第八章　排序技术

8-5　二路归并排序

讲什么？

- 二路归并递归/非递归排序的基本思想
- 一次合并的算法
- 二路归并排序的递归算法
- 一趟二路归并的算法
- 二路归并排序的非递归算法
- 二路归并排序的时空性能、稳定性

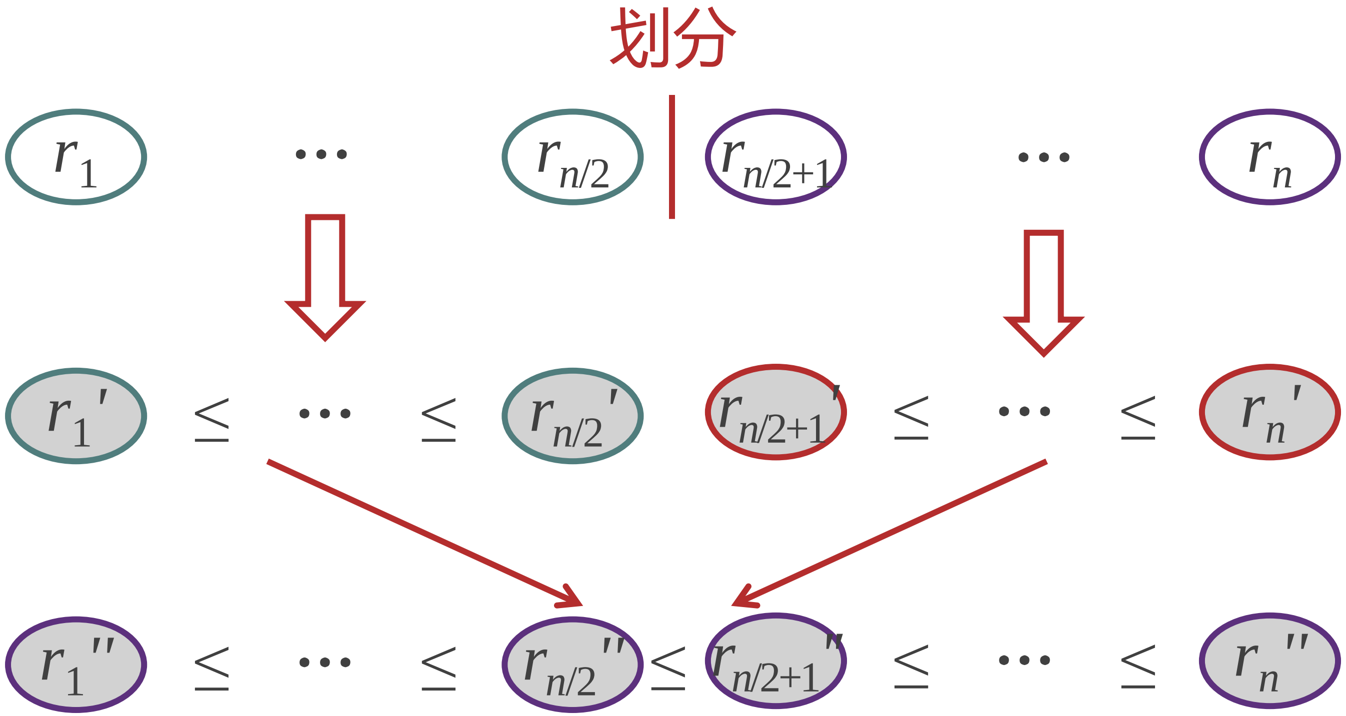
归并算法

- 二路归并递归算法
- 二路归并非递归算法

二路归并排序的递归算法

基本思想

二路归并递归排序(自顶向下)的基本思想：将待排序序列划分为两个长度相等的子序列，分别对这两个子序列进行排序，得到两个有序子序列，再将这两个有序子序列合并成一个有序序列。



