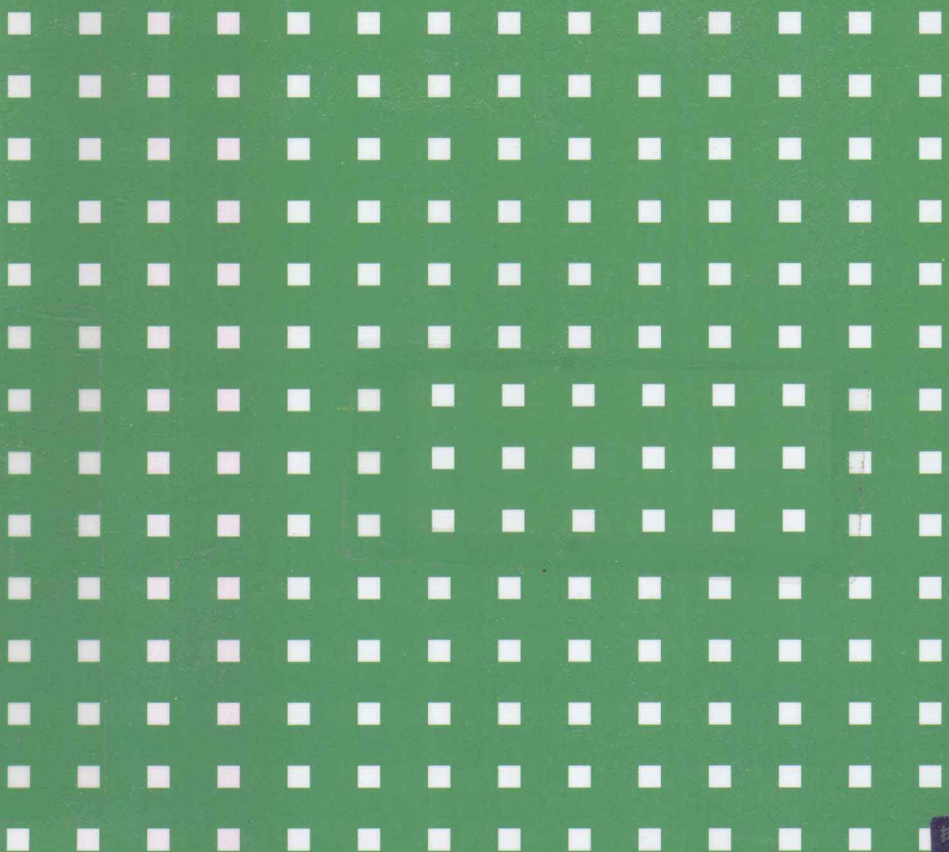
高等学校计算机专业教材精选·算法与程序设计

算法设计与分析

温敬和 主编



清华大学出版社

高等学校计算机专业教材精选·算法与程序设计

算法设计与分析

温敬和 主编

清华大学出版社
北京

内 容 简 介

本书根据课程教学要求编写，内容包括算法分析基本概念、堆和不相交集数据结构、归纳法、分治法、动态规划法、贪心法和回溯法。各章的主要算法(包括算法说明、算法伪代码描述、算法分析和算法实现程序)、习题解答和上机题以及书中出现的所有源程序均可以从清华大学出版社网站(www. tup. com. cn)下载。

本书既可作为“算法设计与分析”课程的主讲教材，也可作为其辅助教材，还可以作为软件工程师学习算法设计的参考教材。

图书在版编目(CIP)数据

算法设计与分析/温敬和主编．—北京：清华大学出版社，2011.6
(高等学校计算机专业教材精选·算法与程序设计)
ISBN 978-7-302-24473-8

Ⅰ. ①算… Ⅱ. ①温… Ⅲ. ①电子计算机—算法设计—高等学校—教材 ②电子计算机—算法分析—高等学校—教材 Ⅳ. ①TP301.6

中国版本图书馆 CIP 数据核字(2010)第 265234 号

责任编辑：白立军　薛　阳
责任校对：时翠兰
责任印制：杨　艳

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：010-62795954，jsjjc@tup. tsinghua. edu. cn
质　量　反　馈：010-62772015，zhiliang@tup. tsinghua. edu. cn
印 装 者：北京市清华园胶印厂
经　　销：全国新华书店
开　　本：185×260　　**印　　张**：12.75　　**字　　数**：308 千字
版　　次：2011 年 6 月第 1 版　　**印　　次**：2011 年 6 月第 1 次印刷
印　　数：1～3000
定　　价：22.00 元

