



选择排序

- 基本思想：每次从待排序的数据元素中**选取关键字值最小**的数据元素，顺序放在已排序的数据元素的最后，直到全部排完。
- 方法：**简单选择排序，堆排序**



简单（直接）选择排序

- 进行 $n-1$ 趟扫描
- 第1趟扫描选出 n 个数据元素中关键字值最小的数据元素，与第1个数据元素交换；
-
- 第 i 趟扫描选出剩下的 $n-i+1$ 个数据元素中关键字值最小的数据元素，并与第 i 个数据元素交换；
-



简单（直接）选择排序

8 3 9 6 1

第一趟 1 3 9 6 8

第二趟 1 3 9 6 8

第三趟 1 3 6 9 8

第四趟 1 3 6 8 9



简单（直接）选择排序

```
void SelectSort(SqList &L)
{ int i,j,k;
  for (i=1,i<L.length;i++)
  { k=i;
    for(j=i+1;j<=L.length;++j)
    if ( L.r[j].key <L.r[k].key)  k=j;
    if(k!=i) { L.r[0]=L.r[i];
              L.r[i]=L.r[k];  L.r[k]=L.r[0]; }
  }
}
```



简单（直接）选择排序

- 时间复杂度为 $O(n^2)$ 。
- 适用于待排序元素较少的情况。
- 一个额外的辅助空间, $O(1)$ 。
- 简单选择排序是不稳定排序法。请用简单选择排序法对 2, 2, 1 进行排序, 考察两个 2 在排序前后的相对位置。

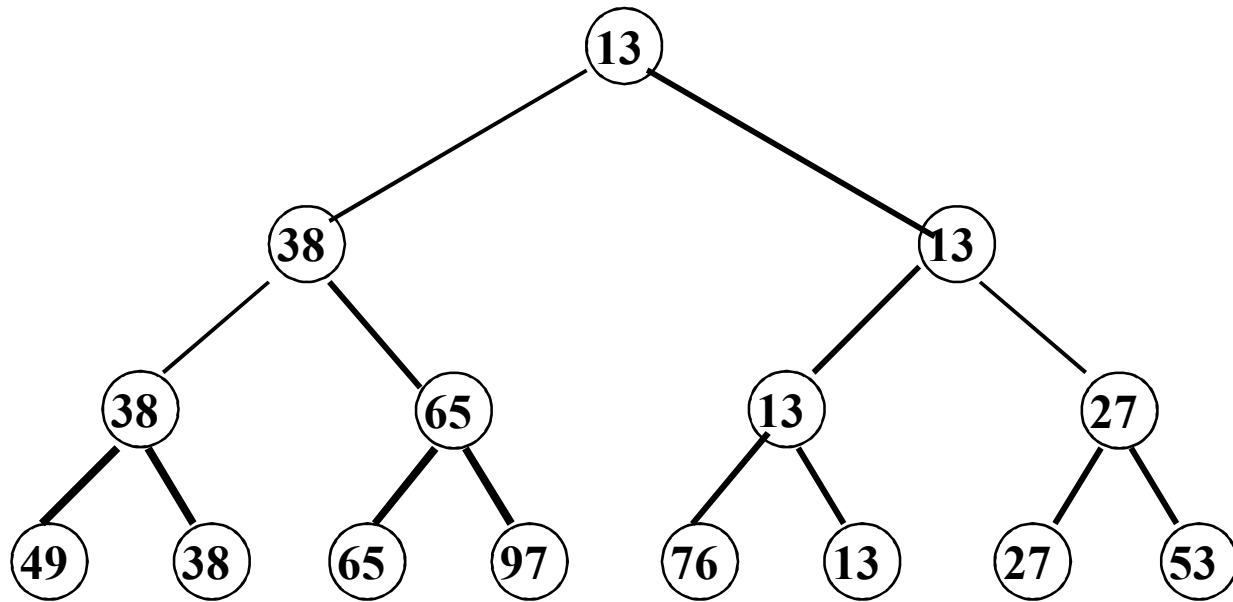


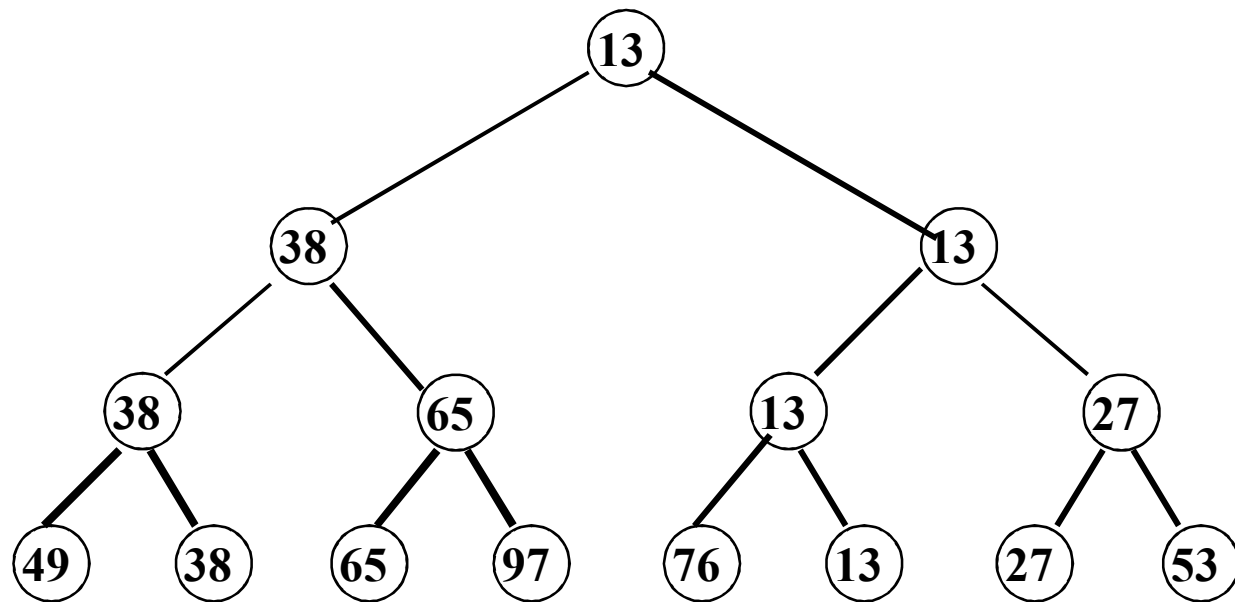
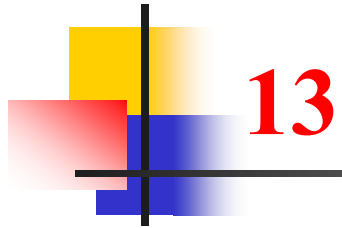
简单（直接）选择排序

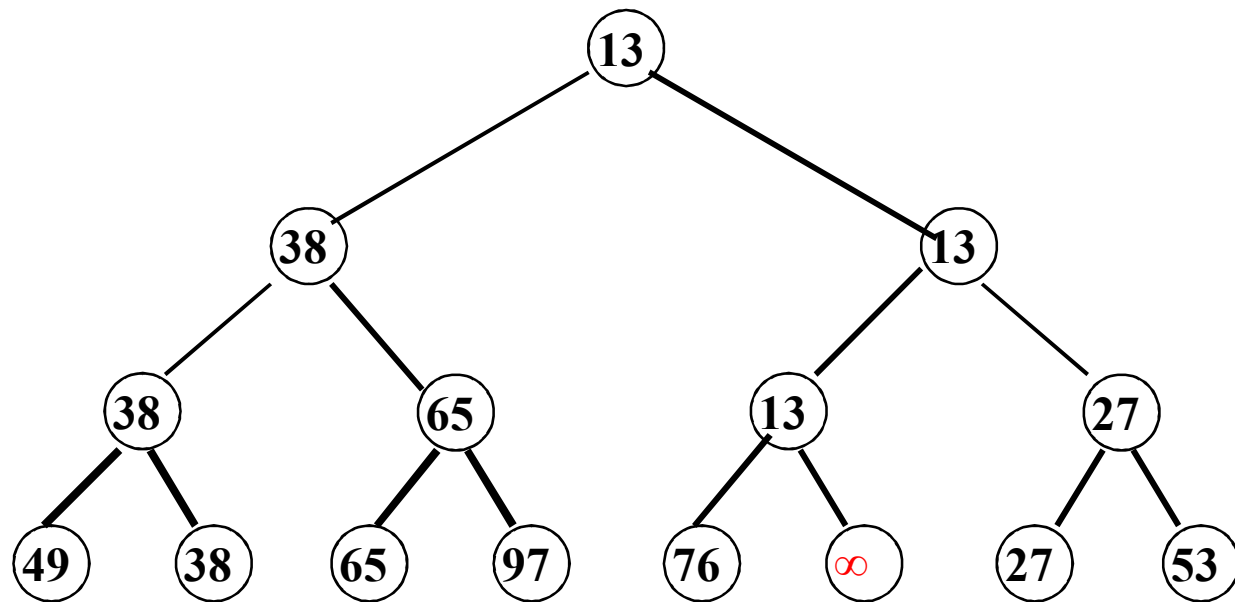
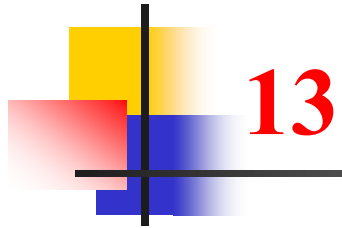
- 最好情况：
 - $(n-1)+(n-2)+\cdots+2+1=n(n-1)/2$ 次数据比较，0次数据移动。如：待排序的数据已经有序(正序)
 - 最好情况时间复杂度 $O(n^2)$
- 最坏情况：
 - $(n-1)+(n-2)+\cdots+2+1=n(n-1)/2$ 次数据比较， $3(n-1)$ 次数据移动。
 - 最好情况时间复杂度 $O(n^2)$



