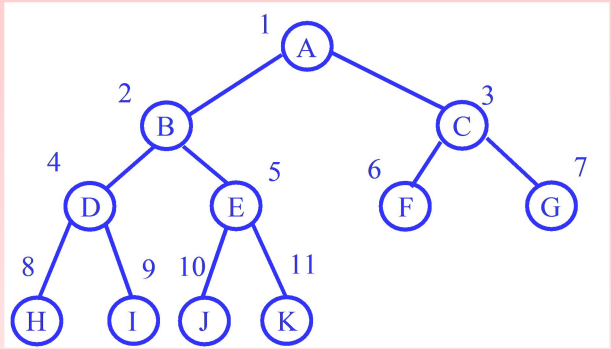
本节主题：

堆排序

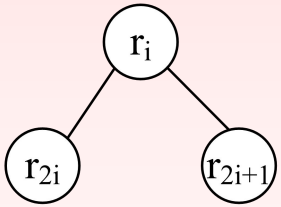
回顾：二叉树的顺序存储(将顺序表看成二叉树)

- 按编号次序存储结点
  - 对树中每个节点进行编号
  - 其编号从小到大的顺序就是节点在连续存储单元的先后次序。



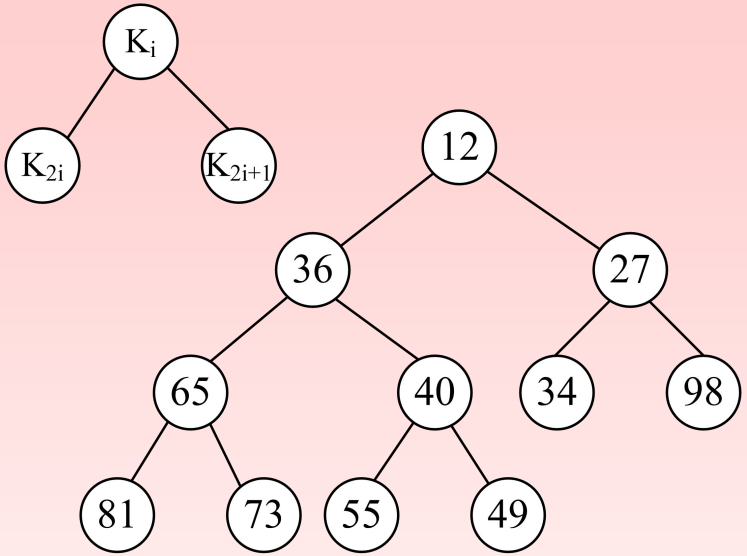
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	...
--	A	B	C	D	E	F	G	H	I	J	K		

- 编号为i的节点的左孩子节点的编号为2i；右孩子节点的编号为(2i+1)。
- 除树根节点外,若一个节点的编号为i，则它的双亲节点的编号为⌊i/2⌋
- 若将数列R[1..n]视作完全二叉树，则 r_{2i} 是 r_i 的左孩子； r_{2i+1} 是 r_i 的右孩子。



堆的定义和堆排序

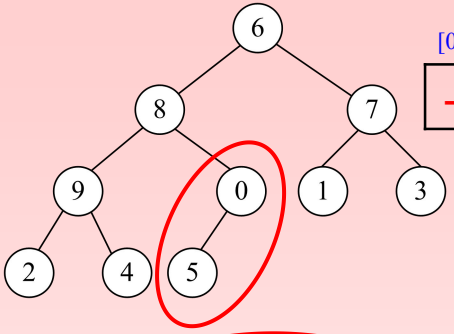
- 堆的定义
  - n个关键字序列$K_1,K_2,\ldots,K_n$称为**堆**，当且仅当该序列满足如下性质
  - （1）$K_i \le K_{2i}$ 且 $K_i \le K_{2i+1}$ 小根堆
  - 或（2）$K_i \ge K_{2i}$ 且 $K_i \ge K_{2i+1}$（$1 \le i \le \lfloor n/2 \rfloor$） 大根堆
- 堆排序思想
  - 堆排序是一树形选择排序
  - 在排序过程中，将R[1..n]看成是一棵完全二叉树的顺序存储结构，利用完全二叉树中双亲结点和孩子结点之间的内在关系，在当前无序区中选择关键字最大（或最小）的记录。



