中南大学

数据结构试验报告

题 目 实验七

学生姓名 王云鹏

学 号 8213180228

指导老师 郑 瑾

学 院 计算机学院

专业班级 物联网1802

完成时间 2020.6

指导老师评定 签名

实验七

1. 需求分析

本实验目的是使读者，掌握常用排序方法的基本思想，通过实验加深理解各种排序算法，通过实

验掌握各种排序方法的时间复杂度分析，了解各种排序方法的优缺点及适用范围。

1．排序算法的实现（设计性实验）

问题描述

排序是计算机领域的一项重要技术，是程序设计中的一种重要运算。它的功能是将一个数据元素

的任意序列重新排列成一个按键有序的序列。学习和研究各种排序方法是计算机工作者的一项重要工

作课题。

基本要求

随机产生*n*个整数（依次为*n*赋值100、200、300、1 000和2 000），将其存于数组A[1..n]中。对主

要算法（顺序查找、插入排序、归并排序、堆排序和快速排序）进行实验比较，计算出平均比较次数

*cn*、平均移动次数*mn*及执行时间*tn*。*cn*和*mn*由程序自动计算，*tn*由系统时间计算。

对实验结果数据进行对比分析。

测试数据

由随机数生成器决定。

实现提示

(1) 设计一个驱动程序，其任务是，随机输入一组原始数据*A*[1..*n*]，对于查找还需准备一个键码

值（可以随机产生，也可在*A*[1..*n*]中按索引号挑选若干有代表性的数据）。对每组原始数据，分别调用

查找或排序算法过程。

(2) 为了自动计算*cn*和*mn*需要在每个算法过程中的适当位置插入计数操作。

手工计算每个算法的执行时间*tn*时，为了减少计时误差，删除所插入的计数操作，并按*n*的大小确

定一个*K*，对每个查找或排序算法过程反复调用*K*次，在调用开始前设置一个计时起点*t*0，在*K*次调用

执行后可打印信息。设计时的终点为*t*1，则(*t*1−*t*0)/*K*便是算法的大致执行时间。

选作内容

对不同的输入表长做试验，观察含义相同的变量相对于表长的变化关系，还可以对稳定性做验证。

2．统计成绩（综合性实验）

问题描述

给出*n*个学生的*m*门考试的成绩表，每个学生的信息由学号、姓名及各科成绩组成。对学生的考试

成绩进行有关统计，并打印统计表。

基本要求

(1) 按总数高低次序，打印出名次表，分数相同的为同一名次。

(2) 按名次打印出每个学生的学号、姓名、总分以及各科成绩。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据。

