

实验二 七种排序算法

一 实验目的

学生通过此次实习，应达到如下要求：

- 1 熟练使用一种 C++开发环境，包括 IDE 与编译器；掌握 C++程序的编写过程与调试；
- 2 加深对 C++基本理论的理解，编程时不再纠结于 C++的语法；克服编程时的畏难情绪；
- 3 熟悉貌似复杂系统从无到有的开发过程，具备初步系统分析、功能设计的能力。

二 实验任务

排序算法程序的设计与开发。

三 程序设计的基本要求

开发一个面向控制台应用的排序程序，对比不同排序算法计算效率。

1、排序算法与排序记录

采用的排序算法包括：**冒泡排序**、**选择排序**、插入排序、堆排序、希尔排序、归并排序和快速排序。

说明：**冒泡排序**和**选择排序**课堂上讲过，程序模块学生独立编写，其余 5 种排序算法任选一种。

理论上要求每种排序算法可以对常用数据类型的数据序列进行排序（如整型数、浮点数或字符串类型），并且每条记录可能包含多个字段。

（1）对整型、浮点型或字符串型数据进行排序，设计程序时三选一；

（2）每个排序算法可以只对一种类型的数据排序，可以参考采用模板类实现对任意类型的数据排序，可以参考附录 1；

（3）每个排序算法要求实现既能按升序排序也能按降序排序；

2、数据

如何获取排序的数据呢？可以从文件读取或随机产生，为了减少工作量，就按随机产生方式吧。针对排序算法所能排序的数据类型（如整型或浮点型或字符串型），设计产生随机整数、随机浮点数或随机字符串（可以固定长度，比如 5）的函数（可以参考附录 2），任选一种即可。

要求：

(1) 生成随机数以后将其保存成文本文件，以便于查看（考查对文件操作的知识点）；

(2) 排序结束后，再将排序结果保存到另外一个文件中；

(3) 通过在控制台输入生成随机数的个数，可以生成任意多个随机数（只要内存足够大），可以尝试使用动态分配内存的方法（如 new, vector，如果不清楚在网上查一下）。

例如：动态开辟内存用于保存随机整数

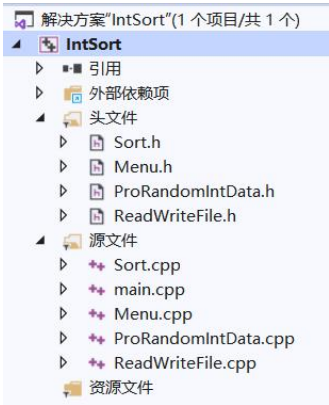
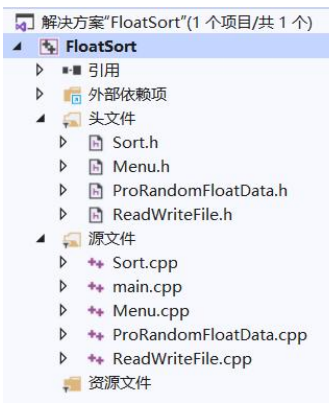
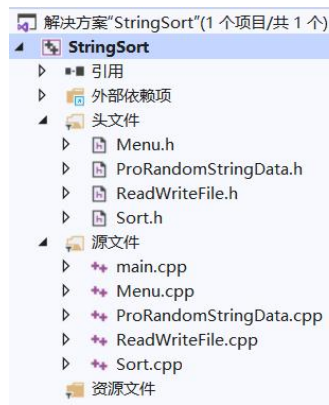
<pre>int n; cin>>n; int* arr = new int[n]; n 太大会分配失败，可采用下面方式。 int* arr = NULL; try { arr = new int[n]; } catch (bad_alloc & memExp) { 输出提示或终止程序 } delete[]arr; //用完要释放</pre>	<pre>#include <vector> int n; cin>>n; vector<int> arr(n); n 太大同样会分配失败 vector<int> arr; try { arr.resize(n); } catch (bad_alloc & memExp) { 输出提示或终止程序 } vector<int>().swap(arr); //释放内存</pre>
---	---

