

沈阳航空航天大学

课 程 设 计 报 告

课程设计名称：数据结构课程设计


课程设计题目：排序综合

学 院	计算机学院
专 业	计算机科学与技术
班 级	计算机 2001
学 号	203405010114
姓 名	徐新宇
指导教师	周唯

2021 年 12 月

沈阳航空航天大学

课程设计任务书

课程设计名称	数据结构课程设计			专业	计算机科学与技术
学生姓名	徐新宇	班级	计科 2001	学号	203405010114
题目名称	排序综合				
起止日期	2021 年 12 月 20 日起至 2021 年 12 月 31 日止				
<p>课设内容和要求:</p> <p>问题描述:</p> <p>对直接插入排序、冒泡排序、选择排序、快速排序、归并排序、堆排序、基数排序的算法进行演示，并输出排序结果情况。</p> <p>基本要求:</p> <ol style="list-style-type: none"> 1. 程序的输入: 排序种类的输入、排序数的个数的输入和所需排序的具体数字的输入; 2. 程序的输出: 主菜单的输出和排序结果的输出; 3. 排序结果输出要求保存到 txt 文件中; 4. 综合排序时, 要求比较两种较快排序, 输出结果, 并将所有排序所花费的时间全部写入到指定的文件中。 <p>参考资料:</p> <p>[1] 严蔚敏, 吴伟民. 数据结构[M]. 北京: 清华大学出版社, 2012</p> <p>[2] (美) 萨尼著, 王立柱, 刘志红等译. 数据结构、算法与应用: C++语言描述. 北京: 机械工业出版社, 2015</p> <p>[3] 何钦铭. C 语言程序设计. 北京: 高等教育出版社, 2015</p>					
<p>课程负责人审核意见: <input checked="" type="checkbox"/> 同意 <input type="checkbox"/> 不同意 课程负责人签字(盖章): </p>					
指导教师(签名)	周唯		2021 年 12 月 20 日		
学生(签名)	徐新宇		2021 年 12 月 20 日		

目录

1	题目介绍.....	1
1.1	问题描述.....	1
1.1.1	问题背景.....	1
1.1.2	主要任务.....	1
1.2	问题分析.....	1
2	系统总体设计.....	2
3	数据结构与算法设计.....	4
3.1	数据结构设计.....	4
3.2	算法设计.....	4
4	系统运行与测试.....	12
4.1	调试及调试分析.....	12
4.2	测试用例.....	12
5	总 结.....	15
	参考文献.....	16
	附 录 （程序清单）.....	17

1 题目介绍

1.1 问题描述

1、输入 n 个数，选择排序方法（直接插入排序、冒泡排序、选择排序、快速排序、归并排序、堆排序、基数排序），输出结果和所用的时间，并将结果保存到 txt 文件中。

2、综合排序: 随机生成 n 个数，用以上排序方法分别进行排序，输出每种方法排序完所用时间，并将所有排序所花费的时间全部写入到指定的文件中，并比较出两种较快排序。

1.1.1 问题背景

排序是计算机程序设计中的一种重要操作。它的功能是将一些数据元素的任意序列，重新排列成有序序列。

排序的方法很多，但是就其全面性能而言，很难找到一个被认为最好的算法，每一种算法都有各自的优缺点，适合在不同环境下使用。

1.1.2 主要任务

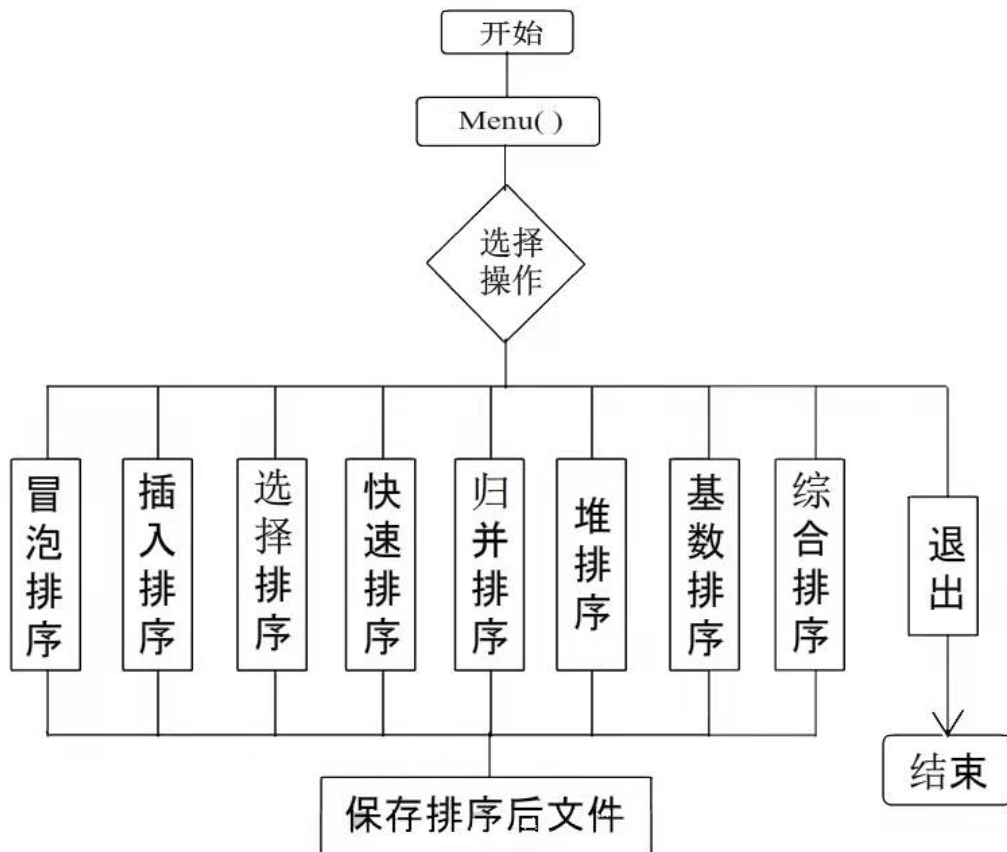
1、写出基本的排序算法（直接插入排序、冒泡排序、选择排序、快速排序、归并排序、堆排序、基数排序），输出每种算法排序 n 个数所用的时间（ n 个数为用户输入）。

