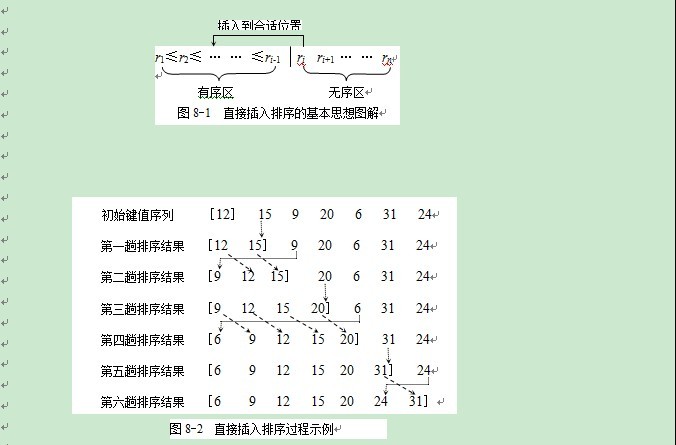
排序技术有很多种，下面简单介绍一下几种。

**一  插入排序**

**1.1  直接插入排序**

基本思想：每次将一个待排序额记录按其关键码的大小插入到一个已经排好序的有序序列中，直到全部记录排好序。

图解：



代码实现：

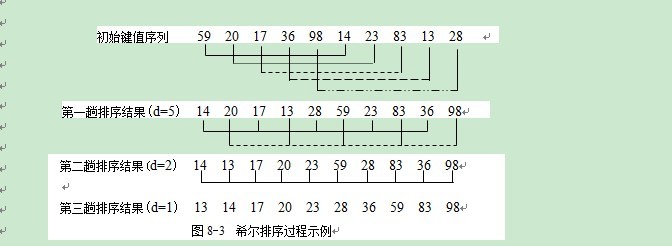
**[cpp]** [view plain](http://blog.csdn.net/ggxxkkll/article/details/8667429) [copy](http://blog.csdn.net/ggxxkkll/article/details/8667429)

1. //直接顺序排序
2. **void** InsertSort(**int** r[], **int** n)
3. {
4. **for** (**int** i=2; i<n; i++)
5. {
6. r[0]=r[i];                        //设置哨兵
7. **for** (**int** j=i-1; r[0]<r[j]; j--)   //寻找插入位置
8. r[j+1]=r[j];                //记录后移
9. r[j+1]=r[0];
10. }
11. **for**(**int** k=1;k<n;k++)
12. cout<<r[k]<<" ";
13. cout<<"\n";
14. }

**1.2 希尔排序**

基本思想是： 先将整个待排序记录序列分割成若干个子序列，在在序列内分别进行直接插入排序，待整个序列基本有序时，再对全体记录进行一次直接插入排序。

**图解：**

****

**代码实现：**

**[cpp]** [view plain](http://blog.csdn.net/ggxxkkll/article/details/8667429) [copy](http://blog.csdn.net/ggxxkkll/article/details/8667429)

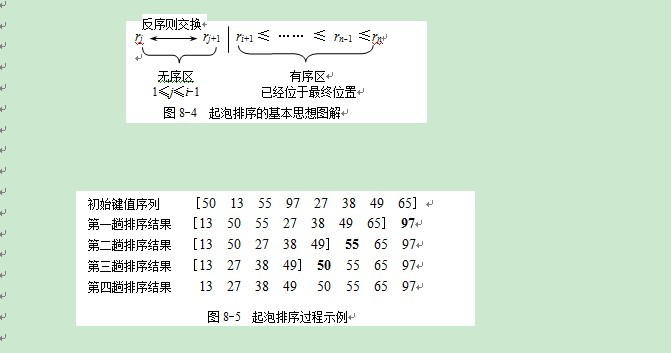
1. <span style="font-size:14px;">//希尔排序
2. **void** ShellSort(**int** r[], **int** n)
3. {
4. **int** i;
5. **int** d;
6. **int** j;
7. **for** (d=n/2; d>=1; d=d/2)            //以增量为d进行直接插入排序
8. {
9. **for** (i=d+1; i<n; i++)
10. {
11. r[0]=r[i];                 //暂存被插入记录
12. **for** (j=i-d; j>0 && r[0]<r[j]; j=j-d)
13. r[j+d]=r[j];       //记录后移d个位置
14. r[j+d]=r[0];
15. }
16. }
17. **for**(i=1;i<n;i++)
18. cout<<r[i]<<" ";
19. cout<<"\n";
20. }</span>