产品编号：040034-01

出版说明

我国高等学校计算机教育近年来迅猛发展，应用所学计算机知识解决实际问题，已经成为当代大学生的必备能力。

时代的进步与社会的发展对高等学校计算机教育的质量提出了更高、更新的要求。现在，很多高等学校都在积极探索符合自身特点的教学模式，涌现出一大批非常优秀的精品课程。

为了适应社会的需求，满足计算机教育的发展需要，清华大学出版社在进行了大量调查研究的基础上，组织编写了《高等学校计算机专业教材精选》。本套教材从全国各高校的优秀计算机教材中精挑细选了一批很有代表性且特色鲜明的计算机精品教材，把作者们对各自所授计算机课程的独特理解和先进经验推荐给全国师生。

本系列教材特点如下。

(1) 编写目的明确。本套教材主要面向广大高校的计算机专业学生，使学生通过本套教材，学习计算机科学与技术方面的基本理论和基本知识，接受应用计算机解决实际问题的基本训练。

(2) 注重编写理念。本套教材作者群为各高校相应课程的主讲，有一定经验积累，且编写思路清晰，有独特的教学思路和指导思想，其教学经验具有推广价值。本套教材中不乏各类精品课配套教材，并力图努力把不同学校的教学特点反映到每本教材中。

(3) 理论知识与实践相结合。本套教材贯彻从实践中来到实践中去的原则，书中的许多必须掌握的理论都将结合实例来讲，同时注重培养学生分析问题、解决问题的能力，满足社会用人要求。

(4) 易教易用，合理适当。本套教材编写时注意结合教学实际的课时数，把握教材的篇幅。同时，对一些知识点按教育部教学指导委员会的最新精神进行合理取舍与难易控制。

(5) 注重教材的立体化配套。大多数教材都将配套教师用课件、习题及其解答，学生上机实验指导、教学网站等辅助教学资源，方便教学。

随着本套教材陆续出版，我们相信它能够得到广大读者的认可和支持，为我国计算机教材建设及计算机教学水平的提高，为计算机教育事业的发展做出应有的贡献。

清华大学出版社

前　言

作者1982年2月毕业于上海交通大学，2010年1月退休，在上海第二工业大学工作了近三十年。在此期间，主要从事“编译程序”和“算法”这两门学科的教学和科研。2005年4月清华大学出版社出版了由本人编著的《编译原理实用教程》，该书至今仍用于我校和国内其他普通高等院校“编译原理”课程的教学。相对于“编译原理”课程而言，“算法设计与分析”课程教材的数量不多，辅助教材更少。多年来，我校“算法设计与分析”课程使用的是由电子工业出版社出版的《算法设计技巧与分析》一书，作者是国际著名算法专家M. H. Alsuwaiyel[沙特]。该书涉及范围广、难度大，可称得上是“算法设计与分析”课程的经典教材。使用该教材，在教学中存在一定难度。为了提高“算法设计与分析”课程的教学水平，同时也想抛砖引玉和同行教师交流教学经验，退休后打算根据自己的教学心得写一本该书的辅助教材。在清华大学出版社的大力支持下，我的愿望终于得以实现。

根据课程教学要求，本书对应《算法设计技巧与分析》中的7章。它们是：第1章　算法分析基本概念，第4章　堆和不相交集数据结构，第5章　归纳法，第6章　分治，第7章　动态规划，第8章　贪心算法和第13章　回溯法。主要内容为：各章的主要算法（包括算法说明、算法伪代码描述、算法分析和算法程序实现）、习题解答和上机题。在书中出现的所有源程序，可以从清华大学出版社指定网站下载。在习题答案中，凡是用伪代码描述的具有一定难度的答案，都有相应的源程序，也可以从清华大学出版社指定的网站下载。另外，由本人编写的“算法设计与分析”课程的电子教案可从“中国高等学校教学资源网”下载。

