全国计算机技术与软件专业技术资格（水平）考试参考用书

程序员考试同步辅导
(下午科目)(第2版)

全国计算机专业技术资格考试办公室推荐

陈海燕 程勇 李为健 主编

清华大学出版社

全国计算机技术与软件专业技术资格(水平)考试参考用书

程序员考试同步辅导
(下午科目)(第2版)

全国计算机专业技术资格考试办公室推荐

陈海燕　程　勇　李为健　主编

清华大学出版社
北京

内容简介

本书是按照人事部(现为人力资源和社会保障部)、信息产业部(现为工业和信息化部)最新颁布的全国计算机技术与软件专业技术资格(水平)考试大纲和指定教材而编写的考试用书。全书分为6章，内容包括：常用算法和数据结构，程序流程图和N-S图，C语言，C++语言，Java语言程序设计，程序员考试(下午科目)样卷与答案解析等，主要从考试大纲要求、考点辅导、典型例题分析和专项习题训练几个方面对各部分内容加以系统的阐释。

本书具有考点分析透彻、例题典型、习题丰富等特点，非常适合参加程序员考试的考生使用，也可作为高等院校或培训班的教材。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

程序员考试同步辅导(下午科目)/陈海燕，程勇，李为健主编. —2版. —北京：清华大学出版社，2010.6
(全国计算机技术与软件专业技术资格(水平)考试参考用书)
ISBN 978-7-302-22511-9

Ⅰ. 程… Ⅱ. ①陈… ②程… ③李… Ⅲ. 程序设计—水平考试—自学参考资料 Ⅳ. TP311.1

中国版本图书馆CIP数据核字(2010)第066341号

责任编辑：章忆文　张丽娜
装帧设计：何凤霞
责任印制：孟凡玉
出版发行：清华大学出版社　　地　址：北京清华大学学研大厦A座
http://www.tup.com.cn　　邮　编：100084
社　总　机：010-62770175　　邮　购：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京四季青印刷厂
装 订 者：三河市李旗庄少明装订厂
经　　销：全国新华书店
开　　本：185×260　印　张：22.75　插　页：2　字　数：552千字
版　　次：2010年6月第2版　　印　次：2010年6月第1次印刷
印　　数：1～4000
定　　价：38.00元

产品编号：036482-01

再版前言

全国计算机技术与软件专业技术资格(水平)考试自实施起至今已经历了 20 多年，在社会上产生了很大的影响，其权威性得到社会各界的广泛认可。为了适应我国信息化发展的需求，人事部(现为人力资源和社会保障部)、信息产业部(现为工业和信息化部)在 2004 年决定将考试的级别拓展到计算机技术与软件各个方面，将网络程序员级别考试改为网络管理员级别考试，2009 年又对网络管理员级别考试大纲进行了重新的调整，以满足社会上对各种信息技术人才的需要。本书第 1 版自 2005 年出版以来，被众多考生选用为考试参考教材，多次重印，深受广大读者好评。为了帮助考生复习迎考，根据 2009 年考试大纲的最新变化及网络新技术的发展，本书对第 1 版同名书进行修订，主要修订内容如下。

(1) 知识点更新。2009 年新大纲对知识点有所调整与变动，使其更注重实践性。本书在第 1 版的基础上，严格按照 2009 年新版大纲，对知识点做了彻底的更新，更符合新大纲新教程对考试的要求。

(2) 结构调整。参考最新指定官方教程、最新考试大纲及最新题型编写章节，便于考生使用《程序员教程(第 3 版)》同步复习，同时更加突出重点与难点，针对性强，减轻了考生复习的工作量。

(3) 例题与习题更新。书中原有例题与习题进行了彻底更新，最近 4 年(2006—2009)8 次考试真题全部分类解析到例题中去，并在其中同时增加了根据最新考试大纲精心设计的例题，具有典型性和代表性，而 2005 年及之前历次考试真题全部分类归入同步练习中。考生能从以前的考题中，更好地熟悉考试的难度与广度，顺利通过考试。