**二 交换排序**

**2.1 起泡排序**

起泡排序是交换排序中最简单的排序方法，其基本思想是： 两两比较相邻记录的关键码，如果反序则交换，直到没有反序的记录为止。

图解：



代码实现：

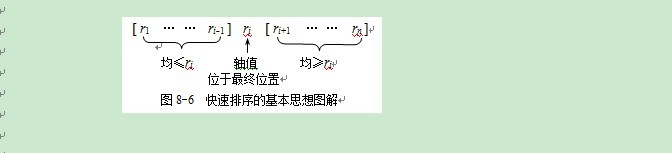
**[cpp]** [view plain](http://blog.csdn.net/ggxxkkll/article/details/8667429) [copy](http://blog.csdn.net/ggxxkkll/article/details/8667429)

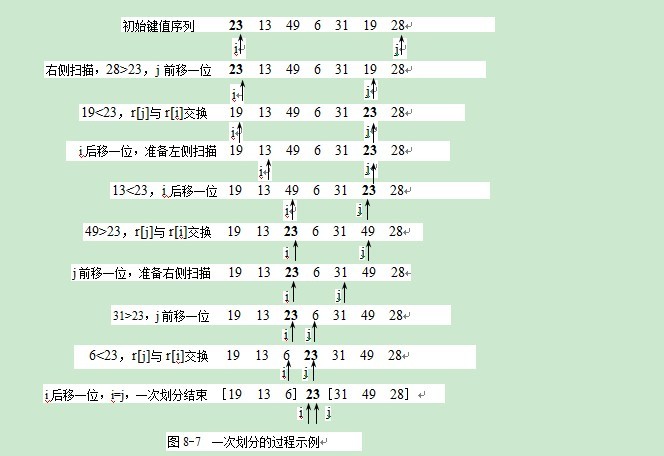
1. <span style="font-size:14px;">//起泡排序
2. **void** BubbleSort(**int** r[], **int** n)
3. {
4. **int** temp;
5. **int** exchange;
6. **int** bound;
7. exchange=n-1;                       //第一趟起泡排序的范围是r[0]到r[n-1]
8. **while** (exchange)                    //仅当上一趟排序有记录交换才进行本趟排序
9. {
10. bound=exchange;
11. exchange=0;
12. **for** (**int** j=0; j<bound; j++)     //一趟起泡排序
13. **if** (r[j]>r[j+1])
14. {
15. temp=r[j];
16. r[j]=r[j+1];
17. r[j+1]=temp;
18. exchange=j;                   //记录每一次发生记录交换的位置
19. }
20. }
21. **for**(**int** i=0;i<n;i++)
22. cout<<r[i]<<" ";
23. cout<<"\n";
24. }</span>

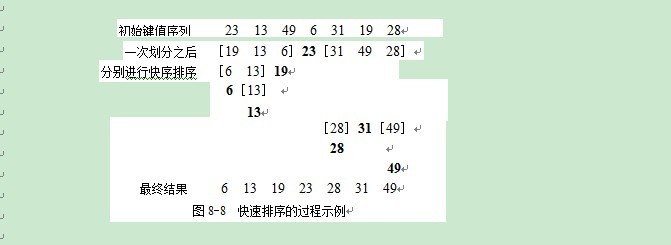
**2.2快速排序**

基本思想:通过一趟[排序](http://baike.baidu.com/view/58783.htm)将要排序的[数据分割](http://baike.baidu.com/view/4466818.htm)成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以[递归](http://baike.baidu.com/view/96473.htm)进行，以此达到整个数据变成有序序列。

