北京交通大學
BEIJING JIAOTONG UNIVERSITY

第四章 线性表的排序

交换排序和选择排序

本章内容

- 排序的概念
- 交换排序
  - 冒泡排序
  - 快速排序
- 选择排序
  - 简单选择排序
- 插入排序
  - 直接插入排序
  - 折半插入排序
  - 2路插入排序
- 希尔排序

10.1 排序的概念

- 排序分为两大类:内部排序和外部排序
- 内部排序按排序原则分类:
  - 插入排序
  - 交换排序
  - 选择排序
  - 归并排序
  - 计数排序
- 按排序所需工作量分类:
  - 简单排序 (n^2)
  - 先进排序 (nlogn)
  - 基数排序 (d •n)

10.1 排序的概念

假设含有n个记录的序列为: $\{R_1, R_2, \cdots, R_n\}$

其相应的关键字序列为: $\{K_1, K_2, \cdots, K_n\}$

需确定1, 2, …, n的一种排列$p_1, p_2, \cdots, p_n$,使其相应的关键字满足如下的非递减(或非递增)关系

$$Kp_1 \le Kp_2 \le \cdots, \le Kp_n$$

即使序列 {R1, R2, …, Rn}成为一个关键有序的序列

$$\{ Rp_1, Rp_2, \cdots, \le Rp_n \}$$

这样一种操作称为 “排序”

10.1 排序的概念

排序算法的性能参数

- 时间复杂度
- 空间复杂度
- 排序稳定性
  - 对于关键字相等的记录，如果**排序前 Ri 领先于 Rj (i<j)，排序后 Ri 仍然领先于 Rj**，则称该算法稳定。

序号	学号	姓名	数学	语文	物理	英语
0	1004	王芸	84	70	78	77
1	1002	张鹏	75	88	92	85
2	1012	李成	90	84	66	80
3	1008	陈红	80	95	77	84
n-1	1022	楚姗	90	95	88	100

10.2 交换排序

10.2 交换排序

- **基本思想：**两两比较待排序对象的关键码，如果发生“逆序”（即排列顺序与排序后的次序正好相反），则交换之，直到所有对象都排好序为止。

  - 10.2.1 冒泡排序
  - 10.2.2 快速排序

10.2.1 冒泡排序

- 简单交换算法

```
00043: /*对 顺 序 表 L作 交 换 排 序 （ 冒 泡 排 序 初 级 版 ） */
00044: void BubbleSort0(SqList *L)
00045: {
00046:     int i,j;
00047:     for(i=1;i<L->length;i++)
00048:     {
00049:         for(j=i+1;j<=L->length;j++)
00050:         {
00051:             if(L->r[i]>L->r[j])
00052:             {
00053:                 swap(L,i,j);/*交 换 L->r[i]与 L->r[j]的 值 */
00054:             }
00055:         }
00056:     }
00057: }
```

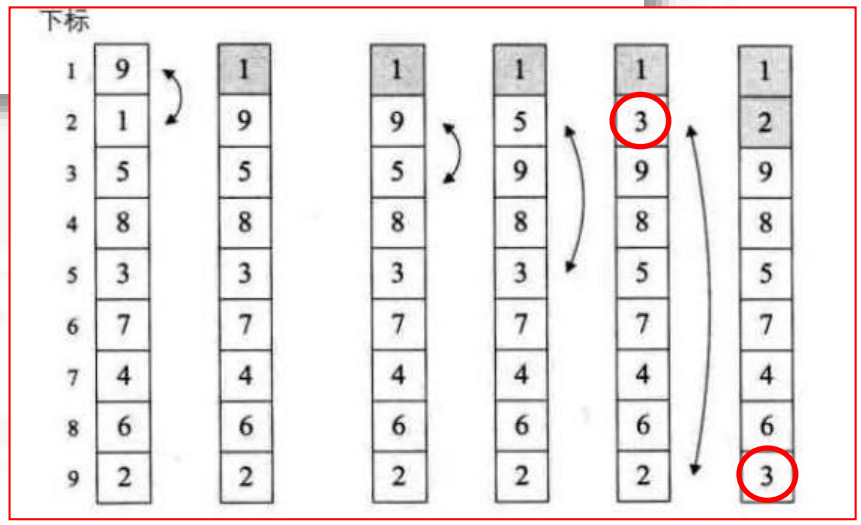
```
00026: /*交 换 L中 数 组 r的 下 标 为 i和 j的 值 */
00027: void swap(SqList *L,int i,int j)
00028: {
00029:     int temp=L->r[i];
00030:     L->r[i]=L->r[j];
00031:     L->r[j]=temp;
00032: }
```

10.2.1 冒泡排序

简单交换算法

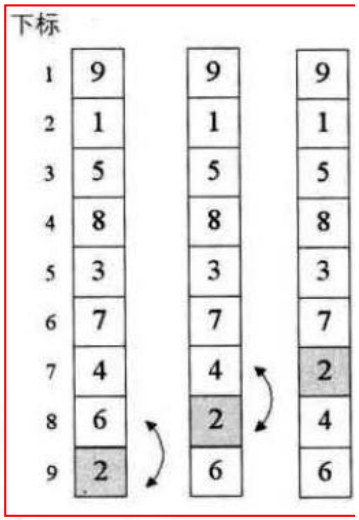
```
00043: /*对 顺 序 表 L作 交 换 排 序 （ 冒 泡 排 序 初 级 版 ） */
00044: void BubbleSort0(SqList *L)
00045: {
00046:     int i,j;
00047:     for(i=1;i<L->length;i++)
00048:     {
00049:         for(j=i+1;j<=L->length;j++)
00050:         {
00051:             if(L->r[i]>L->r[j])
00052:             {
00053:                 swap(L,i,j);/*交 换 L->r[i]与 L->r[j]的 值 */
00054:             }
00055:         }
00056:     }
00057: }
```

?



10.2.1 冒泡排序

- 从底向上



10.2.1 冒泡排序

- 冒泡算法

```
00043: /*对 顺 序 表 L作 交 换 排 序 （ 冒 泡 排 序 初 级 版 ） */
00044: void BubbleSort0(SqList *L)
00045: {
00046:     int i,j;
00047:     for(i=1;i<L->length;i++)
00048:     {
00049:         for(j=i+1;j<=L->length;j++)
00050:         {
00051:             if(L->r[i]>L->r[j])
00052:             {
00053:                 swap(L,i,j);/*交 换 L->r
00054:             }
00055:         }
00056:     }
00057: }
```

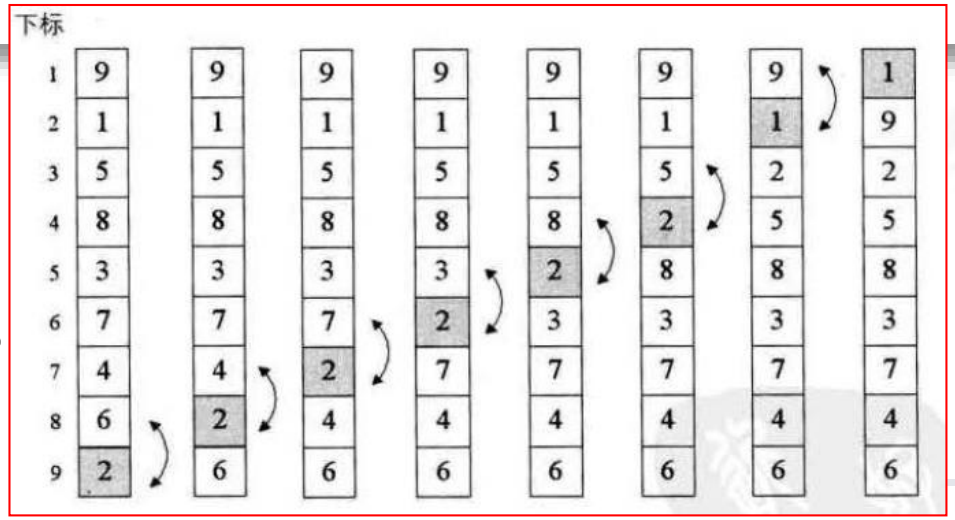
```
00060: void BubbleSort(SqList *L)
00061: {
00062:     int i,j;
00063:     for(i=1;i<L->length;i++)
00064:     {
00065:         for(j=L->length-1;j>=i;j--)
00066:         {
00067:             if(L->r[j]>L->r[j+1]) /
00068:             {
00069:                 swap(L,j,j+1);/*交
00070:             }
00071:         }
00072:     }
00073: }
```

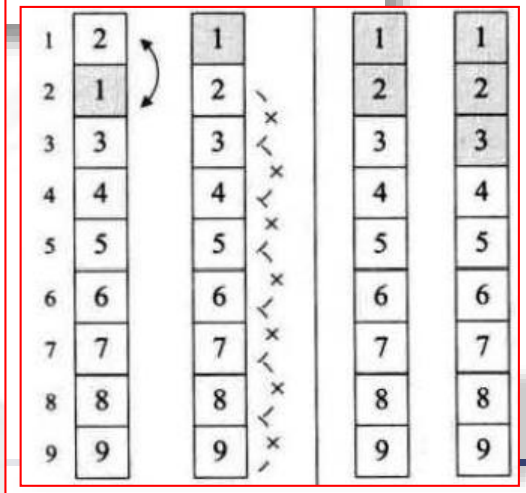
10.2.1 冒泡排序

冒泡算法的问题

```
00059: /*对 顺 序 表 L作 冒 泡 排 序 */
00060: void BubbleSort(SqList *L)
00061: {
00062:     int i,j;
00063:     for(i=1;i<L->length;i++)
00064:     {
00065:         for(j=L->length-1;j>=i;j--) /*注 意 j是 从 后 往 前 循 环 */
00066:         {
00067:             if(L->r[j]>L->r[j+1]) /*若 前 者 大 于 后 者 （ 注 意 这 里 与
00068:             {
00069:                 swap(L,j,j+1);/*交 换 L->r[j]与 L->r[j+1]的 值 */
00070:             }
00071:         }
00072:     }
00073: }
```

?





10.2.1 冒泡排序

- 带交换标记的起泡算法

```
00059: /*对 顺 序 表 L作 冒 泡 排 序 */
00060: void BubbleSort(SqList *L)
00061: {
00062:     int i,j;
00063:     for(i=1;i<L->length;i++)
00064:     {
00065:         for(j=L->length-1;j>=i;j--)  /*注
00066:         {
00067:             if(L->r[j]>L->r[j+1]) /*若
00068:             {
00069:                 swap(L,j,j+1);/*交 换
00070:             }
00071:         }
00072:     }
00073: }
```

```
00076: void BubbleSort2(SqList *L)
00077: {
00078:     int i,j;
00079:     Status flag=TRUE;           /*fl
00080:     for(i=1;i<L->length && flag;i++)
00081:     {
00082:         flag=FALSE;             /*初
00083:         for(j=L->length-1;j>=i;j--)
00084:         {
00085:             if(L->r[j]>L->r[j+1])
00086:             {
00087:                 swap(L,j,j+1); /*交
00088:                 flag=TRUE;     /*如
00089:             }
00090:         }
00091:     }
00092: }
```