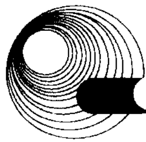
(4) 更加突出重点。第 2 版沿袭第 1 版的框架，每一小节分 4 个板块：考点辅导、典型例题分析、同步练习、同步练习参考答案。其中，考点辅导部分主要以专题的方式，重点介绍程序员上午考试所需的各个方面的知识；典型例题分析是本书的重点，它详尽细致地剖析了近 4 年(2006—2009)所有的真题和例题；同步练习每一道题都配有标准的答案；每章还配有一定数量的习题及答案，对读者所学的知识和能力起到巩固、拓宽和提高的作用。

(5) 语言进行了锤炼，讲述更准确、概念清晰，覆盖所有大纲考点，并突出重难点。

(6) 书中所有例题与习题进行了精选，确保所有题目符合考纲要求，例题选取要典型、有梯度、有广度，分析要详尽；题目的难易度、分布率与真实考试相当；题目答案正确、解析科学；无重复题目、雷同题目。

本书非常适合备考程序员的考生使用，也可作为高等学校相关专业或培训班的教材。

本书第 1 版由杨明、杨萍编写。第 2 版是对第 1 版的修订与升级，具体由陈海燕、程勇、李为健完成编写与升级工作。此外，参与本书编写的还有陈智、郭龙源、何光明、蒋道霞、李佐勇、马常霞、祁云嵩、申继年、孙建东、王珊珊、徐军、许勇、张宏等。在此对作品的全体参与人员表示衷心的感谢。



在本书编写的过程中，参考了许多相关的书籍和资料，目录详见参考文献，本书从中汲取了许多营养，在此表示感谢。需要特别提出感谢的是来自互联网的各位不知道姓名的网友们的无私奉献，正是由于你们，才使本书的内容更完善，更详尽。

由于时间仓促和水平有限，书中难免存在错漏和不妥之处，敬请读者批评指正。

编　者

试读结束：需要全本请在线购买： www.ertongbook.com

目　　录

第 1 章　常用算法和数据结构......................1

1.1　排序算法..1

1.1.1　考点辅导......................................1

1.1.2　典型例题分析............................10

1.1.3　同步练习....................................11

1.1.4　同步练习答案............................14

1.2　查找算法..16

1.2.1　考点辅导....................................16

1.2.2　典型例题分析............................21

1.2.3　同步练习....................................25

1.2.4　同步练习答案............................29

1.3　数据结构..32

1.3.1　考点辅导....................................32

1.3.2　典型例题分析............................69

1.3.3　同步练习....................................77

1.3.4　同步练习答案............................87

1.4　本章小结..92

1.5　达标训练题及参考答案......................93

1.5.1　达标训练题................................93

1.5.2　参考答案....................................98

第 2 章　程序流程图和 N-S 图..................101

2.1　流程图和 N-S 图的基础知识............101

2.1.1　考点辅导..................................101

2.1.2　典型例题分析..........................103

2.1.3　同步练习..................................111

2.1.4　同步练习答案..........................112

2.2　经典算法的描述................................113

2.2.1　考点辅导..................................113

2.2.2　典型例题分析..........................113

2.2.3　同步练习..................................115

2.2.4　同步练习答案..........................116

2.3　信息处理的模拟................................117

2.3.1　考点辅导..................................117

2.3.2　典型例题分析..........................118

2.3.3　同步练习..................................123

2.3.4　同步练习答案..........................123

2.4　本章小结..124

2.5　达标训练题及参考答案....................124

2.5.1　达标训练题..............................124

2.5.2　参考答案..................................127

第 3 章　C 语言..128

3.1　C 语言的程序结构............................128

3.1.1　考点辅导..................................128

3.1.2　典型例题分析..........................129

3.1.3　同步练习..................................131

3.1.4　同步练习答案..........................131

3.2　C 语言的数据类型、运算符和表达式..........131

3.2.1　考点辅导..................................131

3.2.2　典型例题分析..........................133

3.2.3　同步练习..................................135

3.2.4　同步练习答案..........................137

3.3　C 语言的基本语句............................139

3.3.1　考点辅导..................................139

3.3.2　典型例题分析..........................139

3.3.3　同步练习..................................145

3.3.4　同步练习答案..........................148

3.4　标准输入输出函数............................149

3.4.1　考点辅导..................................149

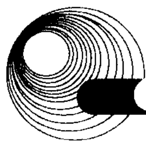
3.4.2　典型例题分析..........................150