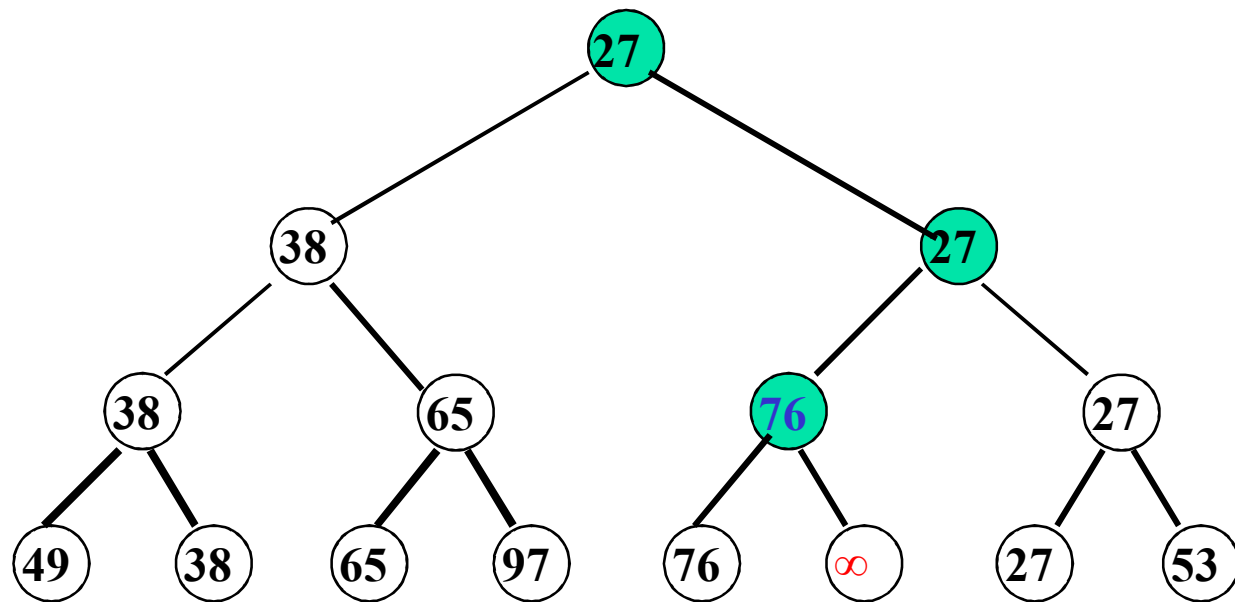
树排序

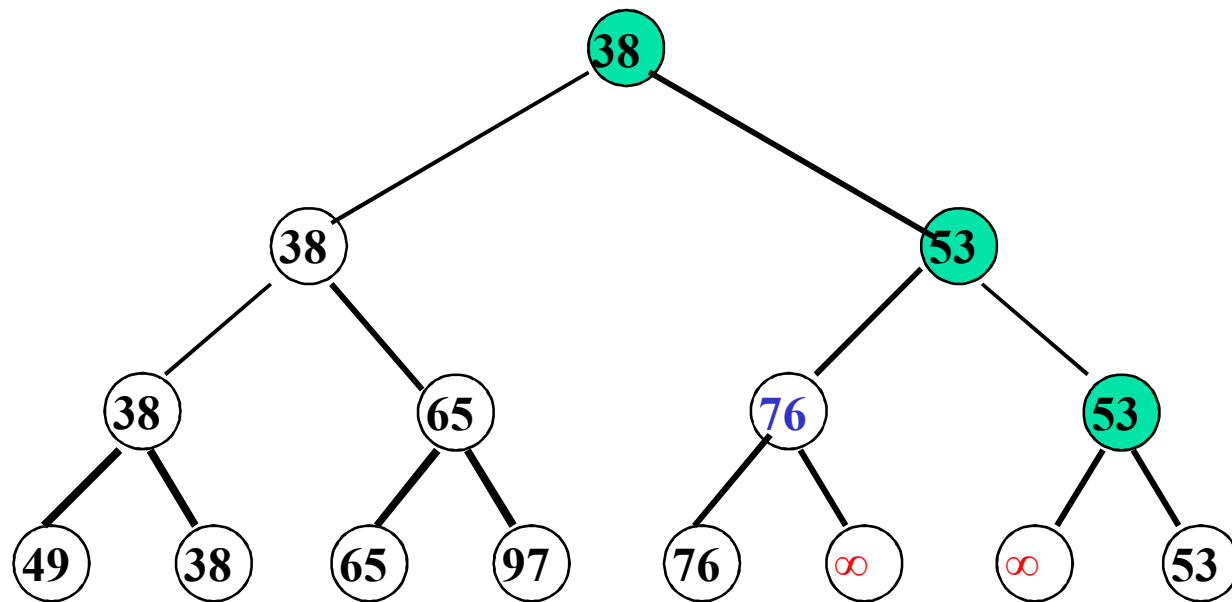
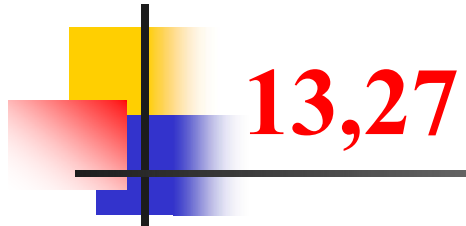