10.2.1 冒泡排序

基本思路： 每趟不断将记录两两比较，并按“前小后大”（或“前大后小”）规则交换。

优点： 每趟结束时，不仅能挤出一个最大值到最后面位置，还能同时“部分理顺”其他元素；一旦下趟没有交换发生，还可以提前结束排序。

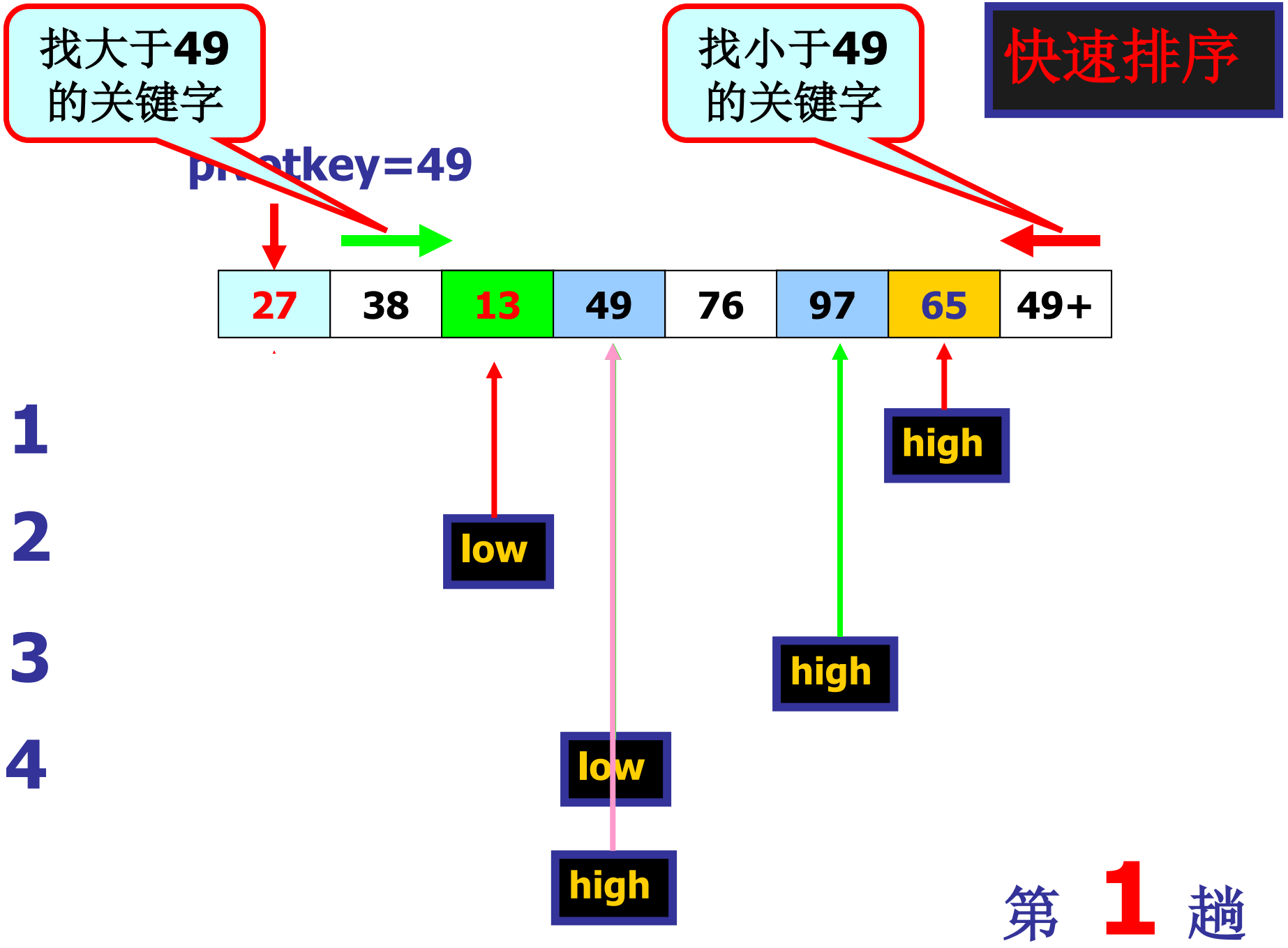
前提： 顺序存储结构

10.2.2 快速排序

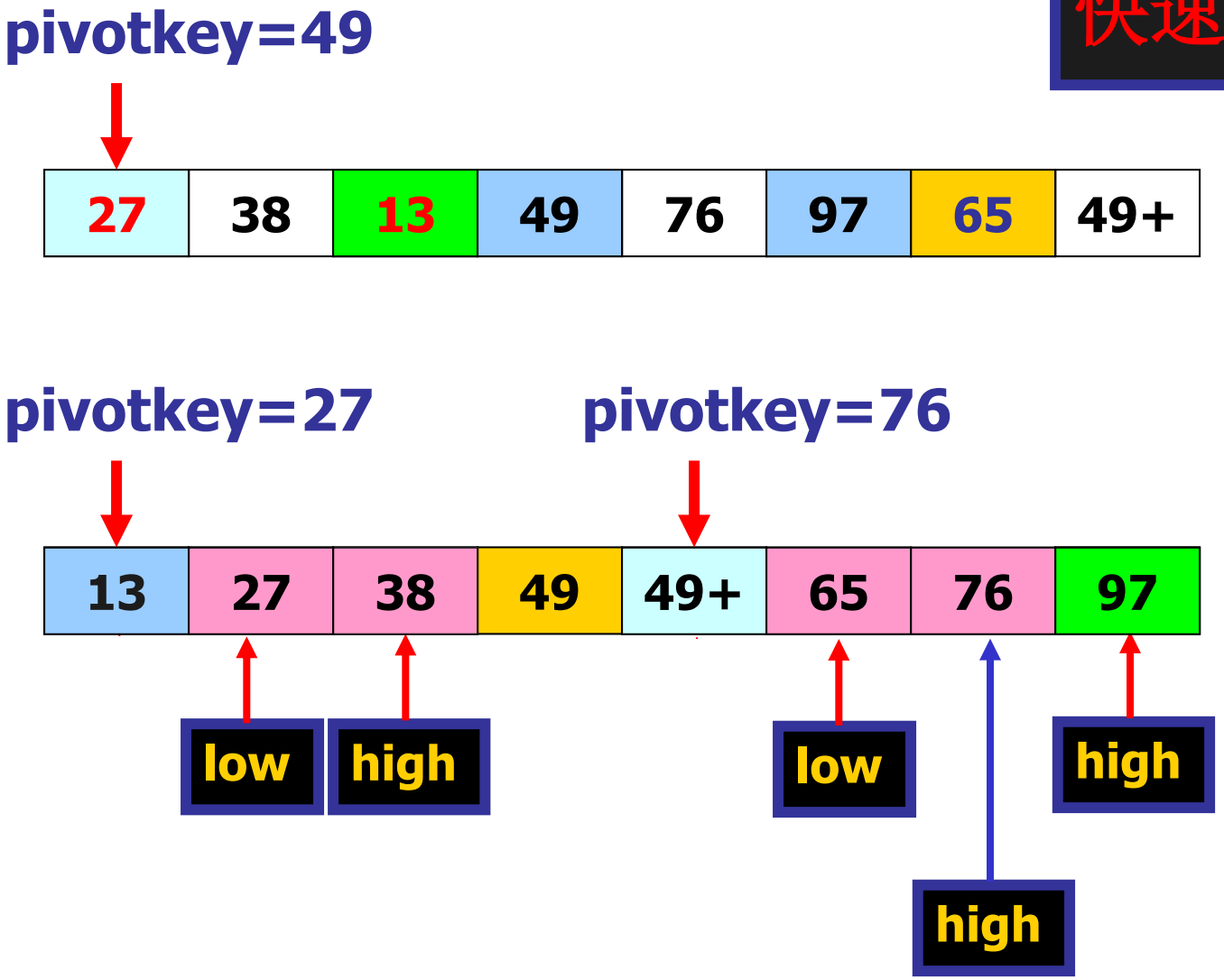
基本思想：通过一趟排序将待排序记录序列分割成成独立的**两部分**，其中一部分记录的关键字均比另一部分记录的关键字小，则可分别对这两部分记录继续进行排序，以达到整个序列有序。

优点：因为每趟可以确定不止一个元素的位置，而且呈指数增加，所以**特别快！**

前提：顺序存储结构


找大于49
的关键字
找小于49
的关键字
快速排序
pivotkey=49
27
38
13
49
76
97
65
49+
1
2
3
4
low
high
low
high
high
第 1 趟

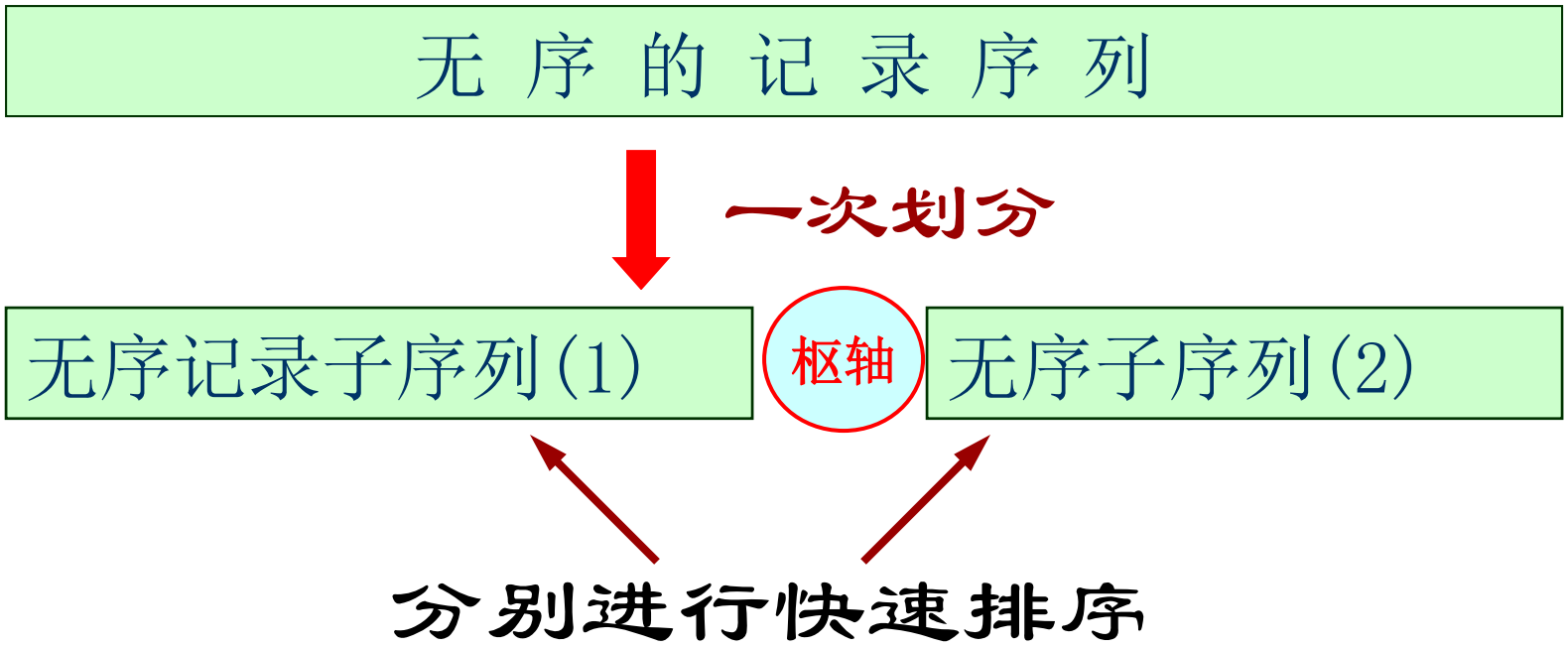
快速排序


pivotkey=49
27
38
13
49
76
97
65
49+
pivotkey=27
pivotkey=76
13
27
38
49
49+
65
76
97
low
high
low
high
high

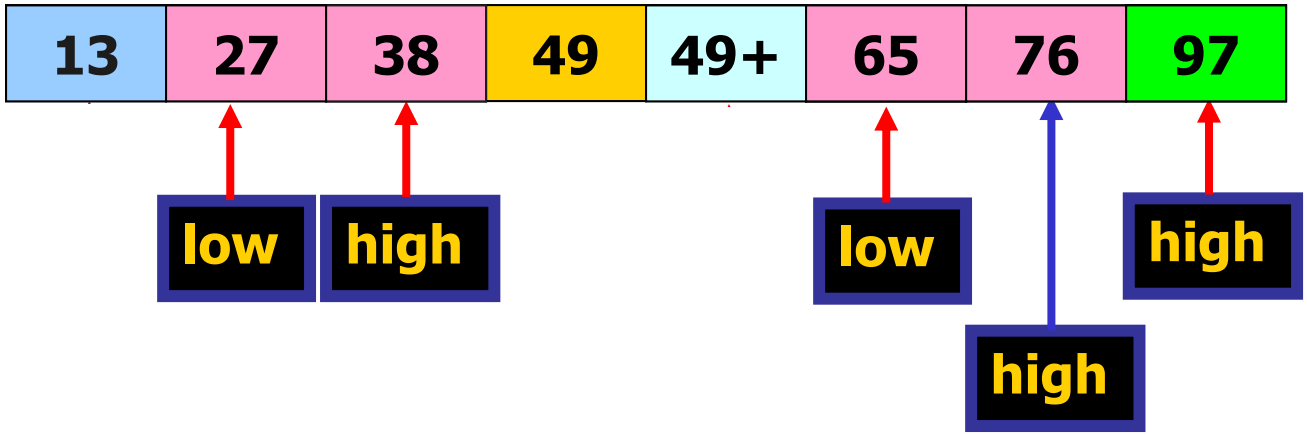
第 2 趟

快速排序

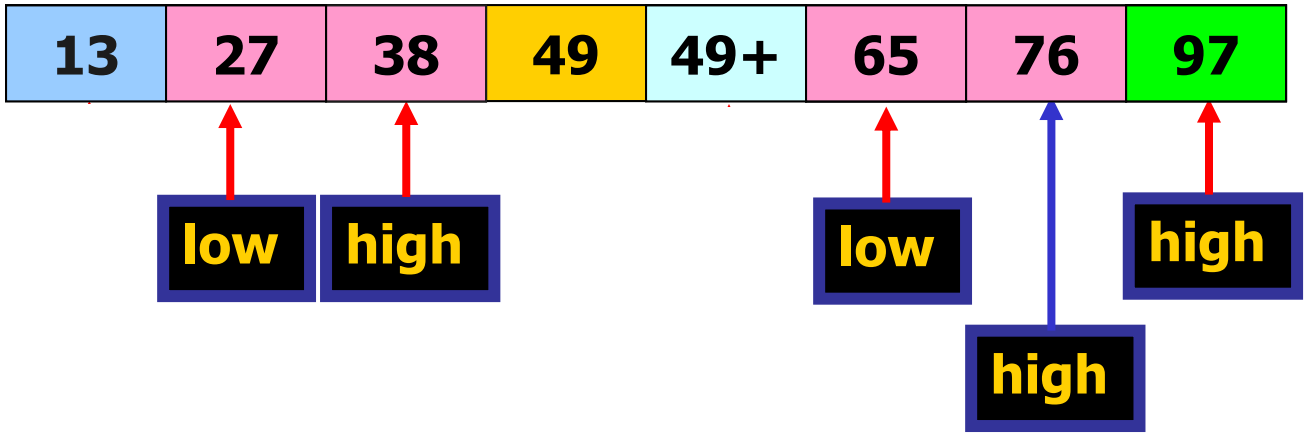
首先对无序的记录序列进行“一次划分”，之后**分别**对分割所得两个子序列“**递归**”**进行快速排序**。



```
int Partition(SqList &L,int low,int high)
{ // 交换顺序表L中子表L.r[low..high]的记录，使枢轴记录到位，
  // 并返回其所在位置，此时在它之前(后)的记录均不大(小)于它
  pivotkey=L.r[low].key;          //用子表的第一个记录作枢轴记录
  while(low<high) {               // 从表的两端交替地向中间扫描
    while(low<high&&L.r[high].key>=pivotkey)   --high;
    L.r[low] <-> L.r[high];       //将比枢轴记录小的记录交换到低端
    while(low<high&&L.r[low].key<=pivotkey)   ++low;
    L.r[low] <-> L.r[high];       //将比枢轴记录大的记录交换到高端
  }
  return low;  //返回枢轴所在位置
}
```



```
int Partition(SqList &L,int low,int high)
{ // 交换顺序表L中子表L.r[low..high]的记录，使枢轴记录到位，
  // 并返回其所在位置，此时在它之前(后)的记录均不大(小)于它
  pivotkey=L.r[low].key;  //用子表的第一个记录作枢轴记录
  while(low<high) {    // 从表的两端交替地向中间扫描
     while(low<high && L.r[high].key>=pivotkey)   --high;
     L.r[low] = L.r[high];     //将比枢轴记录小的记录交换到低端
     while(low<high && L.r[low].key<=pivotkey)   ++low;
     L.r[high] = L.r[low];    //将比枢轴记录大的记录交换到高端
  }
  L.r[low]=L.r[0];  return low;   //返回枢轴所在位置
}
```



10.2.2 快速排序

性能分析

(1) 划分元素的选取是影响快速排序性能的关键

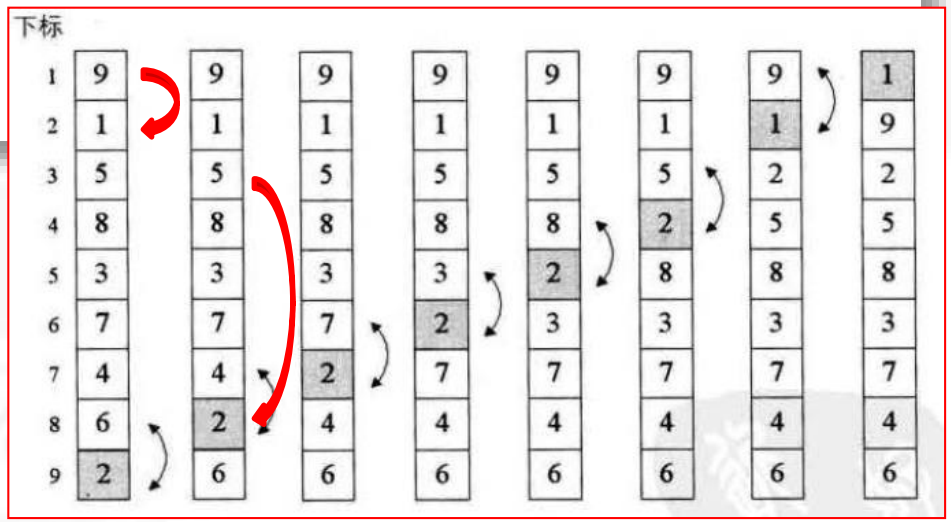
(2) 输入顺序越乱，所划分元素的随机性越好，排序速度越快。相反，输入数据越有序，排序速度越慢？

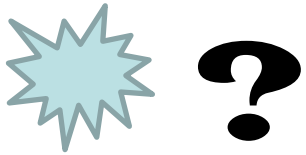
(3) 改变划分元素的选取方法，至多只能改变算法平均情况下的时间性能，无法改变最坏情况下的时间性能。

(4) 快速排序是一种不稳定的排序算法。

10.3 选择排序

```
00059: /*对 顺 序 表 L作 冒 泡 排 序 */
00060: void BubbleSort(SqList *L)
00061: {
00062:     int i,j;
00063:     for(i=1;i<L->length;i++)
00064:     {
00065:         for(j=L->length-1;j>=i;j--) /*注 意 j是 从 后 往 前 循 环 */
00066:         {
00067:             if(L->r[j]>L->r[j+1]) /*若 前 者 大 于 后 者 （ 注 意 这 里 与
00068:             {
00069:                 swap(L,j,j+1);/*交 换 L->r[j]与 L->r[j+1]的 值 */
00070:             }
00071:         }
00072:     }
00073: }
```





?