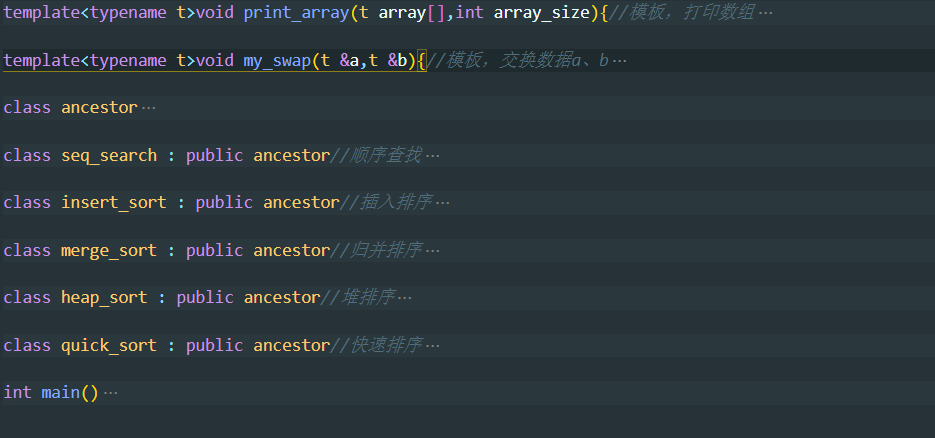
选作内容

对各科成绩设置不同的权值

1. 概要设计

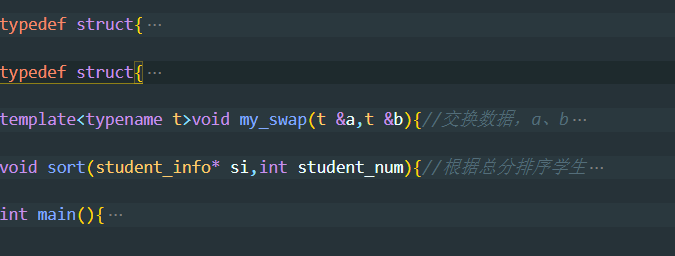
* 设计性实验

函数：



* 综合性实验

函数：



1. 详细设计

* 设计性实验

|  |
| --- |
| *#include* <iostream>  *#include* <fstream>  *#include* <ctime>  using namespace std;  template<typename t>void print\_array(t array[],int array\_size){*//模板，打印数组*  *for*(int i=0;i<array\_size;i++){          cout<<array[i]<<" ";  *if*((i+1)%50==0)cout<<endl;      }      cout<<endl;  }  template<typename t>void my\_swap(t &a,t &b){*//模板，交换数据a、b*      t temp=a;  a=b;      b=temp;  }  class ancestor  {  public:      long cn = 0;*//平均比较次数*      long mn = 0;*//平均移动次数*      long tn = 0;*//平均执行时间,单位：CPU时钟数*      int array\_size;      int \*array;      ancestor(int n)*//初始化array,存放待用数字序列*      {          array\_size = n;  *// array=new int[array\_size];*          array = (int \*)malloc(array\_size \* sizeof(int));*//位array分配空间*  *// int x[array\_size];*  *// array=x;*          srand((unsigned)time(NULL));*//随机数种子*  *for* (int i = 0; i < array\_size; i++)          {              array[i] = rand() % array\_size;*//用随机数序列初始化array*          }      }  };  class seq\_search : public ancestor*//顺序查找*  {      public:      seq\_search(int n,int dest):ancestor(n)      {          clock\_t start\_time,end\_time;          start\_time=clock();*//计时程序*          int flag=0;*//是否查找到的标志位*  *for*(int i=0;i<n;i++){*//遍历查找*              cn++;  *if*(array[i]==dest){*//找到*                  flag=1;  *// cout<<dest<<"位于序列的第"<<i+1<<"位"<<endl;*              }          }          end\_time=clock();*//停止计时*  *// tn=(double)(end\_time-start\_time)/CLOCKS\_PER\_SEC;*          tn=end\_time-start\_time;  *if*(flag==0){*//没找到*  *// cout<<"数字序列中没有"<<dest<<endl;*          }      }  };  class insert\_sort : public ancestor*//插入排序*  {      public:      insert\_sort(int n):ancestor(n)      {          clock\_t start\_time,end\_time;          start\_time=clock();*//计时*  *for*(int i=1;i<n;i++){*//遍历所有数据，看能否往前插入*              cn++;  *if*(array[i]<array[i-1]){*//若前一个比当前的大，就接着往前寻找，否则说明当前数据有序，不需要移动*                  int temp=array[i];                  int j;  *for*(j=i-1;temp<array[j] && j>=0;j--){*//移动比当前所有数据大的数，向后移动一位，覆盖*                      array[j+1]=array[j];                      cn++;*//计数*                      mn++;  *// cout<<j<<endl;*                  }                  array[j+1]=temp;*//最后，插入当前数据到比他略小的数据之后*              }          }          end\_time=clock();          tn=end\_time-start\_time;      }  };  class merge\_sort : public ancestor*//归并排序*  {      public:      merge\_sort(int n):ancestor(n)      {          clock\_t start\_time,end\_time;          start\_time=clock();*//计时*          my\_sort(array,array\_size,0,array\_size-1);*//排序*          end\_time=clock();          tn=end\_time-start\_time;      }      void my\_sort(int array[],int array\_size,int left,int right){*//array为待排序数组，left为拆开的下界，right为上界*  *if*(left<right){              int mid=(int)((left+right)/2);              my\_sort(array,array\_size,left,mid);*//拆开上半部分*              my\_sort(array,array\_size,mid+1,right);*//拆开下半部分*              merge(array,array\_size,left,mid,right);*//归并拆开的部分*          }      }      void merge(int array[],int array\_size,int left,int mid,int right){*//归并*          int i,j,k;          int temp[right-left+1];*//存放需要参加比较的数据*  *for*(int i=left;i<=right;i++){*//从array赋值，初始化temp*              temp[i-left]=array[i];          }  *for*(i=left,j=mid+1,k=left;i<=mid && j<=right;k++){*//两条数据合并，挨个比较并在一起*  *if*(temp[i-left]<=temp[j-left]){*//i小，将i所在数据存入array*                  array[k]=temp[i-left];                  i++;              }*else*{*//j小，将j所在数据存入array*                  array[k]=temp[j-left];                  j++;              }              cn++;              mn++;          }  *while*(i<=mid){*//某条数据放完，将剩下的数据直接放入array*              array[k++]=temp[i-left];              i++;              mn++;          }  *while*(j<=right){              array[k++]=temp[j-left];              j++;              mn++;          }      }  };  class heap\_sort : public ancestor*//堆排序*  {      public:      heap\_sort(int n):ancestor(n)      {          clock\_t start\_time,end\_time;          start\_time=clock();*//计时*  *for*(int i=array\_size/2;i>=0;i--){              heap\_adjust(i,array\_size-1);*//建立一个大顶堆*          }          swap(array[0],array[array\_size-1]);*//将堆顶与堆底互换*          mn=mn+3;  *for*(int i=array\_size-2;i>0;i--){*//不断将堆顶与堆底互换，画完调整堆，使之保持大顶堆。