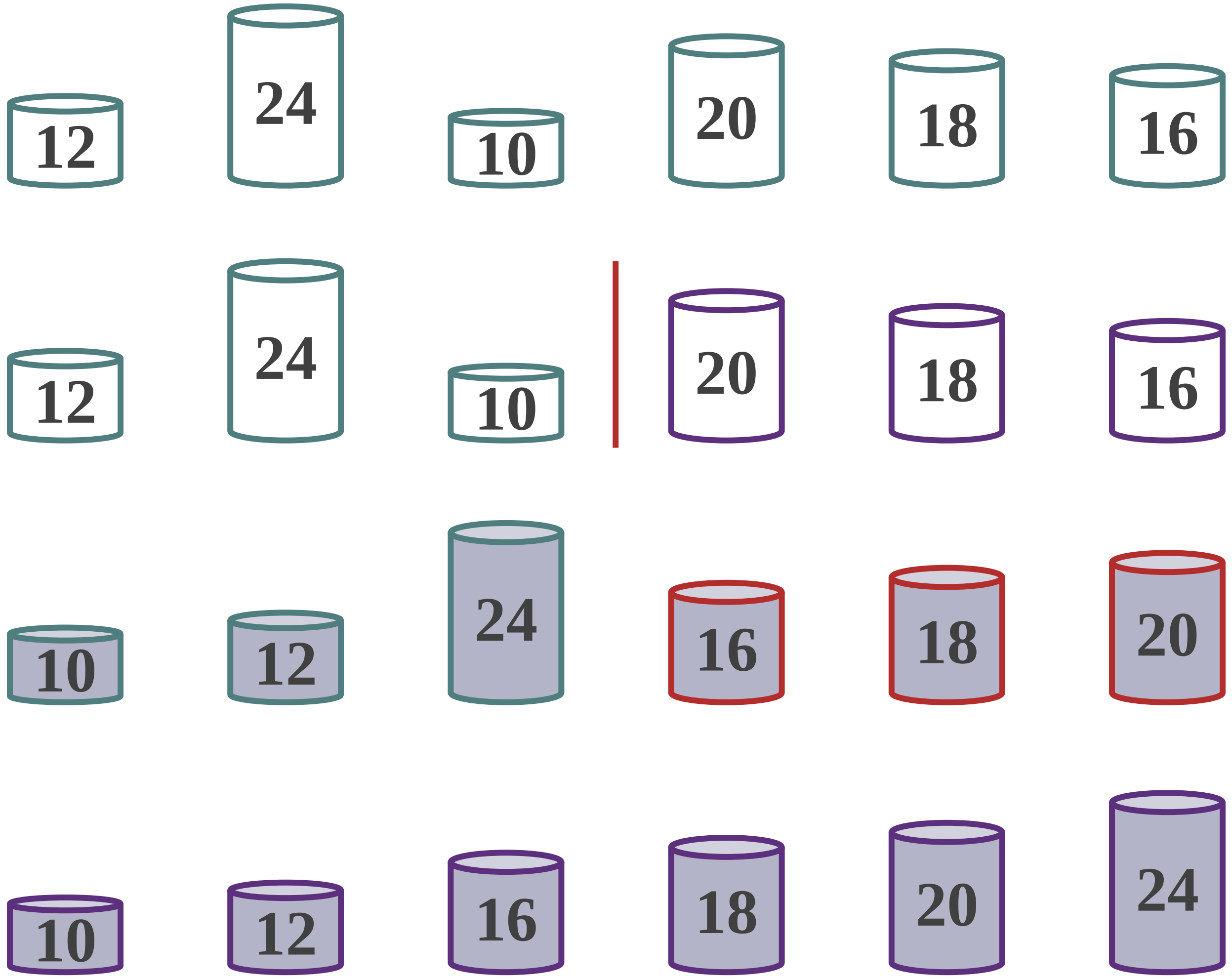
运行实例

待排序序列

划分

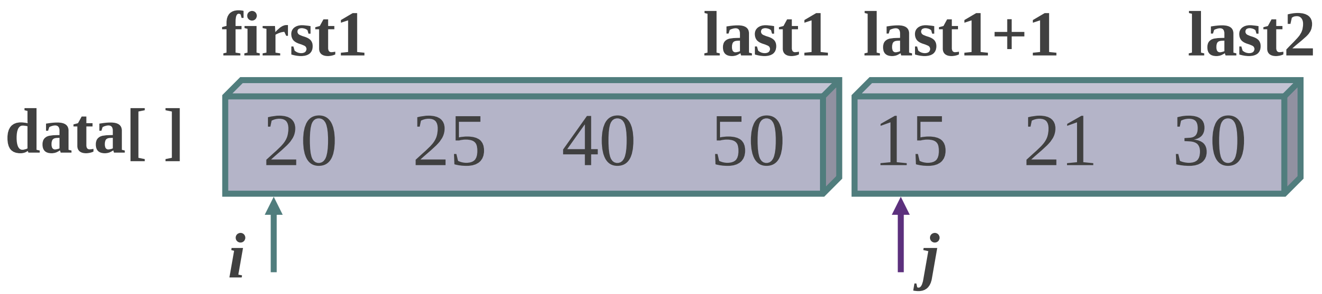
分别排序

合并



关键问题

一次合并：合并两个相邻的有序子序列



如何表示相邻子序列？

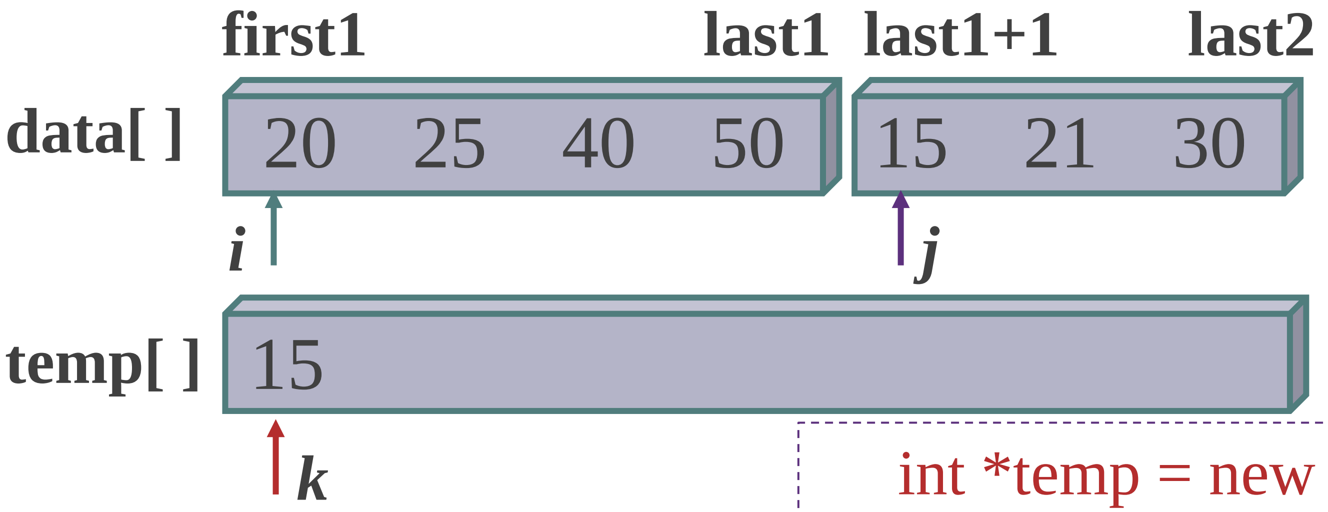
算法描述：

```
void Sort :: Merge(int first1, int last1, int last2)
{
    int i = first1, j = last1 + 1;


}
```