2、比较每种排序算法排序 n 个数所用的时间（ n 为随机生成），并将所有排序所花费的时间全部写入到指定的文件中，并比较出两种较快排序。

1.2 问题分析

为完成课设任务，需要熟练掌握直接插入排序、冒泡排序、选择排序、快速排序、归并排序、堆排序、基数排序等算法，并能深入了解算法中每条语句的含义，能够分析每种算法的时间复杂度和稳定性。

2 系统总体设计



主程序

程序开始运行后，调用主菜单函数 Menu()，输入 c 的值并判断 c 的值：

如果 c 的值等于 1，调用 InsertSort()函数，用插入排序的算法进行排序，并将排后的结果和所用时间输出，并调用 File()函数将排序后的结果和所用时间保存到 txt 文件中。

如果 c 的值等于 2，调用 BubbleSort()函数，用冒泡排序的算法进行排序，并将排后的结果和所用时间输出，并调用 File()函数将排序后的结果和所用时间保存到 txt 文件中。

如果 c 的值等于 3，调用 SelectSort()函数，用选择排序的算法进行排序，并将排后的结果和所用时间输出，并调用 File()函数将排序后的结果和所用时间保存到 txt 文件中。

如果 c 的值等于 4，调用 QuickSort()函数，用快速排序的算法进行排序，并将排后的结果和所用时间输出，并调用 File()函数将排序后的结果和所用时间保存到 txt 文件中。

如果 c 的值等于 5，调用 MergeSort ()函数，用归并排序的算法进行排序，并将排后的结果和所用时间输出，并调用 File()函数将排序后的结果和所用时间保存到 txt 文件中。

如果 c 的值等于 6，调用 HeapSort ()函数，用堆排序的算法进行排序，并将排后的结果和所用时间输出，并将排序后的结果和所用时间保存到 txt 文件中。

如果 c 的值等于 7，调用 radixsort()函数，用基数排序的算法进行排序，并将排后的结果和所用时间输出，并调用 File()函数将排序后的结果和所用时间保存到 txt 文件中。

如果 c 的值等于 8，进行综合排序，随机生成 n 个数，分别进行以上 7 种排序，输出每种排序所需时间，并比较出两种较快排序，并将所需时间和排序结果保存到 txt 文件中。

如果 c 的值等于 9，退出该程序。

如果 c 的值为其他值，则会提示“没有你选择的项目请重新选择！”