而有序的数据都挨个沉到堆底，最终从小到大排序*              heap\_adjust(0,i);              swap(array[0],array[i]);              mn=mn+3;          }          end\_time=clock();          tn=end\_time-start\_time;      }      void heap\_adjust(int s,int m){*//大顶堆，调整堆，从array[s...m]使保持大顶堆（假设：除了以s的位置不对，其他都是对的）*          int temp=array[s];          mn++;  *for*(int j=2\*s+1;j<=m;j=j\*2+1){*//调整以s为顶的堆*  *if*(j<m && array[j]<array[j+1]){*//找到更大的儿子，j为儿子的下标*                  j++;              }  *if*(temp>=array[j]){*//若s所在已经是最大的位置，则是大顶堆，推出循环*  *break*;              }              array[s]=array[j];*//更大的向上浮动*              s=j;*//使s记录已经向上浮动数据的位置*              mn++;          }          array[s]=temp;*//s所在数据放回堆中*          mn++;      }  };  class quick\_sort : public ancestor*//快速排序*  {      public:      quick\_sort(int n):ancestor(n)      {          clock\_t start\_time,end\_time;          start\_time=clock();*//计时*          q\_sort(0,array\_size-1);*//排序*          end\_time=clock();          tn=end\_time-start\_time;      }      int partition(int low,int high){*//分段，并且使一次排序*          int temp=array[low];*//分界数，根据分界数将待排序列分成大小的两段*          mn++;  *while*(low<high){*//遍历，分别从左右向中间靠近，左右互换，完成一次排序*  *while*(low<high && array[high]>=temp){*//从右边，挨个寻找比分界数小的数*                  high--;                  cn++;              }  *if*(low<high){*//判断是处理特殊情况*                  array[low++]=array[high];*//找到比分界数小的数，放大左边去*                  mn++;              }    *while*(low<high && array[low]<=temp){*//从左边，挨个寻找比分界数大的数*                  low++;                  cn++;              }  *if*(low<high){                  array[high--]=array[low];*//找到比分界数大的数，放到右边去（此时，右边该位置的数，已经放到左边）*                  mn++;              }            }          array[low]=temp;*//将分界数放回*          mn++;  *return* low;      }      void q\_sort(int low,int high){*//快排*  *if*(low<high){              int mid=partition(low,high);*//分段，排一次，此时，左边都是比分界数小的数，右边都是比分界数大的数*              q\_sort(low,mid-1);*//排列左半边的数字序列*              q\_sort(mid+1,high);*//排列右半边的数字序列*          }      }  };  int main()  {      int times\_test[]={100,200,300,1000,2000,5000,10000,50000};*//用来排序的数字个数，组成的集合*      int times\_test\_size=sizeof(times\_test)/sizeof(int);  *for*(int i=0;i<times\_test\_size;i++){*//测试数据*          seq\_search ss=seq\_search(times\_test[i],times\_test[i]/2);          insert\_sort is=insert\_sort(times\_test[i]);          merge\_sort ms=merge\_sort(times\_test[i]);          heap\_sort hs=heap\_sort(times\_test[i]);          quick\_sort qs=quick\_sort(times\_test[i]);          cout<<"-----------"<<times\_test[i]<<"个数"<<"------------"<<endl;          cout<<"方法"<<"\t\t"<<"cn"<<"\t"<<"mn"<<"\t"<<"tn"<<"\t"<<endl;          cout<<"顺序查找"<<"\t"<<ss.cn<<"\t"<<ss.mn<<"\t"<<ss.tn<<"\t"<<endl;          cout<<"插入排序"<<"\t"<<is.cn<<"\t"<<is.mn<<"\t"<<is.tn<<"\t"<<endl;          cout<<"归并排序"<<"\t"<<ms.cn<<"\t"<<ms.mn<<"\t"<<ms.tn<<"\t"<<endl;          cout<<"堆排序"<<"\t\t"<<hs.cn<<"\t"<<hs.mn<<"\t"<<hs.tn<<"\t"<<endl;          cout<<"快速排序"<<"\t"<<qs.cn<<"\t"<<qs.mn<<"\t"<<qs.tn<<"\t"<<endl;      }  *// ancestor a=ancestor(100);*  *// print\_array(a.array,a.array\_size);*  *// seq\_search ss=seq\_search(100,50);*  *// print\_array(ss.array,ss.array\_size);*  *// cout<<"执行时间(单位：CPU时钟数)"<<ss.tn<<endl;*  *// insert\_sort is=insert\_sort(100);*  *// print\_array(is.array,is.array\_size);*  *// cout<<"执行时间(单位：CPU时钟数)"<<is.tn<<endl;*  *// merge\_sort ms=merge\_sort(100);*  *// print\_array(ms.array,ms.array\_size);*  *// cout<<"执行时间(单位：CPU时钟数)"<<ms.tn<<endl;*  *// heap\_sort hs=heap\_sort(100);*  *// print\_array(hs.array,hs.array\_size);*  *// cout<<"执行时间(单位：CPU时钟数)"<<hs.tn<<endl;*  *// quick\_sort qs=quick\_sort(100);*  *// print\_array(qs.array,qs.array\_size);*  *// cout<<"执行时间(单位：CPU时钟数)"<<qs.tn<<endl;*  *return* 0;  } |

