

一 . 实验任务

1. 直接插入排序
2. 希尔排序
3. 冒泡排序
4. 快速排序
5. 堆排序
6. 归并排序

二 . 实验上机时间

三 . 知识点

1. 直接插入排序
2. 希尔排序
3. 冒泡排序
4. 快速排序
5. 堆排序
6. 归并排序

四 . 源代码及结果截屏

1. 直接插入排序

a. InsertSort.h

```
#ifndef INSERTSORT_H
#define INSERTSORT_H

#include <stdlib.h>
#include "Status.h"

#define MAXSIZE 20//待排顺序表最大长度
typedef int KeyType;//关键字类型为整数类型
typedef int InfoType;
typedef struct
{
    KeyType key;//关键字项
    InfoType otherinfo;//其他数据项
}RcdType;//记录类型
typedef struct
{
    RcdType r[MAXSIZE + 1];//R[0]闲置或用作哨兵单元
    int length;//顺序表长度
}SqList;//顺序表类型

Status InitList_Sq(SqList* L);
void InsertSort(SqList* L);
Status ListTraverse_Sq(SqList L, void(Visit)(ElemType));
```

```
#endif
```

b. InsertSort.c

```
#define _CRT_SECURE_NO_WARNINGS
#include "InsertSort.h"
#include "Status.h"
#include <stdio.h>
#include <malloc.h>

Status InitList_Sq(SqList* L)
{
    int i;
    for (i = 0; i <= MAXSIZE + 1; i++)
    {
        (*L).r[i].key = 0;
        (*L).r[i].otherinfo = 0;
    }
    (*L).length = 0;           //空表长度为0
    return OK;
}

void InsertSort(SqList* L)
{
    //对顺序表L作直接插入排序
    int i, j;
    for (i = 2; i <= (*L).length; ++i)
    {
        if ((*L).r[i].key < (*L).r[i - 1].key)
        {
            (*L).r[0] = (*L).r[i];
            (*L).r[i] = (*L).r[i - 1];
            for (j = i - 2; (*L).r[0].key < (*L).r[j].key; --j)
                (*L).r[j + 1] = (*L).r[j]; //记录后移
            (*L).r[j + 1] = (*L).r[0]; //插入到正确位置
        } //if
    } //for
} //InsertSort

Status ListTraverse_Sq(SqList L, void(Visit)(ElemType))
{
    int i;
    for (i = 1; i <= L.length; i++)
        Visit(L.r[i].key);
    return OK;
}
```


```
}
```

c. test.c

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "InsertSort.h"
void visit_user(KeyType e);
void visit_user(KeyType e)
{
    printf("%d ", e);
}

int main()
{
    SqList L;
    int i, n, m;
    InitList_Sq(&L);
    printf("请输入关键字数目: ");
    scanf("%d", &n);
    L.length = n;
    getchar();
    for (i = 1; i <= n; i++)
    {
        printf("请输入第%d个关键字:", i);
        scanf("%d", &m);
        getchar();
        L.r[i].key = m;
    }
    InsertSort(&L);
    ListTraverse_Sq(L, visit_user);

    system("pause");
    return 0;
}
```

 E:\大二上\数据结构\代码保存\InsertionSort\Debug\InsertionSort.exe

```
请输入关键字数目: 8
请输入第1个关键字: 34
请输入第2个关键字: 12
请输入第3个关键字: 49
请输入第4个关键字: 28
请输入第5个关键字: 31
请输入第6个关键字: 52
请输入第7个关键字: 51
请输入第8个关键字: 49
12 28 31 34 49 49 51 52 请按任意键继续. . .
```

2. 希尔排序

a. ShellSort.h

```
#ifndef SHELLSORT_H
#define SHELLSORT_H

#include <stdlib.h>
#include "Status.h"

#define MAXSIZE 20//待排顺序表最大长度
typedef int KeyType;//关键字类型为整数类型
typedef int InfoType;
typedef struct
{
    KeyType key;//关键字项
    InfoType otherinfo;//其他数据项
}RcdType;//记录类型
typedef struct
{
    RcdType r[MAXSIZE + 1];//R[0]闲置或用作哨兵单元
    int length;//顺序表长度
}SqList;//顺序表类型