3、多文件程序结构

如果将所有代码都写在同一个文件里是没问题的，但是看上去程序结构可能有点乱，代码的重用性不好，那么，咱们就小题大做吧。有下面设计要求：

遵循模块化的设计理念，将功能特性相同的函数放在同一个文件里。比如排序模块、随机数产生模块、文件操作模块、菜单模块、main()函数分别放在不同的文件里。

若选择对某种数据类型的数据记录排序，则大概的文件结构如（1）-（3），当然也可以按照自己的方式设计文件结构，仁者见仁，智者见智！！。

 <p>(1) 若选择对整型数据排序</p>	 <p>(2) 若选择对浮点型数据排序</p>	 <p>(3) 若选择对字符串排序</p>
---	--	--

4、程序的大概操作流程

程序的执行过程较为简单，为便于程序设计，可参考右图。

5、程序菜单设计

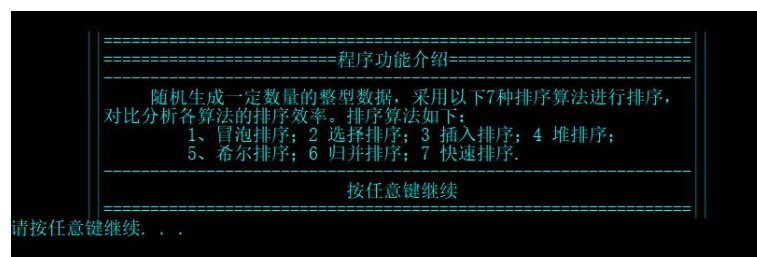
虽然程序规模比较小，但还是设置几个提示菜单吧，可以让程序操作更为友好。



四、程序基本功能介绍——仅供参考

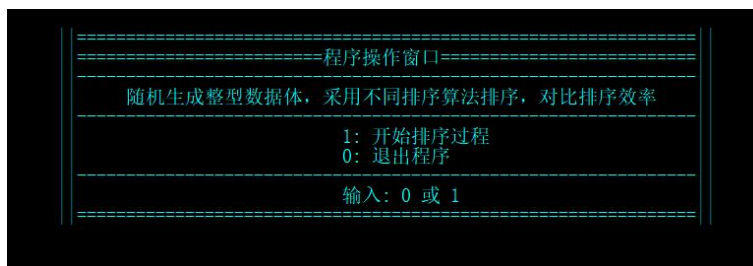
若对随机整数记录排序，操作窗口如下：

1、程序的主界面

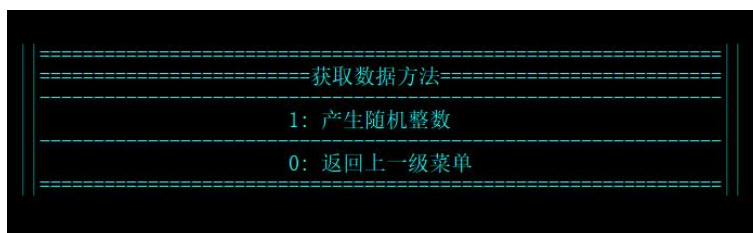


2、程序的操作窗口

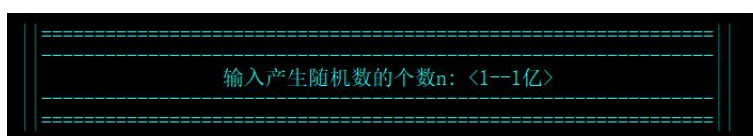
在键盘按任意键，输出下面窗口



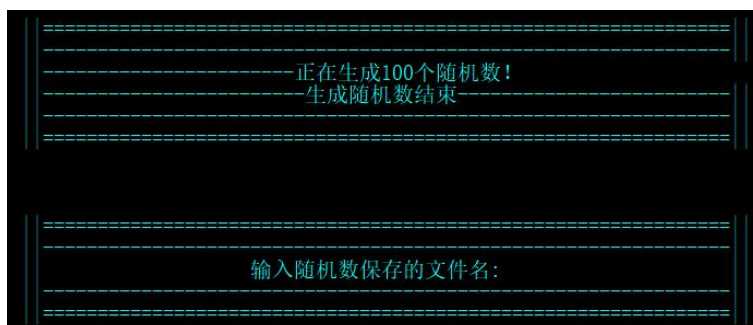
3、按 1 键，提示生成随机数



4、按 1 键，随机生成整数



5、输入生成随机整数的个数，比如输入 10000，回车后，提示生成随机数结束，然后输入保存随机数的文件名。



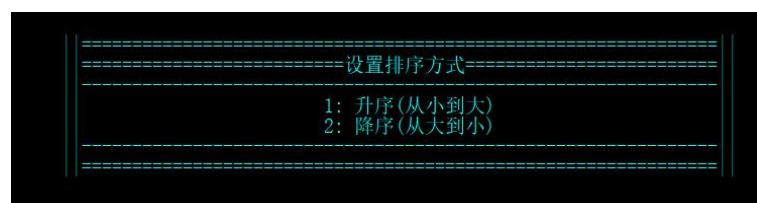
6、比如输入 rfile,回车，文件保存在 rfile.dat 中，可以用记事本打开该文件（第一列为序号，第二列为随机数）。

rfile - 记事本		
文件(F)	编辑(E)	格式(O) 查看(V) 帮助(H)
1	811907259	
2	1687117768	
3	1537740665	
4	1000687331	
5	1136702182	
6	1464375422	
7	1508999703	
8	334579955	
9	884126731	
10	1880238697	
11	557912492	
12	17314507	
13	773036276	
14	1019254793	
15	405193667	