3 数据结构与算法设计

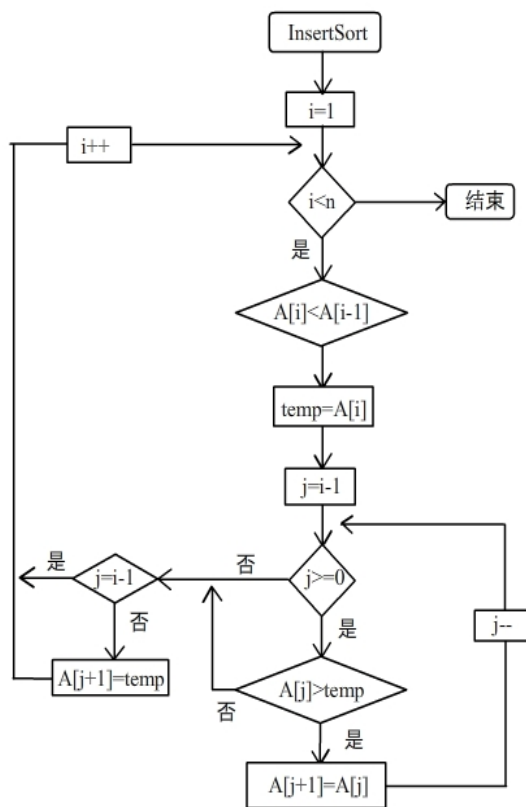
3.1 数据结构设计

```
int a[MaxSize];
```

数组 $a[\text{MaxSize}]$ 用来存放待排序元素， MaxSize 是表示数组的最大容量。

用数组存放，方便排序。

3.2 算法设计

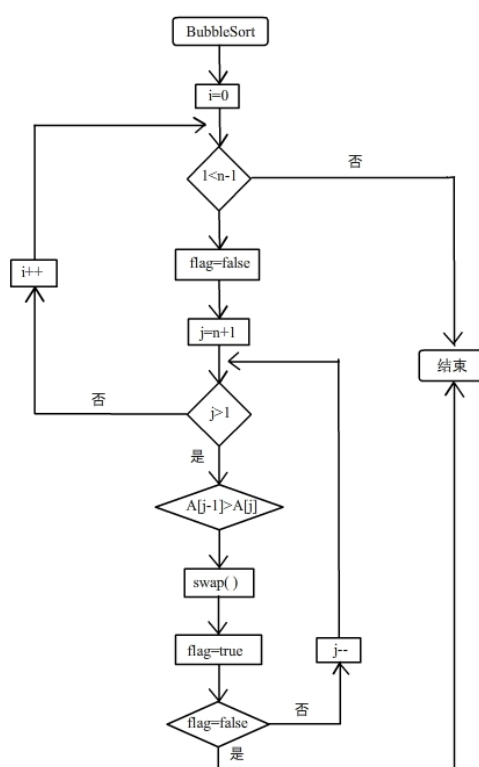


插入排序流程图

直接插入排序的基本思想：

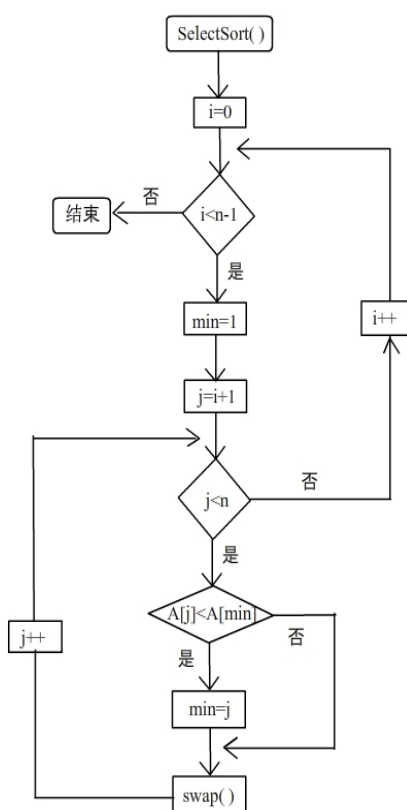
直接插入排序的基本思想是：把 n 个待排序的元素看成为一个有序表和一个无序表，开始时有序表中只包含一个元素，无序表中包含有 $n-1$ 个元素，排序过程中每次从无序表中取出第一个元素，将它插入到有序表中的适当位置，使之成为新的有序表，重复 $n-1$ 次可完成排序过程。

把 $a[i]$ 插入到 $a[0], a[1], \dots, a[i-1]$ 之中的具体实施过程为:先把 $a[i]$ 赋值给变量 $temp$,然后将 $temp$ 依次与 $a[i-1], a[i-2], \dots$ 进行比较,将比 t 大的元素右移一个位置,直到发现某个 $j(0 \leq j \leq i-1)$,使得 $a[j] \leq t$ 或 j 为 (-1) ,把 t 赋值给 $a[j+1]$.



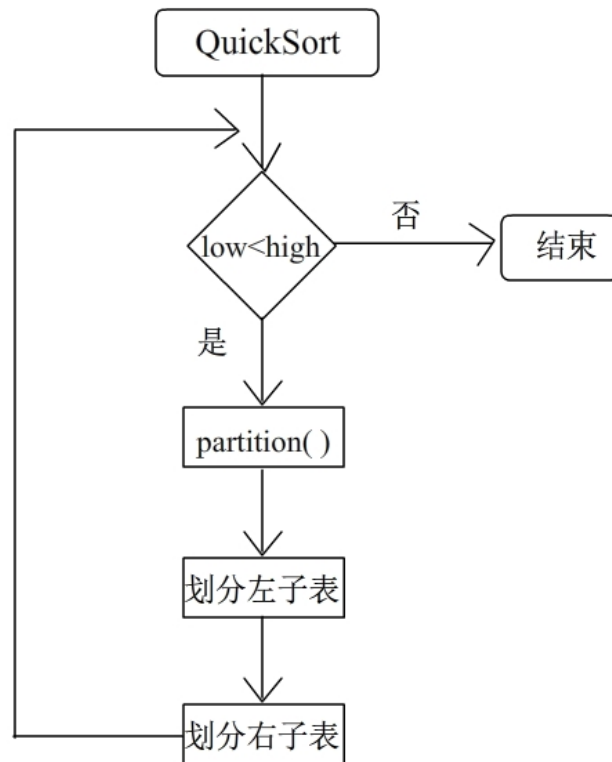
冒泡排序流程图

冒泡排序是一种稳定的排序算法，他的基本思想是：重复地走访过要排序的元素列，依次比较两个相邻的元素，如果顺序（如从大到小、首字母从 **Z** 到 **A**）错误就把他们交换过来。走访元素的工作是重复地进行直到没有相邻元素需要交换，也就是说该元素列已经排序完成。



选择排序流程图

选择排序法是一种不稳定的排序算法。它的工作原理是每一次从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。以此类推，直到全部待排序的数据元素排完。



快速排序流程图

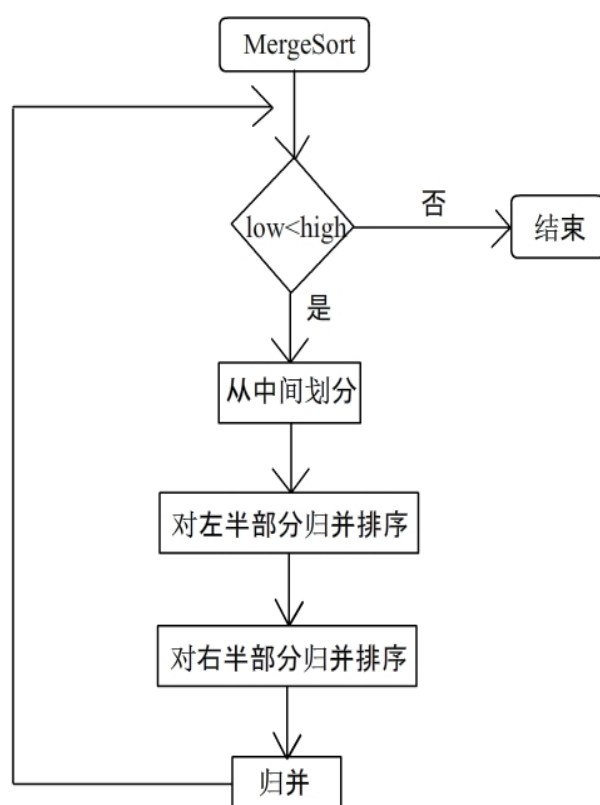
快速排序是一种不稳定的排序算法，他的基本思想是：快速排序使用分治的思想，通过一趟排序将待排序列分割成两部分，其中一部分记录的关键字均比另一部分记录的关键字小。之后分别对这两部分记录继续进行排序，以达到整个序列有序的目的。

