

**数据结构与算法实验  
实验报告**

课程名称： 数据结构与算法

学 院： 计算机与信息安全学院

专 业： 智能科学与技术

学 号： 2200340126

姓 名： 江佳俊

指导教师： 王诗文

报告日期： 2023年 11 月15 日

目录

[一、实验目的 3](#_Toc27329)

[二、实验内容和要求 3](#_Toc30926)

[1. 直接插入排序 3](#_Toc14009)

[2. 二分插入排序 3](#_Toc20005)

[3. shell排序 3](#_Toc25544)

[4. 直接选择排序 3](#_Toc24790)

[5. 堆排序 3](#_Toc31418)

[6. 冒泡排序 3](#_Toc22804)

[7. 快速排序 4](#_Toc23654)

[三、实验背景 4](#_Toc20483)

[四、实验思路与方案 4](#_Toc12894)

[4.1.1. 算法介绍（直接插入排序） 4](#_Toc29161)

[4.1.2 函数设计 4](#_Toc25830)

[4.1.3. 复杂性分析 5](#_Toc16823)

[4.1.4. 测试检验 6](#_Toc15480)

[4.2.1. 算法介绍（二分插入排序） 6](#_Toc24996)

[4.2.2 函数设计 6](#_Toc10375)

[4.2.3. 复杂性分析 7](#_Toc25178)

[4.2.4. 测试检验 7](#_Toc5792)

[4.3.1. 算法介绍（Shell排序） 8](#_Toc26343)

[4.3.2 函数设计 8](#_Toc17848)

[4.3.3. 复杂性分析 9](#_Toc13702)

[4.3.4. 测试检验 9](#_Toc4042)

[4.4.1. 算法介绍（直接选择排序） 10](#_Toc14603)

[4.4.2 函数设计 10](#_Toc24865)

[4.4.3. 复杂性分析 11](#_Toc24189)

[4.4.4. 测试检验 11](#_Toc20798)

[4.5.1. 算法介绍（堆排序） 12](#_Toc5492)

[4.5.2 函数设计 12](#_Toc7320)

[4.5.3. 复杂性分析 13](#_Toc20553)

[4.5.4. 测试检验 13](#_Toc19598)

[4.6.1. 算法介绍（冒泡排序） 14](#_Toc31138)

[4.6.2 函数设计 14](#_Toc5355)

[4.6.3. 复杂性分析 15](#_Toc757)

[4.6.4. 测试检验 15](#_Toc17107)

[4.7.1. 算法介绍（快速排序） 15](#_Toc18485)

[4.7.2 函数设计 15](#_Toc76)

[4.7.3. 复杂性分析 16](#_Toc31172)

[4.7.4. 测试检验 16](#_Toc30933)

[五、实验总结 17](#_Toc1827)

[六、附录 18](#_Toc4072)

[6.1.结构体定义 18](#_Toc6136)

[6.2.打印函数代码 18](#_Toc8286)

[6.3.直接插入排序代码 18](#_Toc20059)

[6.4.二分插入排序代码 19](#_Toc13317)

[6.5.Shell排序代码 20](#_Toc9365)

[6.6.直接选择排序代码 20](#_Toc23792)

[6.7.堆排序代码 21](#_Toc2473)

[6.5.冒泡排序代码 21](#_Toc26189)

[6.5.快速排序代码 22](#_Toc15207)

**一、实验目的**

1.能实现经典的排序方法，并针对不同规模的数据量条件下进行算法性能测试；

2.能根据给定问题进行排序实验方案设计与分析。

**二、实验内容和要求**

1. **直接插入排序**

要求：以线性表L中的课程名称为关键字，完成直接插入排序算法； 输出每趟排序后的课程名称。最后输出排序完成后的线性表内容。

1. **二分插入排序**

要求：以线性表L中的课程名称为关键字，完成二分插入排序算法； 输出每趟排序后的课程名称。

1. **shell排序**

要求：以线性表L中的课程名称为关键字，完成shell插入排序算法； 输出每趟排序后的课程名称。

1. **直接选择排序**

要求：以线性表L中的课程名称为关键字，完成直接选择排序算法； 输出每趟排序后的课程名称。

1. **堆排序**

要求：以线性表L中的课程名称为关键字，完成堆插入排序算法； 输出每趟排序后的课程名称。

1. **冒泡排序**

要求：以线性表L中的课程名称为关键字，完成冒泡排序算法； 输出每趟排序后的课程名称。

1. **快速排序**

要求：以线性表L中的课程名称为关键字，完成快速排序算法； 输出每趟排序后的课程名称。

**三、实验背景**

本次实验基于前面的实验而来，在前面的实验中，我们定义了线性表L用于存储课程信息，其中信息具体包括课程名称、学分、讲授学时、实践学时、课程类别、开课学期。（结构体具体定义见附录）