3.4.3　同步练习..................................152

3.4.4　同步练习答案..........................153

3.5　数组和函数..154

3.5.1　考点辅导..................................154



3.5.2 典型例题分析 158
3.5.3 同步练习 164
3.5.4 同步练习答案 169
3.6 指针 171
3.6.1 考点辅导 171
3.6.2 典型例题分析 176
3.6.3 同步练习 180
3.6.4 同步练习答案 181
3.7 本章小结 181
3.8 达标训练题及参考答案 182
3.8.1 达标训练题 182
3.8.2 参考答案 187

第4章 C++语言 190

4.1 C++程序基础 190
4.1.1 考点辅导 190
4.1.2 典型例题分析 192
4.1.3 同步练习 194
4.1.4 同步练习答案 195
4.2 类、成员、构造函数及析构函数 196
4.2.1 考点辅导 196
4.2.2 典型例题分析 199
4.2.3 同步练习 201
4.2.4 同步练习答案 204
4.3 模板 205
4.3.1 考点辅导 205
4.3.2 典型例题分析 207
4.3.3 同步练习 212
4.3.4 同步练习答案 215
4.4 继承和多态 216
4.4.1 考点辅导 216
4.4.2 典型例题分析 219
4.4.3 同步练习 222
4.4.4 同步练习答案 223
4.5 本章小结 224
4.6 达标训练题及参考答案 224
4.6.1 达标训练题 224
4.6.2 参考答案 227

第5章 Java语言程序设计 228

5.1 Java语言的程序结构和基本语法 228
5.1.1 考点辅导 228
5.1.2 典型例题分析 230
5.1.3 同步练习 232
5.1.4 同步练习答案 233
5.2 类、成员、构造函数 234
5.2.1 考点辅导 234
5.2.2 典型例题分析 237
5.2.3 同步练习 244
5.2.4 同步练习答案 246
5.3 继承及接口 247
5.3.1 考点辅导 247
5.3.2 典型例题分析 250
5.3.3 同步练习 252
5.3.4 同步练习答案 254
5.4 本章小结 254
5.5 达标训练题及参考答案 254
5.5.1 达标训练题 254
5.5.2 参考答案 258

第6章 程序员考试(下午科目)样卷与答案解析 260

6.1 样卷 260
6.1.1 样卷一 260
6.1.2 样卷二 265
6.1.3 样卷三 273
6.1.4 样卷四 278
6.1.5 样卷五 285
6.1.6 样卷六 291
6.1.7 样卷七 298
6.1.8 样卷八 305
6.1.9 样卷九 312
6.1.10 样卷十 319
6.2 答案解析 325
6.2.1 样卷一答案解析 325
6.2.2 样卷二答案解析 328
6.2.3 样卷三答案解析 331

6.2.4 样卷四答案解析 334
6.2.5 样卷五答案解析 337
6.2.6 样卷六答案解析 340
6.2.7 样卷七答案解析 343
6.2.8 样卷八答案解析 346
6.2.9 样卷九答案解析 349
6.2.10 样卷十答案解析 352
参考文献..356

第 1 章　常用算法和数据结构

大纲要求：

- 排序算法。
- 查找算法。
- 数据结构(线性表、栈、队列、数组、树、图)。

1.1　排 序 算 法

1.1.1　考点辅导

1.1.1.1　选择排序

若设 R[1...n]为待排序的 n 个记录，R[1...i-1]已按照主关键字由小到大排序，且任意 x∈R[1...i-1]，y∈R[i...n]满足 x.key≤y.key，则选择排序的主要思路如下。