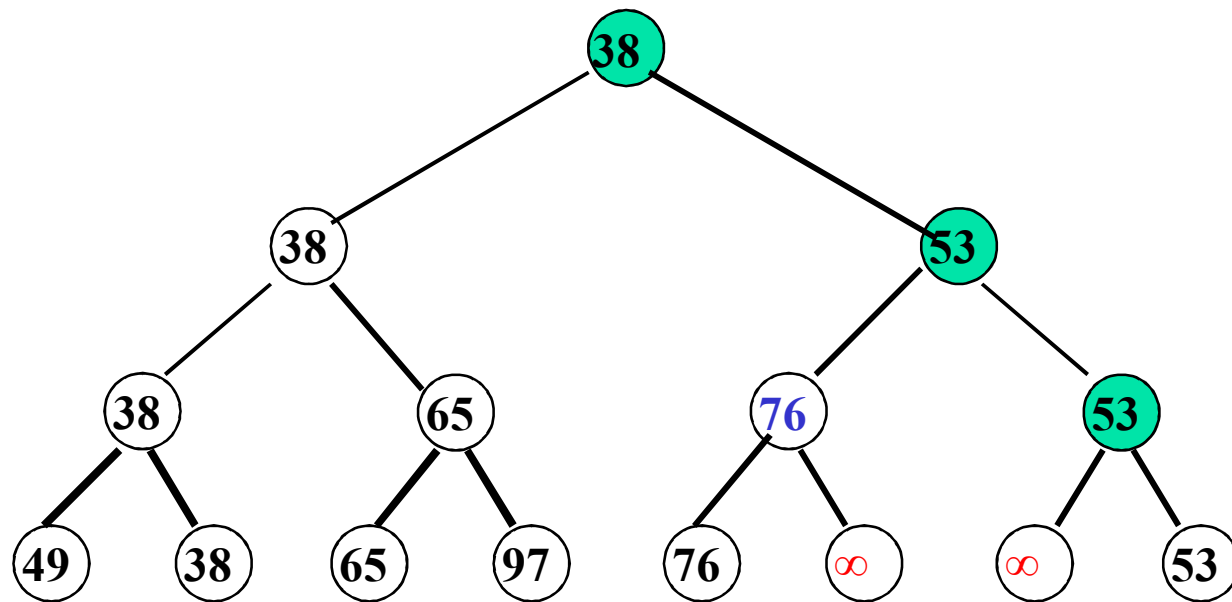
13,27



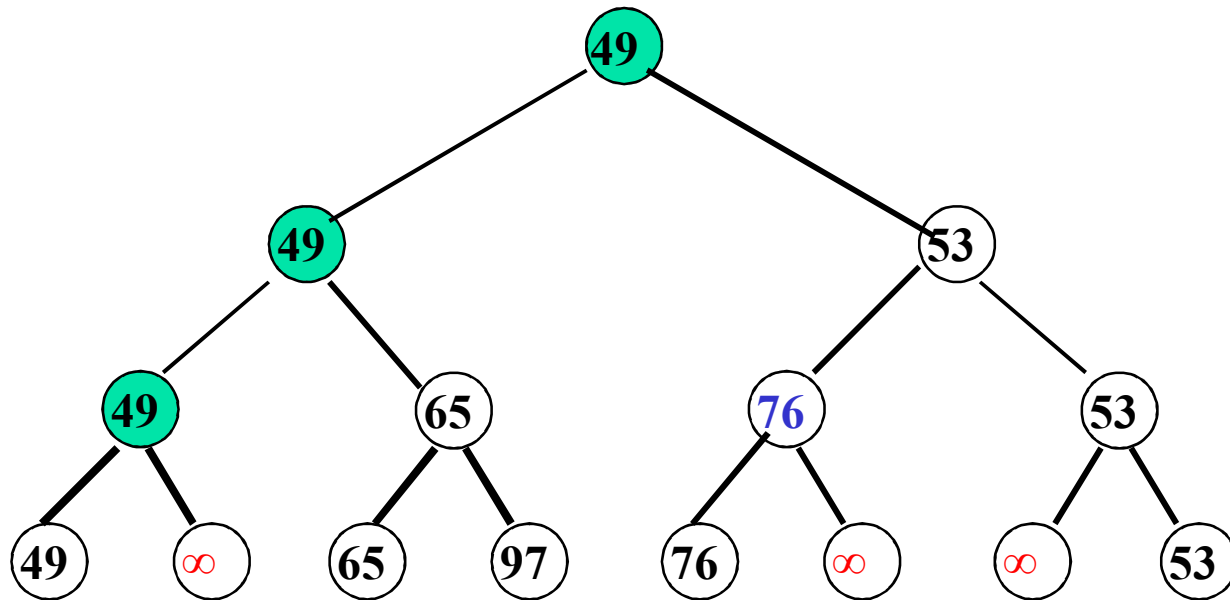




13,27,38



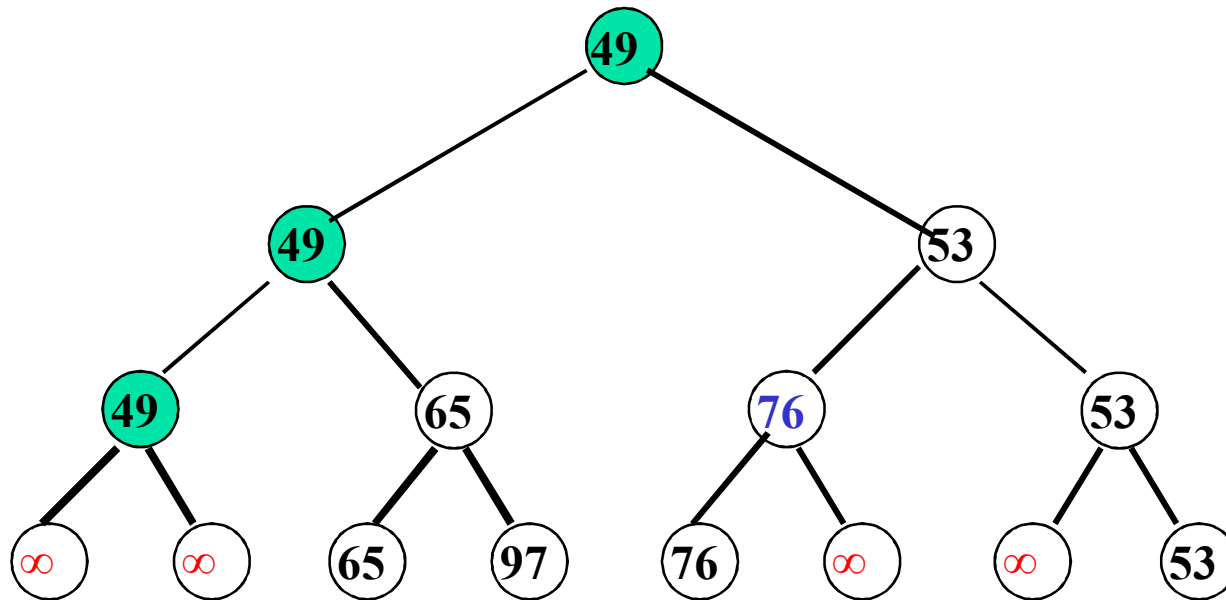
13,27,38





13,27,38,49

树排序将时间复杂度将为 $O(n\log n)$
但需要的辅助空间增加



堆排序

小顶堆

- 堆：是一个键值序列 $\{k_1, k_2, \dots, k_n\}$, 并具有如下特性：
 $k_i \leq k_{2i}, k_i \leq k_{2i+1} \quad (i=1, 2, \dots, n/2);$

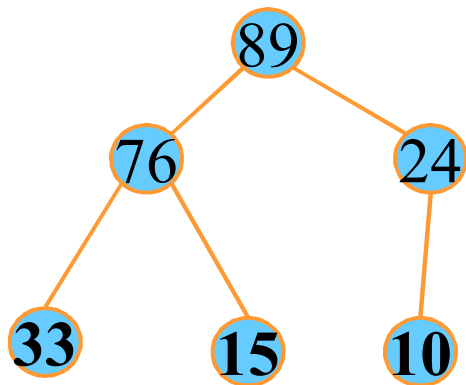
或

$$k_i \geq k_{2i}, k_i \geq k_{2i+1} \quad (i=1, 2, \dots, n/2)。$$

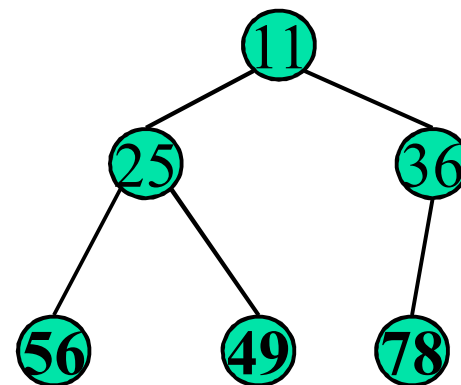
大顶堆

堆用一个一维数组存放，可看成是一棵**完全二叉树**的**顺序存储表示**（即按完全二叉树的层次编号顺序存放）

89	76	24	33	15	10			11	25	36	56	49	78		
----	----	----	----	----	----	--	--	----	----	----	----	----	----	--	--



大顶堆



小顶堆

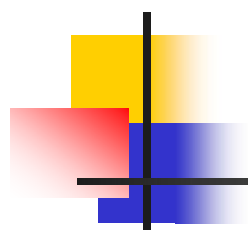
小（**大**）顶堆的特性在此完全二叉树解释为：完全二叉树中任一非叶结点的值均不大（**小**）于它左、右孩子结点的值。