这次实验需要我们设计排序算法来对已经存有课程信息的线性表中的课程进行排序，并且输出每次排序后的课程名称。

除此之外，该实验作为课程管理系统的一部分，能让我们理解项目的构成，不同排序算法的实现也让我们对其进行比较并思考，体会不同排序算法的效率。

**四、实验思路与方案**

**实验思路：**

本次实验可以分为七个排序算法的实现，在该部分，我将针对不同排序算法给出相应的实现方案。另外，在完成这七个排序算法后，为了增加我对算法的印象，我将对比其时间复杂度，加深对排序算法的理解。

**实验方案：**

**实验中首先定义了一个print函数用于输出当前线性表的课程名称，这样在执行一轮排序后只需要调用print函数即可输出每趟排序后的课程名称。于是，我们将重心放在排序算法的实现上。**

**4.1.1. 算法介绍（直接插入排序）**

插入排序(InsertionSort)，一般也被称为直接插入排序。对于少量元素的排序，它是一个有效的算法。插入排序是一种最简单的排序方法，它的基本思想是将一个记录插入到已经排好序的有序表中，从而一个新的、记录数增 1 的有序表。在其实现过程使用双层循环，外层循环对除了第一个元素之外的所有元素，内层循环对当前元素前面有序表进行待插入位置查找，并进行移动。

**4.1.2 函数设计**

**void insert\_sort(PseqList &L)**

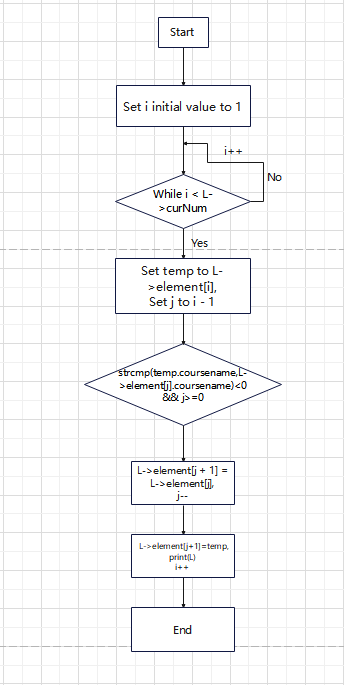
函数功能：以课程名称为关键字，利用直接插入排序对线性表L中课程进行排序，并输出每趟排序后的课程名称。

函数参数：线性表L

返回值：空

**（详细代码见附录）**

函数流程图如下图所示：



**直接插入排序算法流程图**

**4.1.3. 复杂性分析**

插入排序的平均时间复杂度也是 O(n^2)，空间复杂度为常数阶 O(1)，具体时间复杂度和数组的有序性也是有关联的。

插入排序中，当待排序数组是有序时，是最优的情况，只需当前数跟前一个数比较一下就可以了，这时一共需要比较 N-1 次，时间复杂度为 O(N)。最坏的情况是待排序数组是逆序的，此时需要比较次数最多，最坏的情况是 O(n^2)。

**4.1.4. 测试检验**

**直接插入排序检验图**

根据头歌平台评测结果可知，直接插入排序算法运行正常，在正确排序的同时，输出每一趟排序后的课程名称。

**4.2.1. 算法介绍（二分插入排序）**

二分插入排序是插入排序的一种优化算法，尤其适用于小到中等规模的数据集。其基本思想是通过二分查找快速找到当前元素在已排序部分的正确插入位置。通常采用迭代二分查找进行实现，但也可以根据编程上下文和需求选择使用递归二分查找。