图解：







代码实现：

**[cpp]** [view plain](http://blog.csdn.net/ggxxkkll/article/details/8667429) [copy](http://blog.csdn.net/ggxxkkll/article/details/8667429)

1. //快速排序一次划分
2. **int** Partition(**int** r[], **int** first, **int** end)
3. {
4. **int** i=first;                        //初始化
5. **int** j=end;
6. **int** temp;
8. **while** (i<j)
9. {
10. **while** (i<j && r[i]<= r[j])
11. j--;                        //右侧扫描
12. **if** (i<j)
13. {
14. temp=r[i];                 //将较小记录交换到前面
15. r[i]=r[j];
16. r[j]=temp;
17. i++;
18. }
19. **while** (i<j && r[i]<= r[j])
20. i++;                         //左侧扫描
21. **if** (i<j)
22. {
23. temp=r[j];
24. r[j]=r[i];
25. r[i]=temp;                //将较大记录交换到后面
26. j--;
27. }
28. }
29. **return** i;                           //i为轴值记录的最终位置
30. }
32. //快速排序
33. **void** QuickSort(**int** r[], **int** first, **int** end)
34. {
35. **if** (first<end)
36. {                                   //递归结束
37. **int** pivot=Partition(r, first, end);  //一次划分
38. QuickSort(r, first, pivot-1);//递归地对左侧子序列进行快速排序
39. QuickSort(r, pivot+1, end);  //递归地对右侧子序列进行快速排序
40. }
42. }

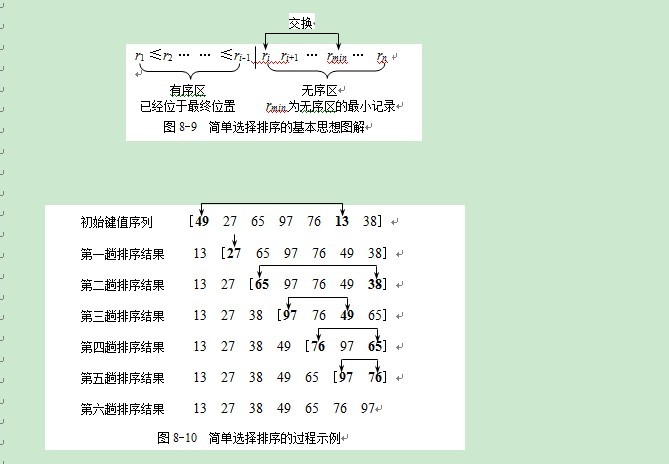
**三 选择排序**

**3.1 简单选择排序**

基本思想：设所排序序列的记录个数为n。i取1,2,…,n-1,从所有n-i+1个记录（Ri,Ri+1,…,Rn）中找出排序码最小的记录，与第i个记录交换。执行n-1趟

后就完成了记录序列的排序。

图解：



代码实现：

**[cpp]** [view plain](http://blog.csdn.net/ggxxkkll/article/details/8667429) [copy](http://blog.csdn.net/ggxxkkll/article/details/8667429)

1. //简单选择排序
2. **void** SelectSort(**int** r[ ], **int** n)
3. {
4. **int** i;
5. **int** j;
6. **int** index;
7. **int** temp;
8. **for** (i=0; i<n-1; i++)                //对n个记录进行n-1趟简单选择排序
9. {
10. index=i;
11. **for** (j=i+1; j<n; j++)            //在无序区中选取最小记录
12. **if** (r[j]<r[index])
13. index=j;
14. **if** (index!=i)
15. {
16. temp=r[i];
17. r[i]=r[index];
18. r[index]=temp;
19. }
20. }
21. **for**(i=0;i<n;i++)
22. cout<<r[i]<<" ";
23. cout<<"\n";
24. }