根结点是完全二叉树中值最**小**（**大**）的结点，也是堆中的最**小**（**大**）值。

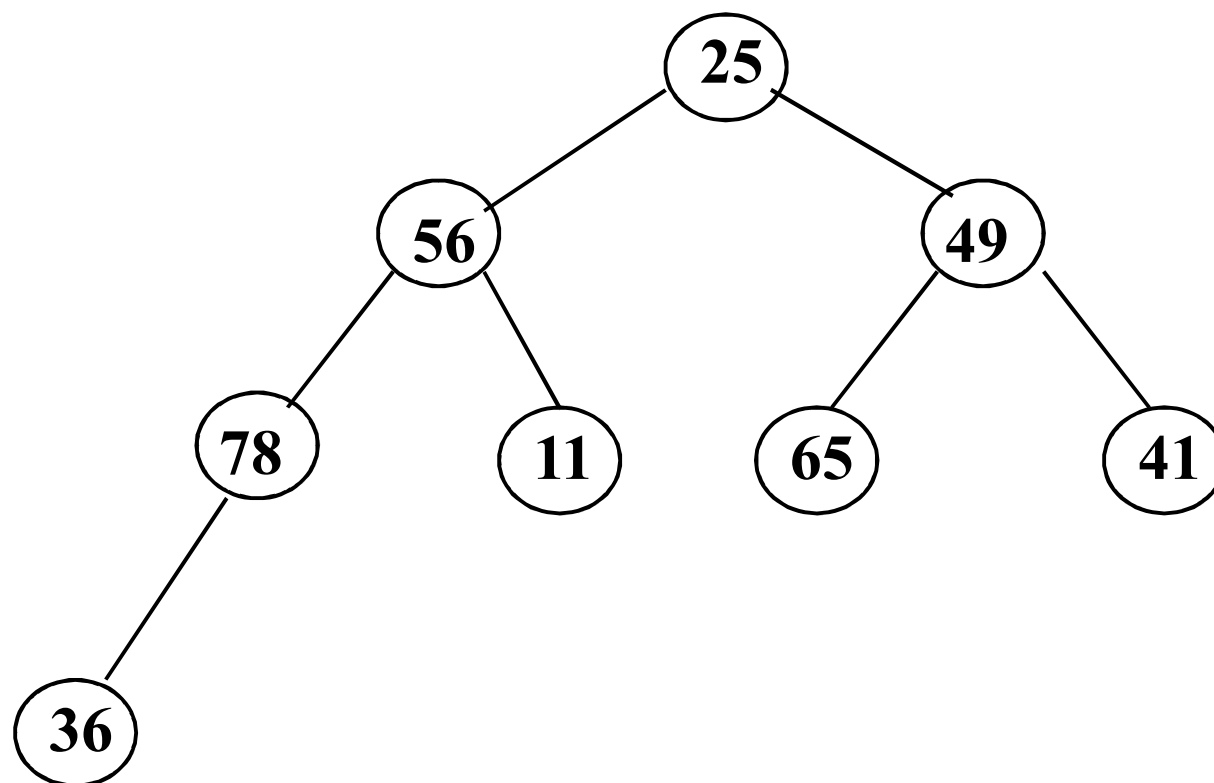


堆排序方法

- 对一组待排序的数据元素，将它们建成一个堆（称为**初建堆**），关键字值最大的数据为堆序列的第一个数据。**建堆方法：筛选法、插入法**
- 将关键字值最大的数据取出（通常与尚未排序的最后一个数据交换存储位置），
- 用剩下的数据元素重建堆（称为一次调整），便得到关键字值次大的数据元素
- 如此反复，直到全部数据排好序。