主要分为三个步骤：

(1)选择基准：在待排序列中，按照某种方式挑出一个元素，作为 "基准" (pivot)

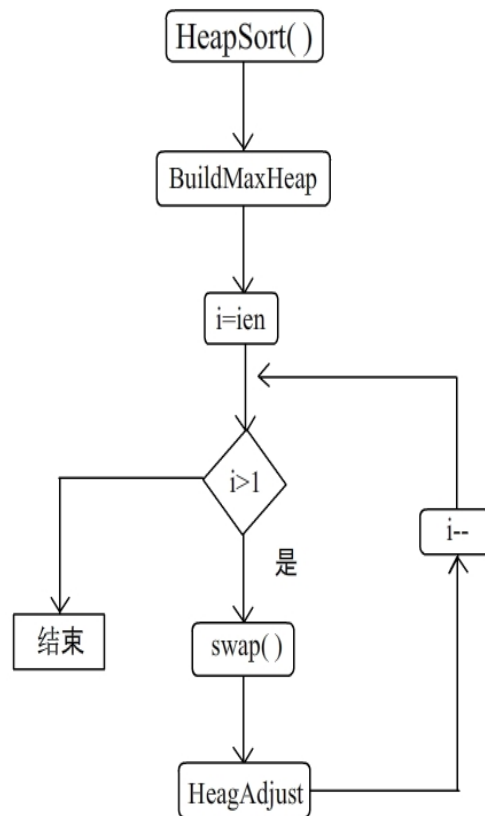
(2)分割操作：以该基准在序列中的实际位置，把序列分成两个子序列。此时，在基准左边的元素都比该基准小，在基准右边的元素都比基准大

(3)递归地对两个序列进行快速排序，直到序列为空或者只有一个元素。



归并排序流程图

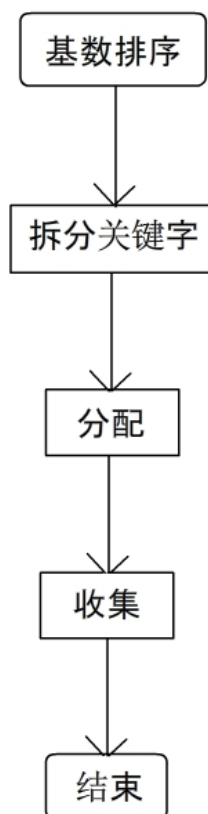
归并排序是一种稳定的排序算法，他的基本思想是将待排序文件看成为 n 个长度为 1 的有序子文件，把这些子文件两两归并，使得得到「 $n/2$ 」个长度为 2 的有序子文件；然后再把这「 $n/2$ 」个有序文件的子文件两两归并，如此反复，直到最后得到一个长度为 n 的有序文件为止，这种排序方法成为二路归并排序。



堆排序流程图

堆排序的基本思想是：将待排序序列构造成一个大顶堆，此时，整个序列的最大值就是堆顶的根节点。将其与末尾元素进行交换，此时末尾就为最大值。然后将剩余 $n-1$ 个元素重新构造成一个堆，这样会得到 n 个元素的次小值。如此反复执行，便能得到一个有序序列了。

堆排序是一种不稳定的排序。



基数排序流程图

基数排序的是一种稳定的排序，他的基本思想是将所有待比较数值统一为同样的数位长度，数位较短的数前面补零。然后，从最低位开始，依次进行一次排序。这样从最低位排序一直到最高位排序完成以后，数列就变成一个有序序列。