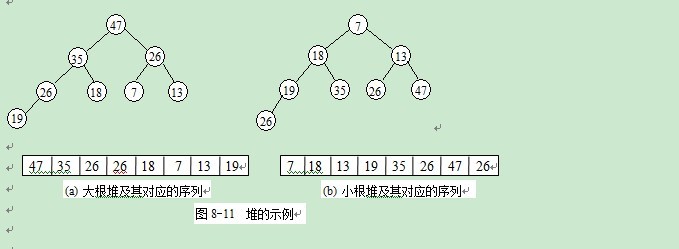
**3.2 堆排序**

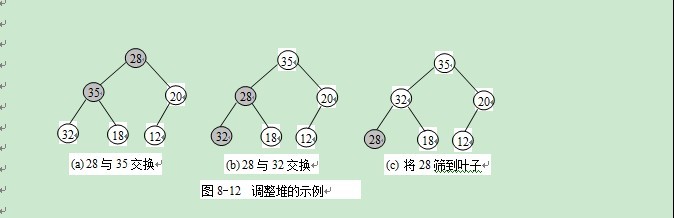
**堆的定义**

**堆**是具有下列性质的完全二叉树：每个结点的值都小于或等于其左右孩子结点的值（**小根堆**）；或者每个结点的值都大于或等于其左右孩子结点的值（**大根堆**）。

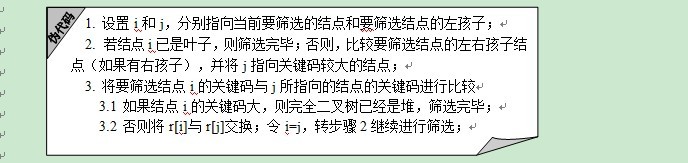
大根堆和小根堆：根结点（亦称为堆顶）的[关键字](http://baike.baidu.com/view/390935.htm)是堆里所有结点关键字中最小者的堆称为小根堆，又称[最小堆](http://baike.baidu.com/view/1251056.htm)。根结点（亦称为堆顶）的[关键字](http://baike.baidu.com/view/390935.htm)是堆里所有结点关键字中最大者，称为大根堆，又称最大堆。注意：①堆中任一子树亦是堆。②以上讨论的堆实际上是[二叉堆](http://baike.baidu.com/view/668854.htm)（Binary

Heap），类似地可定义k叉堆。





假设当前要筛选结点的编号为k，堆中最后一个结点的编号为m，并且结点k的左右子树均是堆（即r[k+1] ~ r[m]满足堆的条件），则筛选算法用伪代码可描述为：



具体的筛选代码如下：

**[cpp]** [view plain](http://blog.csdn.net/ggxxkkll/article/details/8667429) [copy](http://blog.csdn.net/ggxxkkll/article/details/8667429)

1. //筛选法调整堆
2. **void** Sift(**int** r[], **int** k, **int** m)
3. {
5. **int** i;
6. **int** j;
7. **int** temp;
8. i=k;
9. j=2\*i+1;                            //置i为要筛的结点，j为i的左孩子
10. **while** (j<=m)                          //筛选还没有进行到叶子
11. {
12. **if** (j<m && r[j]<r[j+1])
13. j++;                          //比较i的左右孩子，j为较大者
14. **if** (r[i]>r[j]) **break**;             //根结点已经大于左右孩子中的较大者
15. **else**
16. {
17. temp=r[i];
18. r[i]=r[j];
19. r[j]=temp;                   //将根结点与结点j交换
20. i=j;
21. j=2\*i+1;                     //被筛结点位于原来结点j的位置
22. }
23. }
24. }

**堆排序**

堆排序的基本思想是：首先将待排序的记录序列构造成一个堆，此时，选出了堆中所有记录的最大者即堆顶记录，然后将它从堆中移走（通常将堆顶记录和堆中最后一个记录交换），并将剩余的记录再调整成堆，这样又找出了次大的记录，以此类推，直到堆中只有一个记录为止。

（1）用大根堆排序的基本思想

① 先将初始文件R[1..n]建成一个大根堆，此堆为初始的无序区

② 再将关键字最大的记录R[1]（即堆顶）和无序区的最后一个记录R[n]交换，由此得到新的无序区R[1..n-1]和有序区R[n]，且满足R[1..n-1].keys≤R[n].key