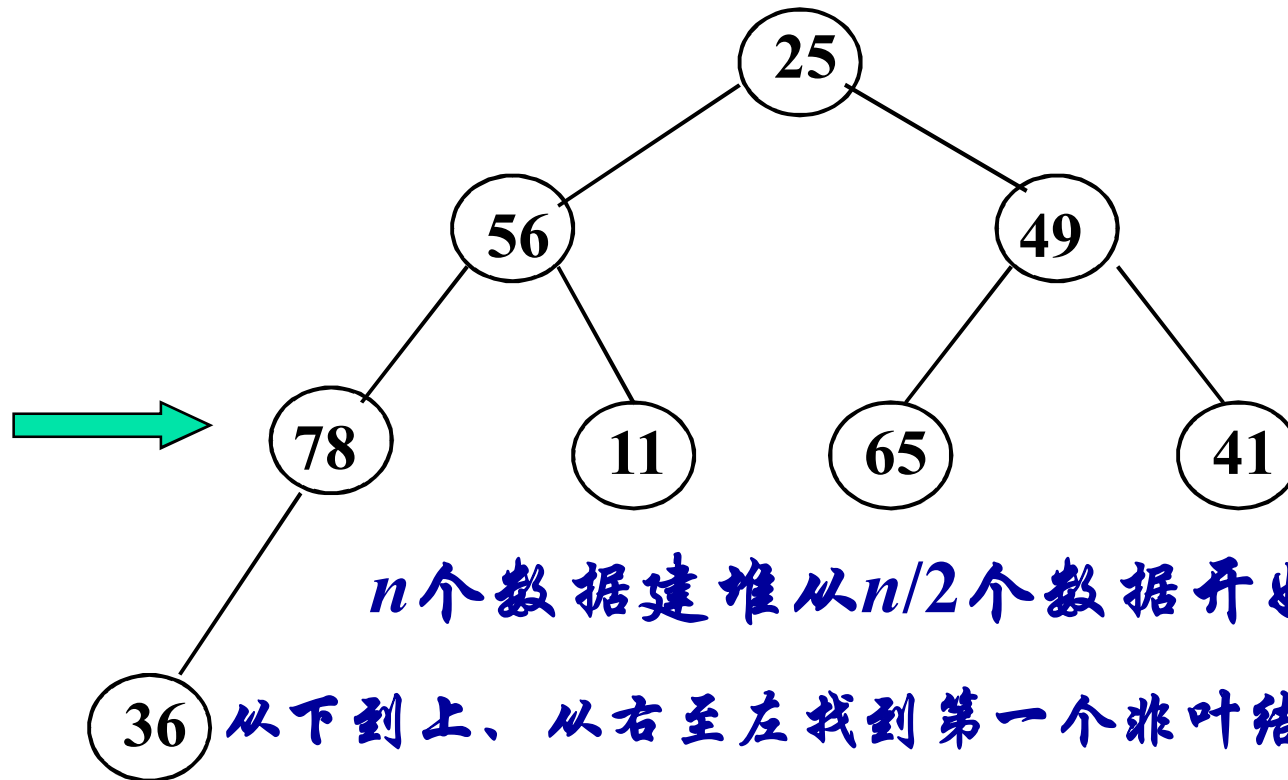
1	2	3	4	5	6	7	8
25	56	49	78	11	65	41	36



筛选法建堆过程演示

以建大顶堆为例，对某个结点进行调整：保证以该结点为根的子树为大顶堆

1	2	3	4	5	6	7	8
25	56	49	78	11	65	41	36



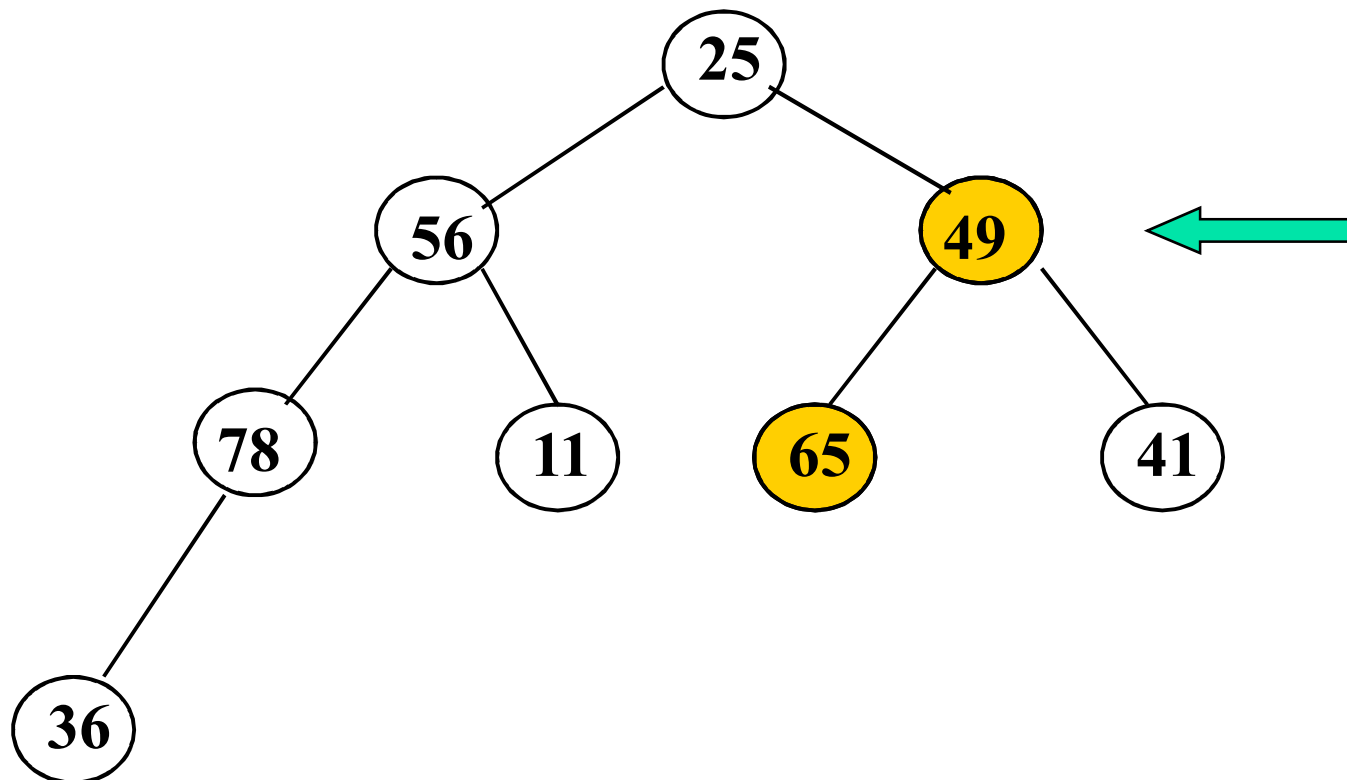
n 个数据建堆从 $n/2$ 个数据开始进行调整

36 从下到上、从右至左找到第一个非叶结点开始进行调整

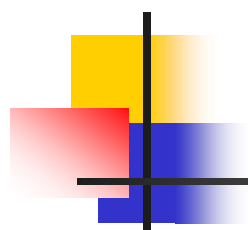
筛选法建堆过程演示

一个结点调整结束后，按照**下到上、从右至左**的顺序找到下一个结点继续调整，直至根结点

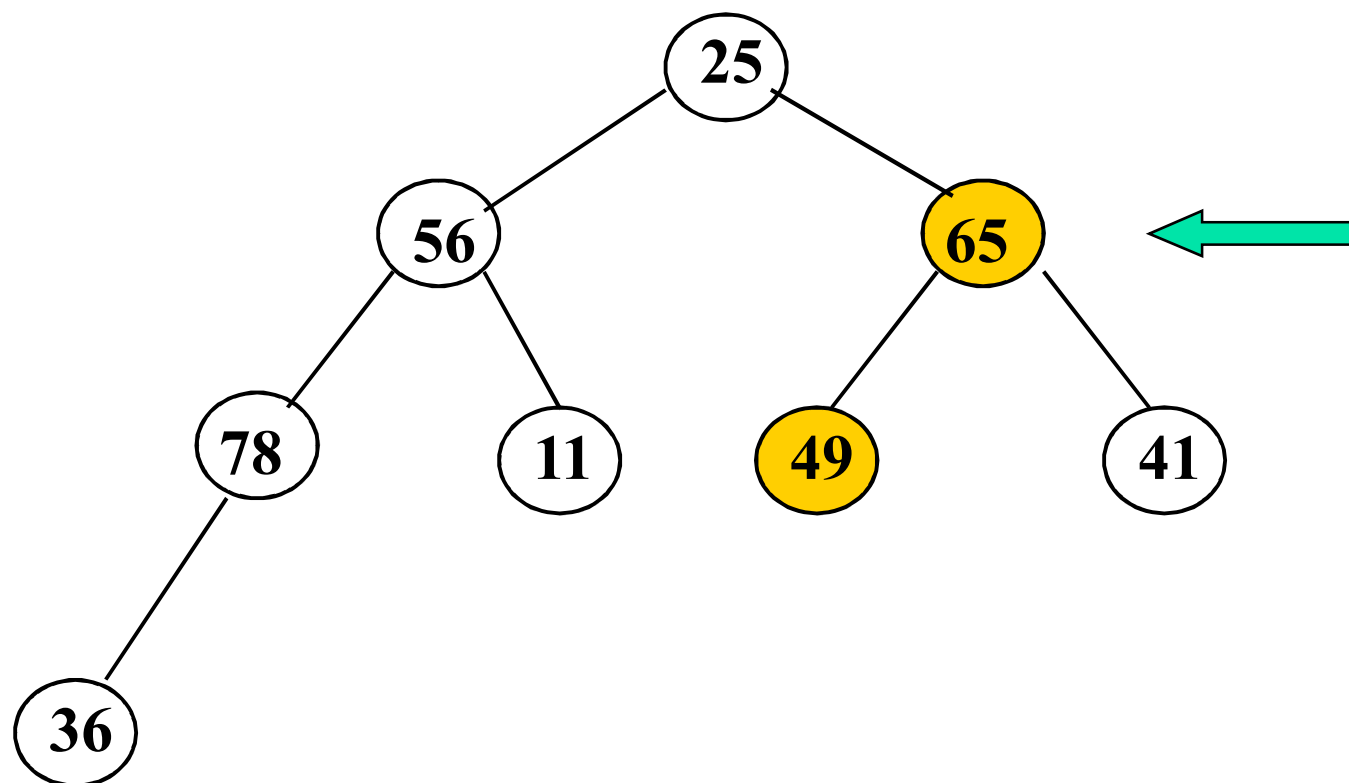