二分插入排序结合了插入排序的简单性和二分查找的效率，使其成为某些场景下的实用选择。

**4.2.2 函数设计**

**void biinsert\_sort(PseqList &L)**

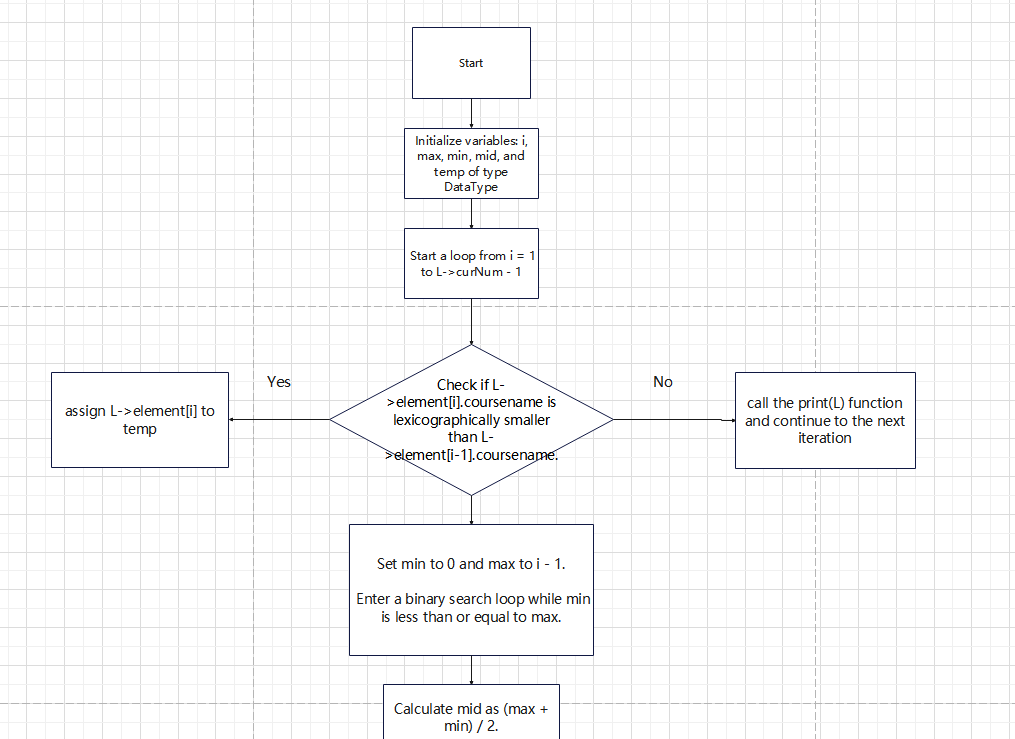
函数功能：以课程名称为关键字，利用二分插入排序对线性表L中课程进行排序，并输出每趟排序后的课程名称。

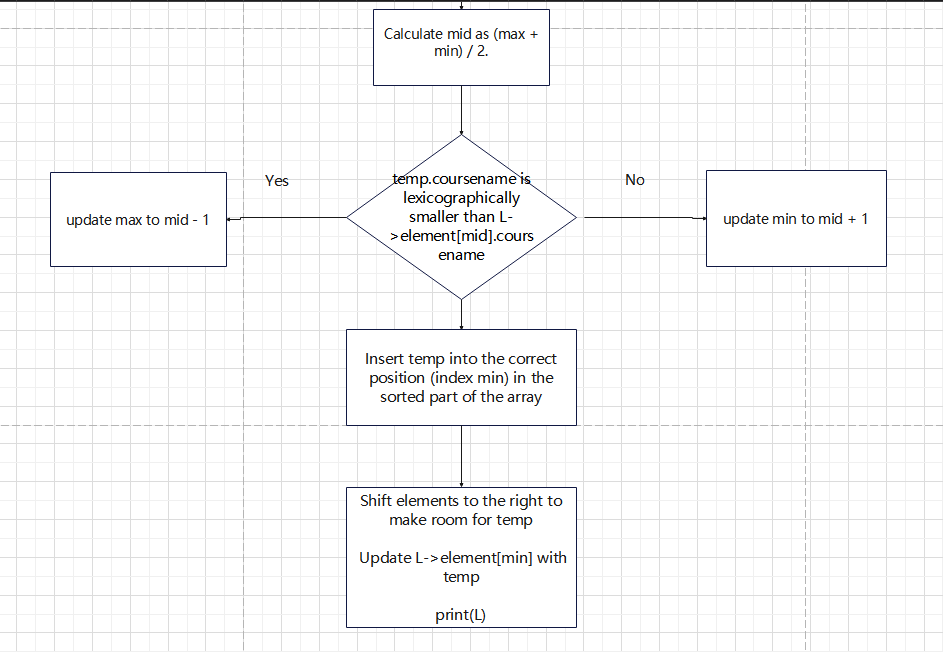
函数参数：线性表L

返回值：空

**（详细代码见附录）**

函数流程图如下图所示：





**二分插入排序算法流程图**

**4.2.3. 复杂性分析**

最坏情况：每次都在有序序列的起始位置插入，则整个有序序列的元素需要后移，时间复杂度为O(n^2)

最好情况：待排序数组本身就是正序的，每个元素所在位置即为它的插入位置，此时时间复杂度仅为比较时的时间复杂度，为O(log2n)