|  |
| --- |
| *#include*<iostream>  *#include*<fstream>  *#include*<string>  *#include*<io.h>  using namespace std;  typedef struct{      int chinese;      int math;      int english;  }score;*//存放分数信息*  typedef struct{      string ID;      string name;      score ss;  }student\_info;*//存放学生信息*  template<typename t>void my\_swap(t &a,t &b){*//交换数据，a、b*      t temp=a;      a=b;      b=temp;  }  void sort(student\_info\* si,int student\_num){*//根据总分排序学生*      int sum[student\_num];  *for*(int i=0;i<student\_num;i++){*//求出总分*          sum[i]=si[i].ss.chinese+si[i].ss.math+si[i].ss.english;      }  *for*(int i=0;i<student\_num;i++){*//选择排序*  *for*(int j=i;j<student\_num;j++){  *if*(sum[i]<sum[j]){                  my\_swap(si[i],si[j]);                  my\_swap(sum[i],sum[j]);              }          }      }      int temp\_score=sum[0];      int rank=1;      cout<<"total\t"<<"rank\t"<<"ID\t"<<"name\t"<<"chinese\t"<<"math\t"<<"english\t"<<endl;*//输出排好的学生数组*  *for*(int i=0;i<student\_num;i++){  *if*(sum[i]!=temp\_score){              temp\_score=sum[i];              rank++;          }          cout<<sum[i]<<"\t"<<rank<<"\t"<<si[i].ID<<"\t"<<si[i].name<<"\t"<<si[i].ss.chinese<<"\t"<<si[i].ss.math<<"\t"<<si[i].ss.english<<"\t"<<endl;      }  }  int main(){      ifstream infile;*//打开文件*      infile.open("./student\_info.txt",ios::in);      student\_info si[10];*//假定不超过10个学生*      int index=0;  *while*(!infile.eof()){*//从文件读入学生信息*          infile>>si[index].ID;          infile>>si[index].name;          infile>>si[index].ss.chinese;          infile>>si[index].ss.math;          infile>>si[index].ss.english;          index++;      }      sort(si,index);*//排序展示*  *return* 0;  } |