1	2	3	4	5	6	7	8
25	56	49	78	11	65	41	36



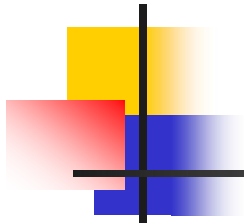
筛选法建堆过程演示



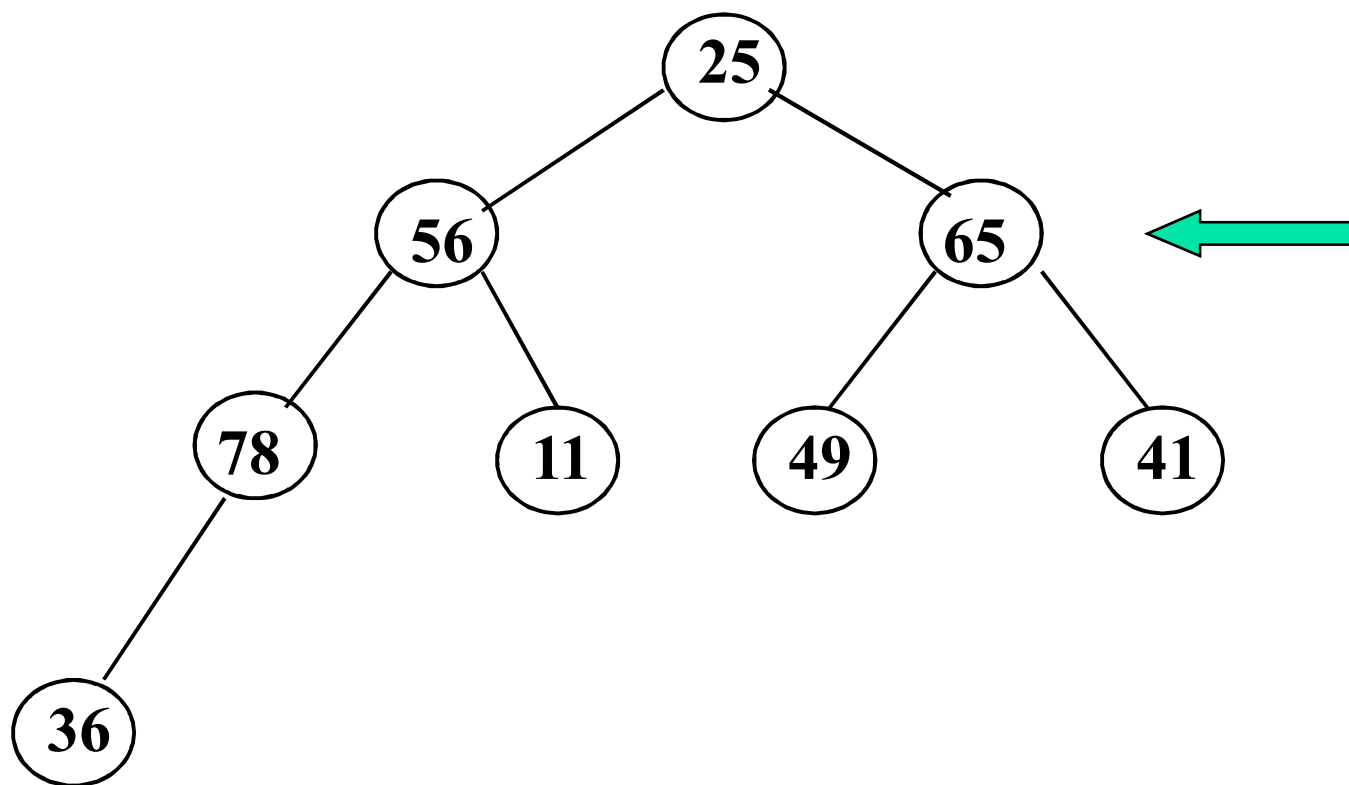
1	2	3	4	5	6	7	8
25	56	65	78	11	49	41	36



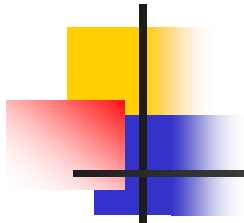
筛选法建堆过程演示



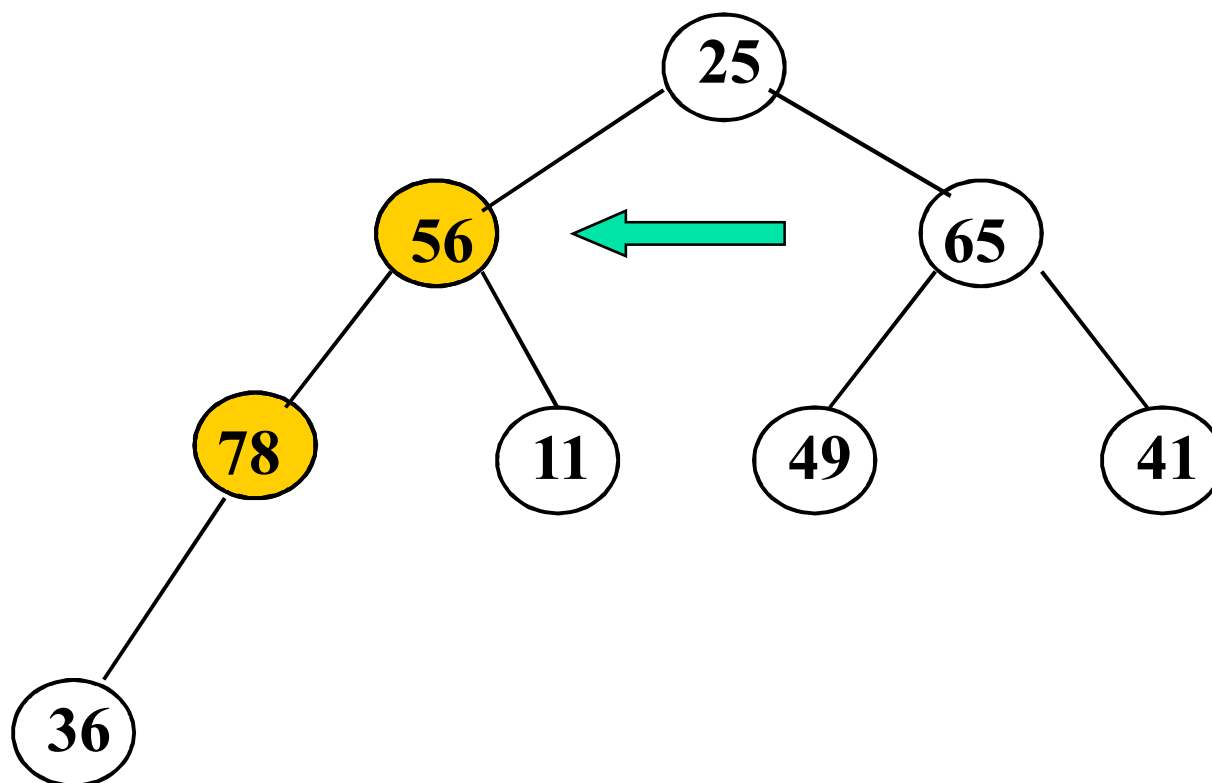
1	2	3	4	5	6	7	8
25	56	65	78	11	49	41	36



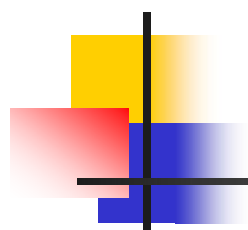
筛选法建堆过程演示



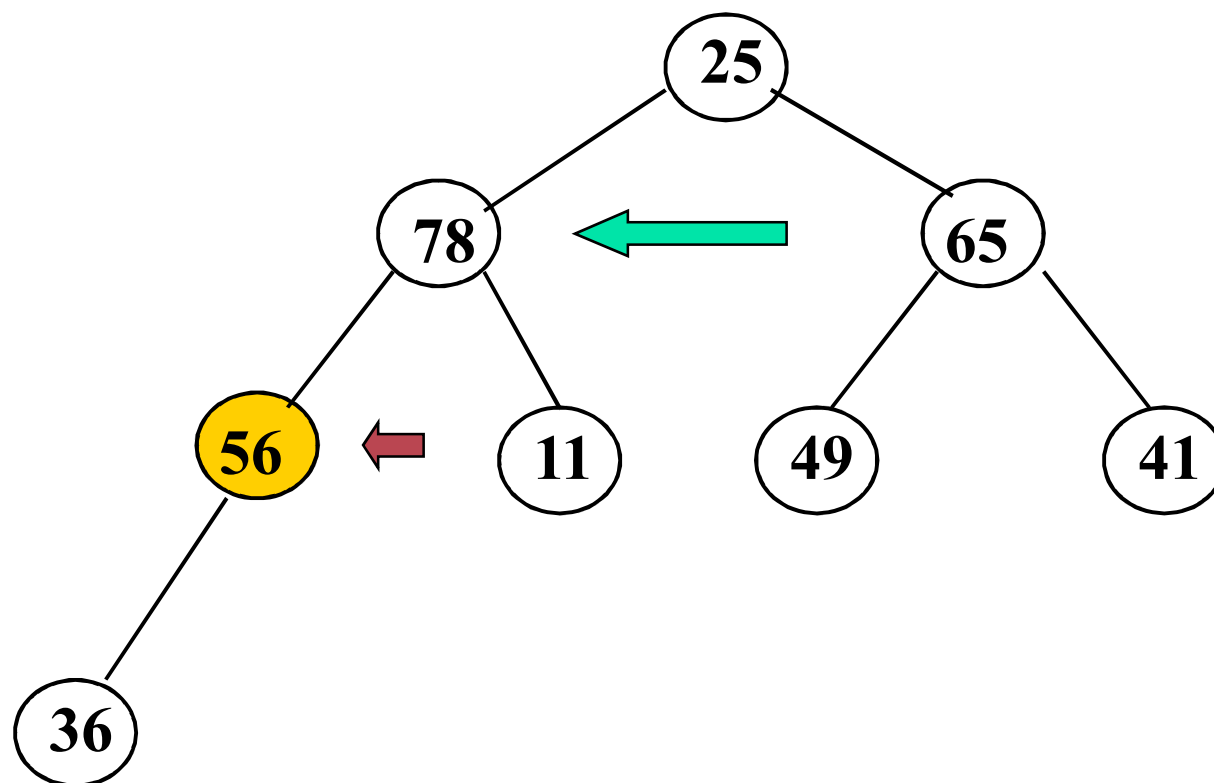