10.3 选择排序

- 基本思想：每一趟在后面$n-i$个待排记录中选取关键字最小的记录作为有序序列中的第i个记录。
  - 10.3.1 简单选择排序
  - 10.3.2 树形选择排序（锦标赛排序）（扩展学习）
  - 10.3.3 堆排序（扩展学习）

10.4.1 简单选择排序

思路简单：每经过一趟比较就找出一个最小值，与待排序列最前面的位置互换即可。

——首先，在n个记录中选择最小者放到r[1]位置；然后，从剩余的n-1个记录中选择最小者放到r[2]位置；…如此进行下去，直到全部有序为止。

优点：实现简单

缺点：每趟只能确定一个元素，表长为n时需要n−1趟

前提：顺序存储结构

10.3 选择排序

```
00059: /*对 顺 序 表 L作 冒 泡 排 序 */
00060: void BubbleSort(SqList *L)
00061: {
00062:     int i,j;
00063:     for(i=1;i<L->length;i++)
00064:     {
00065:         for(j=L->length-1;j>=i;j--)  /*注
00066:         {
00067:             if(L->r[j]>L->r[j+1]) /*若 前
00068:             {
00069:                 swap(L,j,j+1);/*交 换 l
00070:             }
00071:         }
00072:     }
00073: }
```

```
00096: void SelectSort(SqList *L)
00097: {
00098:     int i,j,min;
00099:     for(i=1;i<L->length;i++)
00100:     {
00101:         min = i;
00102:         for (j = i+1;j<=L->length;j++)
00103:         {
00104:             if (L->r[min]>L->r[j])  /*
00105:                 min = j;
00106:         }
00107:         if(i!=min)
00108:             swap(L,i,min);
00109:     }
00110: }
```

例： 关键字序列T= （21，25，49，25+，16，08），请给出简单选择排序的具体实现过程。

原始序列： 21，25，49，25+，16，08

第1趟 08，25，49，25+，16，21

第2趟 08，16，49，25+，25，21

第3趟 08，16，21，25+，25，49

第4趟 08，16，21，25+，25，49

第5趟 08，16，21，25+，25，49

结论：简单选择排序是“不稳定”的

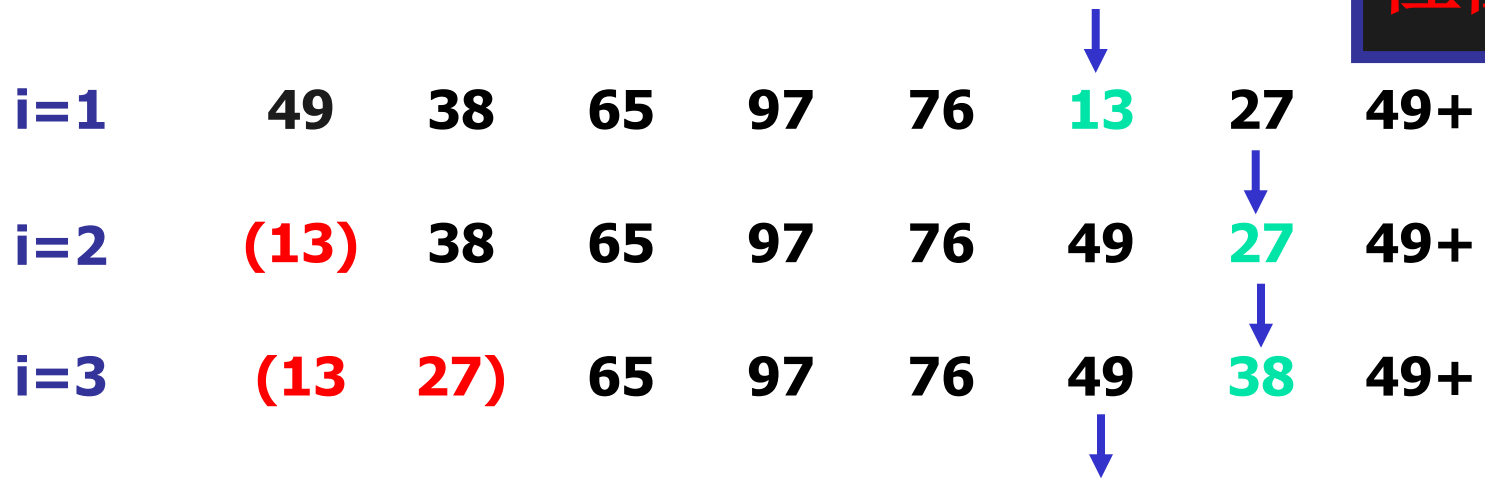
实验5 线性表排序的应用

- （1）输入说明：输入的第一行包括两个正整数N（N<10000）和C，其中N是记录的条数，C是指定排序的列号。之后有N行，每行包括一条学生记录。每条学生记录由学号（5位数字，保证没有重复的学号）、姓名（不超过8位且不包含空格的字符串）、成绩（[0，100]内的整数）组成，相邻属性用1个空格隔开。
- （2）输出格式：在N行中输出按要求排序后的结果，即：C=1时，按照学号递增排序；当C=2时，按照姓名的非递减字典序排序；当C=3时，按照成绩的非递减排序。当若干学生具有相同的姓名或者成绩时，则按照他们的学号递增排序。
- 要求：使用冒泡排序和选择排序完成。

实验5 线性表排序的应用

```
typedef struct STUDENT_Str
 {
        char name[10];//姓名
        char num[6]; //学号
        int score; //成绩
}STUDENT_STRU;
```

性能分析



简单选择排序中存在大量的“冗余比较”

（1） 元素x与y比较一次之后，就不必再比较它们

（2） 若x>y，y>z，则必有x>z

比较的传递性

简单选择排序不能“记住”比较的结果！

10.3.2 树形选择排序

Tree selection sort 采用比赛树结构记住比较结果

基本思想：与体育比赛时的淘汰赛类似。

首先对 n 个记录的关键字进行两两比较，得到$\lceil n/2 \rceil$个优胜者(关键字小者)，作为第一步比较的结果保留下来。然后在这$\lceil n/2 \rceil$个较小者之间再进行两两比较，...，如此重复，直到选出最小关键字的记录为止。