平均情况：O(n^2)

**4.2.4. 测试检验**



**二分插入排序检验图**

根据头歌平台评测结果可知，二分插入排序算法运行正常，在正确排序的同时，输出每一趟排序后的课程名称。

**4.3.1. 算法介绍（Shell排序）**

希尔排序(Shell Sort)是插入排序的一种，它是针对直接插入排序算法的改进。希尔排序又称缩小增量排序，因 DL.Shell 于 1959 年提出而得名。它通过比较相距一定间隔的元素来进行，各趟比较所用的距离随着算法的进行而减小，直到只比较相邻元素的最后一趟排序为止。

**4.3.2 函数设计**

**void shell\_sort(PseqList &L)**

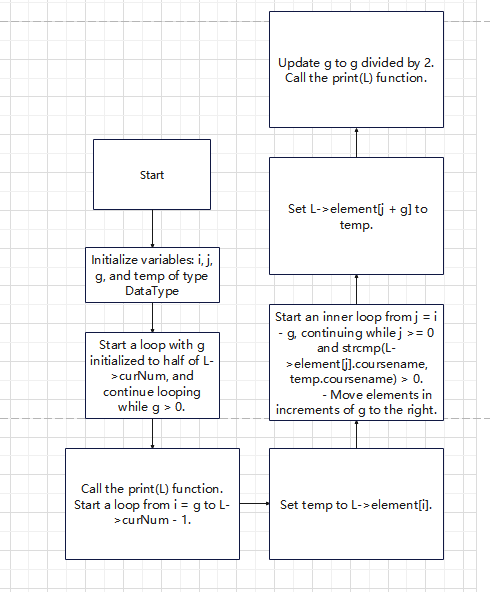
函数功能：以课程名称为关键字，利用Shell排序对线性表L中课程进行排序，并输出每趟排序后的课程名称。

函数参数：线性表L

返回值：空

**（详细代码见附录）**

函数流程图如下图所示：



**Shell排序算法流程图**

**4.3.3. 复杂性分析**

希尔排序时间复杂度是 O(n^(1.3-2))，空间复杂度为常数阶 O(1)。希尔排序没有时间复杂度为 O(n(logn)) 的快速排序算法快 ，因此对中等大小规模表现良好，但对规模非常大的数据排序不是最优选择，总之比一般 O(n^2 ) 复杂度的算法快得多。

**4.3.4. 测试检验**



**Shell排序检验图**

根据头歌平台评测结果可知，Shell排序算法运行正常，在正确排序的同时，输出每一趟排序后的课程名称。

**4.4.1. 算法介绍（直接选择排序）**

选择排序是一种简单直观的排序算法，无论什么数据进去都是 O(n²) 的时间复杂度。所以用到它的时候，数据规模越小越好。一般来说，直接选择排序在实际项目中用的较少，因为在处理数据规模较大的事务时，开销较大，但其胜在简单实现，占用空间较少。

**4.4.2 函数设计**

**void select\_sort(PseqList &L)**

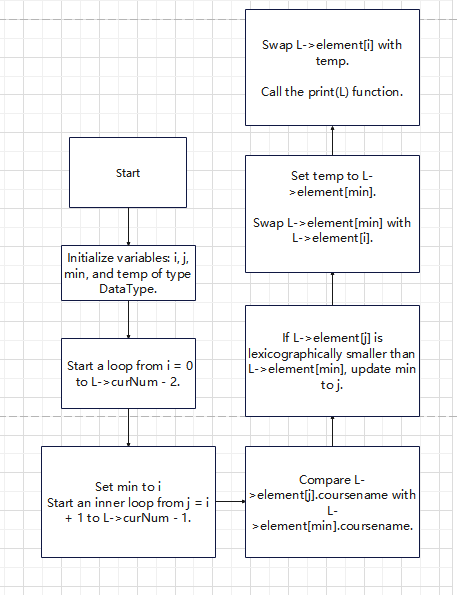
函数功能：以课程名称为关键字，利用直接选择排序对线性表L中课程进行排序，并输出每趟排序后的课程名称。