Status InitList_Sq(SqList* L);
void ShellInsert(SqList* L, int dk);
void ShellSort(SqList* L, int dlt[], int t);
Status ListTraverse_Sq(SqList L, void(Visit)(KeyType));
```

#endif

b. ShellSort.c

```
#define _CRT_SECURE_NO_WARNINGS
#include "ShellSort.h"
#include "Status.h"
#include <stdio.h>
#include <malloc.h>

Status InitList_Sq(SqList* L)
{
    int i;
    for (i = 0; i <= MAXSIZE + 1; i++)
    {
        (*L).r[i].key = 0;
        (*L).r[i].otherinfo = 0;
    }
    (*L).length = 0; //空表长度为0
```

```

        return OK;
    }

void ShellInsert(SqList* L, int dk)
{
    int i, j;
    for (i = dk + 1; i <= (*L).length; ++i)
        if ((*L).r[i].key < (*L).r[i - dk].key)
        {
            (*L).r[0] = (*L).r[i]; //暂存在r[0], 不是监视哨
            for (j = i - dk; j > 0 && ((*L).r[0].key < (*L).r[j].key); j -= dk)
                (*L).r[j + dk] = (*L).r[j]; //记录后移, 查找插入位置
            (*L).r[j + dk] = (*L).r[0]; //插入
        } //if
} //ShellInsert

void ShellSort(SqList* L, int dlta[], int t)
{
    //增量为dlta[0...t-1]的希尔排序
    int k;
    for (k = 0; k < t; ++k)
        ShellInsert(&(*L), dlta[k]);
}

Status ListTraverse_Sq(SqList L, void(Visit)(ElemType))
{
    int i;
    for (i = 1; i <= L.length; i++)
        Visit(L.r[i].key);
    return OK;
}

```

c. Test.c

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "ShellSort.h"
void visit_user(KeyType e);
void visit_user(KeyType e)
{
    printf("%d ", e);
}

int main()
{
    //直接插入法排序

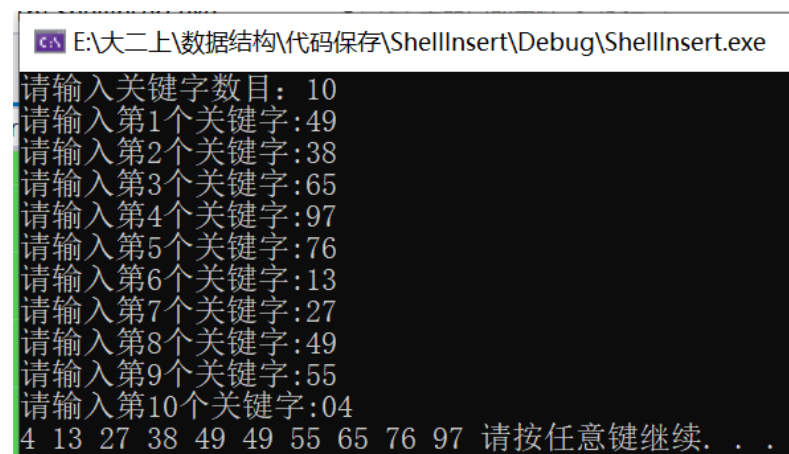
```

```

    SqList L;
    int i, n, m;
    InitList_Sq(&L);
    printf("请输入关键字数目: ");
    scanf("%d", &n);
    L.length = n;
    getchar();
    for (i = 1; i <= n; i++)
    {
        printf("请输入第%d个关键字:", i);
        scanf("%d", &m);
        getchar();
        L.r[i].key = m;
    }
    int t = 3;
    int dlta[3] = { 5, 3, 1 };
    ShellSort(&L, dlta, t);
    ListTraverse_Sq(L, visit_user);

    system("pause");
    return 0;
}

```



```

E:\大二上\数据结构\代码保存\ShellInsert\Debug\ShellInsert.exe
请输入关键字数目: 10
请输入第1个关键字: 49
请输入第2个关键字: 38
请输入第3个关键字: 65
请输入第4个关键字: 97
请输入第5个关键字: 76
请输入第6个关键字: 13
请输入第7个关键字: 27
请输入第8个关键字: 49
请输入第9个关键字: 55
请输入第10个关键字: 04
4 13 27 38 49 49 55 65 76 97 请按任意键继续. . .