优点：减少比较次数，加快排序速度

缺点：空间效率低，需要构造一个比赛树

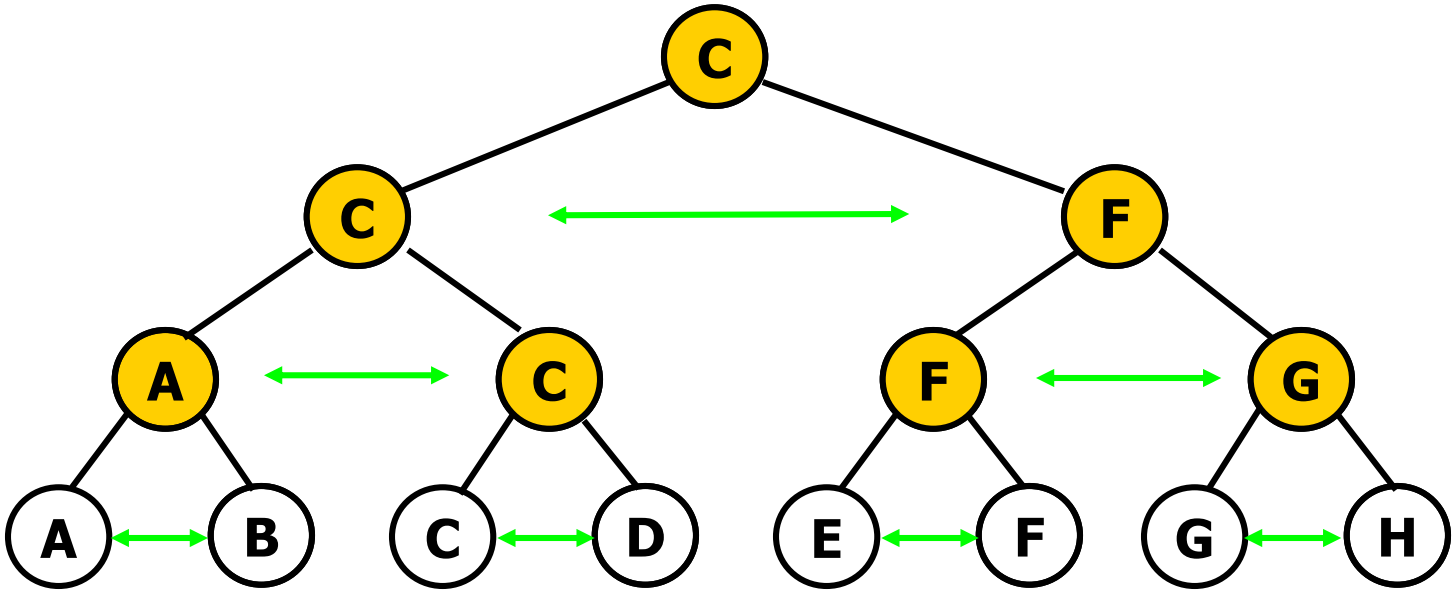
10.3.2 树形选择排序

Tree selection sort 采用比赛树结构记住比较结果

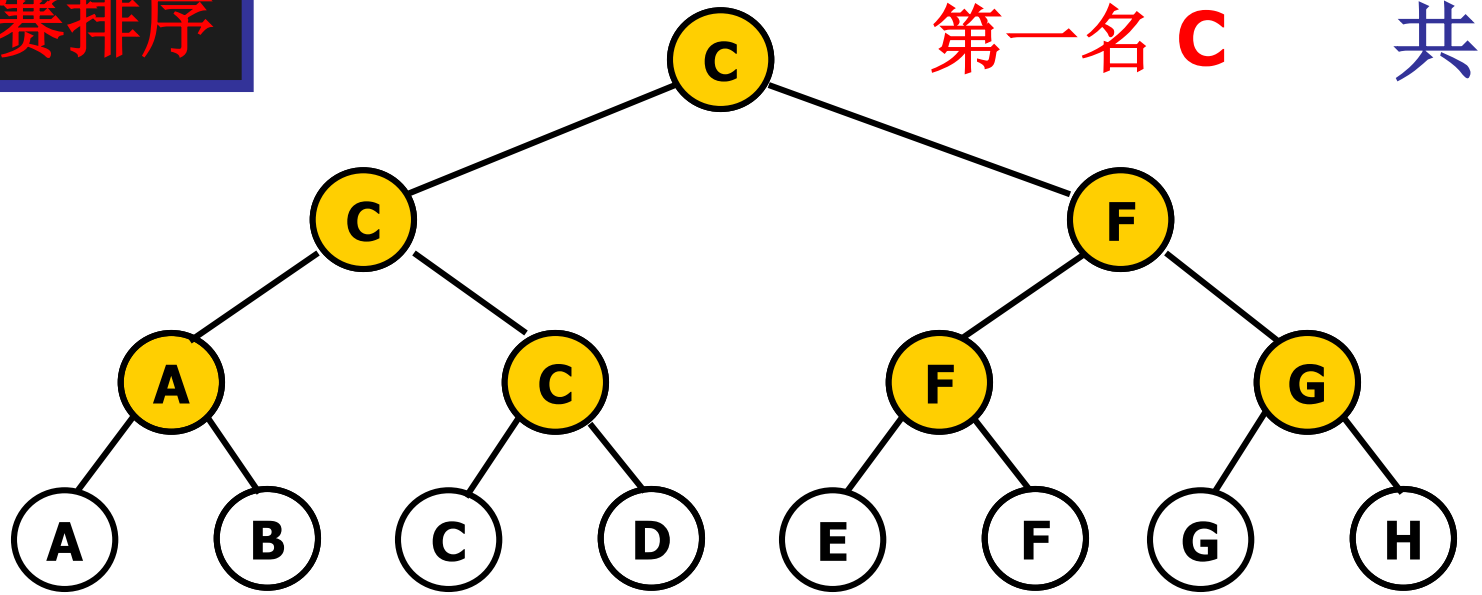
锦标赛排序

第一名 **C**

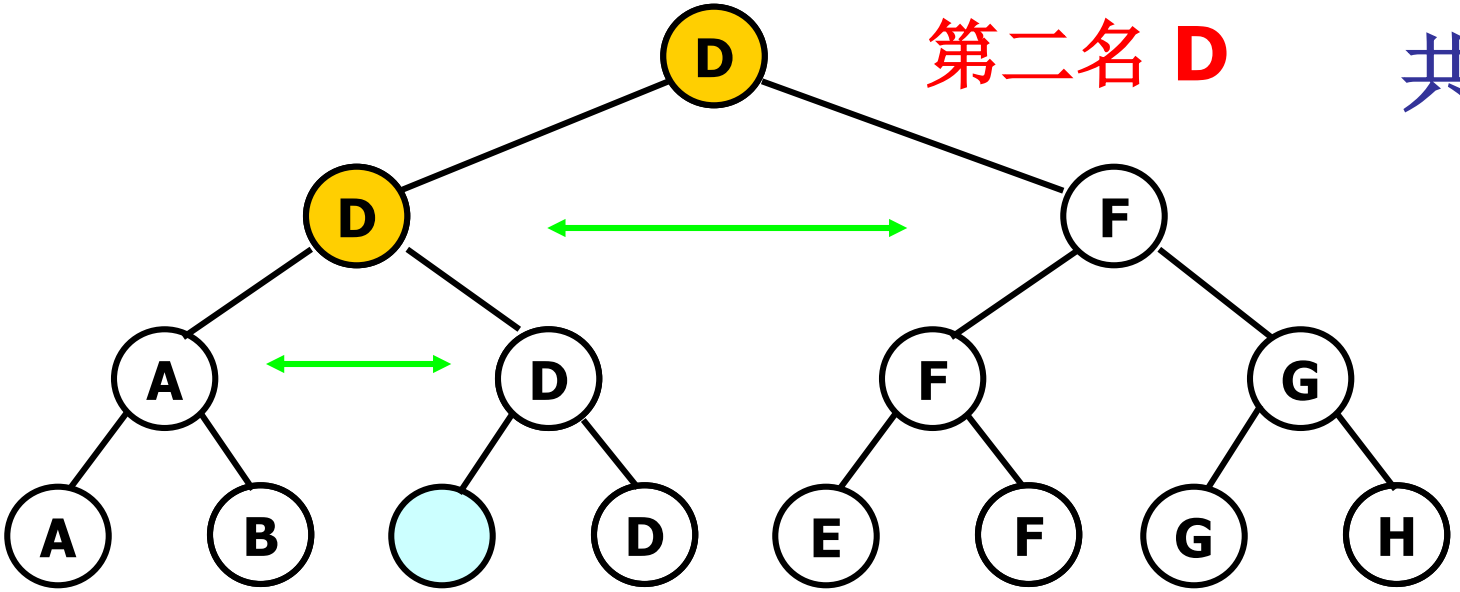
共**7**次



锦标赛排序



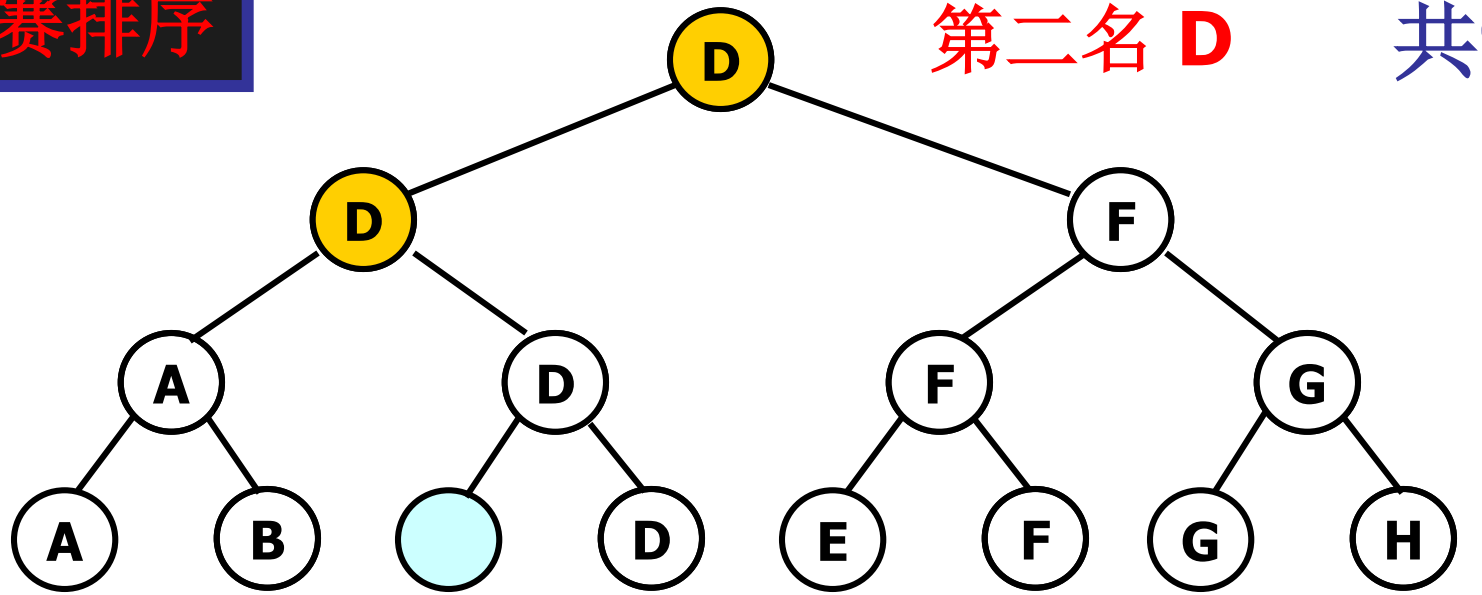
第一名 C　　共7次



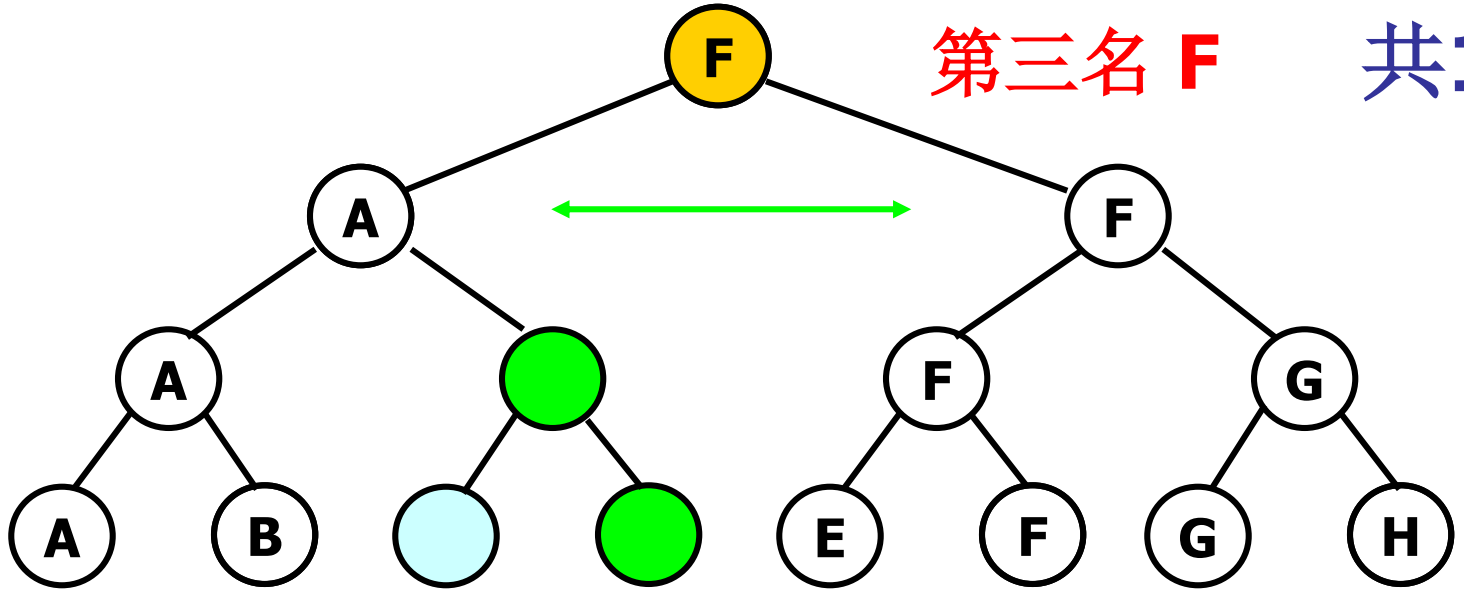
第二名 D　　共9次

锦标赛排序

第二名 D　　共9次

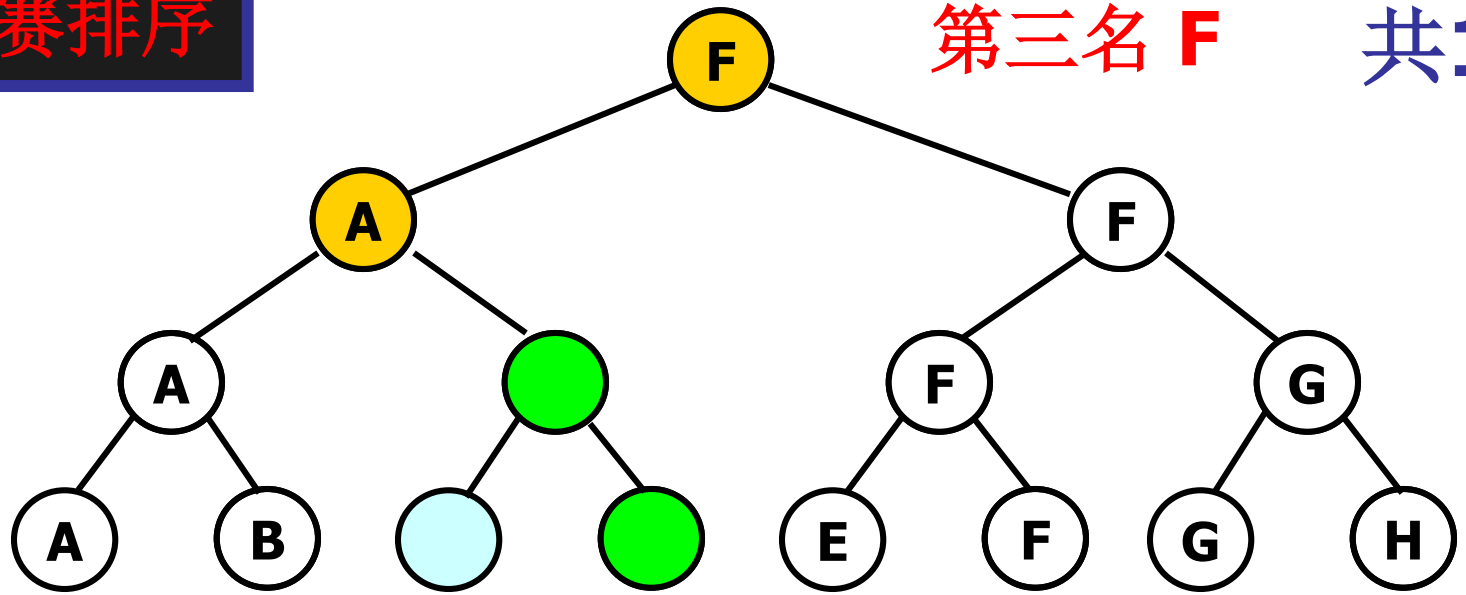


第三名 F　　共10次

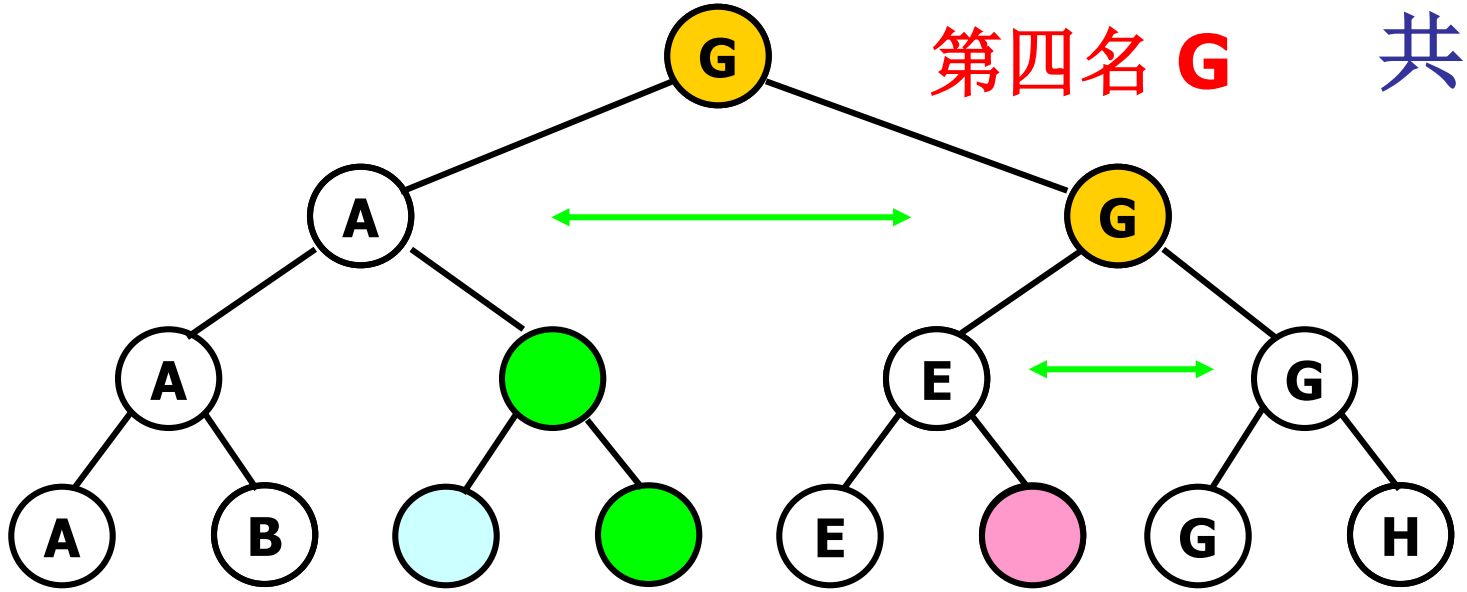


锦标赛排序

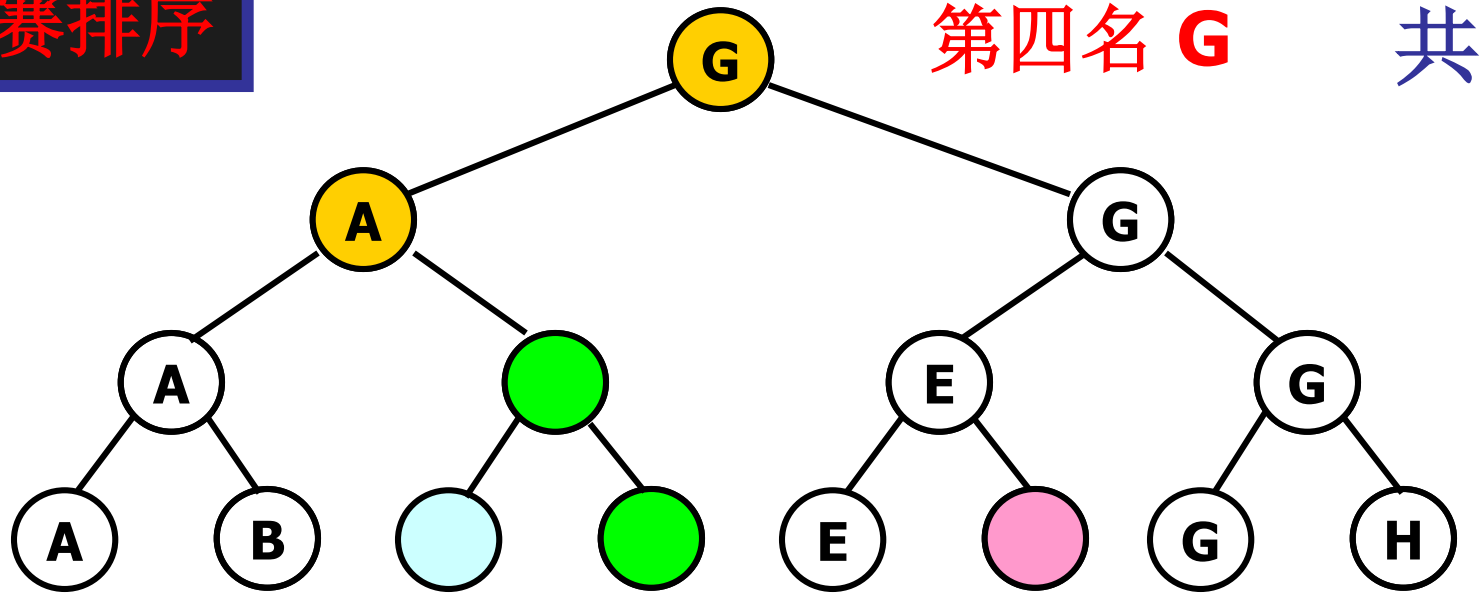
第三名 F　　共10次



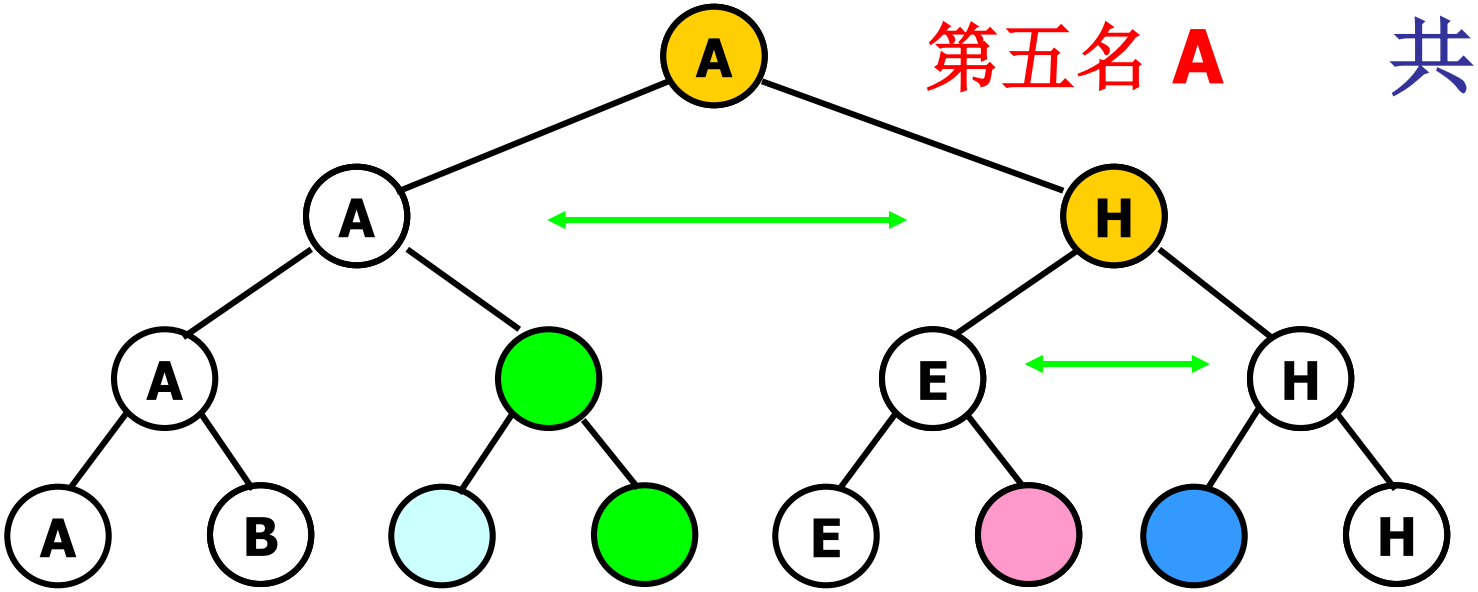
第四名 G　　共12次



锦标赛排序

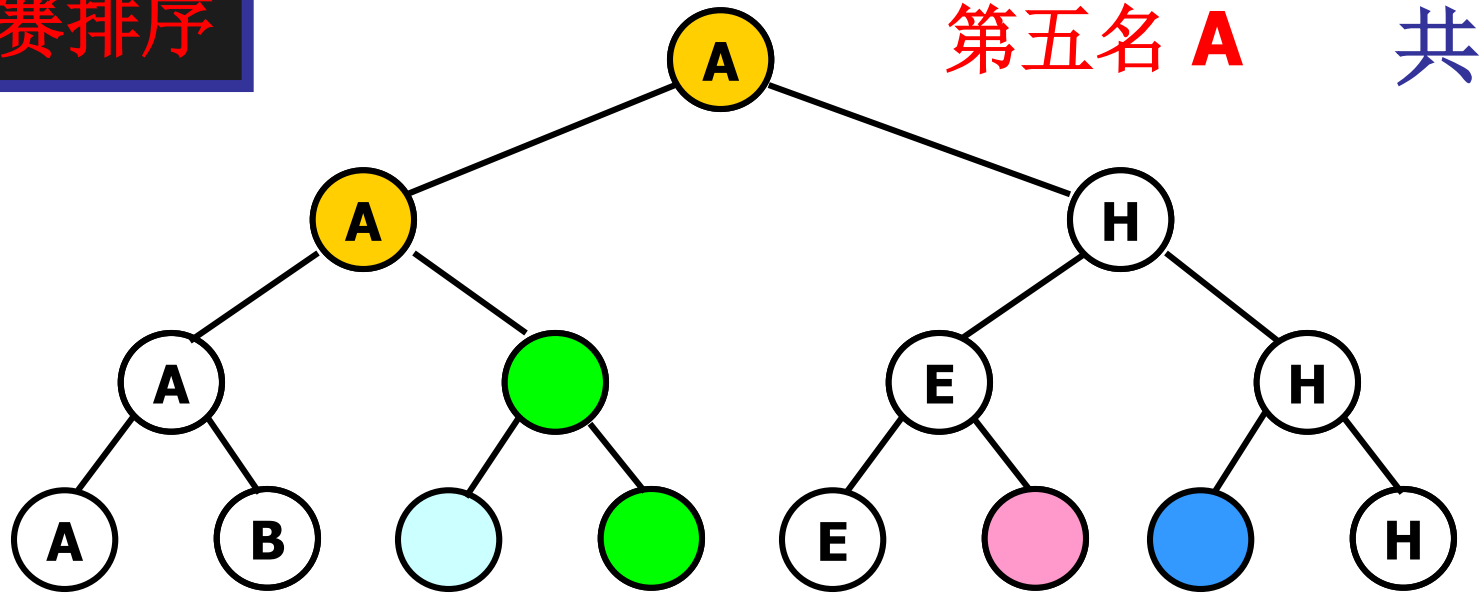


第四名 G　　共12次



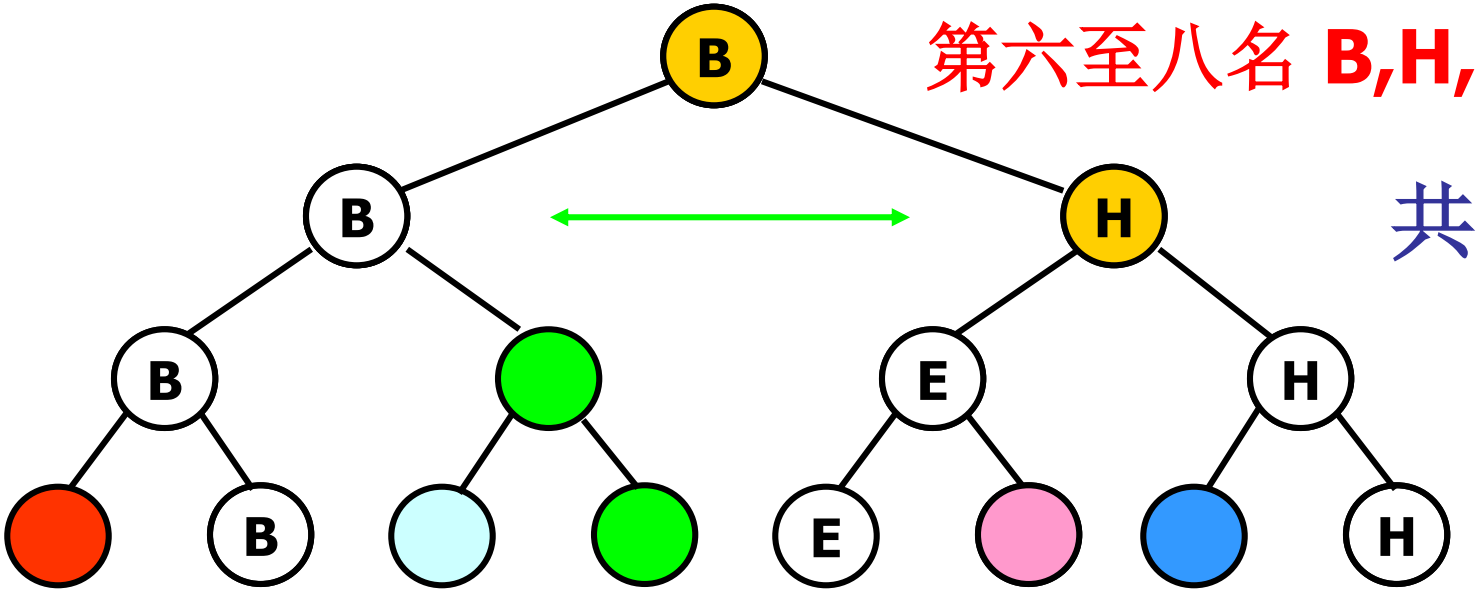
第五名 A　　共14次

锦标赛排序



第五名 A

共14次



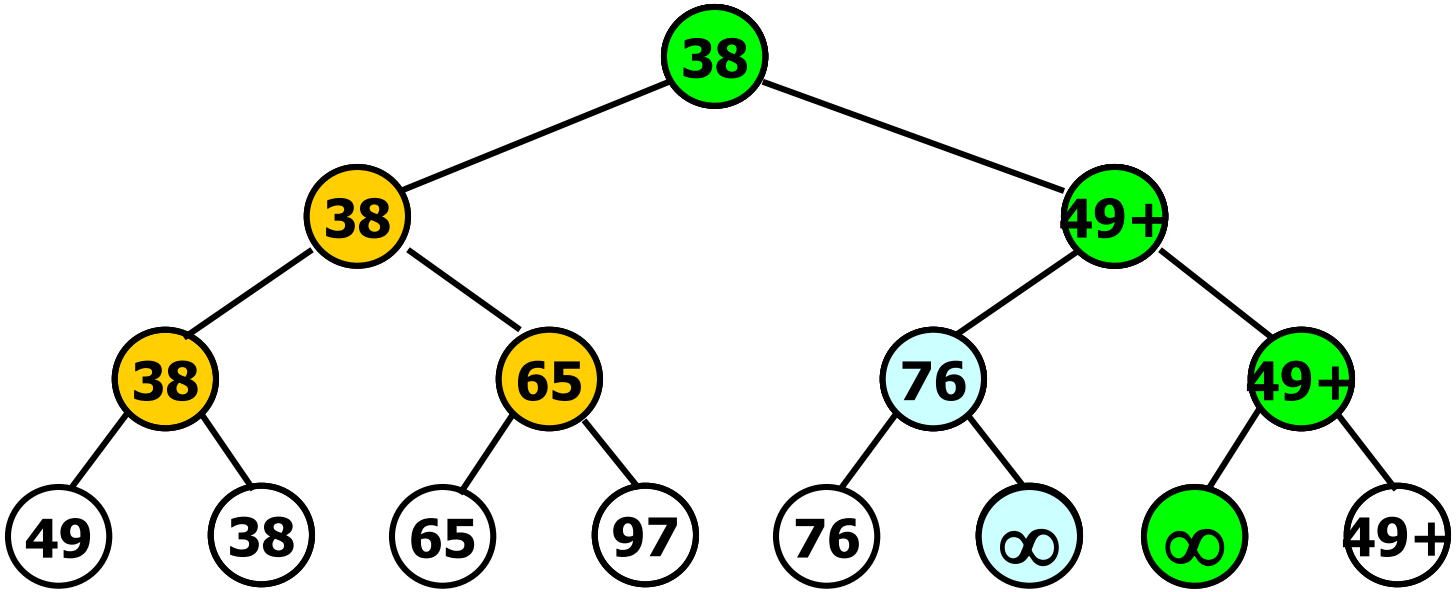
第六至八名 B,H,E

共15次

树形选择排序

3　　1　　2

49	38	65	97	76	13	27	49+