本书既可作为“算法设计与分析”课程的主讲教材，也可作为“算法设计与分析”课程的辅助教材。上海第二工业大学教师闫季鸿参与了各章习题答案的编写，原上海第二工业大学学生董淑芳参与了各章上机题的编写。

由于时间所限，书中存在错误和遗漏之处在所难免，希望广大读者批评指正。

温敬和

2011年3月

目　录

第 1 章　算法分析基本概念

算法是解题方案准确完整的描述，是在有限步骤内求解某一问题所使用的一组定义明确的规则。无论是解题思路还是编写程序，都是在实施某种算法。前者是推理实现，后者是操作实现。算法不是程序，算法是用文字、伪代码或程序来描述的。

算法分析研究的目标是：算法一旦转换成某种语言的程序，该程序在计算机上运行，需要多少时间和存储空间才能完成解题方案。确定一个程序的运行效率，既可以使用数学分析方法，也可以使用实验测试方法，在理论和实践两个方面对算法作出评估。

在计算复杂性领域中，主要是研究算法所需要的时间和空间。当给出合法的输入时，为了得到输出，执行该算法需要多少时间和空间，其中时间尤为重要。

称“一个算法对于输入 x 要用 y 秒来运行”是没有意义的，也是没有必要的。因为一个算法的实际运行时间与诸多因素有关，例如机器性能、语言选择、编译程序优劣和程序员素质等。甚至无须计算出算法运行时间的近似数，理由如下：

(1) 算法分析通常是将解决同一问题的各种算法进行相互比较，执行时间是相对的，而不是绝对的。

(2) 算法应独立于机器，无论科技如何进步，对算法运行时间的评估始终成立。

(3) 算法分析不拘泥于小规模数据的输入，主要关心在大规模的输入实例时，算法运行的状况。

在算法分析中，通常是用渐近运行时间来度量一个算法，渐近运行时间通常称为时间复杂性。算法分析中的渐近符号有 O、Ω、Θ 和 o。

1.1　主要算法及程序实现

1.1.1　二分搜索

1. 算法文字说明

A[low..high]中的元素按升序排列，A[mid]为中间元素，判定给定元素 x 是否在 A 中。假定 x>A[mid]，如果 x 在 A 中，则它必定是 A[mid+1..high]中的一个，接下来只需在 A[mid+1..high]中搜索 x 即可，A[low..mid]中的元素在后续比较中被舍弃了。类似地，如果 x<A[mid]，只需在 A[low..mid−1]中搜索 x。

2. 算法伪代码描述

算法 BinarySearch

输入：n 个元素的数组 A[1..n](按升序排列)和元素 x。

输出：如果 x=A[j],1≤j≤n,则输出 j,否则输出 0。

```
1  low←1;high←n;j←0                    //j=0表示未找到
2  while (low≤high) and (j=0)
```

```
3    mid←⌊(low+high)/2⌋                    //⌊⌋表示取整
4    if x=A[mid] then j←mid
5    else if x<A[mid] then high←mid-1
6    else low←mid+1
7  end while
8  return j
```

3. 算法分析

假定每个三向比较(if-then-else)计为一次比较，显然最小比较次数为 1，即搜索元素 x 在序列中间位置。为了计算最大比较次数，不失一般性，可假定 x≥A[high]。

(1) 第 2 次迭代前(即第 1 次迭代结束后)，数组 A 中剩余元素计算如下：

若 n 是偶数，A[mid+1..n]共有 n/2 个元素。

若 n 是奇数，A[mid+1..n]共有(n−1)/2 个元素。

两种情况都有：$A[mid+1..n]=\lfloor n/2\rfloor=\lfloor n/2^{2-1}\rfloor$。

(2) 第 3 次迭代前，数组 A 中剩余元素为：$\lfloor\lfloor n/2\rfloor/2\rfloor=\lfloor n/4\rfloor=\lfloor n/2^2\rfloor=\lfloor n/2^{3-1}\rfloor$。