(1) 反复从 R[i...n]中选出关键字最小的结点 R[k]。

(2) 若 i≠k，则将 R[i]与 R[k]交换，使得 R[1...i]有序且保持原来的性质。

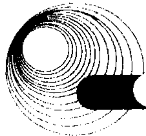
(3) i 增 1，直到 i 为 n。

为方便描述，被查找的顺序表 C 类型定义如下：

```
#define MAXSIZE 1000                    /*顺序表的长度*/
typedef int KeyType;                    /*关键字类型为整数类型*/
typedef struct{
    KeyType key;                        /*关键字项*/
    InfoType otherinfo;                 /*其他数据项*/
}RecType;                               /*记录类型*/
typedef struct {
    RecType  r[MAXSIZE+1];              /*r[0]空作为哨兵*/
    int length;                         /*顺序表长度*/
}SqList;                                /*顺序表类型*/
```

顺序存储线性表的选择排序算法如下：

```
void Sqsort(SqList &q)
{
    int i, j, k, temp;
    for(i=0;i<q.length-1;i++)
    {
        k=i;
        for(j=i+1;j<q.length;j++)
            if(q.r[j].key<q.r[k].key)k=j;  /*选择关键字最小的记录*/
```




```
            if(k!=i)
            {
                temp=q.r[k];
                q.r[k]=q.r[i];
                q.r[i]=temp;
            }
        }
    }
```

可见，选择排序不管原先序列是否有序，其排序需要比较的次数均为 n*(n-1)/2；同时，由于相等的两个元素，位置相对在前的可能被交换到后面，故该选择排序是不稳定的。

1.1.1.2 直接插入排序

若设 R[1...n]为待排序的 n 个记录，R[1...i-1]已按照主关键字由小到大排序，则直接插入排序的主要思路如下。

(1) 寻找 R[i]在 R[1...i-1]中的插入位置，确保 R[i]插入后保持有序。

(2) i 增 1，若 i 小于等于 n，则转到(1)执行，否则结束。

顺序线性存储结构下的直接插入排序算法如下：

```
void Dinsert(SqList &q)
{
  int i,j,k;
  for(i=1;i<q.length;i++)                                   /*q.r[0]为有序*/
  {
      for(t=q.r[i],j=i-1;j>=0 && t.key<q.r[j].key;j--)/*找到插入的位置*/
          q.r[j+1]=q.r[j];
      q.r[j+1]=t;
  }
}
```

【点评】

(1) 对 n 个结点的线性表采用直接插入排序，当线性表已是从小到大排序时，内循环每执行一次，只需进行 1 次比较，整个排序过程只进行 n-1 次比较。当线性表已是从大到小排序时，对外循环执行 i 次，内循环要进行 i 次比较。整个排序过程需要进行 n*(n-1)/2 次比较。可见，对 n 个结点的线性表采用直接插入排序，最少比较次数为 n-1，最多比较次数为 n*(n-1)/2。

(2) 直接插入排序是在有序表的基础上进行的，所以排序效率较高，且比较稳定。

1.1.1.3 希尔排序

希尔排序(Shell Sort)的基本思路为：把直接插入方法分成插入步长由大到小不同的若干趟来进行，一开始步长较大，相当于把序列分成几个子表。对每个子表来说，因为其结点少，直接插入排序的效率会很高。以后各趟逐步减小步长，子表的结点也越来越多，但是子表中的结点已经进行过前一趟的大步长的直接插入排序，有相当多的结点已基本有序。这使得后一趟的插入排序能充分利用前一趟的排序结果。当步长降到 1 时，只要对基本有序的线性表进行一趟直接插入排序即可。

初次取线性表的一半长度为步长，以后每次减半，直到步长为 1，希尔排序的算法如下：

```
void Shsort(SqList &q)
{
 int j,k,h,y;                          /*h 为步长*/
 for(h=q.length/2;h>0;h/=2)
   for(j=h;j<n;j++)                    /*对每个子表进行直接插入排序*/
    {
   y=q.r[j];
        for(k=j-h;k>=0&&y.key<q.r[k].key;k-=h)
        q.r[k+h]=q.r[k];               /*找到插入的位置并移动*/
   q.[k+h]=y;
    }
}
```