关键问题

一次合并：合并两个相邻的有序子序列



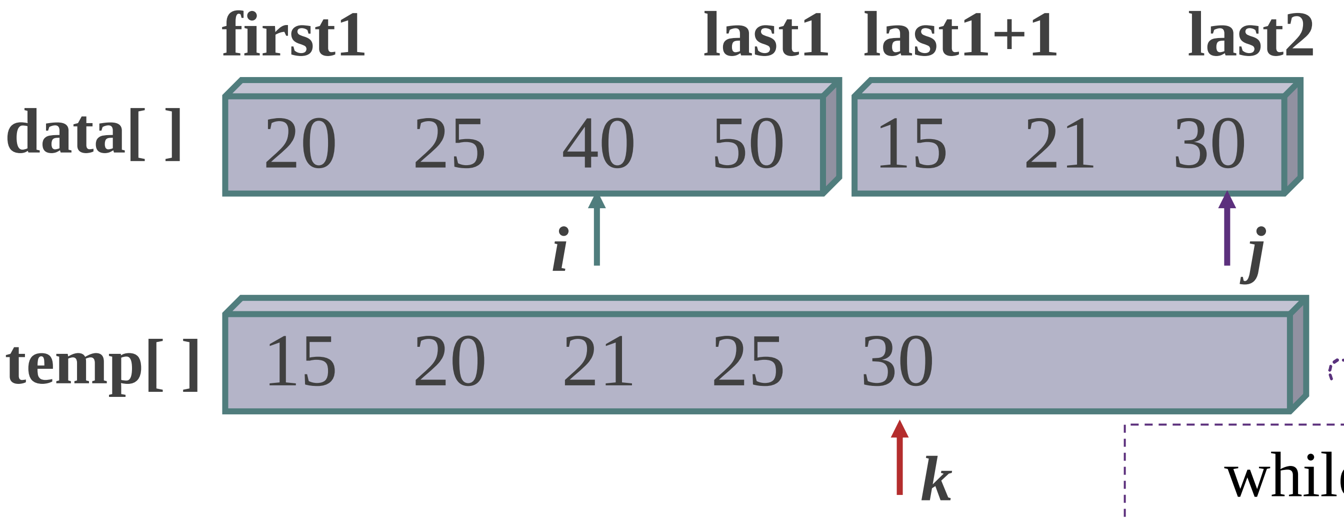
算法描述：

```
int *temp = new int[length];
int i = first1, j = last1 + 1, k = first1;
while (i <= last1 && j <= last2)
{
    if (data[i] <= data[j]) temp[k++] = data[i++];
    else  temp[k++] = data[j++];
}
```

合并可以就地进行吗？

关键问题

一次合并：合并两个相邻的有序子序列的过程。



算法描述：

```
while (i <= last1)
    temp[k++] = data[i++];
while (j <= last2)
    temp[k++] = data[j++];
for (i = first1; i <= last2; i++)
    data[i] = temp[i];
delete[ ] temp;
```

某个子序列比较完毕，做什么？

一次合并的结果在哪里？

算法描述

```
void Sort::Merge(int first1, int last1, int last2)
{
    int *temp = new int[length];
    int i = first1, j = last1 + 1, k = first1;
    while (i <= last1 && j <= last2)
    {
        if (data[i] <= data[j])
            temp[k++] = data[i++];
        else
            temp[k++] = data[j++];
    }
    while (i <= last1)  temp[k++] = data[i++];
    while (j <= last2)  temp[k++] = data[j++];
    for (i = first1; i <= last2; i++)
        data[i] = temp[i];
    delete[ ] temp;
}
```

时空性能

时间复杂度是多少?

将数组扫描一趟，$O(n)$

空间复杂度是多少?