* 综合性实验

4.调试分析

1. 采用IDE中自带的调试功能进行调试，手动添加断点和查看程序。
2. 对设计和编码的讨论和分析。该程序实现了顺序栈的操作。分析程序代码的质量，主要从以下几个方面考虑。

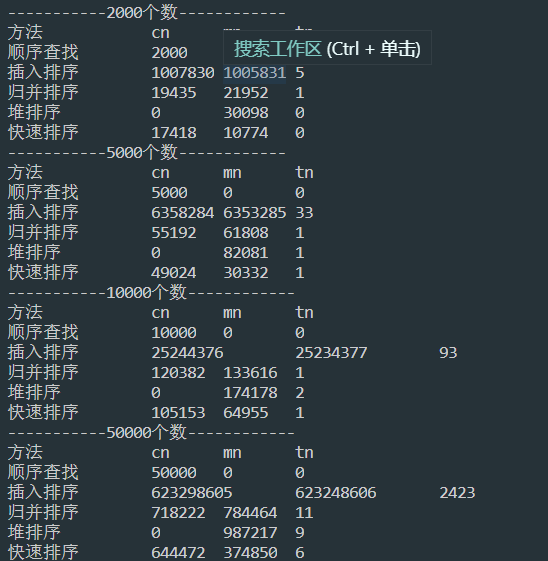
* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

5.使用说明

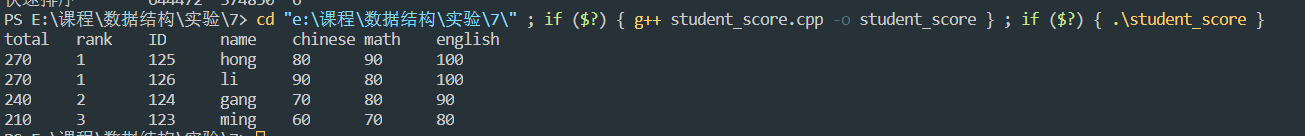
按照屏幕提示，选择想要的功能并输入对应数字，按下ENTER键后，根据屏幕提示进行输入，即可得到想要的结果。

6.测试程序运行结果

* 设计性实验



* 综合性实验



1. **心得体会**

通过本次实验，使我对数据结构有了更深的理解，对指针的运用更加熟练，熟悉了对函数的定义和操作。