(3) ……

(4) 第 j 次迭代前，数组 A 中剩余元素为：$\lfloor n/2^{j-1}\rfloor$。搜索 x 的终止条件是余下的元素仅为 1 个，即$\lfloor n/2^{j-1}\rfloor=1$，j 为最大迭代次数(最大循环次数、最大比较次数)。

$$\lfloor n/2^{j-1}\rfloor=1\Leftrightarrow 1\leqslant n/2^{j-1}<2\Leftrightarrow 2^{j-1}\leqslant n<2^j\Leftrightarrow j-1\leqslant\log_2 n<j$$

因为 $$\lfloor\log_2 n\rfloor\leqslant\log_2 n<\lfloor\log_2 n\rfloor+1$$

所以 $$j-1=\lfloor\log_2 n\rfloor\text{或}\ j=\lfloor\log_2 n\rfloor+1$$

二者均有 $j=\lfloor\log_2 n\rfloor+1$。

由此可得，对于一个大小为 n 的已排序数组，算法执行所需的元素比较最大次数为 $\lfloor\log n\rfloor+1$(在算法分析中，$\log_2 n$ 通常表示为 $\log n$)。

4. 算法程序实现

```
1  #include <iostream.h>
2  int A[]={NULL,                          //A[0]不使用
3      1,4,5,7,8,9,10,12,15,22,23,27,32,35
4  };
5  int BinarySearch(int x)
6  {
7      int low=1,mid,high=sizeof(A)/sizeof(int)-1;
8      int j=0;
9      while(low<=high && j==0){
10         mid=(low+high)/2;
11         if(x==A[mid])
12             j=mid;
13         else
14             if(x<A[mid])
15                 high=mid-1;
16             else
17                 low=mid+1;
```

```
    }
    return j;
}
void main()
{
    int x;
    cin>>x;
    if(BinarySearch(x))
        cout<<"该数找到!"<<endl;
    else
        cout<<"该数不存在!"<<endl;
}
```

1.1.2 合并两个已排序的表

1. 算法文字说明

假定有一个数组 A[1..m],p,q,r 是它的 3 个索引,并有 1≤p≤q<r≤m,两个子数组 A[p..q] 和 A[q+1..r]各自按升序排列。重新排列 A 中元素的位置,使得子数组 A[p..r]按升序排列。

(1) 使用两个指针 s 和 t,初始化时 s=p,t=q+1, s 和 t 各自指向 A[p]和 A[q+1],空数组 B[p..r]用作暂存器。

(2) 每一次比较元素 A[s]和 A[t],将小的元素添加到 B 中然后更新指针。若 A[s]≤A[t],则 s 加 1,否则 t 加 1。

(3) 当 s=q+1,说明子数组 A[p..q]的全部元素已进入 B,无须再比较,此时把子数组余下部分 A[t..r]添加到 B 中。

(4) 当 t=r+1,说明子数组 A[q+1..r]的全部元素已进入 B,无须再比较,此时把子数组余下部分 A[s..q]添加到 B 中。

(5) 最后将 B[p..r]复制到 A[p..r]。

2. 算法伪代码描述

算法 Merge

输入:数组 A[1..m]和它的 3 个索引 p,q,r(1≤p≤q<r≤m),两个子数组 A[p..q]和 A[q+1..r]各自按升序排列。

输出:子数组 A[p..r]按升序排列。

```
comment: B[p..r]为辅助数组          //或 B[1..m]
s←p;t←q+1;k←p                       //s 和 t 分别指向数组 A 的两个子数组第 1 个元素
while (s≤q) and (t≤r)               //k 指向数组 B 当前空白位置
    if A[s]≤A[t] then
        B[k]←A[s]
        s←s+1
    else
        B[k]←A[t]
        t←t+1
```

```
10      end if
11      k←k+1                          //指向数组 B 下一个空白位置
12  end while                          //退出循环时,或 s=q+1 或 t=r+1,二者不可能同时成立
13  if s=q+1 then B[k..r]←A[t..r] //子数组 A[p..q]元素已处理完
14  else B[k..r]←A[s..q]           //否则 t=r+1,子数组 A[q+1..r]已处理完
15  end if
16  A[p..r]←B[p..r]                //复制
```