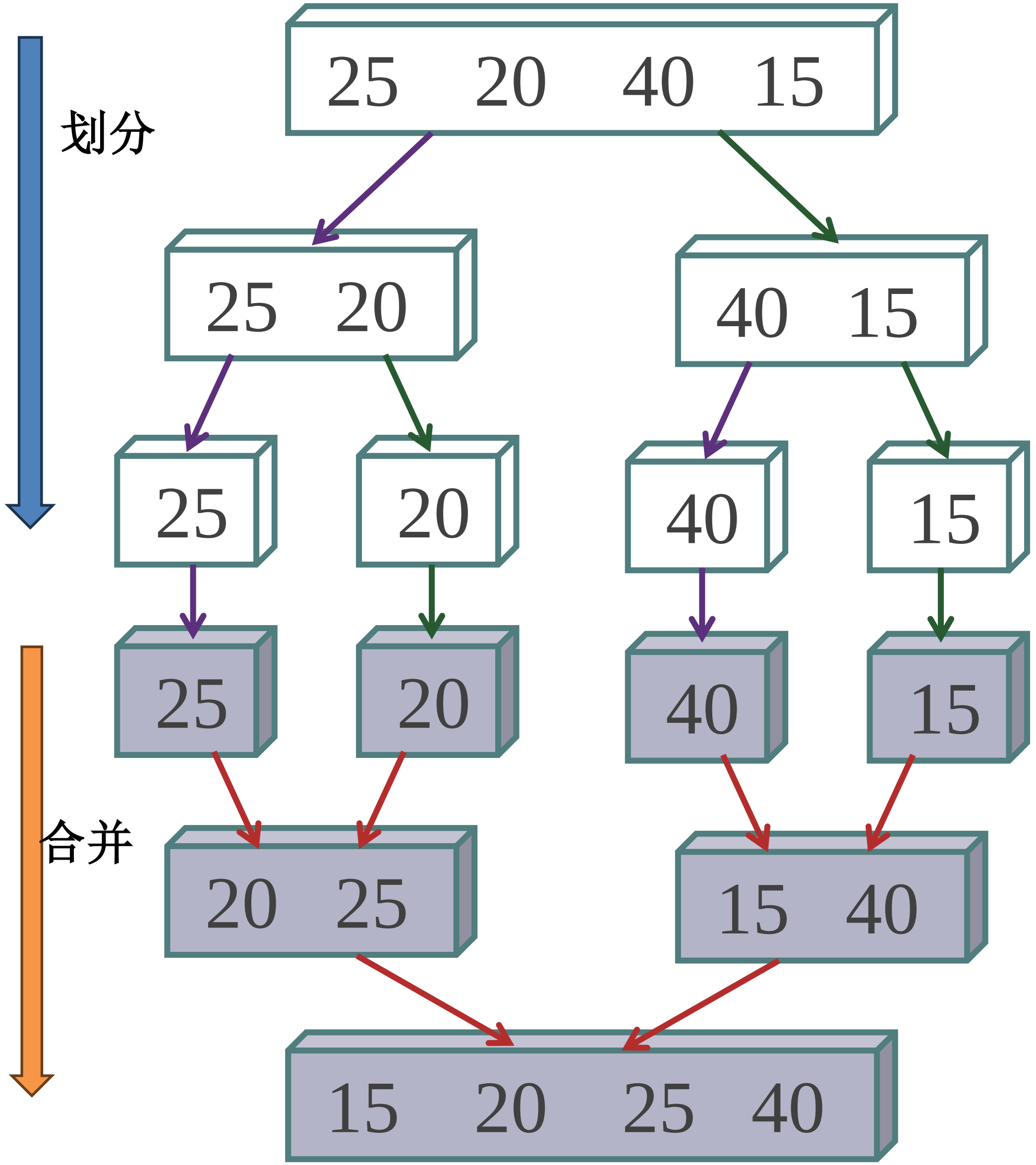
临时数组temp，$O(n)$

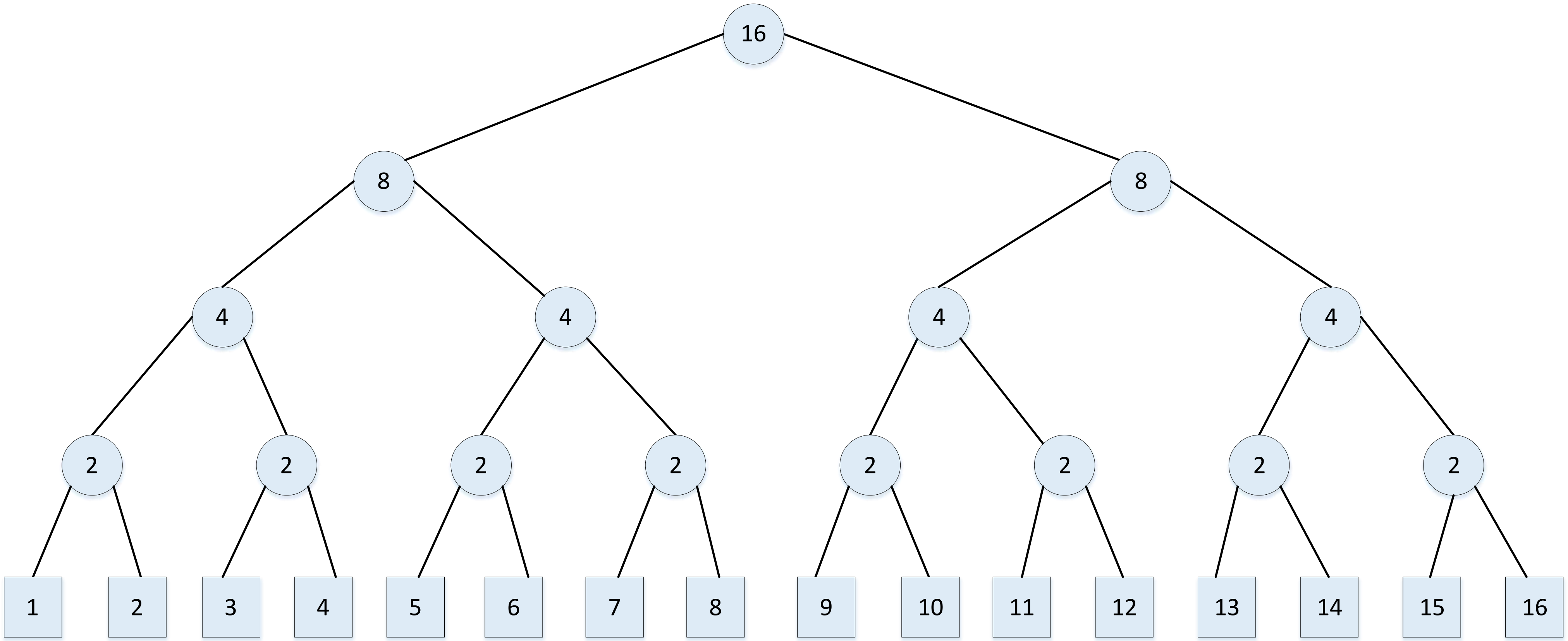
递归算法

```
void Sort::MergeSort1(int first, int last)
{
    if (first >= last) return;
    else {
        int mid = (first + last)/2;
        MergeSort1(first, mid);
        MergeSort1(mid+1, last);
        Merge(first, mid, last);
    }
}
```

子序列长度有什么规律?

执行过程





- 比较发生在merge函数中；
- 最多比较次数小于两表的长度；
- 每一层上表的总长为n；
- 因此每一层上关键字的比较次数小于n；
- 除了叶子之外，调用树共有log_2n层；
- 关键字的比较次数不超过$nlog_2$n。

- 当两个有序表合并到temp表中，记录移动的次数为两个表的总长度；
- 将temp的内容写回原表，移动次数也为两个有序表总长度；
- 每一层上合并操作对应的移动次数为2n次；
- 总共移动次数约为$2nlog_2$n。