4 系统运行与测试

4.1 调试及调试分析

(1) 时间问题

问题描述：当待排序数量过少的时候，无法看出时间差异

解决方法：增大待排序元素的个数

(2) 输出问题

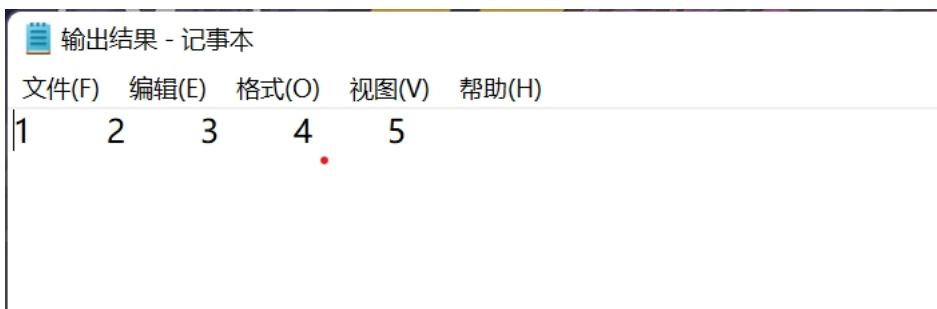
问题描述：输出时间为 0；

解决方法：定义时间应该为 double 类型

4.2 测试用例

```

选择C:\Users\HH\documents\visual studio 2010\Projects\ZONGHE\Debug\ZONGHE.exe
*****
*****欢迎使用排序综合演示系统*****
***      插入排序请按 1      ***
***      冒泡排序请按 2      ***
***      选择排序请按 3      ***
***      快速排序请按 4      ***
***      归并排序请按 5      ***
***      堆 排序请按 6      ***
***      基数排序请按 7      ***
***      综合排序请按 8      ***
***      退出请按    0      ***
*****
5
请输入待排序元素个数：
5
请输入5个元素：
5 4 3 2 1
排序后的结果为：
1 2 3 4 5
排序所用时间：0.000000 秒
请按任意键继续. . .
    
```




```

综合排序输出结果 - 记事本
文件(F) 编辑(E) 格式(O) 视图(V) 帮助(H)

插入排序所用时间: 0.103000 秒
冒泡排序所用时间: 1.025000 秒
选择排序所用时间: 0.159000 秒
快速排序所用时间: 0.002000 秒
归并排序所用时间: 0.004000 秒
堆 排序所用时间: 0.004000 秒
基数排序所用时间: 0.001000 秒

最快两种的排序为: |
基数排序: 0.001000 秒

快速排序: 0.002000 秒

排序的结果为:
0   3   7   9   10  13  15  17  17  21  42  44  47
49  50  58  60  63  63  64  65  69  74  74  79  79
 80  87  95  96  97  98 101 107 114 118 127 132
135 142 142 162 162 166 175 175 180 188 189
190 206 207 211 211 212 213 218 224 224 224 224
 238 240 241 242 242 242 245 252 258 268 269
272 274 274 277 278 278 279 286 286 291 301 301
 303 304 304 315 318 318 323 329 330 334 335
336 338 342 342 344 347 349 354 361 364 367 372
 377 380 386 395 403 404 405 405 406 408 421
    
```

```

选择C:\Users\HH\documents\visual studio 2010\Projects\ZON...
*****
*****欢迎使用排序综合演示系统*****
***      插入排序请按 1      ***
***      冒泡排序请按 2      ***
***      选择排序请按 3      ***
***      快速排序请按 4      ***
***      归并排序请按 5      ***
***      堆 排序请按 6      ***
***      基数排序请按 7      ***
***      综合排序请按 8      ***
***      退出请按 0          ***
*****
999
没有你选择的项目请重新选择!
请按任意键继续. . .
    
```

5 总 结

通过本次课程设计，我深入的了解了并学会了七大基本排序（直接插入排序、冒泡排序、选择排序、快速排序、归并排序、堆排序、基数排序）的算法，时间复杂度，和稳定性，并且在设计过程中虽然遇到了一些问题，但经过一次又一次的思考，一遍又一遍的检查终于找出了原因所在，也暴露出了前期我在这方面的知识欠缺和经验不足。实践出真知，通过亲自动手制作，使我们掌握的知识不再是纸上谈兵。

回顾起此课程设计，至今我仍感慨颇多，从理论到实践，在这段日子里，可以说得是苦多于甜，但是可以学到很多很多的东西，同时不仅可以巩固了以前所学过的知识，而且学到了很多在书本上所没有学到过的知识。同时也暴露出了本人的一些问题，就如比如在学习的时候，只专注了算法的大致思想，在代码实现上还存在这一些细节需要认真学习，不能只专注于理论学习，更要理论与实践相结合。

参考文献

- [1] 严蔚敏, 吴伟民.数据结构[M].北京: 清华大学出版社, 2012
- [2] (美)萨尼著, 王立柱, 刘志红等译.数据结构、算法与应用: C++语言描述. 北京: 机械工业出版社, 2015
- [3] 何钦铭. C 语言程序设计. 北京: 高等教育出版社, 2015
- [4] 高一凡.数据结构算法解析(第二版).北京: 清华大学出版社, 2015

附 录

(程序清单)

```
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<time.h>
#define MaxSize 80000
using namespace std;

//交换两个元素;

void swap(int &a,int &b){
    int temp=a;
    a=b;
    b=temp;
}

//直接插入排序 On2; 算法稳定
void InsertSort(int A[],int n){
    int i,j,temp;
    for(i=1;i<n;i++){
        //将各元素插入已排好的序列中
        if(A[i]<A[i-1]){
            //若 A[i]小于其前驱
            temp=A[i];
            for(j=i-1;j>=0&&A[j]>temp;j--){
                A[j+1]=A[j];
                //所有大于 temp 的元素往后挪
                A[j+1]=temp;
            }
        }
    }
}

//冒泡排序(从后往前冒泡, 最小的数到前面); On2 算法稳定
void BubbleSort(int A[],int n){
    for(int i=0;i<n-1;i++){
        bool flag=false; //表示本趟冒泡是否发生交换的标志
        for(int j=n-1;j>i;j--){
            if(A[j-1]>A[j]){
```

```

        swap(A[j-1],A[j]);
        flag=true;
    }
    if(flag==false)
        return;    //本趟没有发生交换，说明已经有序
}
}

```

//选择排序;

```

void SelectSort(int A[],int n){
    int i,j;
    for(i=0;i<n-1;i++){
        int min=i;
        for(j=i+1;j<n;j++)
            if(A[j]<A[min])
                min=j;
        swap(A[min],A[i]);
    }
}

```

//划分;

```

int Partition(int A[],int low,int high){
    int pivot=A[low];    //用第一个元素作为枢纽
    while(low<high){
        while(low<high&&A[high]>=pivot)
            high--;
        A[low]=A[high];    //比枢纽小的元素移动到左端
        while(low<high&&A[low]<=pivot)
            low++;
        A[high]=A[low];    //比枢纽大的元素移动到右端
    }
    A[low]=pivot;    //枢纽元素存放到最终位置
    return low;
}

```

//快速排序; $n \log n$ 不稳定

```

void QuickSort(int A[],int low,int high){
    if(low<high){    //递归跳出的条件
        int pivotpos=Partition(A,low,high);    //划分
        QuickSort(A,low,pivotpos-1);
        QuickSort(A,pivotpos+1,high);
    }
}