函数参数：线性表L

返回值：空

**（详细代码见附录）**

函数流程图如下图所示：

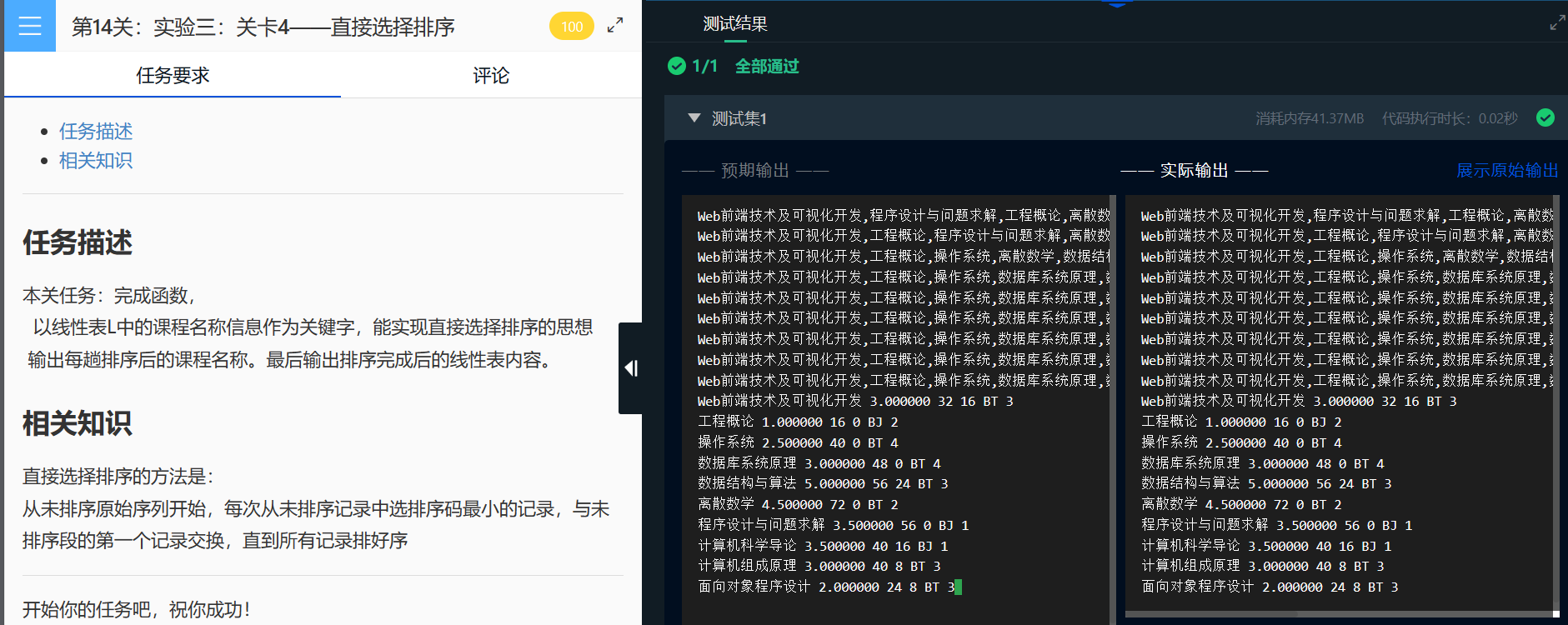


**直接选择排序算法流程图**

**4.4.3. 复杂性分析**

直接选择排序算法时间复杂度恒为O(n^2 )。

**4.4.4. 测试检验**



**直接选择排序检验图**

根据头歌平台评测结果可知，直接选择排序算法运行正常，在正确排序的同时，输出每一趟排序后的课程名称。

**4.5.1. 算法介绍（堆排序）**

堆排序（Heapsort）是指利用堆这种数据结构所设计的一种排序算法。堆积是一个近似完全二叉树的结构，并同时满足堆积的性质：即子结点的键值或索引总是小于（或者大于）它的父节点。堆排序可以说是一种利用堆的概念来排序的选择排序。分为两种方法：

1. 大顶堆：每个节点的值都大于或等于其子节点的值，在堆排序算法中用于升序排列；
2. 小顶堆：每个节点的值都小于或等于其子节点的值，在堆排序算法中用于降序排列；