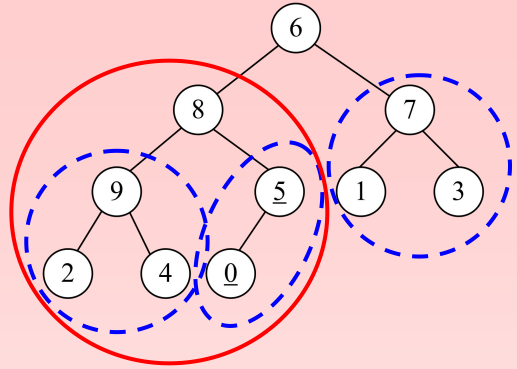
{12, 36, 27, 65, 40, 34, 98, 81, 73, 55, 49} - 小根堆

{12, 36, 27, 65, 40, **14**, 98, 81, 73, 55, 49}- 不是堆

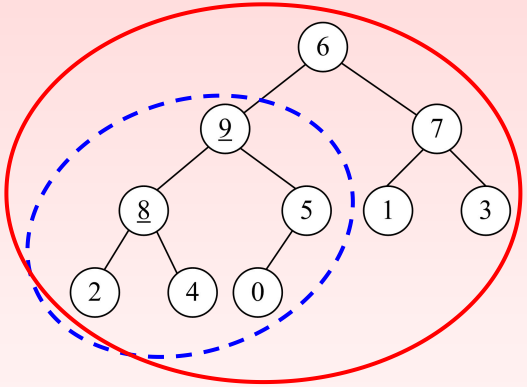
构造初始堆(大根堆)

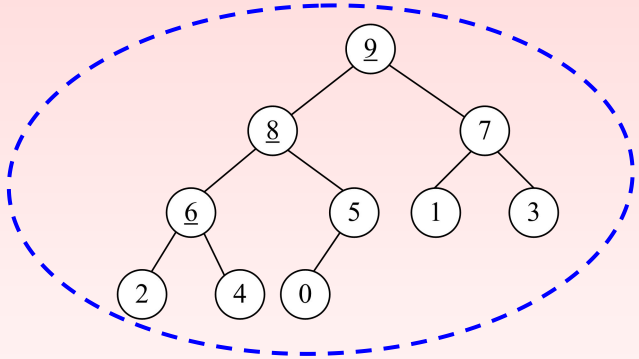


[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	...
-	6	8	7	9	0	1	3	2	4	5		



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	...
-	6	8	7	9	5	1	3	2	4	0		





[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	...
-	6	9	7	8	5	1	3	2	4	0		

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	...
-	9	8	7	6	5	1	3	2	4	0		

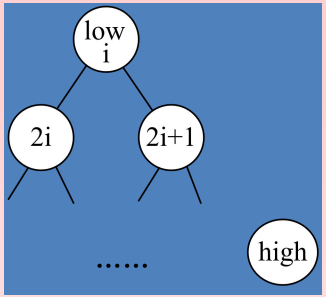
- 方法：以节点i作根，其左、右子树均已成堆，将与“大”分支的根交换，直到成堆。——大者上浮，小者被筛选下去。

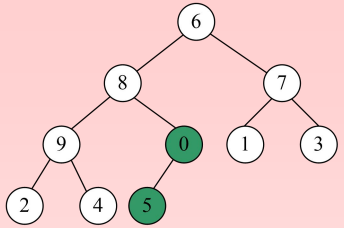
构造初始堆算法

```
for (i=n/2;i>=1;i--)
  sift(R,i,n);
```

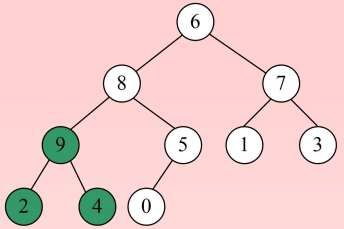
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	...
-	9	8	7	6	5	1	3	2	4	0		

```
void sift(RecType R[],int low,int high)
{
   int i=low,j=2*i;  //R[j]是R[i]的左孩子
   RecType temp=R[i];
   while (j<=high)
   {
      if (j<high && R[j].key<R[j+1].key)
         j++;  //若右孩较大，j指向右孩2i+1
      if (temp.key<R[j].key)
      {
         R[i]=R[j];  //将R[j]调整到双亲结点位置上
         i=j;        //修改i和j值,以便继续向下筛选
         j=2*i;
      }
      else break;    //筛选结束
   }
   R[i]=temp;        //被筛选结点的值放入最终位置
}
```

