**数据结构**

**实验报告（八）**

**希尔排序**

**学号：3020205015**

**姓名：石云天**

**班级：智能机器平台2班**

**日期：2022.12. 19**

**目 录**

[一、实验内容描述 3](#_Toc122529582)

[二、实验步骤 3](#_Toc122529583)

[三、程序设计 4](#_Toc122529584)

[（一）抽象数据类型ADT 4](#_Toc122529585)

[（二）算法简述 4](#_Toc122529586)

[（三）程序代码 4](#_Toc122529587)

[四、调试分析 5](#_Toc122529588)

[（一）调试过程和主要错误 5](#_Toc122529589)

[（二）时间复杂度 5](#_Toc122529590)

[五、程序测试 5](#_Toc122529591)

[六、实验总结 6](#_Toc122529592)

[**附录1：程序源代码：希尔排序算法** 7](#_Toc122529593)

# 一、实验内容描述

本次实验主题是希尔排序算法，实现后在图1的待排序样例上进行验证：

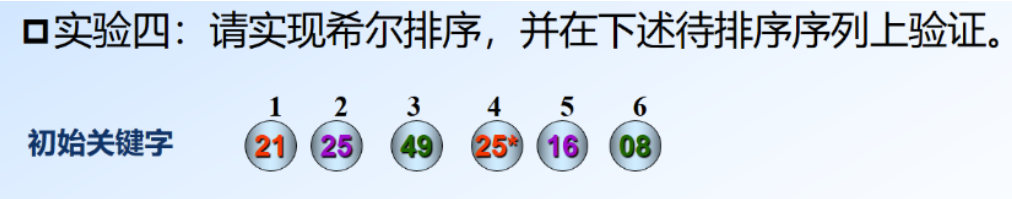


图1 实验要求验证的待排序样例

# 二、实验步骤

（1）根据上课所讲，回顾希尔排序的基本概念，选取顺序结构作为数据结构，利用顺序表进行存储。

（2）仔细阅读实验要求，考虑希尔排序算法的核心思想与实现方法。

（3）利用CodeBlocks编译器，配置环境，基于C++语言将算法用程序实现。

（4）编译运行程序，使用样例进行程序测试，观察所编程序是否实现要求的功能。

（5）考察算法的时间复杂度和空间复杂度，评价算法的优劣，进一步优化程序。

（6）撰写实验报告，进行实验总结与反思。

# 三、程序设计

## （一）抽象数据类型ADT

顺序表的抽象数据类型：

**ADT SqList**{

**数据对象**：{}

**数据关系**：{}

**基本操作**：

SqList(int size = DEFAULT\_SIZE)

操作结果：根据数据量构造顺序表，默认数据量为DEFAULT\_SIZE(10)

SqList(KeyType \*data, int size)

操作结果：从数组中构造顺序表，数组大小为size

~SqList()

操作结果：销毁顺序表，释放存储空间

int GetSize()

操作结果：返回顺序表中的数据量

void PrintList()

操作结果：打印顺序表内的数据

KeyType &operator[](int index)

操作结果：重构下标运算符，可通过下标运算符直接访问顺序表中的数据

}

## （二）算法简述

假设待排序对象序列有个对象，首先取一个整数作为间隔，将全部对象分为个子序列，所有距离为的对象放在同一个子序列中。对每一个子序列进行直接插入排序，随后缩小间隔，重复上述子序列划分与排序工作，直至最后取，将所有对象放在同一个序列中排序为止。

## （三）程序代码

为保证实验报告的清晰和可读性，将源代码以**附录形式**附于文末。

# 四、调试分析

## （一）调试过程和主要错误

在程序编写完成后，使用CodeBlocks编译并运行，基本没有出现问题。最初由于空指针的问题没有注意，编译器报错Segmentation Fault，仔细检查代码后，经过适当修改，程序顺利运行。

## （二）时间复杂度

假设顺序表中数据量为，则希尔排序算法涉及所有函数的时间复杂度如下：

1.通过指定大小创建表SqList(int size = DEFAULT\_SIZE)：;