③由于交换后新的根R[1]可能违反堆性质，故应将当前无序区R[1..n-1]调整为堆。然后再次将R[1..n-1]中关键字最大的记录R[1]和该区间的最后一个记录R[n-1]交换，由此得到新的无序区R[1..n-2]和有序区R[n-1..n]，且仍满足关系R[1..n-2].keys≤R[n-1..n].keys，同样要将R[1..n-2]调整为堆。

……

直到无序区只有一个元素为止。

（2）大根堆排序算法的基本操作：

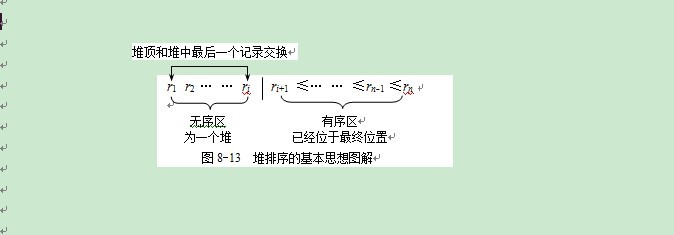
① 初始化操作：将R[1..n]构造为初始堆；

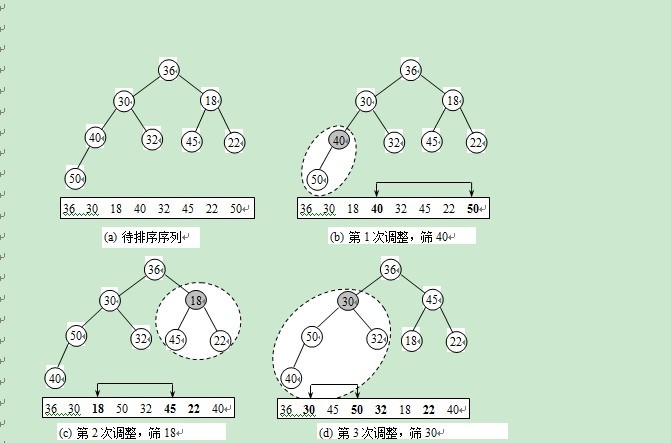
② 每一趟排序的基本操作：将当前无序区的堆顶记录R[1]和该区间的最后一个记录交换，然后将新的无序区调整为堆（亦称重建堆）。

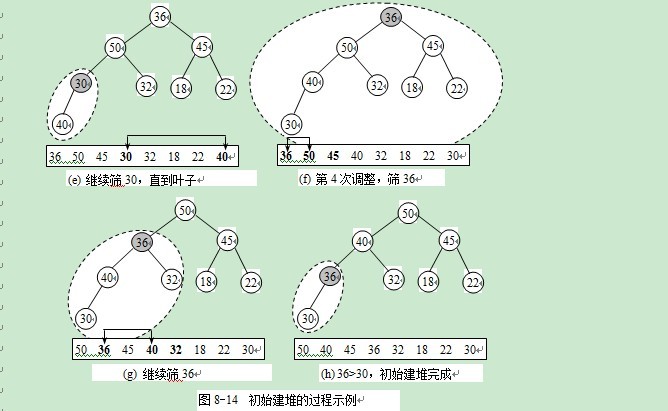
注意：

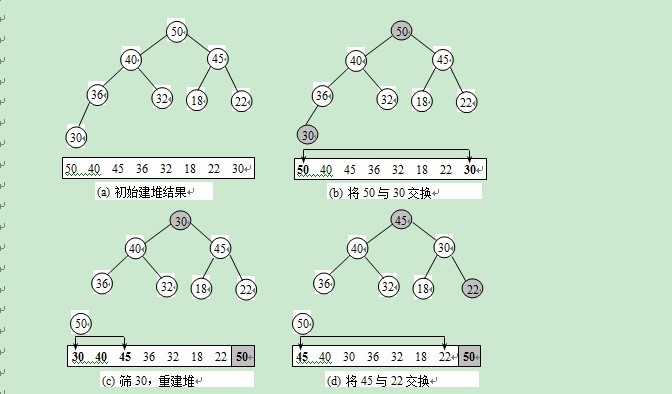
①只需做n-1趟排序，选出较大的n-1个[关键字](http://baike.baidu.com/view/390935.htm)即可以使得文件递增有序。