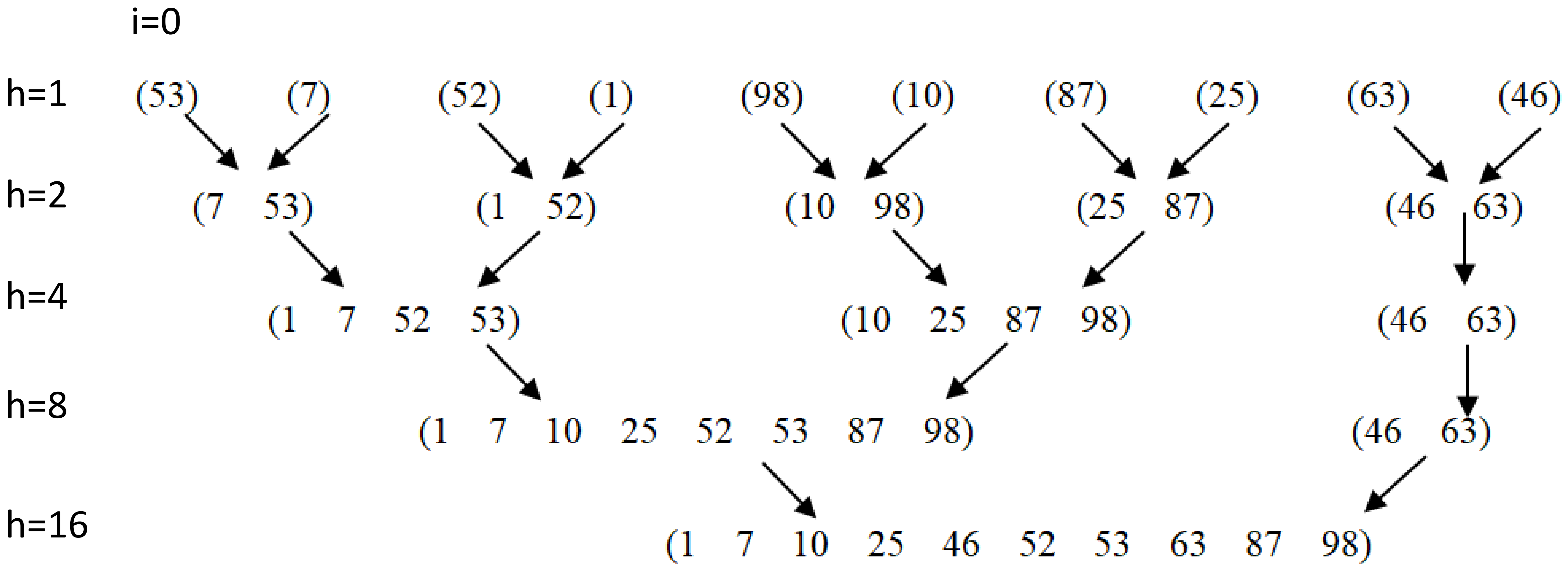
第八章　排序技术

二路归并排序的非递归实现（选学）

非递归归并-自底向上归并排序

- 将待排序记录data[0]~data[n-1]看成是n个长度为1的有序子表，把这些子表依次两两归并，得到约n/2个有序的子表，然后，再把这约n/2个有序的子表两两归并，如此重复，直到得到1个长度为n的有序表为止。