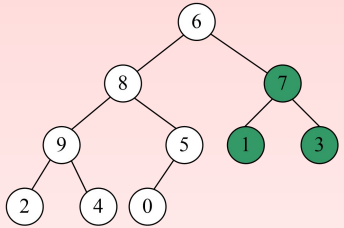




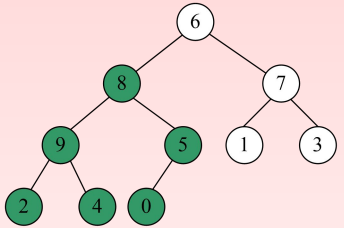
sift(R,5,10)



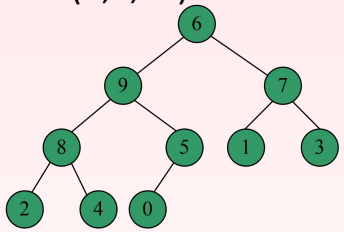
sift(R,4,10)



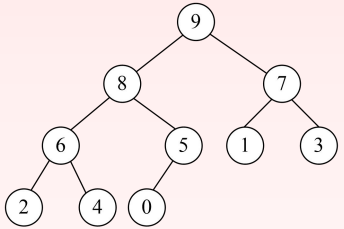
sift(R,3,10)



sift(R,2,10)

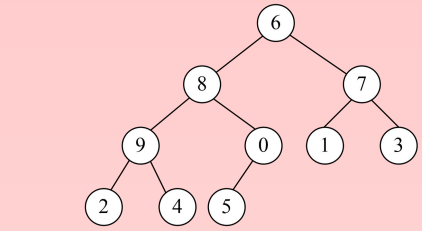


sift(R,1,10)

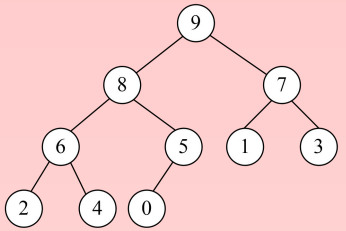


堆排序算法

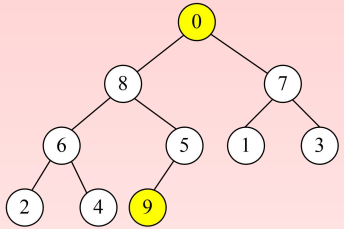
```
void HeapSort(RecType R[],int n)
{
   int i;
   RecType temp;
   //循环建立初始堆
   for (i=n/2; i>=1; i--)
      sift(R,i,n);
   //选最大值置后并调整堆
   for (i=n; i>=2; i--)
   {
      temp=R[1];
      R[1]=R[i];
      R[i]=temp;
      sift(R,1,i-1);
   }
}
```



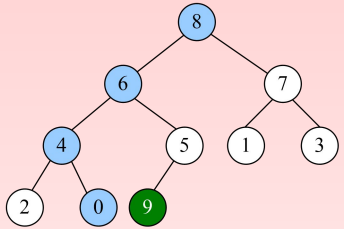
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	...
-	6	8	7	9	0	1	3	2	4	5		



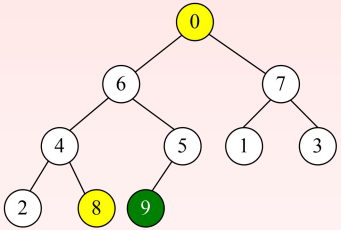
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	...
-	9	8	7	6	5	1	3	2	4	0		



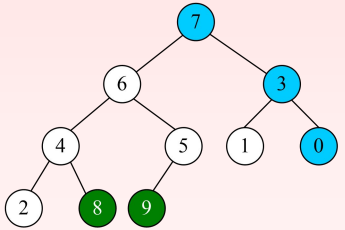
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	...
-	0	8	7	6	5	1	3	2	4	9		



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	...
-	8	6	7	4	5	1	3	2	0	9		



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	...
-	0	6	7	4	5	1	3	2	8	9		



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	...
-	7	6	3	4	5	1	0	2	8	9		

.....

堆排序的时间复杂度分析

1．对深度为 k 的堆，“筛选”所需进行的关键字比较的次数

至多为$2(k-1)$；

2．对 n 个关键字，建成深度为$h(=\lfloor \log_2 n \rfloor+1)$的堆所需进行的关键字比较的次数

不超过$4n$

3．调整“堆顶” n-1 次，总共进行的关键字比较的次数

不超过$2(\lfloor \log_2(n-1) \rfloor+ \lfloor \log_2(n-2) \rfloor+ \ldots+\log_2 2) < 2n(\lfloor \log_2 n \rfloor)$

——堆排序的时间复杂度为$O(n\log_2 n)$。

思考题

- 选择排序中的有序区是全局有序吗？