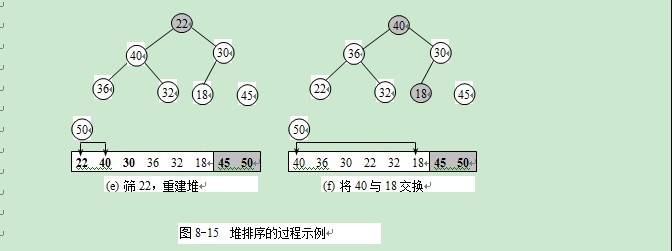
②用小根堆排序与利用大根堆类似，只不过其排序结果是递减有序的。堆排序和直接[选择排序](http://baike.baidu.com/view/547263.htm)相反：在任何时刻堆排序中无序区总是在有序区之前，且有序区是在原向量的尾部由后往前逐步扩大至整个向量为止











代码实现：

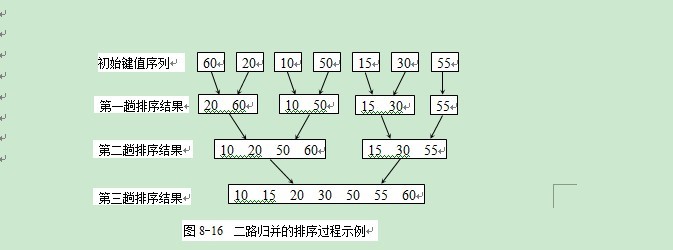
**[cpp]** [view plain](http://blog.csdn.net/ggxxkkll/article/details/8667429) [copy](http://blog.csdn.net/ggxxkkll/article/details/8667429)

1. //堆排序
2. **void** HeapSort(**int** r[ ], **int** n)
3. {
5. **int** i;
6. **int** temp;
7. **for** (i=n/2; i>=0; i--)                //初始建堆，从最后一个非终端结点至根结点
8. Sift(r, i, n) ;
9. **for** (i=n-1; i>0; i--)                //重复执行移走堆顶及重建堆的操作
10. {
11. temp=r[i];
12. r[i]=r[0];
13. r[0]=temp;
14. Sift(r, 0, i-1);
15. }
16. **for**(i=0;i<n;i++)
17. cout<<r[i]<<" ";
18. cout<<"\n";
19. }

**四 归并排序**

二路归并排序

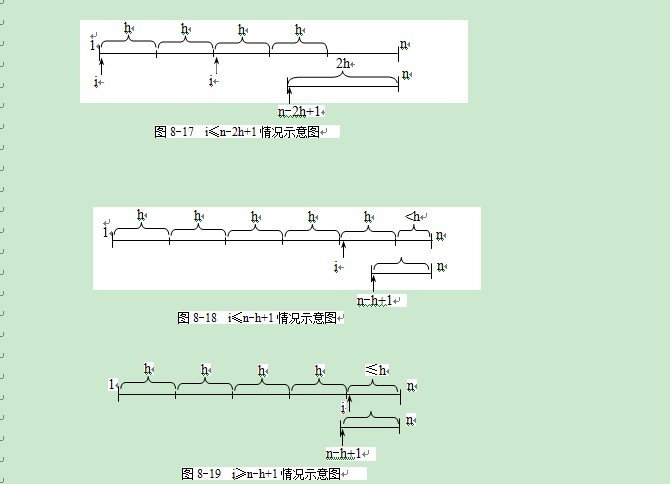
基本思想：将若干个有序序列进行两两归并，直至所有待排序记录都在一个有序序列为止。