3. 算法分析

执行算法 Merge,将个数分别为 n_1 和 n_2 的两个非空数组合并成一个个数为 $n=n_1+n_2$ 的排序数组。设 $n_1 \leqslant n_2$,元素的比较次数在 n_1 和 $n-1$ 之间。特例,如果两个数组大小为 $\lfloor n/2 \rfloor$ 和 $\lceil n/2 \rceil$,需要比较的次数在 $\lfloor n/2 \rfloor$ 和 $n-1$ 之间。

4. 算法程序实现

```
1   #include <iostream.h>
2   void Merge(int A[],int p,int q,int r);
3   void main()
4   {
5       int A[]={NULL,                      //A[0]不使用
6           -1,4,5,7,8,9,110,               //A[1..M]
7           0,3,7,11,13,14,100,101          //A[M+1..N]
8       };
9       const int N=sizeof(A)/sizeof(int)-1;
10      const int M=7;                      //子数组 A[1..M]和 A[M+1..N]各自按升序排列
11      Merge(A,1,M,N);                     //p=1,q=M,r=N
12      for(int i=1;i<=N;i++)
13          cout<<A[i]<<" ";
14      cout<<endl;
15  }
16  void Merge(int A[],int p,int q,int r) //p..q,q+1..r
17  {
18      int B[101],i;                       //辅助数组大小暂定为 100
19      int s=p,t=q+1,k=p;
20      while(s<=q && t<=r){
21          if(A[s]<=A[t])
22              B[k]=A[s],s++;
23          else
24              B[k]=A[t],t++;
25          k++;
26      }                              //退出循环时,或 s==q+1 或 t==r+1,二者不可能同时成立
27      if(s==q+1)                          //因为 s==q+1,所以 t<=r
28          for(;t<=r;t++,k++)              //B[k..r]←A[t..r]
29              B[k]=A[t];
30      else                                //因为 s<=q,所以 t==r+1
31          for(;s<=q;s++,k++)              //B[k..r]←A[s..q]
```

```
32              B[k]=A[s];
33      for(i=p;i<=r;i++)                    //A[p..r]←B[p..r]
34          A[i]=B[i];
35  }
```

1.1.3 选择排序法

1. 算法文字说明

设 A[1..n]为需排序的 n 个元素的数组，选择排序法描述如下：

(1) 首先在 n 个元素中找到最小元素，将其存放在 A[1]中。

(2) 然后在剩下的 n－1 个元素中找到最小元素，将其存放在 A[2]中。

(3) 重复上述过程，直至在两个元素 A[n－1]和 A[n]中找到较小元素，将其存放在 A[n－1] 中。

2. 算法伪代码描述

算法 SelectionSort

输入：n 个元素的数组 A[1..n]。

输出：按升序排列的数组 A[1..n]。

```
1  for i←1 to n-1
2      k←i
3      for j←i+1 to n
4          if A[j]<A[k] then k←j
5      end for
6      if k≠i then 交换 A[i]和 A[k]
7  end for
```

3. 算法分析

SelectionSort 算法执行的元素比较次数为：(n－1)＋(n－2)＋…＋1＝n(n－1)/2。SelectionSort 算法执行的元素交换次数界于 0 到 n－1 之间，由于每次交换需要 3 次元素赋值，因此元素赋值次数界于 0 到 3(n－1)之间。

4. 算法程序实现

```
1  #include <iostream.h>
2  void SelectionSort(int A[],int N)
3  {
4      int i,j,k,t;
5      for(i=1;i<=N-1;i++){
6          k=i;
7          for(j=i+1;j<=N;j++)
8              if(A[j]<A[k])
9                  k=j;
10         if(k!=i)
11             t=A[k],A[k]=A[i],A[i]=t;
12     }
13 }
```

```
14  void main()
15  {
16      int A[]={NULL,                          //A[0]不使用
17          1,400,5,7,8,9,-10,12,15,22,23,27,32,35
18      };
19      const N=sizeof(A)/sizeof(int)-1;
20      SelectionSort(A,N);
21      for(int i=1;i<=N;i++)
22          cout<<A[i]<<" ";
23      cout<<endl;
24  }
```