自底向上归并排序

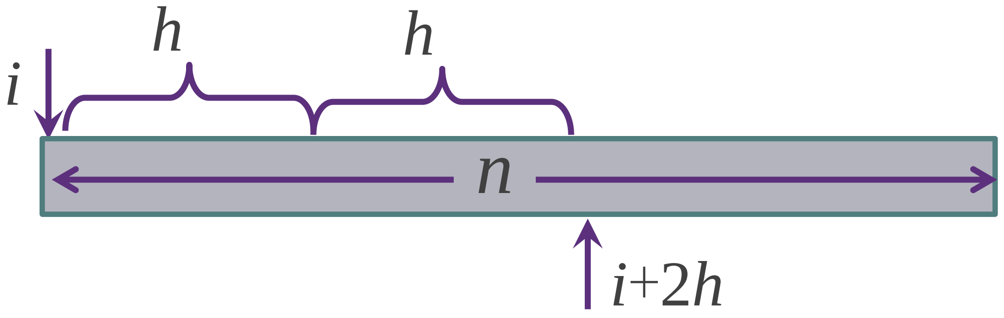


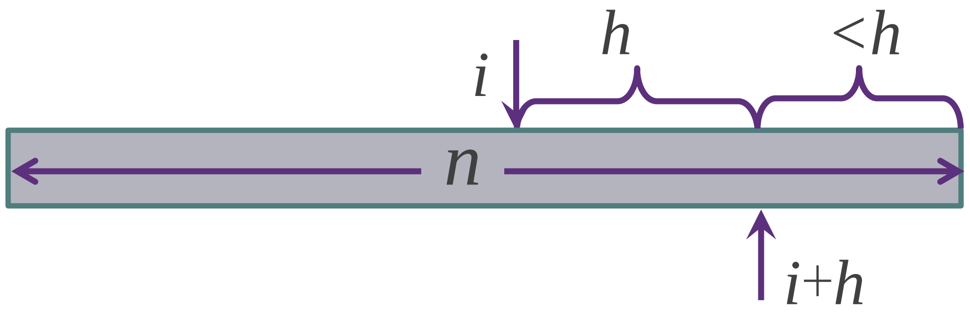
归并趟数（二叉树高度）：$\log_2 n$ 每层待合并的序列长度h，h的规律

一趟归并

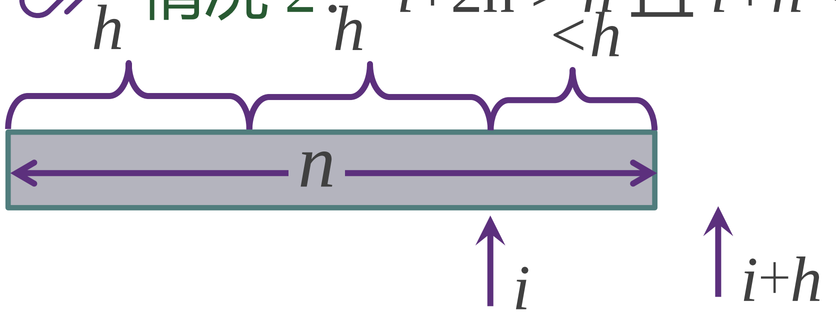
设 i 指向待归并序列的第一个记录，归并的步长是 $2h$

情况 1：$i+2h \le n$，子序列的长度均为 h





情况 2：$i+2h > n$ 且 $i+h < n$，一个长度为 h，另一个长度小于 h



情况 3：只有一个子序列

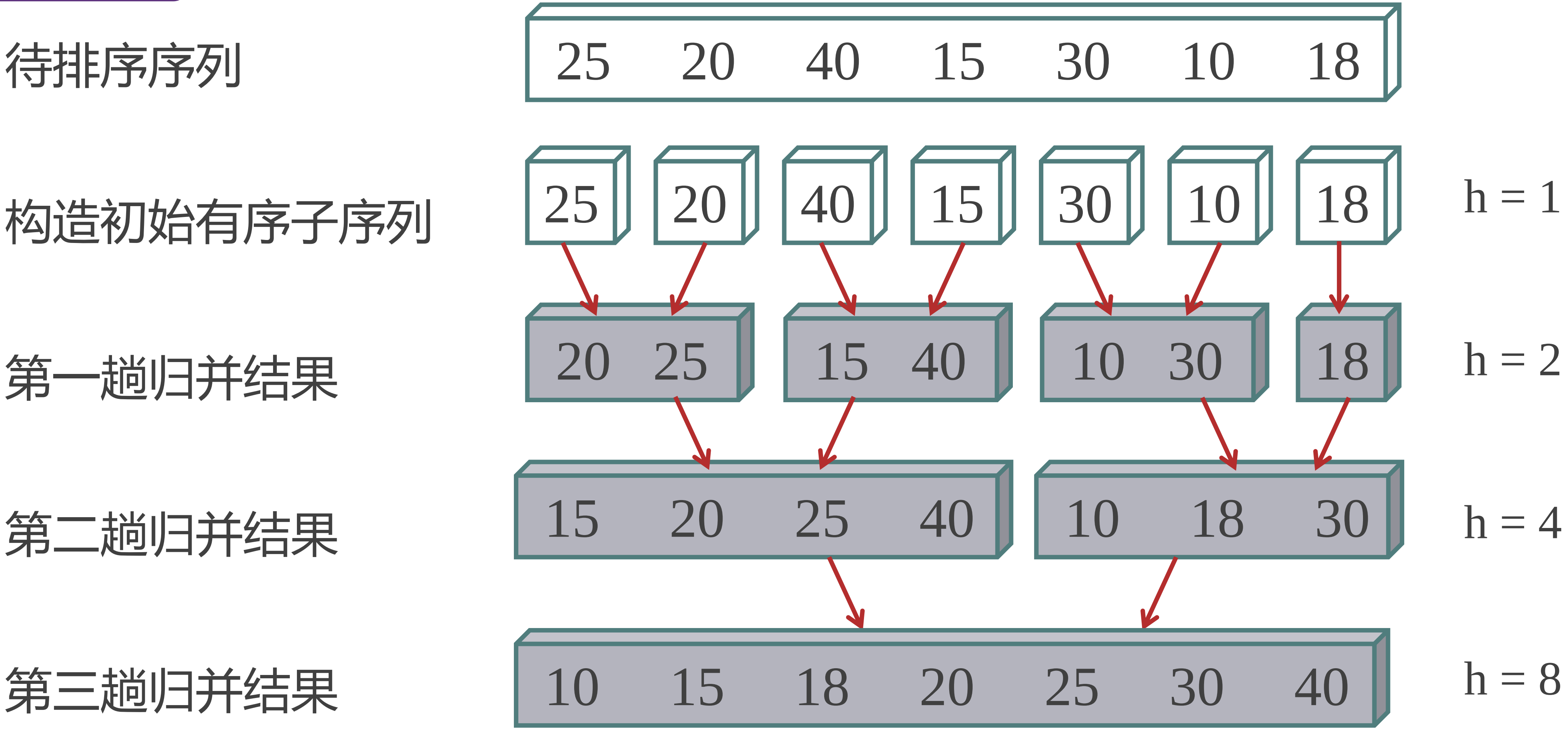
算法描述：

```
while (i + 2 * h <= length)
{
    Merge(i, i+h−1, i+2*h−1);
    i = i + 2 * h;
}
```

```
if (i + h < length)
    Merge(i, i+h−1, length−1);
```

```
/********************
   一趟归并算法
********************/
void Sort::MergePass(int h)
{
        int i = 0;
        while (i + 2 * h <= length)        //待归并记录有两个长度为h的子序列
        {
                Merge(i, i+h-1, i+2*h-1);
                i = i + 2 * h;
        }
        if (i + h < length)
                Merge(i, i+h-1, length-1);    //两个子序列一个长度小于h
}
```

执行过程



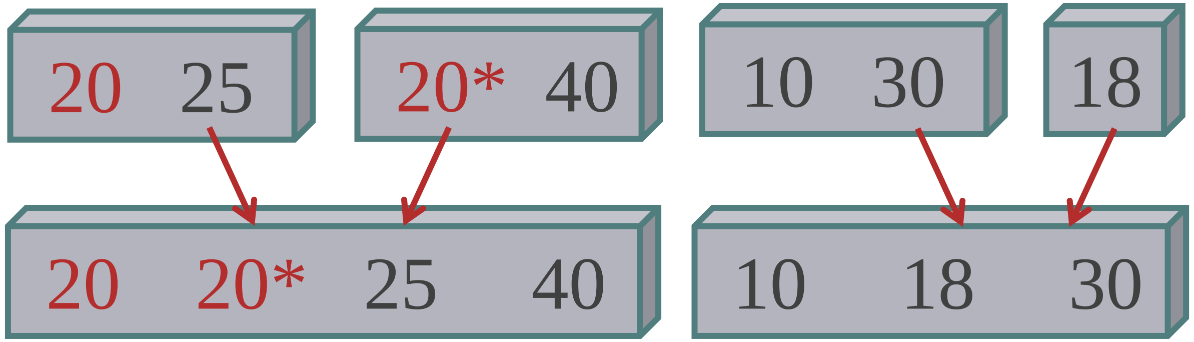
如何控制二路归并的结束？子序列长度有什么规律？

算法描述

```
void Sort :: MergeSort2( )
{
    int h = 1;
    while (h < length)
    {
        MergePass(h);
        h = 2 * h;
    }
}
```