```

```
}
```

//将以 k 为根的子树调整为大根堆;

```
void HeadAdjust(int A[],int k,int len){
    A[0]=A[k];
    for(int i=2*k;i<=len;i*=2){
        if(i<len&&A[i]<A[i+1])
            i++;
        if(A[0]>=A[i]) break;
        else{
            A[k]=A[i];
            k=i;
        }
    }
    A[k]=A[0];
}
```

//建立大根堆

```
void BuildMaxHeap(int A[],int len){
    for(int i=len/2;i>0;i--){
        HeadAdjust(A,i,len);
    }
}
```

//堆排序 不稳定 建堆 $O(n)$ 排序 $O(n \log n)$

```
void HeapSort(int A[],int len){
    BuildMaxHeap(A,len);
    for(int i=len;i>1;i--){
        swap(A[i],A[1]);
        HeadAdjust(A,1,i-1);
    }
}
```

```
int B[MaxSize];
```

```
void Merge(int A[],int low,int mid,int high){
    int i,j,k;
    for(k=low;k<=high;k++)
        B[k]=A[k];
    for(i=low,j=mid+1,k=i;i<=mid&&j<=high;k++){
        if(B[i]<=B[j])
            A[k]=B[i++];    //将较小的数复制到 A 中
```

```

        else
            A[k]=B[j++];
    }
    while(i<=mid) A[k++]=B[i++];
    while(j<=high) A[k++]=B[j++];
}

//归并排序； O nlogn 稳定
void MergeSort(int A[],int low,int high){
    if(low<high){
        int mid=(low+high)/2; //从中间划分
        MergeSort(A,low,mid); //对左半部分归并排序
        MergeSort(A,mid+1,high); //对右半部分归并排序
        Merge(A,low,mid,high); //归并；
    }
}

//辅助函数，求数据的最大位数
int maxbit(int data[], int n) {
    int maxData = data[0];          ///< 最大数
    /// 先求出最大数，再求其位数，这样有原先依次每个数判断其位数，稍微优化点。
    for (int i = 1; i < n; ++i)
    {
        if (maxData < data[i])
            maxData = data[i];
    }
    int d = 1;
    int p = 10;
    while (maxData >= p)
    {
        //p *= 10; // Maybe overflow
        maxData /= 10;
        ++d;
    }
    return d;
}

//基数排序 稳 O(d(n+r))
void radixsort(int data[], int n)
{
    int d = maxbit(data, n);

```

```

int *tmp = new int[n];
int *count = new int[10]; //计数器
int i, j, k;
int radix = 1;
for(i = 1; i <= d; i++) //进行 d 次排序
{
    for(j = 0; j < 10; j++)
        count[j] = 0; //每次分配前清空计数器
    for(j = 0; j < n; j++)
    {
        k = (data[j] / radix) % 10; //统计每个桶中的记录数
        count[k]++;
    }
    for(j = 1; j < 10; j++)
        count[j] = count[j - 1] + count[j]; //将 tmp 中的位置依次分配给每个桶
    for(j = n - 1; j >= 0; j--) //将所有桶中记录依次收集到 tmp 中
    {
        k = (data[j] / radix) % 10;
        tmp[count[k] - 1] = data[j];
        count[k]--;
    }
    for(j = 0; j < n; j++) //将临时数组的内容复制到 data 中
        data[j] = tmp[j];
    radix = radix * 10;
}
delete []tmp;
delete []count;
}

void Menu(){
    cout<<"*****\n";
    cout<<"*****欢迎使用排序综合演示系统*****\n";
    cout<<"***      插入排序请按 1          ***\n";
    cout<<"***      冒泡排序请按 2          ***\n";
    cout<<"***      选择排序请按 3          ***\n";
    cout<<"***      快速排序请按 4          ***\n";
    cout<<"***      归并排序请按 5          ***\n";
    cout<<"***      堆   排序请按 6          ***\n";
    cout<<"***      基数排序请按 7          ***\n";
    cout<<"***      综合排序请按 8          ***\n";
    cout<<"***      退出请按      0          ***\n";
    cout<<"*****\n";
}

```



```

void File(int a[],int n){
    FILE * fp;
    fp= fopen("C:\\Users\\HH\\Desktop\\输出结果.txt","w");
    for(int i=0;i<n;i++)
        fprintf(fp,"%-8d ",a[i]);
    if(fp!=NULL)
        fclose(fp); //关闭文件
}

void main(){
    int c;
    int a[MaxSize];
    int b[MaxSize];
    int arr[MaxSize+1];
    int n=-1;
    clock_t start, finish ;
    double duration = 0.0 ;
    double t[8];
    while(1){
        Menu();
        cin>>c;
        switch(c){

        case 1:
            cout<<"请输入待排序元素个数: \n";
            cin>>n;
            cout<<"请输入" <<n<<"个元素: \n";
            for(int i=0;i<n;i++)
                cin>>a[i];

            start = clock() ;
            InsertSort(a,n);           //插入排序;
            finish = clock() ;
            duration = (double)(finish - start) / CLOCKS_PER_SEC;
            cout<<"排序后的结果为: \n";
            for(int i=0;i<n;i++)
                cout<<a[i]<<" ";
            cout<<"\n";
            printf("排序所用时间: %lf 秒\n",duration);
            File(a,n);
            a[MaxSize]=0;
            system("pause");
        }
    }
}