一路归并算法实现：

**[cpp]** [view plain](http://blog.csdn.net/ggxxkkll/article/details/8667429) [copy](http://blog.csdn.net/ggxxkkll/article/details/8667429)

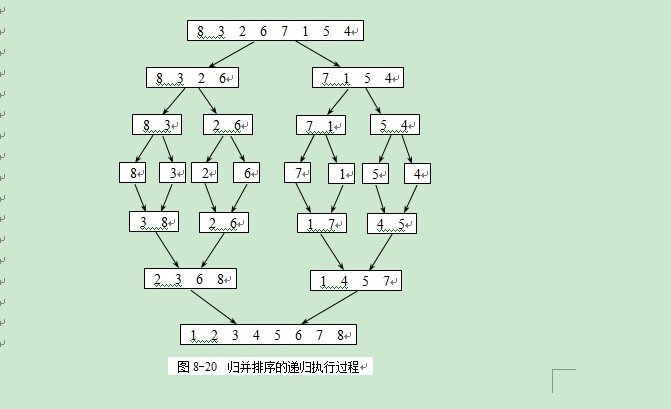
1. //一次归并
2. **void** Merge(**int** r[], **int** r1[], **int** s, **int** m, **int** t)
3. {
5. **int** i=s;
6. **int** j=m+1;
7. **int** k=s;
9. **while** (i<=m && j<=t)
10. {
11. **if** (r[i]<=r[j])
12. r1[k++]=r[i++];            //取r[i]和r[j]中较小者放入r1[k]
13. **else**
14. r1[k++]=r[j++];
15. }
16. **if** (i<=m)
17. **while** (i<=m)                  //若第一个子序列没处理完，则进行收尾处理
18. r1[k++]=r[i++];
19. **else**
20. **while** (j<=t)                  //若第二个子序列没处理完，则进行收尾处理
21. r1[k++]=r[j++];
22. }



**[cpp]** [view plain](http://blog.csdn.net/ggxxkkll/article/details/8667429) [copy](http://blog.csdn.net/ggxxkkll/article/details/8667429)

1. //一趟归并
2. **void** MergePass(**int** r[ ], **int** r1[ ], **int** n, **int** h)
3. {
4. **int** i=0;
5. **int** k;
7. **while** (i<=n-2\*h)                     //待归并记录至少有两个长度为h的子序列
8. {
9. Merge(r, r1, i, i+h-1, i+2\*h-1);
10. i+=2\*h;
11. }
12. **if** (i<n-h)
13. Merge(r, r1, i, i+h-1, n);       //待归并序列中有一个长度小于h
14. **else** **for** (k=i; k<=n; k++)            //待归并序列中只剩一个子序列
15. r1[k]=r[k];
16. }
18. //归并排序的非递归算法
19. **void** MergeSort1(**int** r[ ], **int** r1[ ], **int** n )
20. {
21. **int** h=1;
22. **int** i;
24. **while** (h<n)
25. {
26. MergePass(r, r1, n-1, h);           //归并
27. h=2\*h;
28. MergePass(r1, r, n-1, h);
29. h=2\*h;
30. }
31. **for**(i=0;i<n;i++)
32. cout<<r[i]<<" ";
33. cout<<"\n";
34. }

下面看看二路归并排序的递归实现



**[cpp]** [view plain](http://blog.csdn.net/ggxxkkll/article/details/8667429) [copy](http://blog.csdn.net/ggxxkkll/article/details/8667429)

1. //归并排序的递归算法
2. **void** MergeSort2(**int** r[], **int** r1[], **int** r2[],**int** s, **int** t)
3. {
5. **int** m;
6. **if** (s==t)
7. {
8. r1[s]=r[s];
10. }
11. **else**
12. {
13. m=(s+t)/2;
14. MergeSort2(r, r2, r1, s, m);        //归并排序前半个子序列
15. MergeSort2(r, r2, r1, m+1, t);      //归并排序后半个子序列
16. Merge(r2, r1, s, m, t);             //将两个已排序的子序列归并
17. }
18. }

**总结**

各种排序方法的比较（未完待续）：