时空性能

- 执行趟数：$\log_2 n$
- 每一趟：将记录扫描一遍，$O(n)$
- 最好、最坏、平均：$O(n\log_2 n)$
- 稳定性：稳定



归并排序—性能分析

- 归并排序的时间复杂度为$O(n\log_2 n)$。
- 归并排序采用**temp**列表暂存合并结果，因此空间效率为**O(n)**。
- 归并排序是稳定排序。

基数排序

基数排序

- 多关键字排序
- 链式基数排序

多关键字的排序

例如:学生记录含三个关键字:

系别、**班号**和**班内的序列号**，其中以系别为最主位关键字。

最次位优先法（least significant digit first）LSD法排序过程:

无序序列	3,1,30	1,2,15	3,1,20	2,3,18	2,1,20
对K^2排序	1,2,**15**	2,3,**18**	3,1,**20**	2,1,**20**	3,1,**30**
对K^1排序	3,**1**,20	2,**1**,20	3,**1**,30	1,**2**,15	2,**3**,18
对K^0排序	**1**,2,15	**2**,1,20	**2**,3,18	**3**,1,20	**3**,1,30

基数排序

多关键字的记录序列中，如果每个关键字的取值范围相同，则按LSD法进行排序时，可以采用**“分配-收集”**的方法，其好处是不需要进行关键字间的比较。

对于数值或字符串类型的**单关键字**，可以**看成**是由多个数位或多个字符构成的**多关键字**，此时可以**采用**这种**“分配-收集”**的办法**进行排序，称为基数排序法，**也称为“桶排序”或“箱排序”。

数字是有范围的，均有0-9这十个数字组成，则只需设置十个箱子，相继按个、十、百位进行排序。

例：(614，738，921，485，637，101，215，530，790，306)

第一趟分配(按个位排)

第一趟收集

530 790 921 101 614 485 215 306 637 738

第一趟收集

530	790	921	101	614	485	215	306	637	738

第二趟分配（按十位排）

e[0]	e[1]	e[2]	e[3]	e[4]	e[5]	e[6]	e[7]	e[8]	e[9]
101	614	921	530					485	790
306	215		637						
			738						

第二趟收集

101	306	614	215	921	530	637	738	485	790

第二趟收集

101 306 614 215 921 530 637 738 485 790

第三趟分配（按百位排）

$e[0]$	$e[1]$	$e[2]$	$e[3]$	$e[4]$	$e[5]$	$e[6]$	$e[7]$	$e[8]$	$e[9]$
	101	215	306	485	530	614	738		921
						637	790		

第三趟收集

101 215 306 485 530 614 637 738 790 921

链式基数排序

在计算机上实现基数排序时，为减少所需辅助存储空间，应采用**链表作存储结构**，即链式基数排序，**具体做法为**：

1)待排序记录以指针相链，构成一个**链表**；

2)“**分配**” 时，按当前 “关键字位”所取值，将记录分配到不同的 “链队列” 中，每个队列中记录的 “关键字位” 相同；

3)“**收集**”时，按当前关键字位取值从小到大将各队列首尾相链成一个链表;

4)对每个关键字位均**重复2) 和3)** 两步。

链式基数排序

例如：

p→369→367→167→239→237→138→230→139

进行第一次分配

f[0] →230← r[0]

f[7] →367 →167 →237 ← r[7]

f[8] →138← r[8]

f[9] →369 →239 →139← r[9]

进行第一次收集

p→230 →367→167→237 →138 →368→239→139

链式基数排序

p→230→367→167→237→138→368→239→139

进行第二次分配

f[3] →230 →237 →138 →239 →139 ← r[3]

f[6] →367 →167 →368 ← r[6]

进行第二次收集

p→230→237→138→239→139 →367→167→368

链式基数排序

p→230→237→138→239→139→367→167→368

进行第三次分配

f[1] →138 →139 →167 ← r[1]

f[2] →230 →237 →239 ← r[2]

f[3] →367 →368 ← r[3]

进行第三次收集之后便得到记录的有序序列

p→138→139→167 →230→237→239 →367→368

链式基数排序

提醒：

“**分配**”和“**收集**”的实际操作仅为修改链表中的指针和设置队列的头、尾指针；

链式基数排序–性能分析

- **基数排序的时间复杂度为O(d(n+r))**

  其中：

  (1)每一趟分配的时间复杂度为O(n)，n是参与排序的元素个数;

  (2)每一趟收集的时间复杂度为O(r)， r为关键字的“基数”，如十进制数的基数为10，二进制数的基数为2，决定了需要多少个“箱子”；

  (3)d为“分配-收集”的趟数，可以等于数值关键字的位数，比如3位数字要按个、十、百进行3趟排序；或者等于排序关键字的个数，比如出生日期，可按日、月、年3趟排序，扑克牌可以先按花色然后按数字进行2趟排序等。

- **基数排序所需用的计算时间不仅与n有关，而且还与关键字的位数、关键字的基数有关。**

- r较小的情况下，链式基数排序的时间复杂度也可写作O(*d***n*)。

链式基数排序–性能分析

- 由于分配时形成了r条队列，每条队列设1个首指针和1个尾指针，因此空间效率为O(r)。
- **基数排序是稳定排序。**
- **基数排序的缺点：**
  - 需要知道关键字取值的上下界，且上下界限定的数量是有限的

第八章　排序技术

8-6　各种排序方法的比较

讲什么？

- 时间性能
- 空间性能
- 稳定性及简单性
- 记录本身的信息量
- 关键码的分布情况

时间性能

排序方法	平均情况
直接插入排序	$O(n^2)$
希尔排序	$O(n\log_2 n)\sim O(n^2)$
起泡排序	$O(n^2)$
快速排序	$O(n\log_2 n)$
简单选择排序	$O(n^2)$
堆排序	$O(n\log_2 n)$
归并排序	$O(n\log_2 n)$