1.1.1.4　冒泡排序

若设 R[1...n]为待排序的 n 个记录，且假设为从上至下纵向排列，则要求将 n 个给定记录由小到大排序的冒泡排序(Bubble Sort)的基本思路如下。

(1) 对当前还未排好序的、指定范围内的全部结点，自上而下对相邻的两个结点依次进行比较和调整，让关键字较大的结点往下沉，关键字较小的结点往上冒。即若 R[j].key>R[j+1].keg，则将 R[j].key 与 R[j+1].key 交换。

(2) 初始化时，排序范围是 R[1...n]，以后的排序范围由前一遍的扫描结果确定：当自上而下将当前排序范围内的结点执行一遍比较之后，若最后往下沉的结点是 R[j]，R[j]下沉到 R[j+1]的位置，R[j+1]以下的结点比较后会发现它们都不再需要交换。因此，下一趟的排列范围可以缩减为从 R[1]到 R[j]。

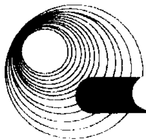
(3) 整个排列过程最多执行 n−1 趟。若某一趟的比较没有结点交换，所有相邻结点的排列顺序都与排序要求一致，则线性表为有序的。

顺序线性存储结构下的冒泡排序算法如下：

```
void bubblesort(SqList &q)
 {
int j,p=q.length-1,h,t;           /*p 用来记录一趟排序最后下沉的结点位置*/
for(h=1;h<=p;)
  for(j=0;j<p-1;j++)
    if(q.r[j].key>q.r[j+1].key){
      temp=q.r[j]; q.r[j]=q.r[j+1];q.r[j+1]=temp;
      p=j;
      }
}
```

可见，对 n 个结点的线性表采用冒泡排序，外循环最多执行 n−1 趟，每一趟最多执行 n−1 次比较；第 2 趟最多执行 n−2 次比较；依次类推，第 n−1 趟最多执行 1 次比较。因此，整个排序过程最多执行 n*(n−1)/2 次比较。由于关键字相等的结点不交换，故冒泡排序算法是稳定的。

试读结束：需要全本请在线购买： www.ertongbook.com



1.1.1.5 快速排序

快速排序(Quick Sort)的主要思路为：通过对线性表序列的一趟扫描使某个结点移到中间的某个位置，且使其左边序列的各结点的关键字都比该结点的关键字小，而其右边序列的各结点的关键字都不比该结点的关键字小，常称这样的一次扫描为“划分”。然后，对左、右序列进行同样的处理，直到所有序列均只包含一个结点为止，这样便可将原线性表排好序。

快速排序可以看做是冒泡排序的改进，冒泡排序可以看做是快速排序的退化，即每趟划分总是在同一端进行。

若设待排序的记录序列{ R1，R2，…，Rn }为 R[1...n]，则对其按关键值的非递减序列进行快速排序的算法如下：

```
void QuickSort1(SqList &R, int s, int t)
{
                                        /*对n个元素的数组R进行由小到大排序*/
    int low, high;
    low=s;
    high=t;
    privotkey=R.r[s].key;
    R.r[0]=R.r[s];
    while(low<high)
    {
        while(high>low && R.r[high].key>privotkey)
            high--;
        if (low<high)
        {
            R.r[low]=R.r[high];     /*将比枢轴小的记录移动到低端*/
            low++;
        }
        while(low<high && R.r[low].key<=privotkey)
            ++low;
        if(low<high)
        {
            R.r[high]=R.r[low];     /*将比枢轴大的记录移动到高端*/
            high--;
        }
    }
    R.r[low]=R.r[0];                /*枢轴记录到位，即完成一趟排序*/
    QuickSort1(R, s, i-1);          /*对左区间递归排序*/
    QuickSort1(R, i+1, t);          /*对右区间递归排序*/
}                                   /*完成快速排序*/
```