1.1.4 插入排序法

1. 算法文字说明

设 A[1..n]为需排序的 n 个元素的数组，插入排序法描述如下：

(1) 从子数组 A[1]开始，单个数可认为有序。将 A[2]插入到 A[1]的前面或后面，使得 A[1..2]有序。

(2) 将 A[3]插入到 A[1..2]中，使得 A[1..3]有序。

(3) ……

(4) 将 A[n]插入到 A[1..n－1]中，使得 A[1..n]有序。

2. 算法伪代码描述

算法 InsertionSort

输入：数组 A[1..n]。

输出：按升序排列的数组 A[1..n]。

```
1  for i←2 to n
2      x←A[i]
3      j←i-1
4      while (j>0) and (A[j]>x)
5          A[j+1]←A[j]
6          j←j-1
7      end while
8      A[j+1]←x
9  end for
```

3. 算法分析

执行 InsertionSort 算法，元素比较次数取决于输入元素的顺序。当元素已按升序排列时，元素比较次数最小。每个元素 A[i](2≤i≤n)只和 A[i－1]比较，比较次数仅为 n－1。当元素已按降序排列，并且所有元素各不相同，比较次数最大。每个元素 A[i](2≤i≤n)需和A[i－1]，A[i－2]，…，A[1]比较，比较次数为 1＋2＋…＋(n－1)＝n(n－1)/2。

4. 算法程序实现

```
1  #include <iostream.h>
```

```
void InsertionSort(int A[],int N)
{
    int i,j,x;
    for(i=2;i<=N;i++){
        x=A[i];
        j=i-1;
        while(j>0 && A[j]>x){
            A[j+1]=A[j];
            j--;
        }
        A[j+1]=x;
    }
}
void main()
{
    int A[]={NULL,                          //A[0]不使用
        1,-400,5,7,8,9,-10,12,15,22,23,27,32,-35
    };
    const int N=sizeof(A)/sizeof(int)-1;
    InsertsionSort(A,N);
    for(int i=1;i<=N;i++)
        cout<<A[i]<<" ";
    cout<<endl;
}
```

1.1.5 自底向上合并排序法

1. 算法文字说明

设 A[1..n]为需排序的 n 个元素的数组。

(1) 第 1 次迭代：

生成个数为 2(=2^1)的段序列，该序列共有⌈n/2⌉=⌈n/2^1⌉段，段内有序。若 n 为奇数，最后一段只有 1 个元素，单独构成一段进入下一轮合并。

(2) 第 2 次迭代：

生成个数为 4(=2^2)的段序列，共有⌈n/4⌉=⌈n/2^2⌉段，段内有序。若 n 不能为 4 整除，最后一段可能由 1 个、2 个或 3 个元素构成。若为 1 个或 2 个元素(局部有序)，单独构成一段进入下一轮合并；如果为 3 个元素，将 2 个元素(局部有序)和另外 1 个元素合并成一个 3 个元素的有序段，然后进入下一轮合并。

(3) 第 j 次迭代(可假设 j=3 来理解)：

生成元素个数为 2^j 个的段序列，该序列共有⌈n/2^j⌉段，段内有序。若 n 不能为 2^j 整除，则最后一段元素个数在 1 至 2^j-1 之间，设最后一段元素个数为 r(例如 j=3，1≤r≤7)。

- 若 1≤r≤2^{j-1}(例如 j=3，1≤r≤4)，单独构成一段进入下一轮合并(局部有序)；
- 若 2^{j-1}<r<2^j(例如 j=3，5≤r≤7)，将 2^{j-1}个元素(局部有序)和另外 r−2^{j-1}个元素(局部有序)合并成个数为 r 的有序段，然后进入下一轮合并。

(4) 总的迭代次数 k 的取值范围为：$2^{k-1}<n\leqslant 2^k$($k-1<\log n\leqslant k$)。

2. 算法伪代码描述

算法 BottomUpSort

输入：n 个元素的数组 A[1..n]。

输出：按升序排列的数组 A[1..n]。

```
1  t←1
2  while t<n
3      s←t;t←2s;i←0
4      while i+t≤n
5          Merge(A,i+1,i+s,i+t)              //Merge(A,p,q,r)
6          i←i+t
7      end while                             //退出循环时 i+t>n
8      if i+s<n then Merge(A,i+1,i+s,n)      //若 i+s<n,则合并
9  end while
```