```

```

system("cls");
break;

case 2:
    cout<<"请输入待排序元素个数: \n";
    cin>>n;
    cout<<"请输入"<<n<<"个元素: \n";
    for(int i=0;i<n;i++)
        cin>>a[i];
    start = clock() ;
    BubbleSort(a,n);           //冒泡排序;
    finish = clock() ;
    duration= (double)(finish - start) / CLOCKS_PER_SEC;
    cout<<"排序后的结果为: \n";
    for(int i=0;i<n;i++)
        cout<<a[i]<<" ";
    cout<<"\n";
    printf("排序所用时间: %lf 秒\n",duration) ;
    File(a,n);
    a[MaxSize]=0;
    system("pause");
    system("cls");
    break;

case 3:
    cout<<"请输入待排序元素个数: \n";
    cin>>n;
    cout<<"请输入"<<n<<"个元素: \n";
    for(int i=0;i<n;i++)
        cin>>a[i];
    start = clock() ;
    SelectSort(a,n);          //选择排序;
    finish = clock() ;
    duration= (double)(finish - start) / CLOCKS_PER_SEC;
    cout<<"排序后的结果为: \n";
    for(int i=0;i<n;i++)
        cout<<a[i]<<" ";
    cout<<"\n";
    printf("排序所用时间: %lf 秒\n",duration) ;
    File(a,n);
    a[MaxSize]=0;
    system("pause");
    system("cls");

```

break;

case 4:

```

    cout<<"请输入待排序元素个数: \n";
    cin>>n;
    cout<<"请输入"<<n<<"个元素: \n";
    for(int i=0;i<n;i++)
        cin>>a[i];
    start = clock() ;
    QuickSort(a,0,n-1);        //快速排序;
    finish = clock() ;
    duration= (double)(finish - start) / CLOCKS_PER_SEC;
    cout<<"排序后的结果为: \n";
    for(int i=0;i<n;i++)
        cout<<a[i]<<" ";
    cout<<"\n";
    printf("排序所用时间: %lf 秒\n",duration) ;
    File(a,n);
    a[MaxSize]=0;

```

system("pause");

system("cls");

break;

case 5:

```

    cout<<"请输入待排序元素个数: \n";
    cin>>n;
    cout<<"请输入"<<n<<"个元素: \n";
    for(int i=0;i<n;i++)
        cin>>a[i];
    start = clock() ;
    MergeSort(a,0,n-1);        //归并排序;
    finish = clock() ;
    duration= (double)(finish - start) / CLOCKS_PER_SEC;
    cout<<"排序后的结果为: \n";
    for(int i=0;i<n;i++)
        cout<<a[i]<<" ";
    cout<<"\n";
    printf("排序所用时间: %lf 秒\n",duration) ;
    File(a,n);
    a[MaxSize]=0;

```

system("pause");

system("cls");

break;

case 6:

```

cout<<"请输入待排序元素个数: \n";
cin>>n;
cout<<"请输入"<<n<<"个元素: \n";
    for(int i=1;i<=n;i++)
        cin>>a[i];
    start = clock() ;
    HeapSort(a,n);          //堆排序;
    finish = clock() ;
    duration= (double)(finish - start) / CLOCKS_PER_SEC;
    cout<<"排序后的结果为: \n";
    for(int i=1;i<=n;i++)
        cout<<a[i]<<" ";
    cout<<"\n";
    printf("排序所用时间: %lf 秒\n",duration) ;
    FILE * fp;
    fp= fopen("C:\\Users\\HH\\Desktop\\输出结果.txt","w") ;
    for(int i=1;i<=n;i++)
        fprintf(fp,"%-8d ",a[i]) ;
if(fp!=NULL)
fclose(fp) ; //关闭文件
a[MaxSize]=0;
system("pause");
system("cls");
break;

```

case 7:

```

cout<<"请输入待排序元素个数: \n";
cin>>n;
cout<<"请输入"<<n<<"个元素: \n";
    for(int i=0;i<n;i++)
        cin>>a[i];
    start = clock() ;
    radixsort(a,n);      //基数排序;
    finish = clock() ;
    duration= (double)(finish - start) / CLOCKS_PER_SEC;
    cout<<"排序后的结果为: \n";
    for(int i=0;i<n;i++)
        cout<<a[i]<<" ";
    cout<<"\n";
    printf("排序所用时间: %lf 秒\n",duration) ;
    File(a,n);

```

```

        a[MaxSize]=0;
    system("pause");
    system("cls");
    break;

case 8:
    cout<<"请输入待排序元素个数: \n";
    cin>>n;
    srand((unsigned int)time(0));
    for(int i=0;i<n;i++){
        a[i]=rand();
        b[i]=a[i];
        arr[i+1]=a[i];
    }
        cout<<"\n";

    start = clock() ;
        InsertSort(b,n);           //插入排序;
    finish = clock() ;
    t[1] = (double)(finish - start) / CLOCKS_PER_SEC;
    printf("插入排序所用时间: %lf 秒\n",t[1]);
    for(int i=0;i<n;i++)
        b[i]=a[i];
    cout<<"\n";

    start = clock() ;
        BubbleSort(b,n);           //冒泡排序;
    finish = clock() ;
    t[2] = (double)(finish - start) / CLOCKS_PER_SEC;
    printf("冒泡排序所用时间: %lf 秒\n",t[2]);
    for(int i=0;i<n;i++)
        b[i]=a[i];
    cout<<"\n";

    start = clock() ;
        SelectSort(b,n);           //选择排序;
    finish = clock() ;
    t[3] = (double)(finish - start) / CLOCKS_PER_SEC;
    printf("选择排序所用时间: %lf 秒\n",t[3]);
    for(int i=0;i<n;i++)
        b[i]=a[i];
    cout<<"\n";

```

```

start = clock() ;
    QuickSort(b,0,n-1);          //快速排序;
finish = clock() ;
t[4] = (double)(finish - start) / CLOCKS_PER_SEC;
    for(int i=0;i<n;i++)
        b[i]=a[i];
printf("快速排序所用时间: %lf 秒\n",t[4]);
cout<<"\n";

```

```

    start = clock() ;
    MergeSort(b,0,n-1);          //归并排序;
    finish = clock() ;
t[5] = (double)(finish - start) / CLOCKS_PER_SEC;
    for(int i=0;i<n;i++)
        b[i]=a[i];
printf("归并排序所用时间: %lf 秒\n",t[5]);
cout<<"\n";