**4.5.2 函数设计**

**void heap\_sort(PseqList &L)**

函数功能：以课程名称为关键字，利用堆排序对线性表L中课程进行排序，并输出每趟排序后的课程名称。

函数参数：线性表L

返回值：空

**void swap(DataType\*a,DataType\*b)**

函数功能：交换输入的两个值

函数参数：课程A和课程BN

返回值：空

**void heapify(DataType arr[],int n,int i)**

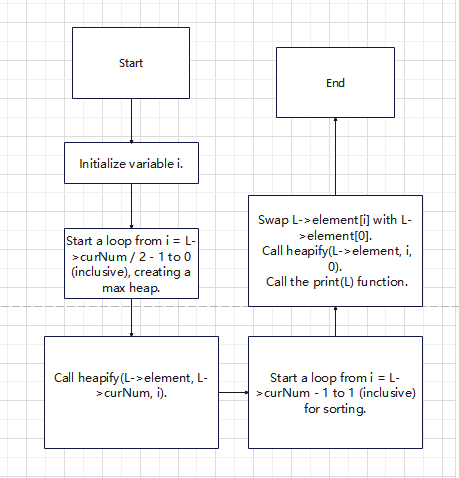
函数功能：将数据堆化处理

函数参数：堆，堆的大小，正在应用堆化的当前节点的索引

返回值：空

**（详细代码见附录）**

函数流程图如下图所示：



**堆排序算法流程图**

**4.5.3. 复杂性分析**

堆排序的平均时间复杂度为 Ο(nlogn)。

**4.5.4. 测试检验**



**堆排序检验图**

根据头歌平台评测结果可知，堆排序算法运行正常，在正确排序的同时，输出每一趟排序后的课程名称。

**4.6.1. 算法介绍（冒泡排序）**

冒泡排序（Bubble Sort）也是一种简单直观的排序算法。它重复地走访过要排序的数列，一次比较两个元素，如果他们的顺序错误就把他们交换过来。走访数列的工作是重复地进行直到没有再需要交换，也就是说该数列已经排序完成。这个算法的名字由来是因为越小的元素会经由交换慢慢"浮"到数列的顶端。作为最简单的排序算法之一，冒泡排序还有一种优化算法，就是立一个 flag，当在一趟序列遍历中元素没有发生交换，则证明该序列已经有序。

**4.6.2 函数设计**

**void bubble\_sort(PseqList &L)**

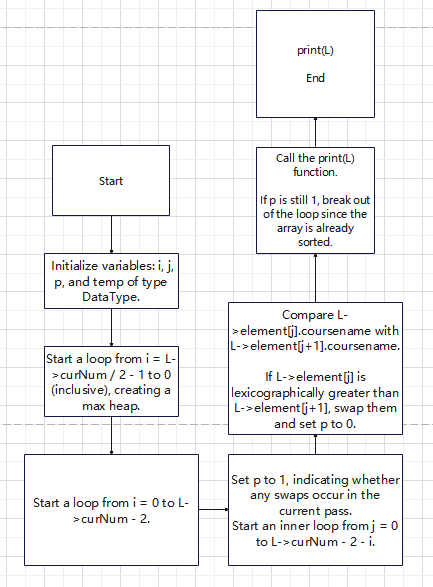
函数功能：以课程名称为关键字，利用冒泡排序对线性表L中课程进行排序，并输出每趟排序后的课程名称。

函数参数：线性表L

返回值：空

**（详细代码见附录）**

函数流程图如下图所示：



**冒泡排序算法流程图**

**4.6.3. 复杂性分析**

最优的情况也就是开始就已经排序好序了，那么就可以不用交换元素了，则时间花销为：[ n(n-1) ] / 2；所以最优的情况时间复杂度为：O( n^2 )；

最差的情况也就是开始的时候元素是逆序的，那么每一次排序都要交换两个元素，则时间花销为：[ 3n(n-1) ] / 2；（其中比上面最优的情况所花的时间就是在于交换元素的三个步骤）；所以最差的情况下时间复杂度为：O( n^2 )；

最优的时间复杂度为：O( n^2 ) ；

最差的时间复杂度为：O( n^2 )；

平均的时间复杂度为：O( n^2 )；

**4.6.4. 测试检验**

**冒泡排序检验图**

根据头歌平台评测结果可知，冒泡排序算法运行正常，在正确排序的同时，输出每一趟排序后的课程名称。

**4.7.1. 算法介绍（快速排序）**

快速排序是由东尼·霍尔所发展的一种排序算法。在平均状况下，排序 n 个项目要 Ο(nlogn) 次比较。在最坏状况下则需要 Ο(n2) 次比较，但这种状况并不常见。事实上，快速排序通常明显比其他 Ο(nlogn) 算法更快，因为它的内部循环（inner loop）可以在大部分的架构上很有效率地被实现出来。