可见，快速排序可能会破坏两个相等记录的原来次序，因而快速排序算法是不稳定的。

1.1.1.6 将顺序存储结构上的排序算法移植到链表上

很多优秀的算法都是建立在顺序存储结构上的，如何在链式存储结构上实现这些优秀

算法，是考生应注意的问题。近年来，在程序员水平考试中也出现了这样的题目。这里，我们通过在顺序存储结构和链式存储结构两种存储结构上实现快速排序来说明。顺序存储结构下的算法 QuickSort1 可以拓展到链式存储结构下的算法 QuickSort2。

下面的算法就是将算法 QuickSort1 拓展到链式存储结构下的算法 QuickSort2。

```
typedef  struct  node {
    int data;                       /*结点的数据域*/
    struct  node  *next;            /*结点的指针域或链域*/
} Slink;
QuickSort2(Slink *head, Slink *tail)
                                    /*将不带头结点的单链表 head 进行由小到大排序*/
                                    /*主程序调用时，tail=NULL*/
{
    Slink *p=head->next , *mid, *midpre, *pre, *r;
    int temp;
    midpre=head;
    mid=p;
    if(!p)return 1;
    pre=p;
    p=p->next;
    while(p!=tail)
    {
        r=p->next;
        if(p->data<=head->next->data)
        {
            pre->next=r;                /*断开与 p 结点的链接*/
            midpre->next=p;             /*将 p 结点插入到 mid 之前*/
            p->next=mid;
            midpre=p;
            p=r ;
        }
        else
        {                               /*结点的位置保持不变*/
            pre=p;
            p=r;
       }
    }
    QuickSort2(head, midpre);           /*对前一部分链表进行快速排序*/
    QuickSort2(mid->next,tail);         /*对后一部分链表进行快速排序*/
}                                       /*QuickSort2*/
```

可见，只要充分领会顺序存储结构下的算法思想，熟悉链表存储结构就可以通过掌握顺序存储结构下的算法得到链表存储结构下的相应算法。

1.1.1.7 堆排序

首先，要认真掌握堆的定义，然后才能进一步理解建堆的算法。堆的定义为：n 个关键字序列 k_1，k_2，k_3，…，k_n 称为堆，当且仅当该序列满足如下性质(也称堆性质)：①$k_i \leq k_{2i}$ 且

$k_i \leqslant k_{2i+1}$ 或 ② $k_i \geqslant k_{2i}$ 且 $k_i \geqslant k_{2i+1}(1 \leqslant i \leqslant n/2)$。满足第①种情况的堆称为小根堆，满足第②种情况的堆称为大根堆。这里仅讨论第①种情况。

本质上，堆排序在排序过程中，是将顺序表中存储的数据看成一棵完全二叉树，利用完全二叉树中双亲结点和孩子结点之间的内在关系来选择关键字最小记录。堆排序分建堆和堆调整两个过程。

建堆的过程为：堆排序的过程是一个不断从堆顶到叶子的调整过程(又称“筛选”)。从一个无序序列建堆的过程就是一个反复筛选的过程。若将此序列看成是一个完全二叉树，则最后一个非终端结点是第 n/2 个元素，因此筛选只需从第 n/2 个元素开始。

堆调整过程为：将该完全二叉树中最后一个元素替代已输出的结点。若新的完全二叉树的根结点小于左右子树的根结点，则直接输出。反之，则比较左右子树根结点的大小。若左子树的根结点小于右子树的根结点(或右子树的根结点小于左子树的根结点)，则将左子树(或右子树)的根结点与该完全二叉树的根结点进行交换。重复上述过程，调整左子树(或右子树)，直至叶子结点，则新的二叉树满足堆的条件。