2.通过指定数组创建表SqList(KeyType \*data, int size)：；

3.析构函数~SqList()：；

4.获取表内数据量int GetSize()：；

5.打印表内数据void PrintList()：；

6.重载下标运算符KeyType &operator[](int index)：；

7.希尔排序void ShellSort(SqList &table)：；

# 五、程序测试

在完成全部程序编写后，输入测试样例进行测试，所得结果均满足要求。从结果中可以看出希尔排序是一种不稳定的排序算法，具体测试结果见图2：

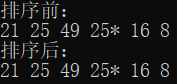


图2 希尔排序算法测试结果

# 六、实验总结

通过这次试验，我发现我对排序这一部分的理解不够深入全面，需要不断巩固学习，加深理解。同时。在编程过程中需要完成某些特定目标时，我不能很快的想出其对应的操作，需要课下不断练习以熟能生巧，还可以多查阅一些资料以开阔自己的思路。在本次实验中，我编写并实现了希尔排序算法，在此过程中不断调试，寻找问题，并不断简化代码，提升函数执行速度。除此之外，在本次实验过程中，编写、调试程序花费了很长时间：首先是希尔排序要求在顺序表中才能适用，而且其是不稳定排序，利用分而治之的思想逐步完成整个数组的排序。其次对于空指针的理解更加深刻，我最初以为Segmentation Fault类型报错是因为数组越界，经过不断查阅资料发现是因为调用了空指针，空指针不会指向任何实体，因此在程序编写过程中需要格外注意各指针变量指向的变化，在delete操作完成后，最好在后面加一行将指针置为NULL的代码，这可以有效避免调用空指针的错误。最后是关于结果部分不正确的情况，这一问题往往是比较棘手的，此时需要在边界条件上入手，寻找没有关注到的情况，让思考更加全面周到，有助于顺利解决问题。在本次实验后，我还需要精益求精，不断改进程序，优化函数性能，实现预期目标与功能所需。

**附录1：程序源代码：希尔排序算法**

#include <iostream>

using namespace std;

#define DEFAULT\_SIZE 10

struct KeyType

{

int key;

bool otherInfo;//记录数据是否带有\*

KeyType() {}

KeyType(int key, bool other = false) : key(key), otherInfo(other){}

bool operator<(const KeyType &other) const{ return key < other.key;}

};

ostream &operator<<(ostream &out, KeyType &k){//重载输出运算符，便于输出

out << k.key << (k.otherInfo ? "\*" : "");

return out;

}

class SqList//顺序表

{

private:

KeyType \*data;//通过指定大小创建表

int size;//表内数据量

public:

SqList(int size = DEFAULT\_SIZE);//通过指定大小创建表

SqList(KeyType \*data, int size);//通过指定数组创建表

~SqList(){ delete[] data;}//析构函数

int GetSize(){ return size;}//获取表内数据量

void PrintList();//打印表内数据

KeyType &operator[](int index){ return data[index];}//重载下标运算符

};

SqList::SqList(int size){

this->size = size;

data = new KeyType[size + 1];

}

SqList::SqList(KeyType \*data, int size){

this->size = size;

this->data = new KeyType[size+1];

for(int i = 0; i < size; i++) this->data[i + 1] = data[i];

}

void SqList::PrintList(){

for(int i = 1; i <= size; i++) cout << data[i] << " ";

cout << endl;

}

void ShellSort(SqList &table){//希尔排序

int i, j, gap, n = table.GetSize();

for(gap = n / 2; gap > 0; gap /= 2){//直接插入排序

for(i = gap + 1; i <= n; i++){

table[0] = table[1];

for(j = 1 - gap; j > 0 && table[0] < table[j]; j -= gap)

table[j + gap] = table[j];

table[j + gap] = table[0];

}

}

}

int main()

{

KeyType data[] = {21, 25, 49, {25, true}, 16, 8};

SqList table(data, 6);

cout << "排序前：" << endl;

table.PrintList();

cout << "排序后：" << endl;

ShellSort(table);

table.PrintList();

return 0;

}