快速排序使用分治法（Divide and conquer）策略来把一个串行（list）分为两个子串行（sub-lists）。

快速排序又是一种分而治之思想在排序算法上的典型应用。本质上来看，快速排序应该算是在冒泡排序基础上的递归分治法。

**4.7.2 函数设计**

**void quick\_sort(PseqList &L,int left,int right)**

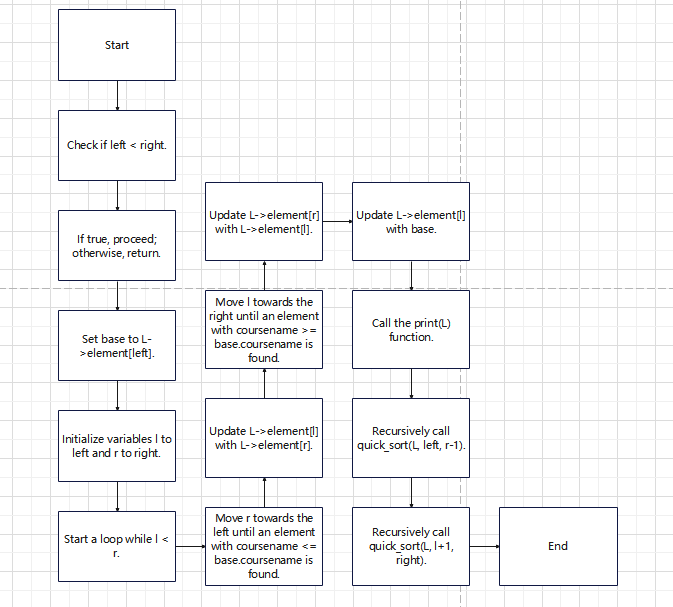
函数功能：以课程名称为关键字，利用快速排序对线性表L中课程进行排序，并输出每趟排序后的课程名称。

函数参数：线性表L，左指针，右指针

返回值：空

**（详细代码见附录）**

函数流程图如下图所示：



**快速排序算法流程图**

**4.7.3. 复杂性分析**

快速排序是在冒泡排序的基础上改进而来的，冒泡排序每次只能交换相邻的两个元素，而快速排序是跳跃式的交换，交换的距离很大，因此总的比较和交换次数少了很多，速度也快了不少。  
 快速排序只是使用数组原本的空间进行排序，所以所占用的空间应该是常量级的，但是由于每次划分之后是递归调用，所以递归调用在运行的过程中会消耗一定的空间，在一般情况下的[空间复杂度](http://data.biancheng.net/view/294.html" \t "http://data.biancheng.net/view/_blank)为 O(logn)，在最差的情况下，若每次只完成了一个元素，那么空间复杂度为 O(n)。所以我们一般认为快速排序的空间复杂度为 O(logn)。

**4.7.4. 测试检验**

**快速排序检验图**

根据头歌平台评测结果可知，快速排序算法运行正常，在正确排序的同时，输出每一趟排序后的课程名称。

**复杂度比较：**

1. 冒泡排序（Bubble Sort）: 时间复杂度为 O(n^2)，空间复杂度为 O(1)。
2. 选择排序（Selection Sort）: 时间复杂度为 O(n^2)，空间复杂度为 O(1)。
3. 插入排序（Insertion Sort）: 时间复杂度为 O(n^2)，空间复杂度为 O(1)。
4. 希尔排序（Shell Sort）: 时间复杂度取决于步长序列的选择，最好情况下可以达到 O(nlogn)，空间复杂度为 O(1)。
5. 快速排序（Quick Sort）: 平均时间复杂度为 O(nlogn)，最坏情况下为 O(n^2)，空间复杂度为 O(logn)。
6. 堆排序（Heap Sort）: 时间复杂度为 O(nlogn)，空间复杂度为 O(1)。
7. 二分插入排序：平均时间复杂度为 O(n^2)，空间复杂度为 O(1)。

**五、实验总结**

通过本次实验，我了解了排序算法的基本原理与实现方法，并且学会了分析排序算法的时间复杂度。除此之外，能够将排序算法利用在实际项目中，对自身有重要的实际意义。更加重要的是，我从排序算法中学到了一些思路，例如分治，二分，这些对编程有着重要的意义。

**六、附录**

**6.1.结构体定义**

typedef struct course

{

    char coursename[50];

    float credit;

    int Tperiod;

    int Operiod;

    char classes[10];

    int term;

} DataType;

struct seqList //顺序线性表结构定义

{

  int MAXNUM;*//用于记录顺序线性表中能存放的最大元素个数的 整型 MAXNUM*

  int curNum;*//用于存放顺序线性表中数据元素的个数  整型  curNum*

  DataType \*element;*//用于存放顺序线性表数据元素的连续空间的起始地址*

};

typedef struct seqList \*PseqList;

typedef struct seqList User;

**6.2.打印函数代码**

void print(PseqList &L)

{

    for(int k=0; k<L->curNum; k++)

    {

        printf("%s,",L->element[k].coursename);