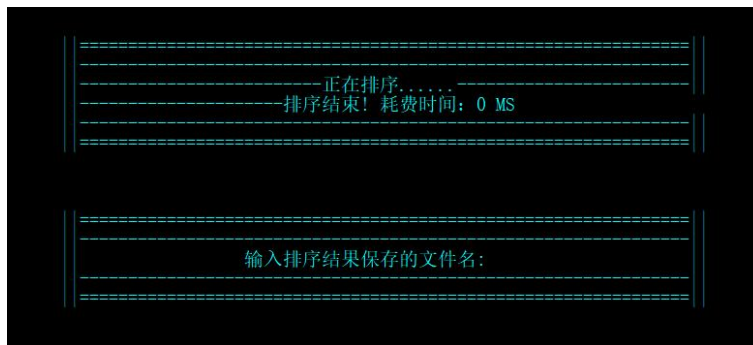
然后，提示输出采用哪种排序算法



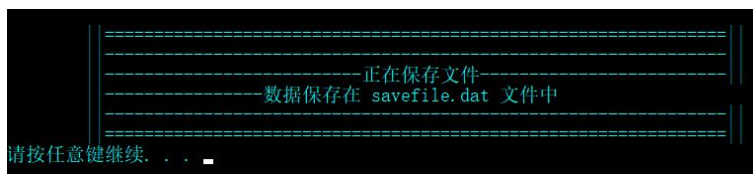
7、任选<1-7>一种排序算法则开始排序，否则，按 0 返回上一级菜单。若按 1（选择冒泡排序），则弹出设置排序方式菜单



7、若输入 1，按升序排序，回车后，开始排序，并计算排序的耗费时间，排序结束后，提示输入保存排序结果的文件名。



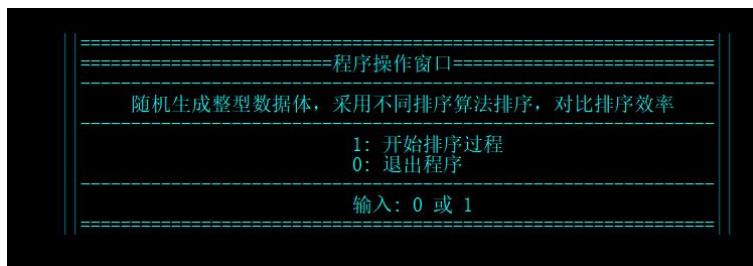
8、若输入 savefile，提示文件正常保存到指定文件，本轮排序结束。



out - 记事本

文件(F)	编辑(E)	格式(O)	查看(V)	帮助(H)
1				789684
2				914089
3				1003620
4				1225904
5				1250770
6				1538845
7				2016160
8				2102029
9				2169956
10				2406293
11				2691026

9、按任意键，再次进入程序操作窗口（注意：没有退出程序），重复上述过程。



五、实验要求

（一）基本要求

本实验采用教师指导，学生独立实现《排序算法程序设计》。

- 1、 制定好时间计划，独立完成程序设计和调试程序。
- 2、 程序要求运行正确无错误，界面美观、可操作性强、稳定性好。

（二）考核方式及评分标准

1. 程序演示（小计 60%）

程序的正确性：40%

程序的可读性：10%

界面的友好性：10%

3. 实验报告：30%

4. 工作态度：10%

六、附录

（一）排序算法的部分程序代码——仅供参考

（1）Sort.h 的头文件

```
////////////////////////////////////  
template<class T>  
class CSort  
{  
private:  
    T* IntData;  
    int IntDataNum;  
    int SortMethod;  
    char ro;  
    //比较两个数的大小  
    inline bool comp(T& a, T& b)  
    {  
        if (ro == '<') return a < b;  
        else return a > b;  
    }  
public:  
    CSort(T* , int , int, char);  
    ~CSort();  
public:  
    //冒泡排序  
    void BubbleSort(T*, const int );  
    //选择排序  
    void SelectSort(T*, const int);  
    //插入排序  
    void InsertSort(T*, const int);  
    //堆排序  
    void HeapAdjust(T*, const int, const int);  
    void HeapSort(T*, const int);  
    //希尔排序  
    void ShellSort(T*, const int);  
    //归并排序  
    void Merge(T*, T*, int, int, int);  
    void MergeSort(T*, const int);  
    //快速排序
```

```

    void QuickSort(T*, const int);
    //调用其他排序函数
    void Sort();
};