1	2	3	4	5	6	7	8
25	56	65	78	11	49	41	36



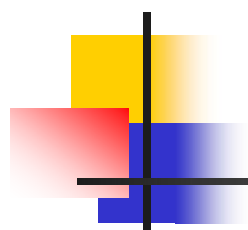
筛选法建堆过程演示



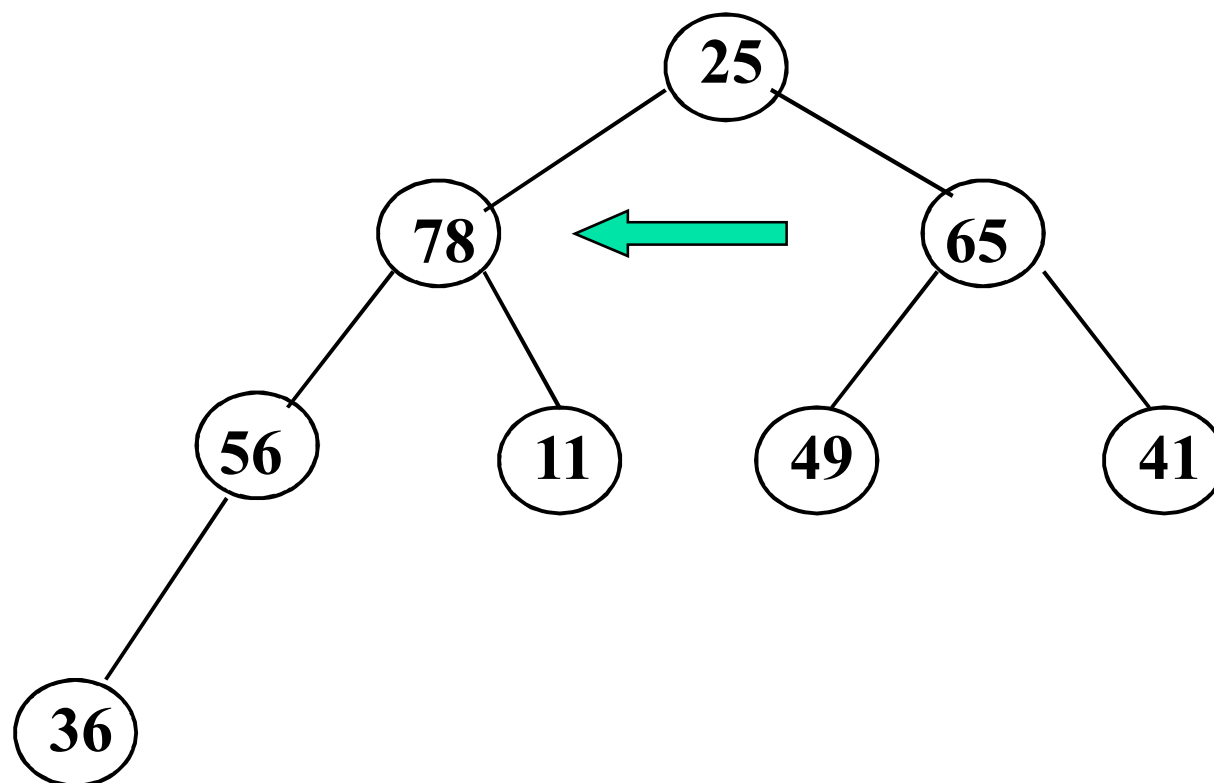
1	2	3	4	5	6	7	8
25	78	65	56	11	49	41	36



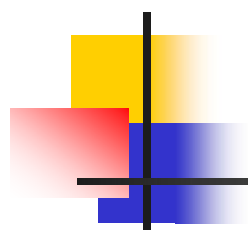
筛选法建堆过程演示



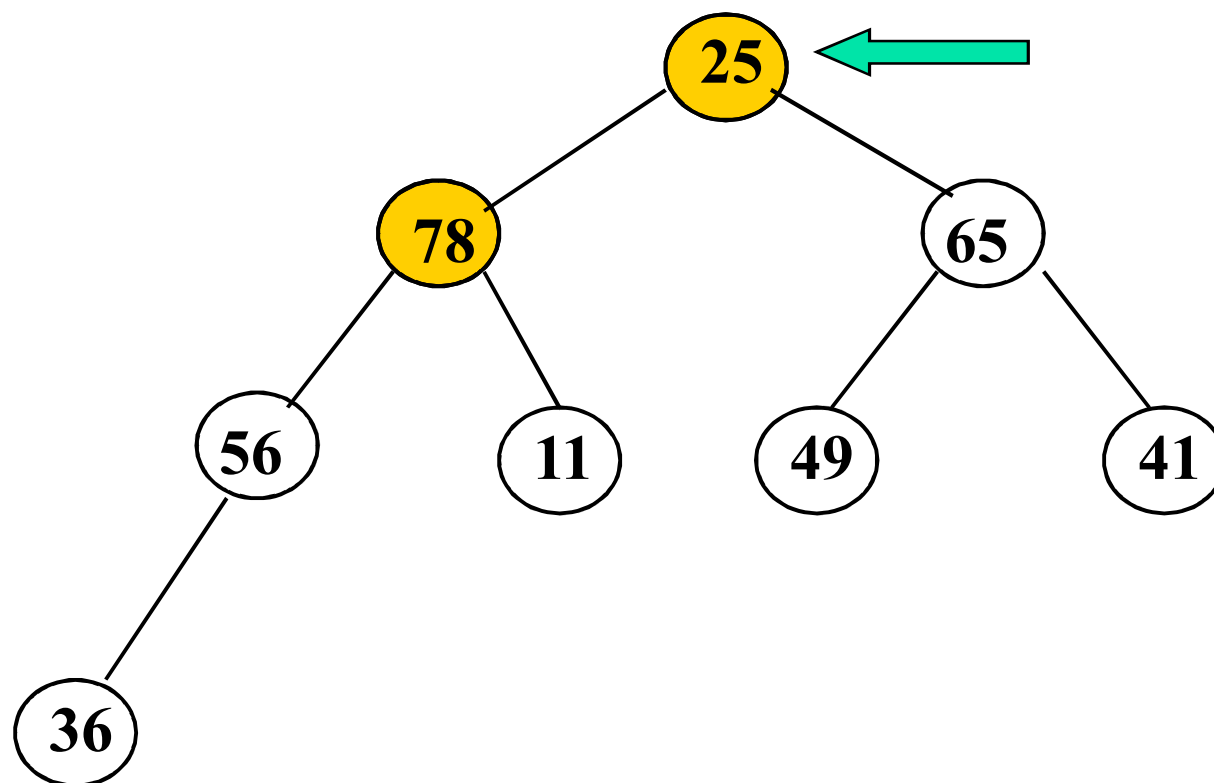
1	2	3	4	5	6	7	8
25	78	65	56	11	49	41	36



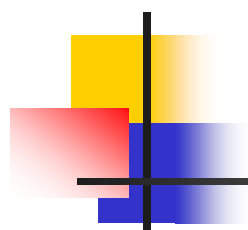
筛选法建堆过程演示