    }

    printf("\n");

}

**6.3.直接插入排序代码**

void insert\_sort(PseqList &L)

{

    int i,j;

    DataType temp;

    for(i=1; i<L->curNum; i++)

    {

        temp=L->element[i];

        j=i-1;

        while(strcmp(temp.coursename,L->element[j].coursename)<0&&j>=0)

        {

            L->element[j+1]=L->element[j];

            j--;

        }

        L->element[j+1]=temp;

  print(L);

    }

}

**6.4.二分插入排序代码**

void biinsert\_sort(PseqList &L)

{

    int i,max,min,mid;

    DataType temp;

    for(i=1; i<L->curNum; i++)

    {

        if(strcmp(L->element[i].coursename,L->element[i-1].coursename)<0)

        {

            temp=L->element[i];

        }

        else

        {

            print(L);

            continue;

        }

        min=0;

        max=i-1;

        while(min<=max)

        {

            mid=(max+min)/2;

            if(strcmp(temp.coursename,L->element[mid].coursename)<0)

                max=mid-1;

            else

                min=mid+1;

        }

        //此时在下标min的元素>=temp的元素

        for(int j=i; j>min; j*--)*

        {

            L->element[j]=L->element[j-1];

        }

        L->element[min]=temp;

        print(L);

    }

}

**6.5.Shell排序代码**

void shell\_sort(PseqList &L)

{

    int i,j,g;

    DataType temp;

    for(g=L->curNum>>1; g>0; g=g>>1)

    {

        print(L);

        for(i=g; i<L->curNum; i++)

        {

            temp=L->element[i];

            for(j=i-g; j>=0&&strcmp(L->element[j].coursename,temp.coursename)>0; j=j-g)

                L->element[j+g]=L->element[j];*//交换*

            L->element[j+g]=temp;*//交换*

        }

    }

    print(L);

}

**6.6.直接选择排序代码**

void select\_sort(PseqList &L)

{

    int i,j,min;

    DataType temp;

    for(i=0; i<L->curNum-1; i++)

    {

        min=i;

        for(j=i+1; j<L->curNum; j++)

            if(strcmp(L->element[j].coursename,L->element[min].coursename)<0)

             min=j;

        temp=L->element[min];

  L->element[min]=L->element[i];

  L->element[i]=temp;

  print(L);

    }

}

void swap(DataType\*a,DataType\*b)

{

 DataType temp=\*a;

 \*a=\*b;

 \*b=temp;

}

void heapify(DataType arr[],int n,int i)*//从i开始维护*

{

    int largest=i;

    int lson=i\*2+1;

    int rson=i\*2+2;

    if(lson<n&&strcmp(arr[largest].coursename,arr[lson].coursename)<0)

        largest=lson;

    if(rson<n&&strcmp(arr[largest].coursename,arr[rson].coursename)<0)

        largest=rson;

    if(largest!=i)

    {

        swap(&arr[largest],&arr[i]);

        heapify(arr,n,largest);

    }

}

6.7.堆排序代码

void heap\_sort(PseqList &L)

{

    int i;

    for(i=L->curNum/2-1; i>=0; i--)*//大顶堆*

        heapify(L->element,L->curNum,i);

    for(i=L->curNum-1; i>0; i--)*//将数组排序*

    {

        swap(&L->element[i],&L->element[0]);*//将每次维护后的大顶从后往前排序*

        heapify(L->element,i,0);

        print(L);

    }

}

6.5.冒泡排序代码

void bubble\_sort(PseqList &L)

{

    int i,j,p;

    DataType temp;

    for(i=0; i<L->curNum-1; i++)

    {

     p=1;

        for(j=0; j<L->curNum-1-i; j++)

        {

            if(strcmp(L->element[j].coursename,L->element[j+1].coursename)>0)

            {

                swap(&L->element[j+1],&L->element[j]);

                p=0;

            }

        }

        print(L);

        if(p==1)

        break;

    }

     print(L);

}

**6.5.快速排序代码**

void quick\_sort(PseqList &L,int left,int right)

{

    if(left<right)

    {

        DataType base=L->element[left];

        int l=left;

        int r=right;

        while(l<r)

        {

            while(l<r&&strcmp(L->element[r].coursename,base.coursename)>0)

                r--;

            L->element[l]=L->element[r];

            while(l<r&&strcmp(L->element[l].coursename,base.coursename)<0)

                l++;

            L->element[r]=L->element[l];

        }

        L->element[l]=base;

        print(L);

        quick\_sort(L,left,r-1);

        quick\_sort(L,l+1,right);

    }

}