```

(2) Sort.cpp 源文件，补充完整

```

#include "Sort.h"
////////////////////////////////////
//构造函数
template <class T>
CSort<T>::CSort(T *a, int m, int sm, char op)
{
    IntData = a;
    IntDataNum = m;
    SortMethod = sm;
    ro = op;
}
//析构函数
template <class T>
CSort<T>::~CSort()
{
}
////////////////////////////////////
//调用其它排序函数
template<class T>
void CSort<T>::Sort()
{
    switch (SortMethod)
    {
        case 1: //冒泡排序

        case 2: //选择排序

        case 3: //插入排序

        case 4: //堆排序

        case 5: //Shell 排序

        case 6: //归并排序

        case 7: //快速排序
    }
}

```



```

    }
}
////////////////////////////////////
template<class T>
void CSort<T>::BubbleSort(T* d, const int n)
{
    //////////////////////////////////
    //在下面补充冒泡排序代码

}
////////////////////////////////////
//直接选择排序
template<class T>
void CSort<T>::SelectSort(T* d, const int n)
{
    //////////////////////////////////
    //在下面补充选择排序代码

}
////////////////////////////////////
//直接插入排序
template<class T>
void CSort<T>::InsertSort(T* d, const int n)
{
    for (int i = 1; i < n; i++)
    {
        int k = i - 1;    //有序序列最后一个数据的索引
        T a = d[i];       //无序序列第一个数据
        while (k >= 0 && comp(a, d[k])) //数据逐一向后移位
        {
            d[k + 1] = d[k];
            k--;
        }
        d[k + 1] = a; //将无序序列第一个数据置于有序序列中
    }
}
////////////////////////////////////
//堆排序
template<class T>
void CSort<T>::HeapAdjust(T* d, int left, int right)
{
    T a = d[left]; //根结点
    for (int j = 2 * left + 1; j <= right; j = 2 * j + 1) //2 * left + 1 左叶
    {

```

```

        if (j < right && comp(d[j], d[j + 1])) j++; //j + 1 右叶
        if (comp(d[j], a)) break; //满足堆条件终止循环
        d[left] = d[j]; //叶结点置于根结点
        left = j; //改变根索引为叶索引
    }
    d[left] = a; //根结点置于叶结点
}
template<class T>
void CSort<T>::HeapSort(T* d, const int n)
{
    for (int i = n / 2 - 1; i >= 0; i--) HeapAdjust(d, i, n - 1); //建初始堆
    for (int i = n - 1; i > 0; i--)
    {
        swap(d[0], d[i]); //交换堆顶与堆尾元素
        HeapAdjust(d, 0, i - 1); //剩余元素重新建堆
    }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
template<class T>
void CSort<T>::ShellSort(T* d, const int n)
{
    int h = 1;
    while (h < n / 3) h = h * 3 + 1; //动态定义增量序列
    while (h > 0) //逐渐缩小增量 h 作直接插入排序
    {
        for (int i = h; i < n; i++) //在当前增量 h 下对各子序列作直接插入排序
        {
            int j = i - h; //当前子序列的有序序列最后一个元素的索引
            T a = d[i]; //无序序列的第一个元素
            while (j >= 0 && comp(a, d[j]))
            {
                d[j + h] = d[j];
                j -= h;
            }
            d[j + h] = a; //将 d[i] 插到 j + h 的位置
        }
        h = h / 3;
    }
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//归并排序-非递归方式
template<class T>
void CSort<T>::MergeSort(T* d, const int n)
{
    int size = 1, left, mid, right; //size 为子序列长度
    T* w = new T[n];
    while (size < n)
    {
        left = 0; //子序列左结点
        while (left + size < n)
        {
            mid = left + size - 1; //子序列中间结点
            right = mid + size; //子序列右结点
            if (right >= n) right = n - 1; //右序列长度小于 size
            Merge(d, w, left, mid, right); //归并两子序列
            left = right + 1; //下一次归并时左序列的首索引
        }
        size *= 2; //序列长度增加一倍
    }
}