每选择一个“次小”关键字，需要_____次比较?

$\lfloor \log_2 n \rfloor$

$T(n)=O(n\log_2 n)$

10.3.2 树形选择排序

- 算法分析
  - 锦标赛排序构成的树是满的完全二叉树，其深度为$\lceil \log_2(n+1) \rceil$，其中 n 为待排序元素个数。
  - 除第一次选择具有最小关键码的对象需要进行 n-1 次关键码比较外，重构胜者树选择具有次小、再次小关键码对象所需的关键码比较次数均为 $O(\log_2 n)$。总关键码比较次数为 $O(n\log_2 n)$。
  - 对象的移动次数不超过关键码的比较次数，

10.3.3 堆排序

Heap sort　　J.Willioms, 1964

堆的定义：一个有n个元素的线性序列$(R_0, R_1, \ldots, R_{n-1})$，其关键字序列$(K_0, K_1, \ldots, K_{n-1})$满足：

$$\begin{cases} K_i \le K_{2i} \\ K_i \le K_{2i+1} \end{cases} \quad \text{或} \quad \begin{cases} K_i \ge K_{2i} \\ K_i \ge K_{2i+1} \end{cases}$$

$$\left(i = 1, 2, \cdots, \left\lfloor \frac{n}{2} \right\rfloor \right)$$

$$\begin{cases} K_i \geq K_{2i} \\ K_i \geq K_{2i+1} \end{cases}$$

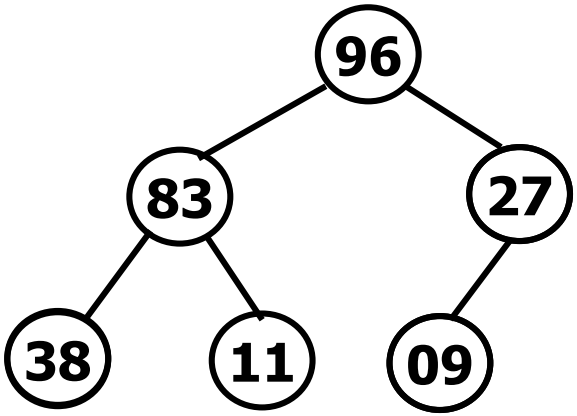
$$\begin{cases} K_i \leq K_{2i} \\ K_i \leq K_{2i+1} \end{cases}$$