```

```

start = clock() ;
    HeapSort(arr,n);             //堆排序;
finish = clock() ;
t[6] = (double)(finish - start) / CLOCKS_PER_SEC;
printf("堆排序所用时间: %lf 秒\n",t[6]);
cout<<"\n";

```

```

start = clock() ;
    radixsort(a,n);              //基数排序;
finish = clock() ;
t[7] = (double)(finish - start) / CLOCKS_PER_SEC;
printf("基数排序所用时间: %lf 秒\n",t[7]);
cout<<"\n";

```

```

// 输出排序结果;
/* cout<<"排序的结果为: \n";
    for(int i=0;i<n;i++)
        printf("%d ",a[i]);
    cout<<"\n";*/

```

```

    {int i,k=0,j,k2=0;
    double t1[8];

```

```

double min1=t[1],min2=99.99;
    for( i=1;i<=7;i++){
        t1[i]=t[i];
        if(t[i]<min1){
            min1=t[i];
            k=i;
        }
    }

    t1[k]=999.99;

    for(j=7;j>0;j--){
        if(t1[j]<=min2){
            min2=t1[j];
            k2=j;
        }
    }
}

```

```

FILE * fp2;
fp2= fopen("C:\\Users\\HH\\Desktop\\综合排序输出结果.txt","w");
    fprintf(fp2,"\n");
    fprintf(fp2,"插入排序所用时间: %lf 秒\n",t[1]);
    fprintf(fp2,"冒泡排序所用时间: %lf 秒\n",t[2]);
    fprintf(fp2,"选择排序所用时间: %lf 秒\n",t[3]);
    fprintf(fp2,"快速排序所用时间: %lf 秒\n",t[4]);
    fprintf(fp2,"归并排序所用时间: %lf 秒\n",t[5]);
    fprintf(fp2,"堆    排序所用时间: %lf 秒\n",t[6]);
    fprintf(fp2,"基数排序所用时间: %lf 秒\n",t[7]);
    fprintf(fp2,"\n");
    fprintf(fp2,"最快两种的排序为: \n");
    if(k==1)
        fprintf(fp2,"插入排序: %lf 秒\n",t[1]);
    else if(k==2)
        fprintf(fp2,"冒泡排序: %lf 秒\n",t[2]);
    else if(k==3)
        fprintf(fp2,"选择排序: %lf 秒\n",t[3]);
    else if(k==4)
        fprintf(fp2,"快速排序: %lf 秒\n",t[4]);
    else if(k==5)
        fprintf(fp2,"归并排序: %lf 秒\n",t[5]);
    else if(k==6)
        fprintf(fp2,"堆    排序: %lf 秒\n",t[6]);
    else if(k==7)

```

```

        fprintf(fp2,"基数排序: %lf 秒\n",t[7]);
        fprintf(fp2,"\n");
        if(k2==1)
            fprintf(fp2,"插入排序: %lf 秒\n",t[1]);
        else if(k2==2)
            fprintf(fp2,"冒泡排序: %lf 秒\n",t[2]);
        else if(k2==3)
            fprintf(fp2,"选择排序: %lf 秒\n",t[3]);
        else if(k2==4)
            fprintf(fp2,"快速排序: %lf 秒\n",t[4]);
        else if(k2==5)
            fprintf(fp2,"归并排序: %lf 秒\n",t[5]);
        else if(k2==6)
            fprintf(fp2,"堆    排序: %lf 秒\n",t[6]);
        else if(k2==7)
            fprintf(fp2,"基数排序: %lf 秒\n",t[7]);
        fprintf(fp2,"\n");
        fprintf(fp2,"\n");
        fprintf(fp2,"排序的结果为: \n");
        for(int i=1;i<=n;i++)
            fprintf(fp2,"%-8d ",arr[i]);
        if(fp2!=NULL)
            fclose(fp2); //关闭文件
    }
    system("pause");
    system("cls");
    break;

case 0:exit(0);

default:cout<<"没有你选择的项目请重新选择! \n";
        system("pause");
        system("cls");
    } //switch
} //while;
} //return;

```


计算机学院

数据结构课程设计评分表

专业：计算机科学与技术

学号：203405010114

姓名：徐新宇

题目	排序综合		成绩	良好
评价内容		评价标准	满分	得分
问题分析设计 (20)	数据结构设计	数据结构设计方案合理，数据结构设计正确	10	
	算法设计	设计方案合理，算法设计正确；流程图布局合理，控制流程正确	10	
编码与调试 (20)	编写代码	能够熟练利用编程语言实现相应算法	10	
	调试工具使用	熟练使用调试工具	10	
系统运行及答辩 (30)	工作量与难度	工作量饱满，难度较大	5	
	程序运行	程序运行结果正确，程序具有健壮性，具有优化设计，运行效率高，高质量完成任务书各项功能要求	10	
	程序测试与验证	测试方案合理，设计多组测试用例；对算法性能进行了深入分析	5	
	答辩	语言表达能力强，有独立见解，逻辑思路清晰，回答问题正确	10	
课设报告质量 (30)	报告内容质量	论述简练清晰，有自己独特观点；逻辑结构合理；内容正确，文字通畅；无抄袭	20	
	报告格式规范性	图表完备优美；报告格式规范；程序清单关键代码有注释	5	
	个人总结	文字通畅，内容真实，有真情实感，无抄袭	5	
其它补充说明			累计得分	
指导教师签名			日期	2022.1.2

注：成绩评定采用五级记分制优秀(90~100分)、良好(80~89分)、中等(70~79分)、及格(60~69分)、不及格(60分以下)。