堆排序的算法描述如下：

```
void HeapSort(Sqlist &R)            /*R.r[1...R.length]看成是完全二叉树的*/
                                    /*顺序存储结构*/
{                                   /*建立小根堆*/
    for(i=R.length/2; i>0; i--)     /*从最后一个内部结点开始调整*/
        HeapAdjust(R, i, R.length);
}
void HeapAdjust(Sqlist & R, int i, int m)
{                               /*i 为堆调整的位置，m 为堆的大小，该函数建立小根堆*/
    R.r[0]=R.r[i];
    for(j=2*i; j<=m; j*=2)
    {
        if(j<m && R.r[j].key>R.r[j+1].key)++j;
                                    /*找孩子结点中关键字最小的*/
        if(R.r[0].key<=R.r[j].key)  /*调整结束*/
            break;
        R.r[i]=R.r[j];          /*使得根结点的关键字小于等于左右孩子的关键字*/
        i=j;
    }
    R.r[s]=R.r[0];
}
```

堆排序算法的时间复杂度为 $O(n\log_2 n)$，这是一个不稳定的排序方法。

1.1.1.8 合并排序

合并排序(Merging Sort)的基本思想是：将两个或者两个以上的有序子表合并成一个新的有序表。对于两个有序子表合并成一个有序表的 2-路合并排序来说，初始时，把含有 n 个结点的待排序序列看做由 n 个长度都为 1 的有序子表所组成，将它们依次两两合并得到长度为 2 的若干有序子表，再对它们做两两合并，直到得到长度为 n 的有序表，排序即告完成。

2-路合并排序算法如下：

```
void mergesort(SqList &q)
{
    SqList r;
    r.length=q.length;
    int len=1,f=0;
    while(len<q.length)
    {                                        /*交替地在q.r[]和r.r[]之间来回合并*/
        if(!f)
            mergepass(&q,&r,len);
        else
            mergepass(&r,&q,len);
        len*=2;                              /*一趟合并后，有序段结点数加倍*/
        f=1-f;                               /*控制交替合并*/
        if(f)
            q.r[0...q.length-1]=r.r[0...r.length-1]; }
}
```

其中，一趟合并和相邻两个有序段合并的函数如下：

```
void mergerpass(SqList &q, SqList &r, int len)
{
    int start, end;
    start=0;
    while(start+len<q.length)
    {                                        /*至少还有两个有序段*/
        end=start+2*len-1;
        if(end>=q.length)
        {
            end=q.length-1;                  /*最后一个段可能不足 len 个结点*/
            mergestep(&q,&r,start, start+len-1, end);
                                             /*相邻有序段合并*/
            start=end+1;
        }
    }
    if(start<n)                        /*还剩下一个有序段时将其从q.r[]复制到r.r[]*/
        for(;start<q.length;start++)
            r.r[start]=q.r[start];
}

void mergestep(SqList &q,SqList &r, int start, int middle, int end)
                                                   /*将两个相邻段合并成一个段*/
{
    int j,k,l;
    k=l=start;
    j=middle+1;
    while(l<=middle&&j<=end)
    {                                              /*当两个有序段都未结束时循环*/
        if(q.r[l]<=q.r[j])r.r[k++]=q.r[l++];       /*取其中小的元素复制*/
```

```
        else r.r[k++]=q.r[j++];
        while(l<=middle)
            r.r[k++]=q.r[l++];                  /*复制还未合并完的剩余部分*/
        while(j<=end)
            r.r[k++]=q.r[j++];
    }
}
```

可以看出，合并排序是一种稳定的排序方法，但需要和待排序序列一样多的辅助存储空间。

1.1.1.9 如何在r进制下运用基数排序

如何依据实际情况，对任意r进制运用基数排序是基数排序算法设计的关键。下面通过例子来说明和理解基数排序。

若下列C程序的功能是：用基数排序法对读入的n个无符号整数进行排序(排成从小到大的次序)，请在程序空缺处填上适当字句，使其能正确执行。

```
#define  M 1000
#define  Radix 16
#define  Bits 4
rsort(int a[], int n)
{
    int Bit,divisor, i;
    int count[Radix];
    divisor=1;
    for(Bit=1;Bit<=Bits; Bit++)
    {
        for(i=0;i<=Radix-1;i++)
            count[i]=0;
        for(i=1;i<=n;i++)
            count[(a[i]/divisor)%Radix]=count[(a[i]/divisor)%Radix]+1;
        for(i=0;i<=Radix-1;i++)
            count[i]=count[i]+count[i-1];
        for(i=n;i>=1;i--)
        {
            t[(1)]=a[i];
            count[(a[i]/ divisor)%Radix]:=(2);
        }
        for(i=1;i<=n;i++)
            (3);
        divisor=(4);
    }
}
```

