = L->l[low];
    pivotkey = L->l[low];
    while (low < high) {
        while (low < high && L->l[high] >= pivotkey) {
            CompareCount++;
            --high;
        }
        CompareCount++;
        swap(L, low, high);
        MoveCount += 3;
        while (low < high && L->l[low] <= pivotkey) {
            CompareCount++;
            low++;
        }
        CompareCount++;
    }
}

```

```

        swap(L, low, high);
        MoveCount += 3;
    }
    return low;
}

void QSort(SqList* L, int low, int high, int& CompareCount, int& MoveCount)
{
    int pivotloc;
    if (low < high) {
        pivotloc = Partition(L, low, high, CompareCount, MoveCount);
        QSort(L, low, pivotloc - 1, CompareCount, MoveCount);
        QSort(L, pivotloc + 1, high, CompareCount, MoveCount);
    }
}

void QuickSort(SqList* L)
{
    int MoveCount = 0;
    int CompareCount = 0;
    QSort(L, MoveCount, L->length, CompareCount, MoveCount);
    printf("The times of comparision of QuickSort is %d\n", CompareCount);
    printf("The times of movement of QuickSort is %d\n", MoveCount);
}

void ShellSort(SqList* L)
{
    int j;
    int MoveCount = 0;
    int CompareCount = 0;
    int increase = L->length;
    do {
        increase = increase / 5 + 1;
        for (int i = increase + 1; i < L->length; i++) {
            CompareCount++;
            if (L->l[i] < L->l[i - increase]) {
                L->l[0] = L->l[i];
                MoveCount++;
                for (j = i - increase; L->l[0] < L->l[j] && j > 0; j = j - increase) {
                    CompareCount++;
                    L->l[j + increase] = L->l[j];
                    MoveCount++;
                }
                CompareCount++;
                L->l[j + increase] = L->l[0];
                MoveCount++;
            }
        }
    } while (increase > 1);
    printf("The times of comparision of ShellSort is %d\n", CompareCount);
    printf("The times of movement of ShellSort is %d\n", MoveCount);
}

void HeapAdjust(SqList* L, int s, int m, int& CompareCount, int& MoveCount)
{
    int rc = L->l[s];
    MoveCount++;
    for (int i = 2 * s; i <= m; i *= 2) {
        CompareCount++;
        if (i < m && L->l[i] < L->l[i + 1]) {
            i++;
        }
    }
}

```

```

        CompareCount++;
        if (rc >= L->l[i])
            break;
        L->l[s] = L->l[i];
        MoveCount++;
        s = i;
    }
    L->l[s] = rc;
    MoveCount++;
}

void HeapSort(Sqlist* L)
{
    int i;
    int MoveCount = 0;
    int CompareCount = 0;
    for (i = L->length / 2; i > 0; i--) {
        HeapAdjust(L, i, L->length, CompareCount, MoveCount);
    }
    for (i = L->length; i > 1; i--) {
        swap(L, 1, i);
        MoveCount += 3;
        HeapAdjust(L, 1, i - 1, CompareCount, MoveCount);
    }
    printf("The times of comparision of HeapSort is %d\n", CompareCount);
    printf("The times of movement of HeapSort is %d\n", MoveCount);
}

```