```

3. 冒泡排序

BubbleSort.h 与 test.c 文件与前面类似,只列举 BubbleSort 函数

```

void BubbleSort(SqList* L)
{
    int i, j, swapTag;
    for (i = 1, swapTag = 1; i < (*L).length && swapTag; ++i)
    {
        swapTag = 0; //swapTag是存在交换操作的标志
        for (j = 1; j < (*L).length - i + 1; ++j)
            if ((*L).r[j].key > (*L).r[j + 1].key)
            {

```

```

        (*L).r[0] = (*L).r[j];
        (*L).r[j] = (*L).r[j + 1];
        (*L).r[j + 1] = (*L).r[0];
        swapTag = 1;
    } //if
} //for
} //BubbleSort

```

E:\大二上\数据结构\代码保存\InsertionSort\Debug\InsertionSort.exe

```

请输入关键字数目: 8
请输入第1个关键字: 49
请输入第2个关键字: 38
请输入第3个关键字: 65
请输入第4个关键字: 97
请输入第5个关键字: 76
请输入第6个关键字: 13
请输入第7个关键字: 27
请输入第8个关键字: 49
13 27 38 49 49 65 76 97 请按任意键继续. . .

```

4. 快速排序

QuickSort.h 与 test.c 文件与前面类似，只列举 QuickSort 函数

```

int Partition(SqlList* L, int low, int high)
{
    (*L).r[0] = (*L).r[low]; //用子表第一个记录作为枢轴记录
    int pivotkey = (*L).r[low].key; //枢轴
    while (low < high)
    {
        while (low < high && (*L).r[high].key >= pivotkey)
            --high; //从右向左搜索
        (*L).r[low] = (*L).r[high];
        while (low < high && (*L).r[low].key <= pivotkey)
            ++low; //从左向右搜索
        (*L).r[high] = (*L).r[low];
    }
    (*L).r[low] = (*L).r[0]; //枢轴记录到位
    return low; //返回枢轴位置
}

void QSort(SqlList* L, int low, int high)
{
    //对顺序表L中的子序列L.r[low...high]进行快速排序
    int pivotloc;
    if (low < high) //长度大于1
    {
        pivotloc = Partition(&(*L), low, high); //一次划分
        QSort(&(*L), low, pivotloc - 1); //对左子序列递归排序
    }
}


```

```

        QSort(&(*L), pivotloc + 1, high); //对右子序列递归排序
    }
} //QSort

void QuickSort(SqList* L)
{
    //对顺序表进行快速排序
    QSort(&(*L), 1, (*L).length);
} //QuickSort

```

 E:\大二上\数据结构\代码保存\InsertionSort\Debug\InsertionSort.exe

```

请输入关键字数目: 9
请输入第1个关键字: 49
请输入第2个关键字: 38
请输入第3个关键字: 65
请输入第4个关键字: 97
请输入第5个关键字: 13
请输入第6个关键字: 27
请输入第7个关键字: 49
请输入第8个关键字: 55
请输入第9个关键字: 04
4 13 27 38 49 49 55 65 97 请按任意键继续. . .

```

5. 堆排序

a. HeapSort.h

```

#ifndef HEAPSORT_H
#define HEAPSORT_H

#include <stdlib.h>
#include "Status.h"

#define MAXSIZE 20 //待排顺序表最大长度
typedef int KeyType; //关键字类型为整数类型
typedef int InfoType;
typedef struct
{
    KeyType key; //关键字项
    InfoType otherinfo; //其他数据项
} RcdType; //记录类型
typedef struct
{
    RcdType r[MAXSIZE + 1]; //R[0]闲置或用作哨兵单元
    int length; //顺序表长度
} SqList; //顺序表类型
typedef SqList HeapType; //定义堆类型: 采用顺序表表示

Status InitList_Sq(HeapType* H);
Status ListTraverse_Sq(HeapType H, void(Visit)(KeyType));