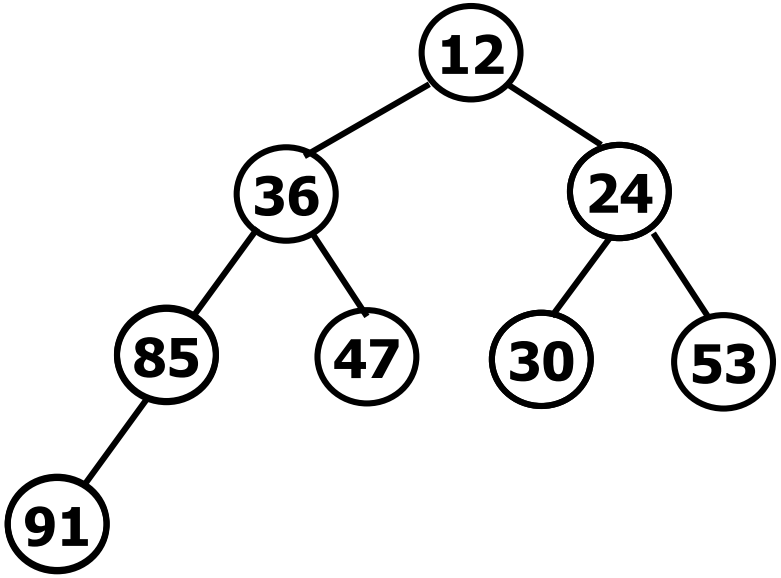
堆顶

堆尾

0	1	2	3	4	5	6
	96	83	27	38	11	09

0	1	2	3	4	5	6	7	8
	12	36	24	85	47	30	53	91



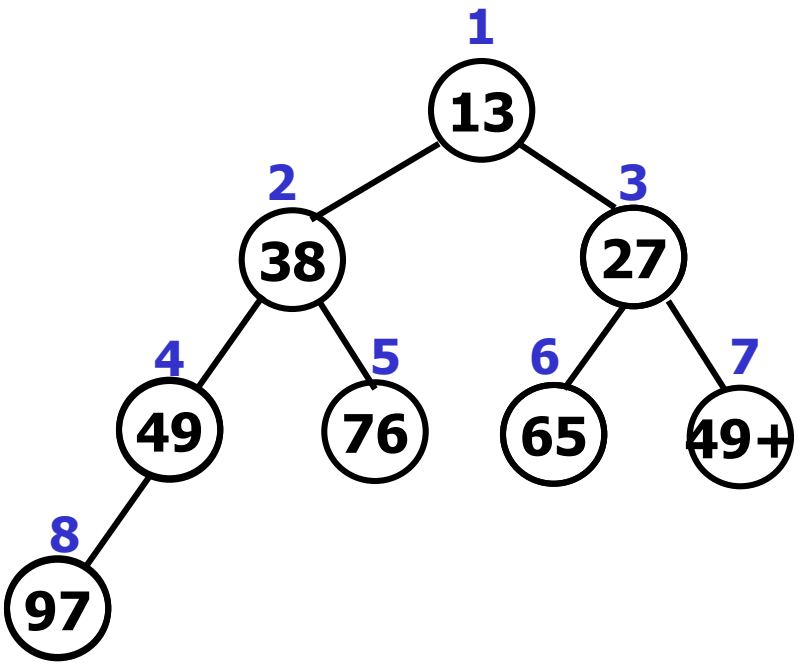


大顶堆

小顶堆

堆排序

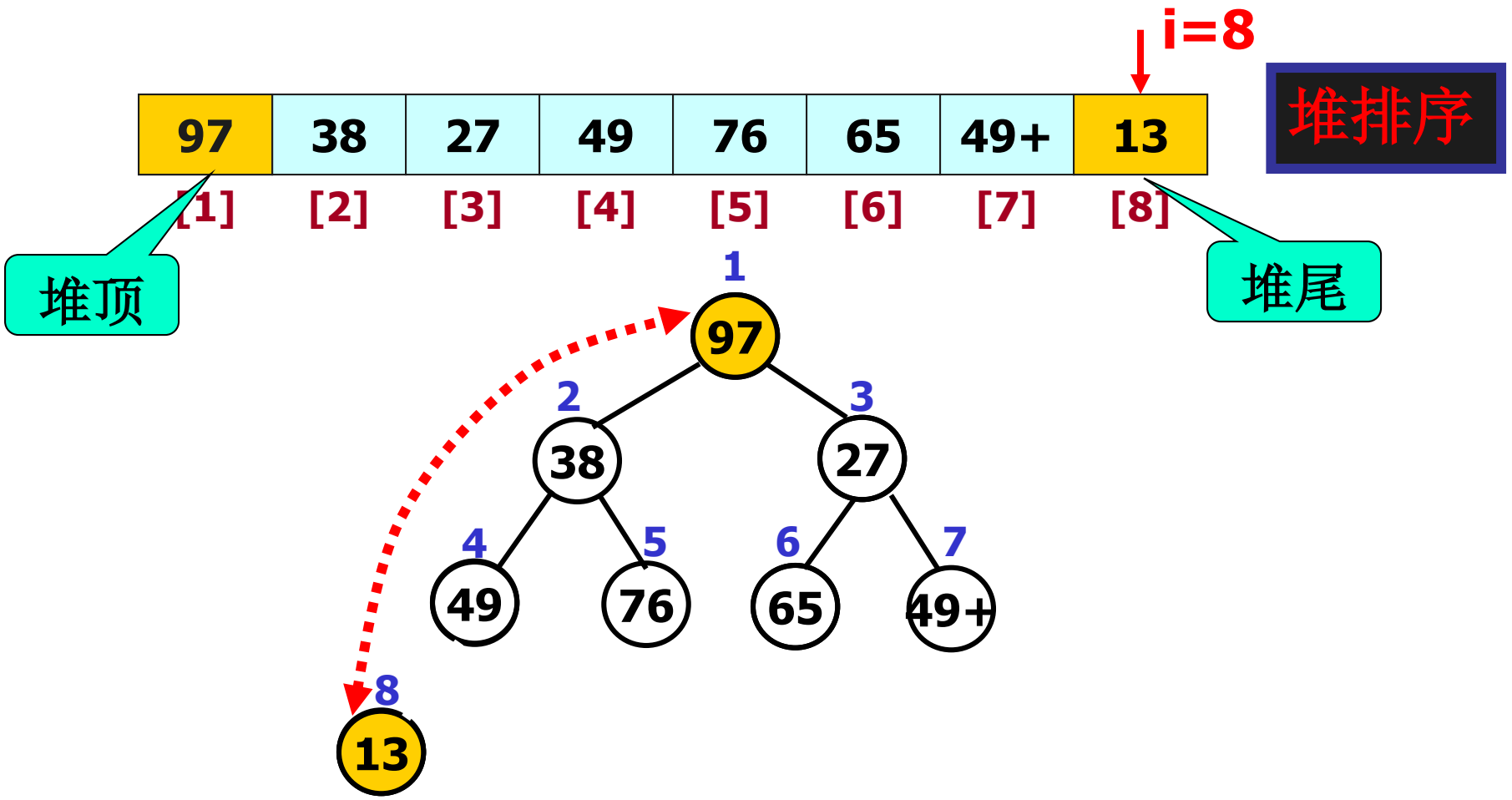
13	38	27	49	76	65	49+	97
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]