从平均情况看

（1）直接插入排序、简单选择排序和起泡排序属于一类，时间复杂度为$O(n^2)$；

（2）堆排序、快速排序和归并排序属于一类，时间复杂度为$O(n\log_2 n)$；

（3）希尔排序的时间性能取决于增量序列，介于$O(n^2)$和$O(n\log_2 n)$之间。

快速排序是目前最快的一种排序方法
在待排序记录个数较多的情况下，归并排序比堆排序更快

时间性能

排序方法	最好情况
直 接 插 入 排 序	$O(n)$
希 尔 排 序	$O(n^{1.3})$
起 泡 排 序	$O(n)$
快 速 排 序	$O(n\log_2 n)$
简 单 选 择 排 序	$O(n^2)$
堆 排 序	$O(n\log_2 n)$
归 并 排 序	$O(n\log_2 n)$

从最好情况看

（1）直接插入排序和起泡排序为 $O(n)$；

（2）其他排序算法的最好情况与平均情况相同。

如果待排序序列接近正序，首选起泡排序和直接插入排序

时间性能

排序方法	最坏情况
直 接 插 入 排 序	$O(n^2)$
希 尔 排 序	$O(n^2)$
起 泡 排 序	$O(n^2)$
快 速 排 序	$O(n^2)$
简 单 选 择 排 序	$O(n^2)$
堆 排 序	$O(n\log_2 n)$
归 并 排 序	$O(n\log_2 n)$

从最坏情况看

（1）快速排序的时间复杂度为 $O(n^2)$；

（2）直接插入排序和起泡排序虽然与平均情况相同，但后者系数大约增加一倍，所以运行速度将降低一半；

（3）最坏情况对直接选择排序、堆排序和归并排序影响不大。

如果待排序序列接近正序或逆序，不使用快速排序

空间性能

排序方法	辅助空间
直接插入排序	$O(1)$
希 尔 排 序	$O(1)$
起 泡 排 序	$O(1)$
快 速 排 序	$O(\log_2 n)$ ~$O(n)$
简单选择排序	$O(1)$
堆 排 序	$O(1)$
归 并 排 序	$O(n)$

从空间性能看

（1）归并排序的空间复杂度为$O(n)$；

（2）快速排序的空间复杂度为$O(\log_2 n)$~$O(n)$；

（3）其它排序方法的空间复杂度为$O(1)$。

稳定性与简单性

乌龟插队直冒泡☺

从稳定性看

（1）稳定：包括直接插入排序、起泡排序和归并排序；

（2）不稳定：包括希尔排序、简单选择排序、快速排序和堆排序。

从算法简单性看

（1）简单算法：包括直接插入排序、简单选择排序和起泡排序，

（2）改进算法，较复杂：包括希尔排序、堆排序、快速排序和归并排序。

记录本身信息量

从记录本身信息量的大小看

记录本身信息量越大，占用的存储空间就越多，移动记录所花费的时间就越多，所以对记录的移动次数较多的算法不利。

排序方法	最好情况	最坏情况	平均情况
直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$
起泡排序	0	$O(n^2)$	$O(n^2)$
简单选择排序	0	$O(n)$	$O(n)$

记录个数不多且记录本身的信息量较大时，首选简单选择排序算法

关键码的分布

从关键码的分布看

（1）当待排序记录按关键码有序时，插入排序和起泡排序能达到$O(n)$的时间复杂度；对于快速排序而言，这是最坏情况，时间性能蜕化为$O(n^2)$；

（2）简单选择排序、堆排序和归并排序的时间性能不随记录序列中关键码的分布而改变。

各种排序算法各有优缺点，
应该根据实际情况选择合适的排序算法

时间复杂度

		最好	最差	平均	空间复杂度	
插入排序	直接插入排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
	希尔排序	$O(n)$	$O(n^2)$	$\sim O(n^{1.3})$	$O(1)$	不稳定
交换排序	冒泡排序	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	稳定
	快速排序	$O(nlogn)$	$O(n^2)$	$O(nlogn)$	$O(nlogn)$	不稳定
选择排序	直接选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
	堆排序	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$	$O(1)$	不稳定
归并排序		$O(nlogn)$	$O(nlogn)$	$O(nlogn)$	$O(n)$	稳定
基数排序 k:待排元素的维数，m为基数的个数		$O(n+m)$	$O(k*(n+m))$	$O(k*(n+m))$	$O(n+m)$	稳定

单选题 5分

设置

1. 合并两个长度为n的有序子序列，时间复杂度是O(n)，空间复杂度是O(1)。

A 正确

B 错误

提交

单选题 5分

设置

2. 归并排序执行的趟数与待排序序列的初始状态无关。

A 正确

B 错误

提交

单选题 5分

设置

3. 二路归并排序的时间性能较好，是不稳定的排序算法。

A 正确

B 错误

提交

4. 对于待排序记录序列{25, 10, 8, 20, 35, 15}，写出二路归并排序每一趟的结果。

作答

单选题 5分

设置

1. 如果待排序记录个数较多且随机排列，应该采用（ ）方法。

A 直接插入排序

B 起泡排序

C 快速排序

D 堆排序

提交

单选题 5分

设置

2. 如果待排序记录个数不多且基本有序，应该采用（　）方法。

A 直接插入排序

B 简单选择排序

C 快速排序

D 堆排序

提交

单选题 5分

设置

3. 如果不断产生待排序记录，随时需要当前记录集合的排序结果，应该采用（　）方法。

A 直接插入排序

B 起泡排序

C 快速排序

D 堆排序

提交

单选题 5分

设置

4. 如果待排序序列每个记录的存储量很大，不应该采用（　）方法。

A 直接插入排序

B 起泡排序

C 快速排序

D 简单选择排序

提交