```



```

void HeapSort(HeapType* H); //对顺序表H进行堆排序
void HeapAdjust(HeapType* H, int s, int m);

```

```

#endif

```

b. HeapSort.c

```

#define _CRT_SECURE_NO_WARNINGS
#include "HeapSort.h"
#include "Status.h"
#include <stdio.h>
#include <malloc.h>

```

```

Status InitList_Sq(HeapType *H)
{
    int i;
    for (i = 0; i <= MAXSIZE + 1; i++)
    {
        (*H).r[i].key = 0;
        (*H).r[i].otherinfo = 0;
    }
    (*H).length = 0; //空表长度为0
    return OK;
}

```

```

Status ListTraverse_Sq(HeapType H, void(Visit)(KeyType))
{
    int i;
    for (i = 1; i <= H.length; i++)
        Visit(H.r[i].key);
    return OK;
}

```

```

void HeapSort(HeapType* H) //对顺序表H进行堆排序
{
    int i;
    RcdType swap;
    for (i = (*H).length / 2; i > 0; --i)
        HeapAdjust(&(*H), i, (*H).length);
    for (i = (*H).length; i > 1; --i)
    {
        swap = (*H).r[1];
        (*H).r[1] = (*H).r[i];
        (*H).r[i] = swap; //将堆顶记录和当前未经排序子序列H.r[1..r-1]重新调整堆
        HeapAdjust(&(*H), 1, i - 1); //对H.r[1..i-1]重新调整堆
    }
}

```

```

} //HeapSort

void HeapAdjust(HeapType* H, int s, int m)
{
    //已知H.r[s..m]中记录的关键字除H.r[s].key之外均满足堆的定义,
    //本函数调整H.r[s]的关键字, 使H.r[s..m]成为一个大顶堆
    int j;
    RcdType rc = (*H).r[s]; //暂存H.r[s]
    for (j = 2 * s; j <= m; j *= 2) //沿key较大的子结点向下筛选
    {
        if (j < m && (*H).r[j].key < (*H).r[j + 1].key)
            ++j; //j为key较大的记录的下标
        if (rc.key >= (*H).r[j].key)
            break; //找到rc插入位置s, 不需继续往下调整
        (*H).r[s] = (*H).r[j];
        s = j; //否则记录上移, 尚需继续往下调整
    }
    (*H).r[s] = rc; //将调整前的堆顶记录插入到s位置
} //HeapAdjust

```

c. Test.c

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "HeapSort.h"
void visit_user(KeyType e);
void visit_user(KeyType e)
{
    printf("%d ", e);
}

int main()
{
    //直接插入法排序
    HeapType H;
    int i, n, m;
    InitList_Sq(&H);
    printf("请输入关键字数目: ");
    scanf("%d", &n);
    H.length = n;
    getchar();
    for (i = 1; i <= n; i++)
    {
        printf("请输入第%d个关键字:", i);
        scanf("%d", &m);
        getchar();
    }
}

```

```

        H.r[i].key = m;
    }
    HeapSort(&H);
    ListTraverse_Sq(H, visit_user);

    system("pause");
    return 0;
}

```

C:\E:\大二上\数据结构\代码保存\InsertionSort\Debug\InsertionSort.exe

```

请输入关键字数目: 8
请输入第1个关键字: 47
请输入第2个关键字: 36
请输入第3个关键字: 53
请输入第4个关键字: 91
请输入第5个关键字: 12
请输入第6个关键字: 30
请输入第7个关键字: 24
请输入第8个关键字: 85
12 24 30 36 47 53 85 91 请按任意键继续. . .

```

6. 归并排序

a. MergingSort.h

```

#ifndef MERGINGSORT_H
#define MERGINGSORT_H

#include <stdlib.h>
#include "Status.h"