基数排序的两种主要方法是：最高位优先MSD(Most Significant Digit first)和最低位优先LSD(Least Significant Digit first)。这里的程序采用最低位优先LSD方法对输入的n个整数进行排序，采用的主要思想是把各整数看成由4个16位数组成；若从低到高分别称为L1、

L2、L3、L4，则执行下面步骤。

(1) 初始化计数器 count[0...15]，使各个 count[i]=0(0≤i≤15)。

(2) 统计 L1 为 0、1、…、15 的各个整数个数分别到 count[0...15]。

(3) 将 count[0]、count[1]、…、count[15]合并起来得到 L1 为 j(0≤j≤15)的整数在一趟排序后的起始位置；如 a[j]在 count[(a[j]> divisor%) Radix]到 count[(a[j]/divisor)%Radix+1]-1 之间；同时 L1 相同的整数的前后位置没有明显的界限，只需根据读入的次序来定。

(4) 按照得到的各个整数 a[i]的位置 count[j]将该元素登记在相应的位置上，同时 count[j]增加 1，以便存放下一个和整数 a[i]的 L1 相同的元素。

(5) 对 L2、L3、L4 重复上述过程。

通过仔细分析，不难得到如下答案。

(1) count[(a[i] / divisor)% Radix]

(2) count[(a[i] / divisor)% Radix]+1

(3) a[i]=t[i]

(4) divisor*16

进一步推广，若把整数看成八进制或其他进制，同样也能得到问题的答案。

1.1.1.10 败者树

采用败者树的目的是为了在进行最小键值的查找时减少比较次数。

败者树是一棵完全二叉树，其中每个结点的键值都取其两个子结点的键值中的较小者，因此，根结点的键值是这棵树中所有结点的键值中最小的。这就像 k 个参加淘汰赛的球队，胜者(值较小者)进入下一轮的比赛，根结点为冠军(值最小者)。

败者树的构造过程是：对具有 k 个记录的序列，首先用这 k 个记录作为叶结点，然后把相邻的两个结点进行比较，把键值小的记录(优胜者)作为这两个结点的父结点，按此方法自下而上一层一层地产生败者树的结点。为了节约内存空间，非叶子结点可不包含整个记录，只要存放记录的键值及指向该记录的指针即可。

败者树的根结点的值是构成败者树的元素中最小的，在后面的应用中，往往把根结点的值输出并用一个新的元素替换，要求构成新的败者树，这时只要在原来的败者树的基础上进行调整即可。调整仅在从根到新加入的叶子结点的树枝上的结点及其兄弟结点之间进行，自下而上进行比较并调整其父结点。

1.1.1.11 k 路归并法

有了 m 个初始归并段(都是有序段)，便可进行 k 路归并了，即将 k 个初始归并段采用某种方法进行归并产生一个段，这样 m 个初始归并段便产生多个更大的段，然后对这些段再进行归并，如此下去，直到只生成一个段为止，这个段就是最后生成的归并段。

在内存里进行 k 路归并的方法很多。当归并路数 k 较大时，为了减少合并时的比较次数，常采用败者树进行合并的方法，其合并过程如下。

(1) 用参加合并的 k 个有序段的第一个记录构造一棵初始败者树，该树中的根结点就是这 k 个记录中具有最小键值的记录。

(2) 把败者树根结点所代表的记录送到输出缓冲区。

试读结束：需要全本请在线购买： www.ertongbook.com