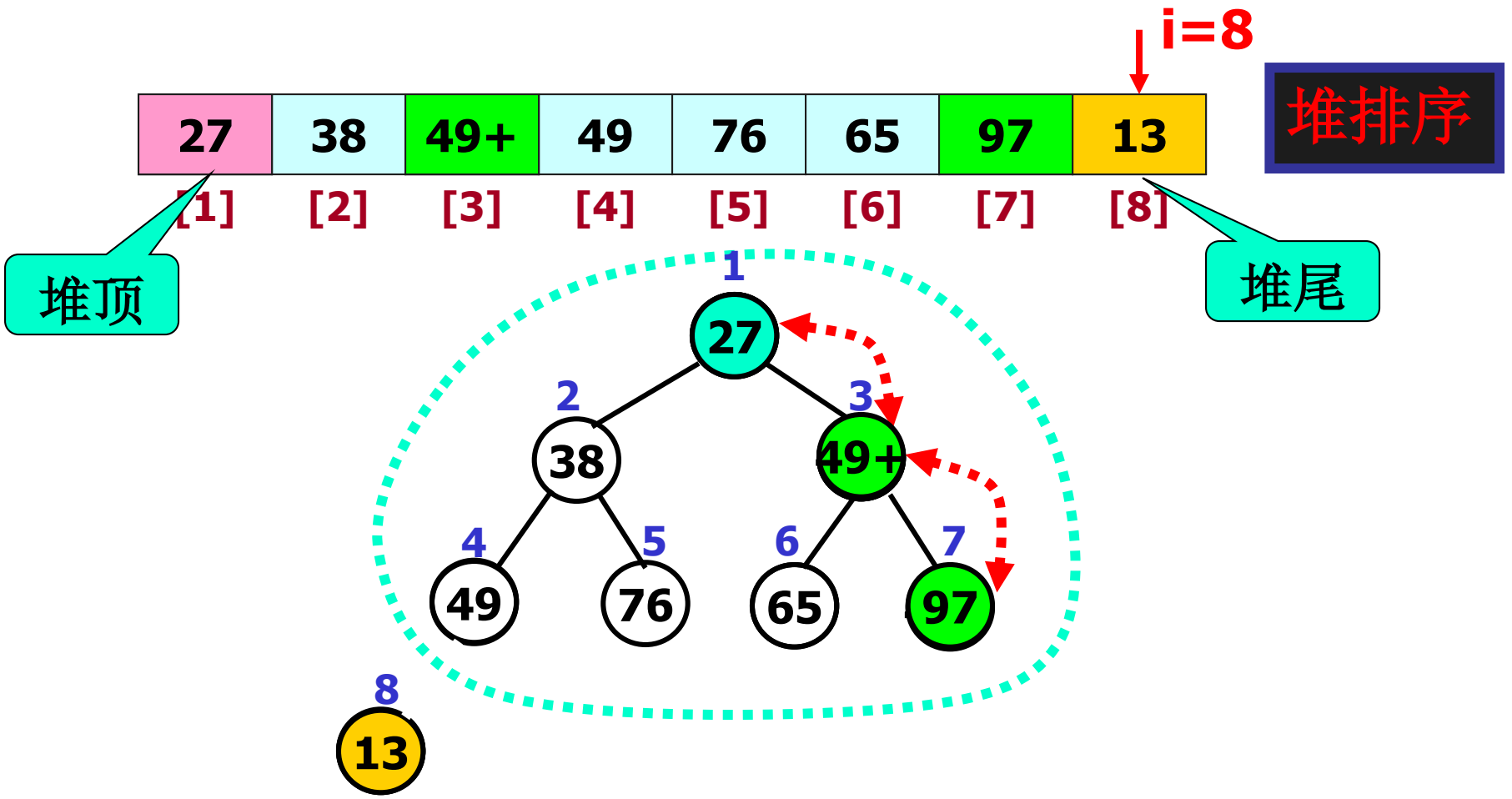
重新堆化

（1）如何由一个无序序列“建立”一个堆？

（2）如何在输出堆顶元素之后，“调整”剩余元素成为一个新的堆？

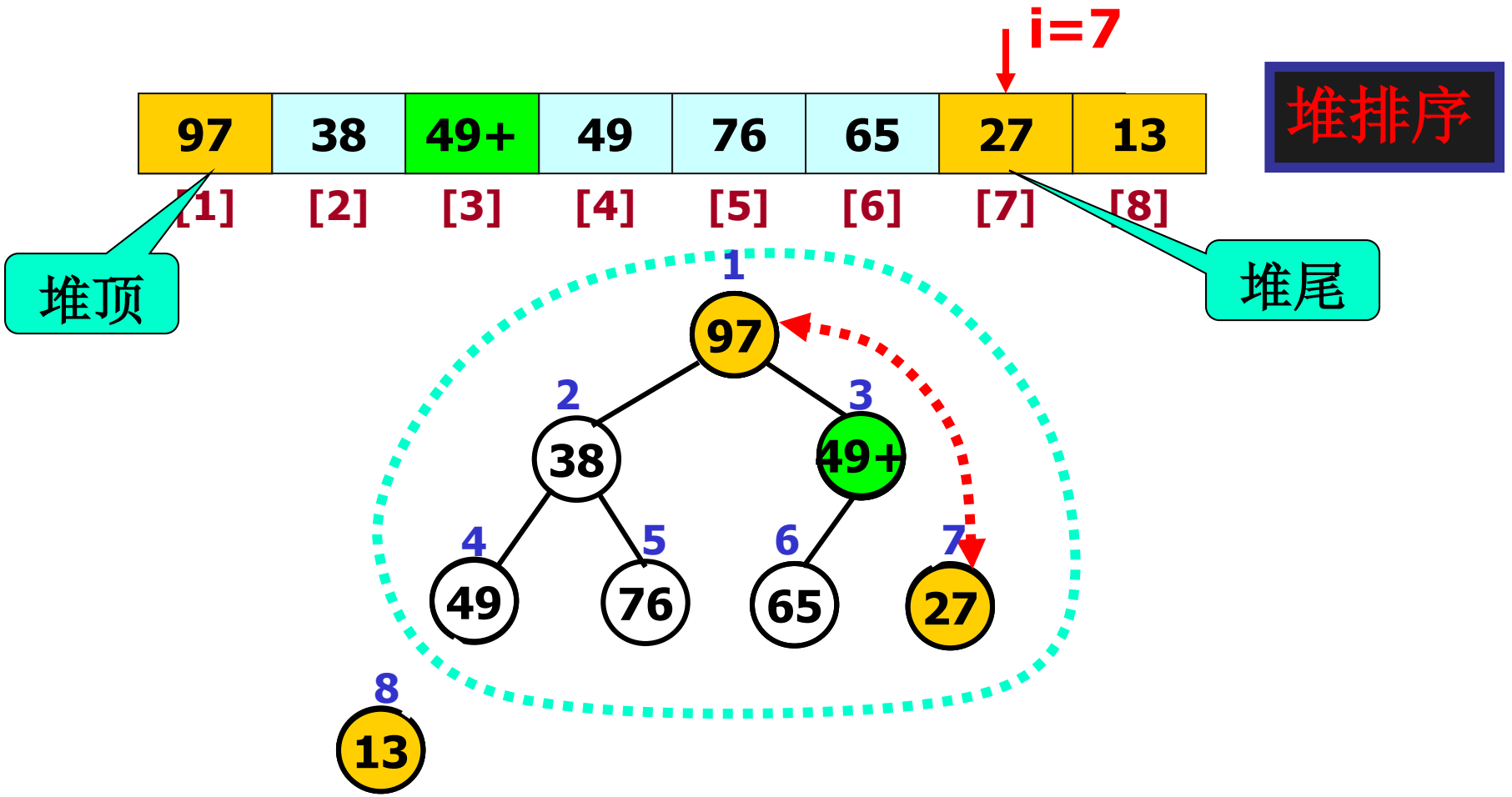


(1) H.r[1] ↔ H.r[i]　　堆尾和堆顶元素交换

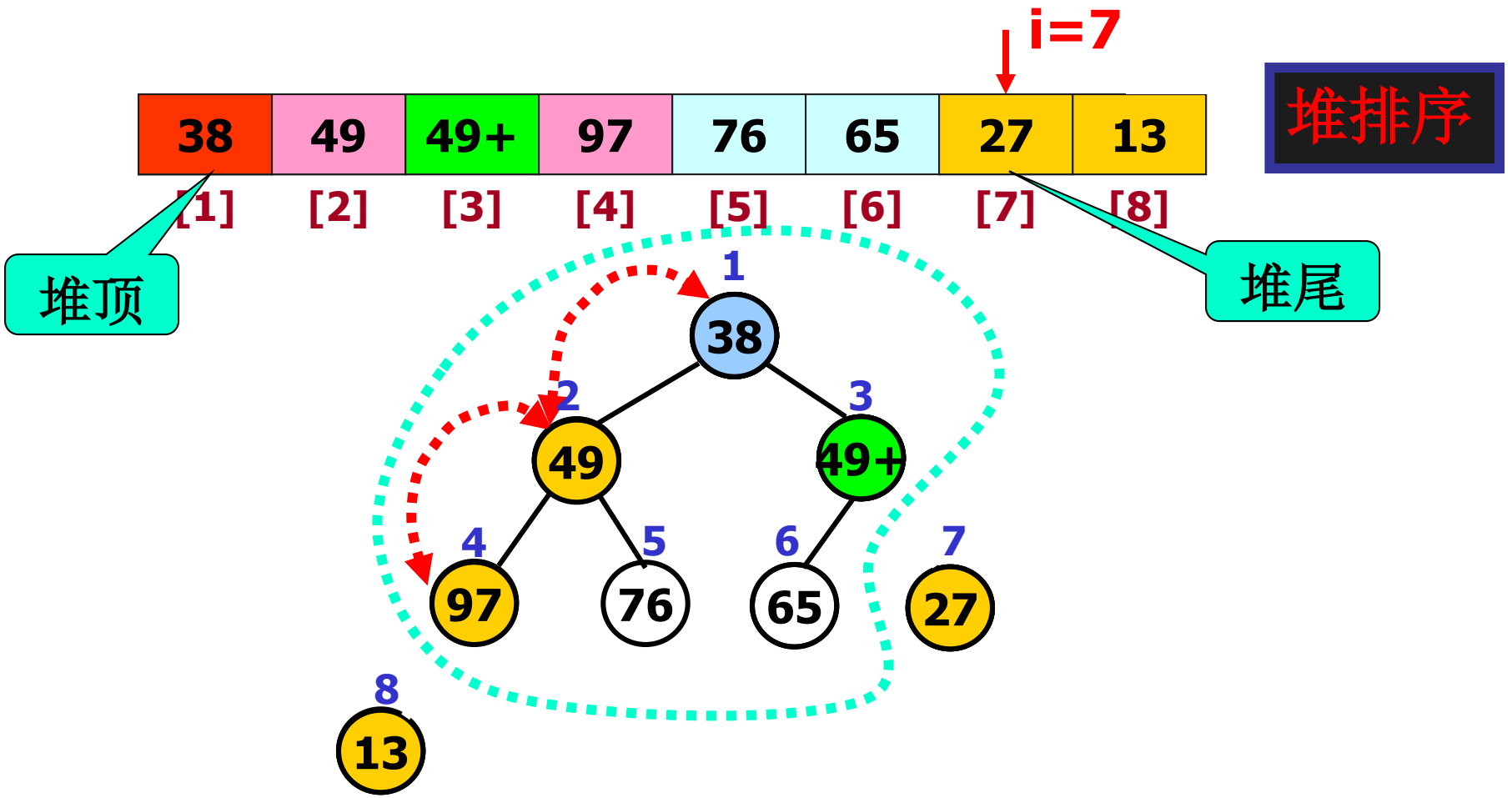


(1) H.r[1] ↔ H.r[i]　　堆尾和堆顶元素交换

(2) 重新堆化 H.r[1..i-1]

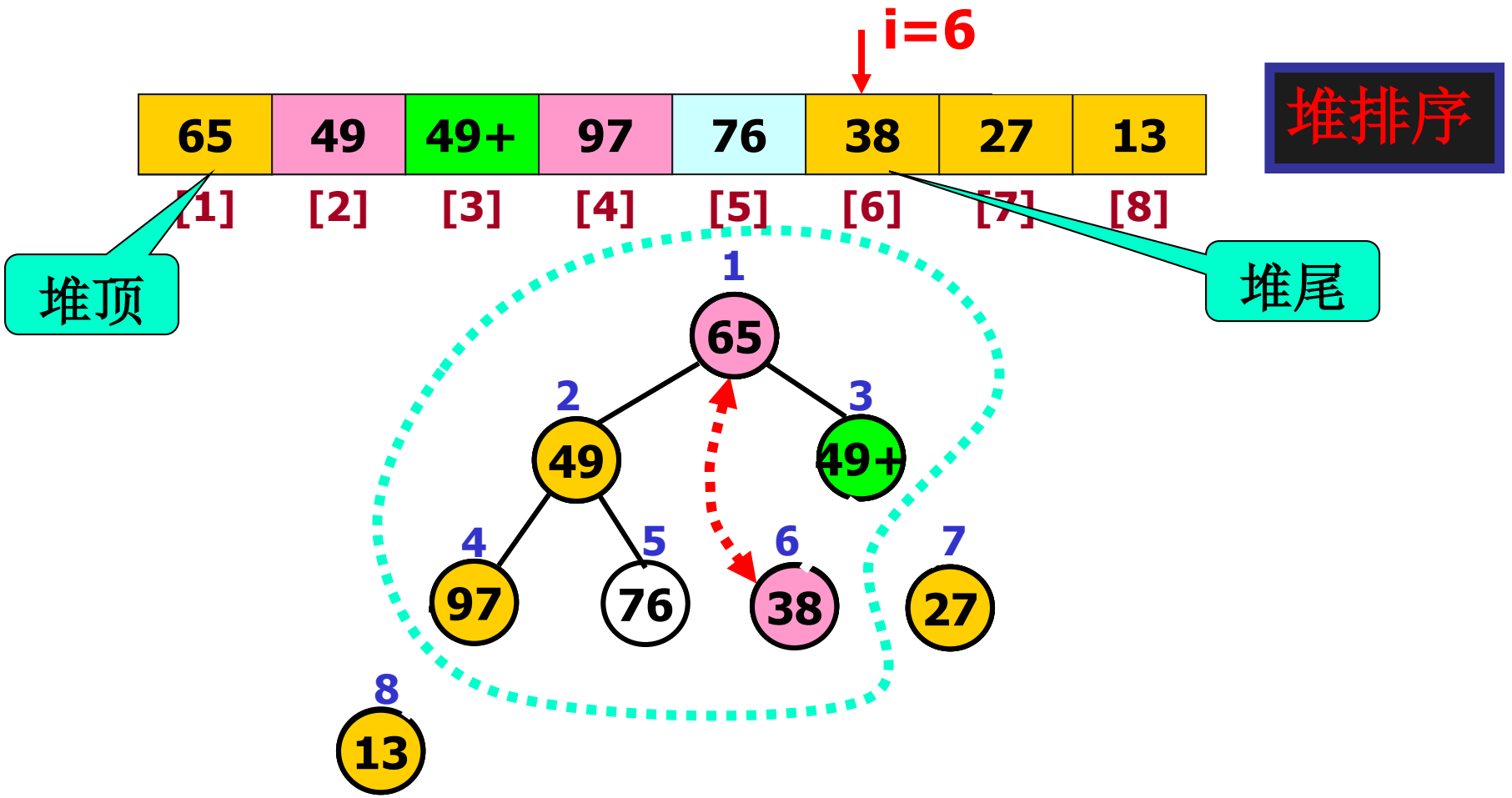


(1) H.r[1] ↔ H.r[i]　　堆尾和堆顶元素交换

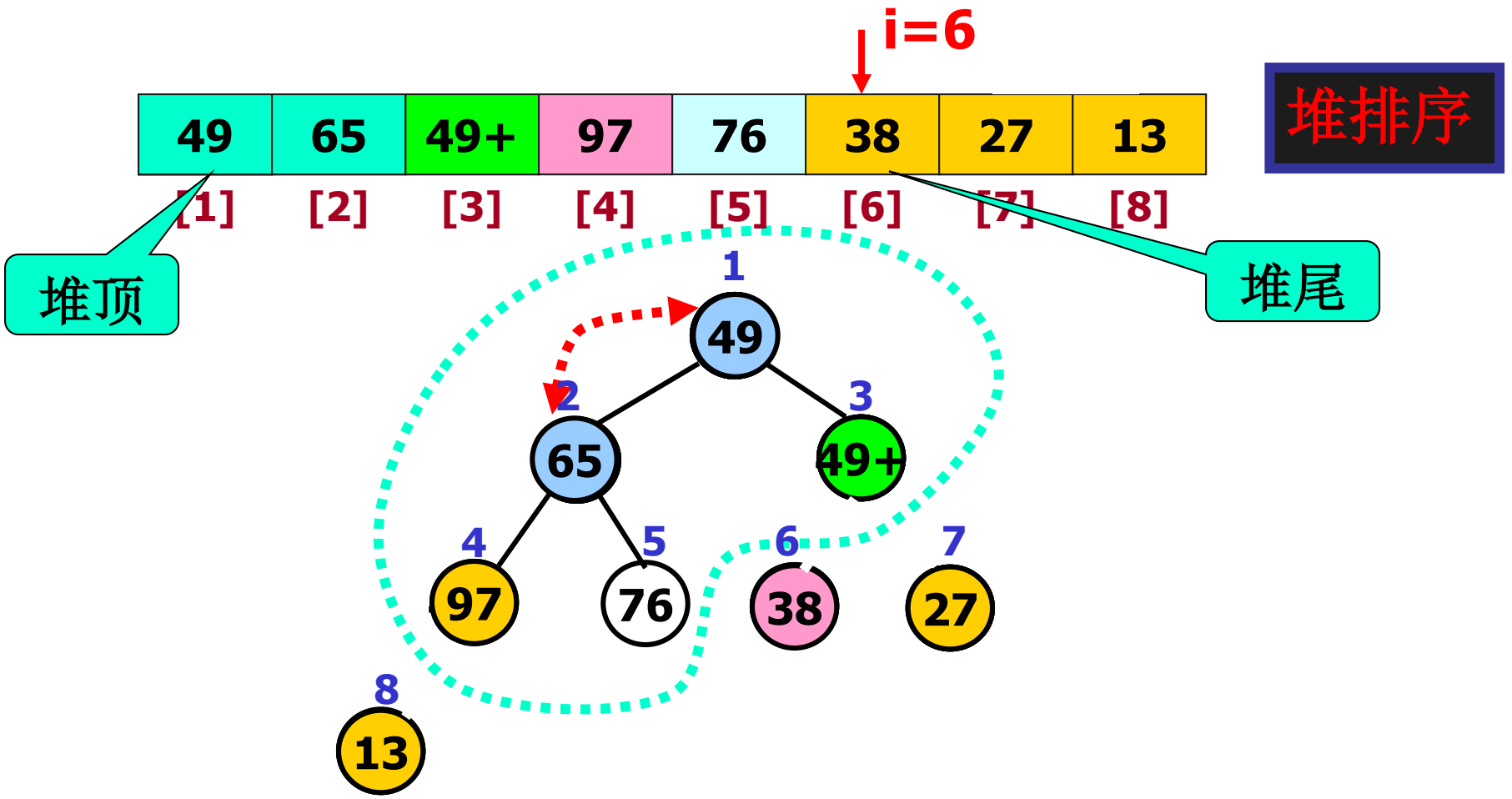


(1) H.r[1] ↔ H.r[i]　　堆尾和堆顶元素交换

(2) 重新堆化 H.r[1..i-1]