#define MAXSIZE 20//待排顺序表最大长度
typedef int KeyType;//关键字类型为整数类型
typedef int InfoType;
typedef struct
{
    KeyType key;//关键字项
    InfoType otherinfo;//其他数据项
}RcdType;//记录类型
typedef struct
{
    RcdType r[MAXSIZE + 1];//R[0]闲置或用作哨兵单元
    int length;//顺序表长度
}SqList;//顺序表类型

Status InitList_Sq(SqList* L);
Status ListTraverse_Sq(SqList L, void(Visit)(KeyType));
void MergeSort(SqList* L);
void MSort(RcdType SR[], RcdType* TR1[], int s, int t);

```

```
void Merge(RcdType SR[], RcdType* TR[], int i, int m, int n);
```

```
#endif
```

b. MergingSort.c

```
#define _CRT_SECURE_NO_WARNINGS
```

```
#include "MergingSort.h"
```

```
#include "Status.h"
```

```
#include <stdio.h>
```

```
#include <malloc.h>
```

```
Status InitList_Sq(SqList* L)
```

```
{
    int i;
    for (i = 0; i <= MAXSIZE + 1; i++)
    {
        (*L).r[i].key = 0;
        (*L).r[i].otherinfo = 0;
    }
    (*L).length = 0;           //空表长度为0
    return OK;
}
```

```
Status ListTraverse_Sq(SqList L, void(Visit)(ElemType))
```

```
{
    int i;
    for (i = 1; i <= L.length; i++)
        Visit(L.r[i].key);
    return OK;
}
```

```
void MergeSort(SqList* L)
```

```
{
    //对顺序表L作归并排序
    MSort((*L).r, &((*L).r), 1, (*L).length);
} //MergeSort
```

```
void MSort(RcdType SR[], RcdType TR1[], int s, int t)
```

```
{
    //将SR[s..t]归并排序为TR1[s..t]
    int m;
    RcdType TR2[MAXSIZE + 1];
    if (s == t)
        TR1[s] = SR[s];
```

```

else
{
    m = (s + t) / 2; //将SR[s..t]平分为SR[s..m]和SR[m+1..t]
    MSort(SR, &TR2, s, m); //将SR[s..m]归并为TR2[s..m]
    MSort(SR, &TR2, m+1, t); //将SR[m+1..t]归并为TR2[m+1..t]
    Merge(TR2, &TR1, s, m, t); //将TR2[s..m]和SR[m+1..t]归并到TR1[s..t]
}
} //MSort

void Merge(RcdType SR[], RcdType* TR[], int i, int m, int n)
{
    //将有序的子序列SR[i..m]和SR[m+1..n]归并为一个有序序列TR[i..n]
    int j, k;
    for (j = m + 1, k = i; i <= m && j <= n; ++k) //将SR中记录由小到大并入TR
    {
        if (SR[i].key <= SR[j].key)
            (*TR)[k] = SR[i++];
        else
            (*TR)[k] = SR[j++];
    }
    while (i <= m)
        (*TR)[k++] = SR[i++];
    while (j <= n)
        (*TR)[k++] = SR[j++];
} //Merge

```

c. Test.c

```

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include "MergingSort.h"
void visit_user(KeyType e);
void visit_user(KeyType e)
{
    printf("%d ", e);
}

int main()
{
    //直接插入法排序
    SqList L;
    int i, n, m;
    InitList_Sq(&L);
    printf("请输入关键字数目: ");
    scanf("%d", &n);
}

```


```

L.length = n;
getchar();
for (i = 1; i <= n; i++)
{
    printf("请输入第%d个关键字:", i);
    scanf("%d", &m);
    getchar();
    L.r[i].key = m;
}

MergeSort(&L);
ListTraverse_Sq(L, visit_user);

system("pause");
return 0;
}

```

 E:\大二上\数据结构\代码保存\ShellInsert\Debug\ShellInsert.exe

```

请输入关键字数目: 6
请输入第1个关键字: 52
请输入第2个关键字: 23
请输入第3个关键字: 80
请输入第4个关键字: 36
请输入第5个关键字: 68
请输入第6个关键字: 14
14 23 36 52 68 80 请按任意键继续. . .

```

五．实验总结

主要利用顺序表，写起来还是挺顺利的