- i 表示要处理数据的起始地址(简称位移)。
- s 为被合并的子序列长度，初始时为 1，每循环一次乘以 2。
- t 为 s 的两倍，即两个子序列合并后的长度。
- 退出内层循环后执行第 8 步。如果剩余元素数目 n－i 大于 s，就要将长度为 s 的子序列和长度为 n－i－s 的子序列进行合并，合并后的长度小于 t＝2s。

3. 算法分析

假设数组元素个数 n 为 2 的幂($n=2^k=2^{\log n}$)，外部循环次数为 $k=\log n$。在执行内部循环后，由于 n 为 2 的幂，永远不会执行第 8 步。

(1) 在第 1 次迭代中，n 个元素被划分为 n/2 个段序列，共执行了 n/2 次比较。段长＝2，段内有序。

(2) 在第 2 次迭代中，n/2 个段序列(段长＝2)被合并成 n/4 个段序列，合并次数为 n/4。段长＝4，段内有序。每对合并需要的比较次数最少为 2，最多为 3。

(3) 在第 3 次迭代中，n/4 个段序列(段长＝4)被合并成 n/8 个段序列，合并次数为 n/8。段长＝8，段内有序。每对合并需要的比较次数最少为 4，最多为 7。

(4) ……

(5) 在第 j 次迭代中，$n/2^{j-1}$ 个段序列(段长＝2^{j-1})被合并成 $n/2^j$ 个段，合并次数为 $n/2^j$。段长＝2^j，段内有序。每对合并需要的比较次数最少是 2^{j-1}，最多为 2^j-1 。

总的最少比较次数($k=\log n$)

$$\frac{n}{2}+\frac{n}{4}*2+\frac{n}{8}*4+\cdots+\frac{n}{2^j}*2^{j-1}+\cdots+\frac{n}{2^k}*2^{k-1}$$

$$=\frac{n}{2}+\frac{n}{2}+\frac{n}{2}+\cdots+\frac{n}{2}+\cdots+\frac{n}{2}$$

$$=\frac{1}{2}nk=\frac{1}{2}n\log n$$

总的最多比较次数

$$\frac{n}{2}+\frac{n}{4}*3+\frac{n}{8}*7+\cdots+\frac{n}{2^j}*(2^j-1)+\cdots+\frac{n}{2^k}*(2^k-1)$$

$$=n*\left(\frac{1}{2}+\frac{3}{4}+\frac{7}{8}+\cdots+\frac{2^j-1}{2^j}+\cdots+\frac{2^k-1}{2^k}\right)$$

$$=n*\left(1-\frac{1}{2}+1-\frac{1}{4}+1-\frac{1}{8}+\cdots+1-\frac{1}{2^j}+\cdots+1-\frac{1}{2^k}\right)$$

$$=n*\left(k-\left(\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots+\frac{1}{2^j}+\cdots+\frac{1}{2^k}\right)\right)$$

$$=n*\left(k-\left(1-\frac{1}{2^k}\right)\right)=nk-n+\frac{n}{2^k}=nk-n+1=n\log n-n+1$$

用算法 BottomUpSort 对 n 个元素进行排序，当 n 为 2 的整数幂时，比较次数在 (n log n)/2 到 n log n－n＋1 之间。执行该算法，元素赋值次数为 2n log n。

4. 算法程序实现

```
1  #include <iostream.h>
2  #include "Merge.h"
3  void BottomUpSort(int A[],int n)
4  {
5      int i,t,s;
6      t=1;
7      while(t<n){
8          s=t;t=2*s;i=0;
9          while(i+t<=n){
10             Merge(A,i+1,i+s,i+t);
11             i=i+t;
12         }
13         if(i+s<n)Merge(A,i+1,i+s,n);
14     }
15 }
16 void main()
17 {
18     int A[]={NULL,                    //A[0]不使用
19         6,10,9,5,3,11,4,8,1,2,7
20     };
21     const int N=sizeof(A)/sizeof(int)-1;
22     BottomUpSort(A,N);
23     for(int i=1;i<=N;i++)
24         cout<<A[i]<<' ';
25     cout<<endl;
26 }
```

1.2 习题答案

1-1 令 A[1..60]＝11,12,…,70，用算法 BinarySearch 搜索下列 x 值时执行了多少比较次数？