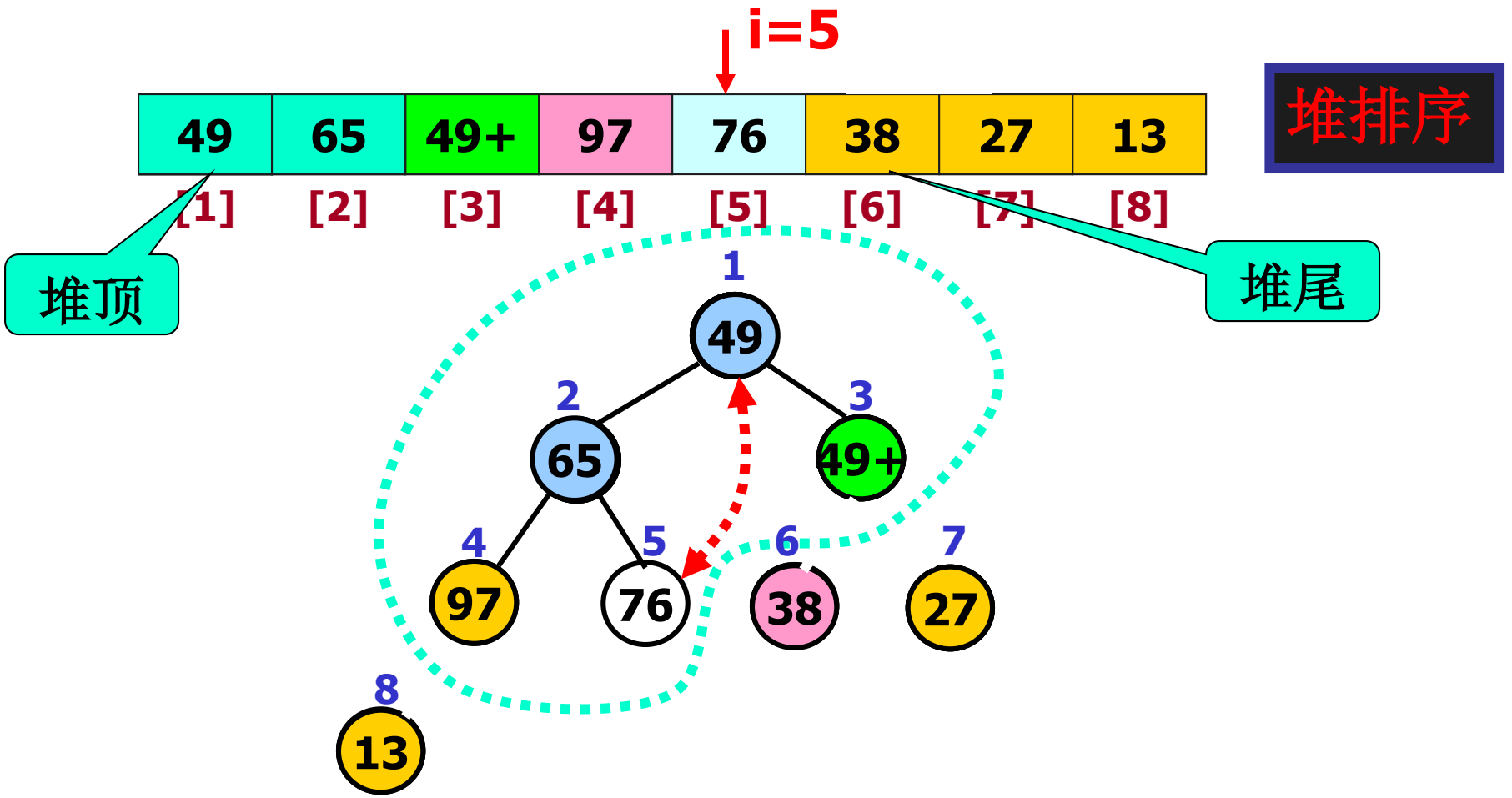


(1) H.r[1] ↔ H.r[i]　　堆尾和堆顶元素交换

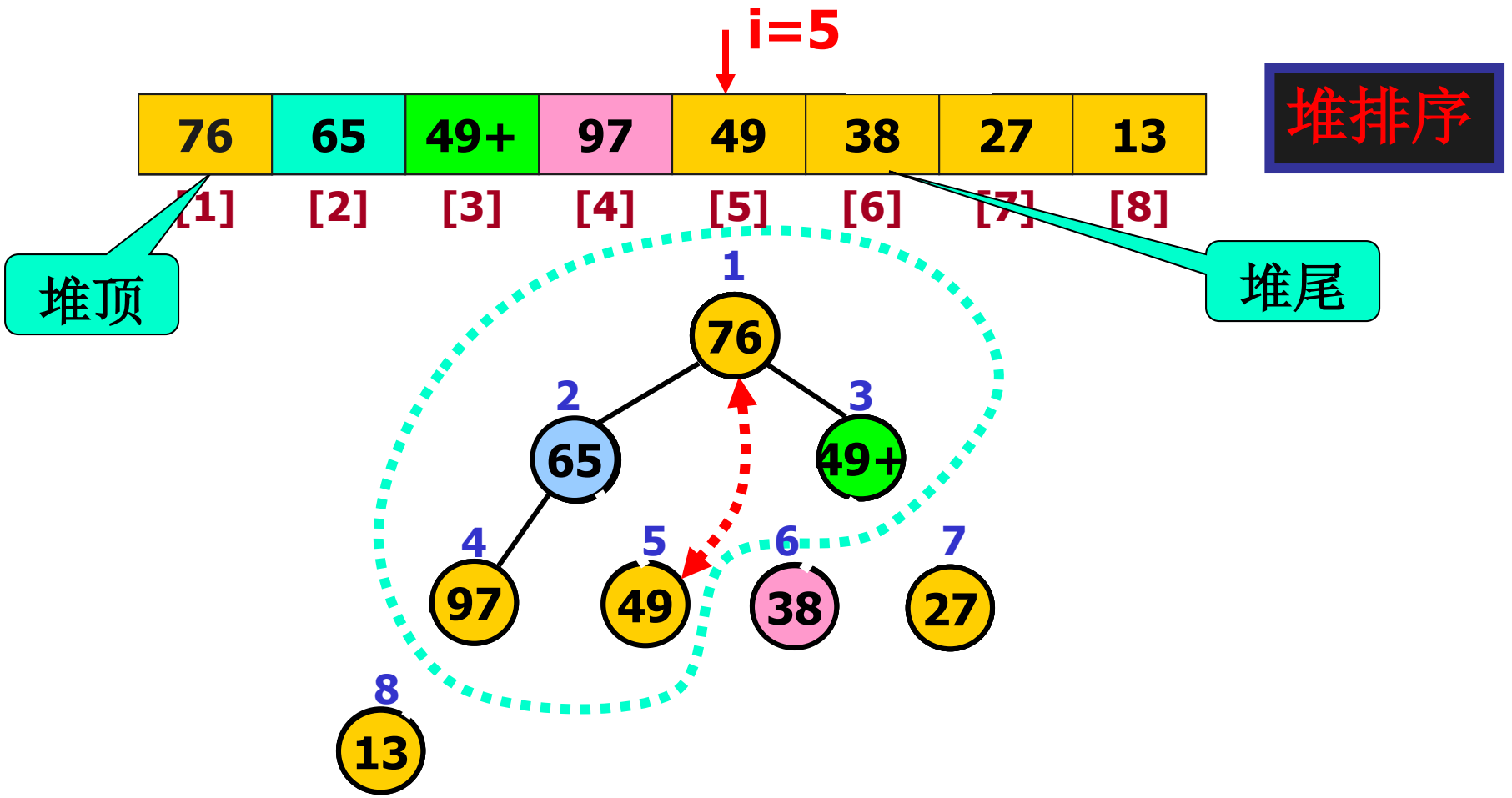


(1) H.r[1] ↔ H.r[i]　　堆尾和堆顶元素交换

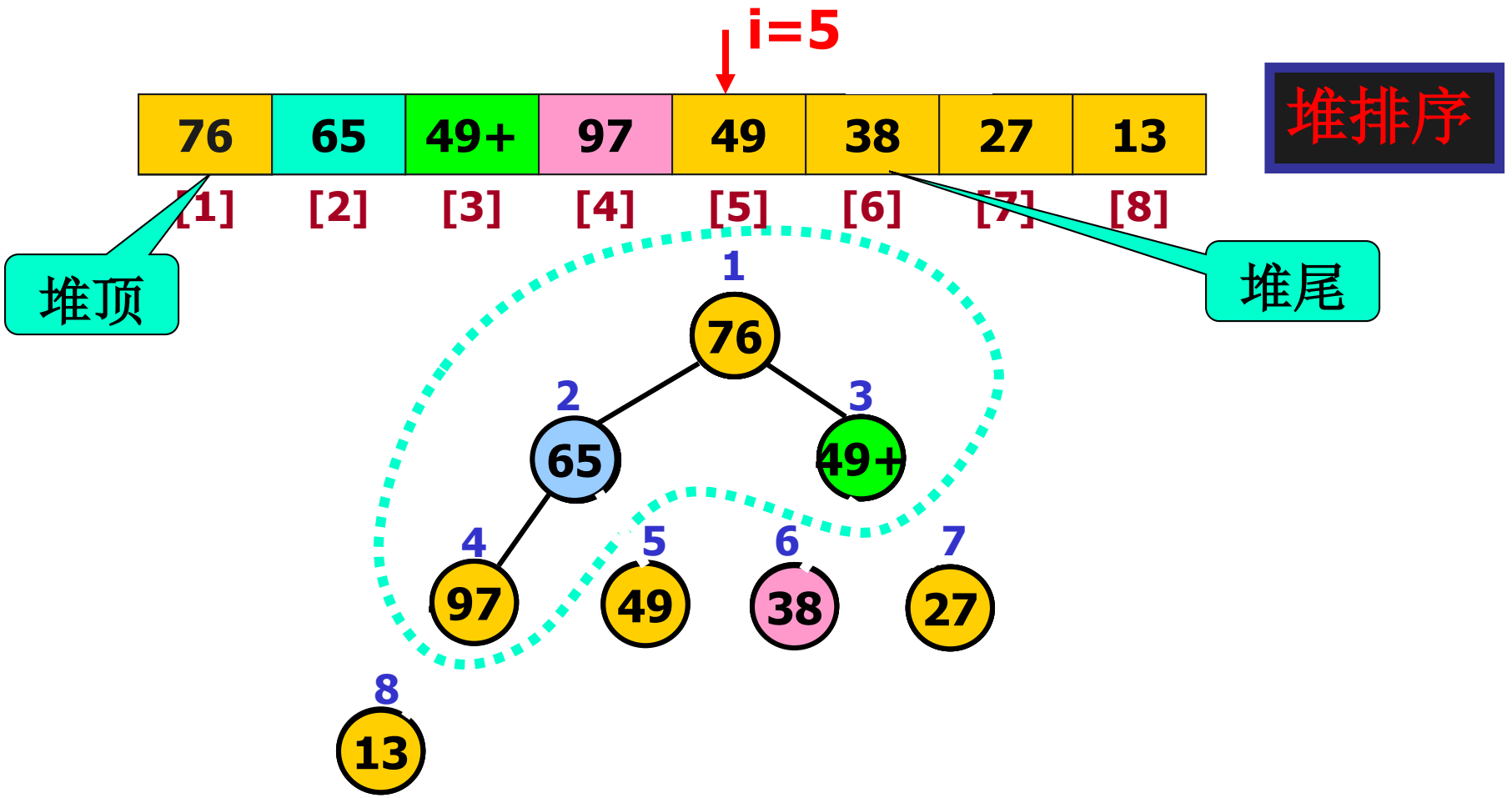
(2) 重新堆化 H.r[1..i-1]



(1) H.r[1] ↔ H.r[i]　　堆尾和堆顶元素交换

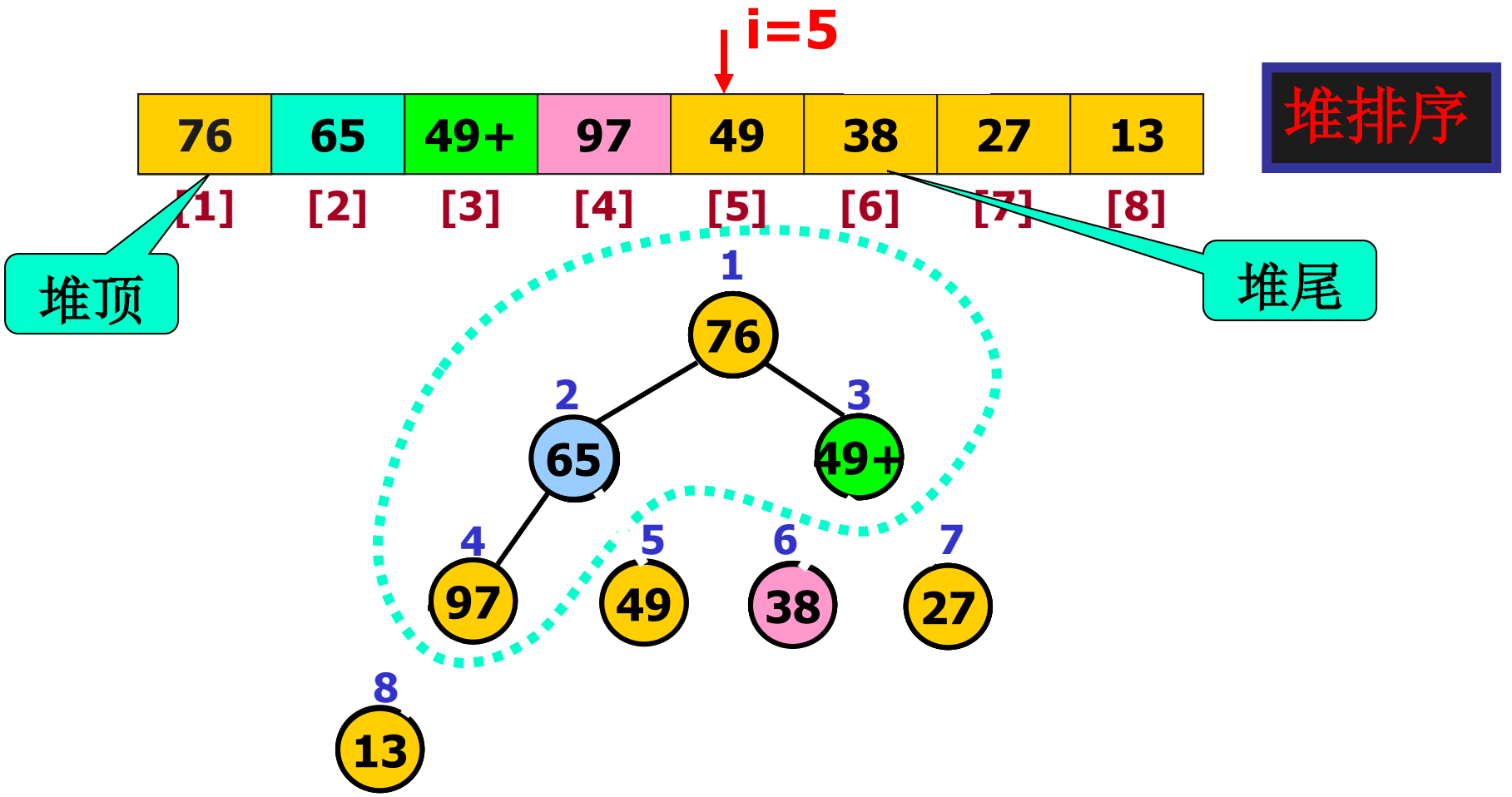


(1) H.r[1] ↔ H.r[i]　　堆尾和堆顶元素交换



(1) H.r[1] ↔ H.r[i]　　堆尾和堆顶元素交换

(2) 重新堆化 H.r[1..i-1]

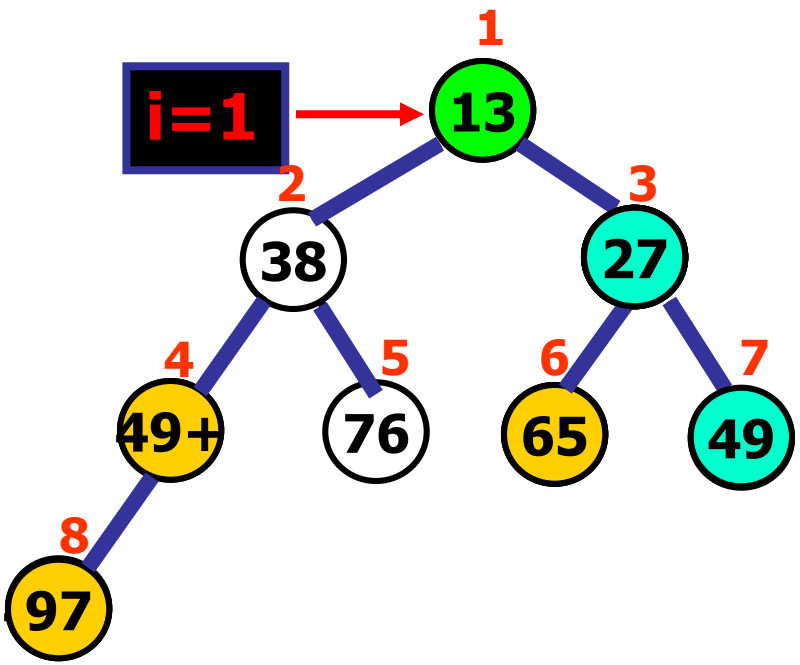


将记录按从大到小排列： 建“小”顶堆

将记录按从小到大排列： 建“大”顶堆

堆的构造

13	38	27	49+	76	65	49	97
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]



从最后一个“非叶子结点”开始，自下而上进行“堆化”

性能分析

(1) 空间性能

堆排序属于原地排序

(2) 时间性能

(a) 建n个元素的新堆，总的关键字比较次数：？

(b) 对n个结点的堆排序时，重新堆化过程中的总的关键字比较次数：？

(a) 建n个元素的新堆，总的关键字比较次数：？

对于每个非终端结点来说，最多进行两次比较操作，因此整个构建端的时间复杂度是O（n）。

(b) 对n个结点的堆排序时，重新堆化过程中的总的关键字比较次数：？

设树高度为h，　$h=\lfloor \log_2 n \rfloor + 1$

完全二叉树的某个结点到根结点的距离为$[\log_2 i]+1$，并且需要进行n-1次重新堆化，因此重新堆化的时间复杂度为O（nlogn）。

结论：

性能分析

(1) 堆排序的运行时间主要耗费在建初始堆和调整建新堆时的反复“筛选”上

(2) 对n个元素的数组，建初始堆的关键字比较次数不超过4n，重新堆化的总的比较次数不超过$2n\log_2 n$

(3) 堆排序在最坏情况下的时间性能达到O(nlogn)

(4) 堆排序对记录数较少时不值得提倡，但是对n较大的文件很有效

(5) 堆排序是不稳定的排序算法