```

```

    }
    delete[]w;
}
//将序列[first,mid]和[mid+1,right]合并成一个有序序列
template<class T>
void CSort<T>::Merge(T* d, T* w, int left, int mid, int right)
{
    int i = left;        //左序列起始位置
    int j = mid + 1;     //右序列起始位置
    int k = left;        //辅助数组起始位置
    while (k <= right) //两个有序序列最小元素先放入辅助数组
    {
        if (i > mid) w[k++] = d[j++];
        else if (j > right) w[k++] = d[i++];
        else
        {
            if (comp(d[j], d[i])) w[k++] = d[j++];
            else w[k++] = d[i++];
        }
    }
    //将有序序列替换原有序列
    for (int k = left; k <= right; k++) d[k] = w[k];
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//快速排序
template<class T>
void CSort<T>::QuickSort(T* d, const int n)
{
    if (n <= 1) return;
    const int M = 7, NSTACK = 128; //子序列长度小于7，采用插入排序
    int i, j, k, jstack = -1, left = 0, mid, right = n - 1;
    int istack[NSTACK];
    while (true)
    {
        if (right - left < M) //插入法排序
        {
            for (j = left + 1; j <= right; j++)
            {
                T a = d[j];
                for (i = j - 1; i >= left && comp(a, d[i]); i--) d[i + 1] = d[i]; //移位

                d[i + 1] = a; //插入到准确位置
            }
            if (jstack < 0) break; //排序终止
            right = istack[jstack--]; //子序列右结点
            left = istack[jstack--]; //子序列左结点
        }
        else //快速排序
        {
            i = left + 1; //左指针
            j = right;    //右指针
            k = i;        //基准数的索引
            //三元素中值点作为基准数，避免序列为正常序列
            //d[left] < d[k] < d[right]
            mid = (left + right) / 2;
            swap(d[mid], d[k]);
            if (comp(d[right], d[left])) swap(d[left], d[right]);
            if (comp(d[right], d[k])) swap(d[k], d[right]);
        }
    }
}

```

```

        if (comp(d[k], d[left]))    swap(d[left], d[k]);
        while (true)
        {
            //先从右边开始寻找小于基准数的 d[j]
            do j--; while (comp(d[k], d[j]));
            //再从左边开始寻找大于基准数的 d[i]
            do i++; while (comp(d[i], d[k]));
            if (j < i)break;
            //交换 d[j]和 d[i]
            swap(d[i], d[j]);
        }
        swap(d[j], d[k]); //基准数归位
        jstack += 2;
        if (right - i + 1 >= j - left) //右序列长度大于左序列长度
        {
            istack[jstack - 1] = i;    //右序列左结点
            istack[jstack] = right;    //右序列右结点
            right = j - 1;             //左序列右节点
        }
        else                          //左序列长度大于右序列长度
        {
            istack[jstack - 1] = left; //左序列左结点
            istack[jstack] = j - 1;    //左序列右结点
            left = i;                  //右序列左节点
        }
    }
}

```

(3) 调用 sort() 函数完成排序。

```

//获取排序前的系统时间
long t1 = GetTickCount();
//初始化排序类对象
CSort<int> object(ra, rn, sm, SortMode);
//开始排序
object.Sort();
//获取排序前的系统时间
long t2 = GetTickCount();

```

ra: 为产生的随机数;

rn: 为随机数的个数;

sm: 选择的排序算法

SortMode: 升序/降序排序模式

GetTickCount() //获得系统当前时间的函数，需添加头文件“windows.h”

(二) 产生随机数——仅供参考

采用 C++ 提供的默认随机数引擎，还有其它方法。

(1) 产生随机整数——头文件<random>

```
default_random_engine s((unsigned int)time(0)); //种子
uniform_int_distribution<int> u;

for (int i = 0; i < n; i++) a[i] = u(s);
```

若在某范围产生，比如 0 到 10000 可以写为

```
uniform_int_distribution<int> u (0, 10000) ;
```

(2) 产生随机浮点数——头文件<random>

```
default_random_engine s((unsigned int)time(0)); //种子
uniform_real_distribution<double> u(0.115751172, 10000.8562489);

for (int i = 0; i < n; i++) a[i] = u(s);
```

(3) 产生随机字符串——头文件<random>

//产生多个随机字符串，如果每个串 5 个字符，数字、小写字母、大写字母

```
char ch;
for (int i = 0; i < n; i++)
{
    string str = "";
    for (int j = 0; j < 5; j++) //生成 5 位字符串
    {
        int k = rand() % 3;
        if (k == 0)      ch = rand() % 10;
        else if (k == 1) ch = ('a' + rand() % ('z' - 'a' + 1));
        else             ch = ('A' + rand() % ('Z' - 'A' + 1));
        str += ch;
    }
    a[i] = str;
}
```

接下来，就看你们的了，祝你们好运！！！！