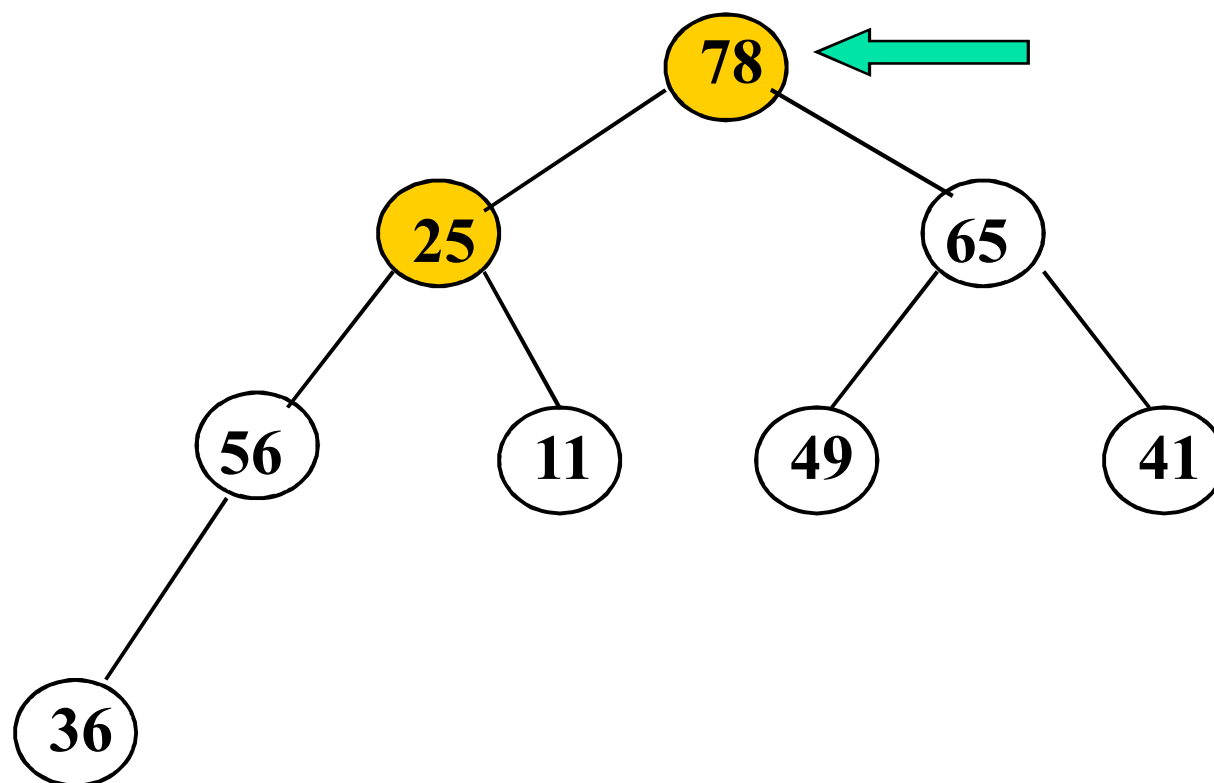
1	2	3	4	5	6	7	8
25	78	65	56	11	49	41	36



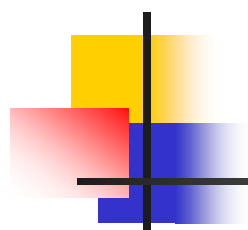
筛选法建堆过程演示



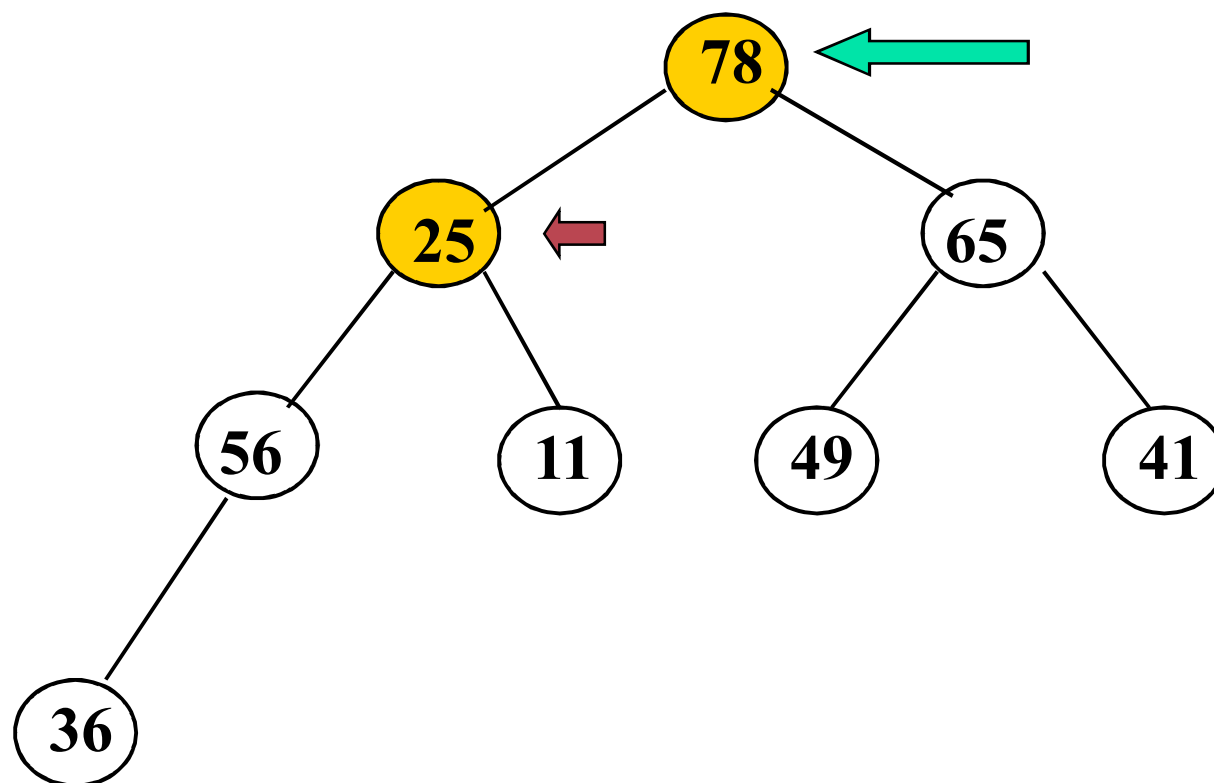
1	2	3	4	5	6	7	8
78	25	65	56	11	49	41	36



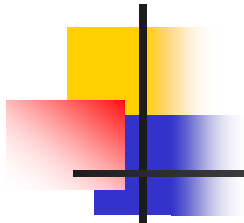
筛选法建堆过程演示



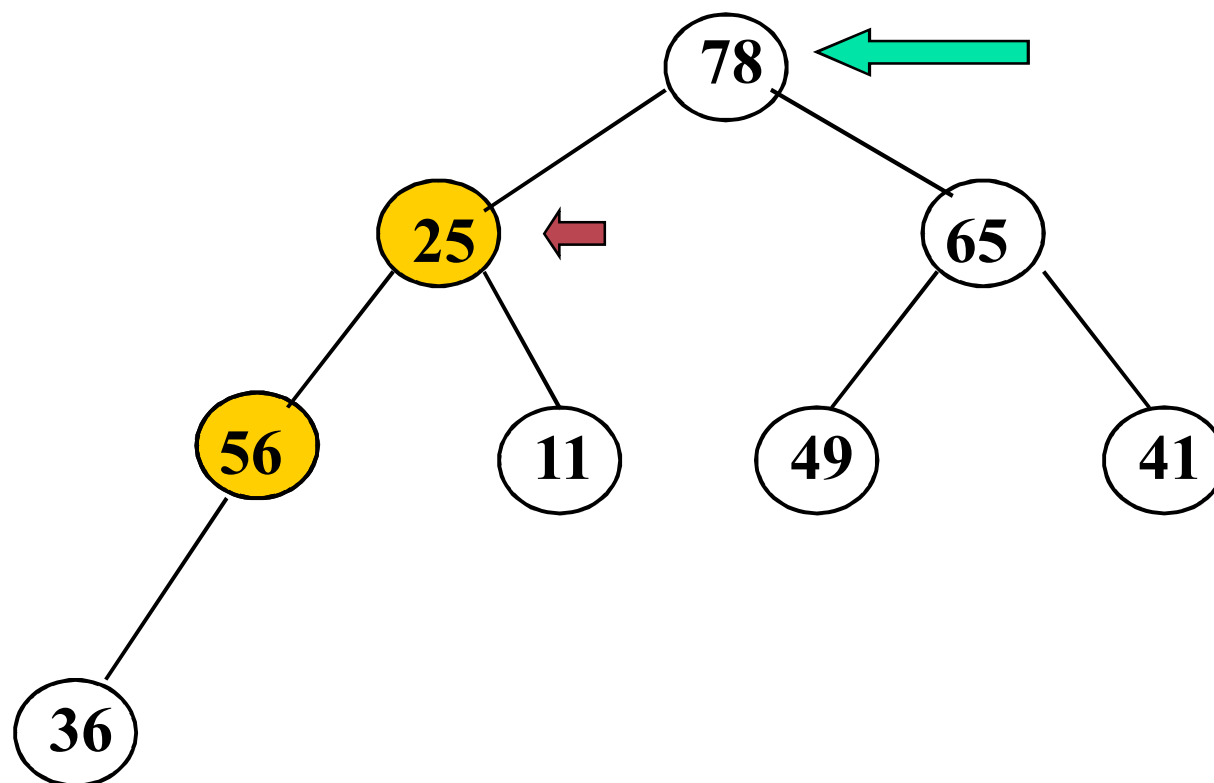
1	2	3	4	5	6	7	8
78	25	65	56	11	49	41	36



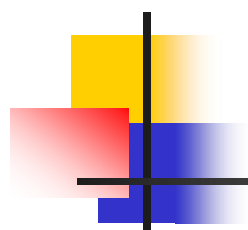
筛选法建堆过程演示



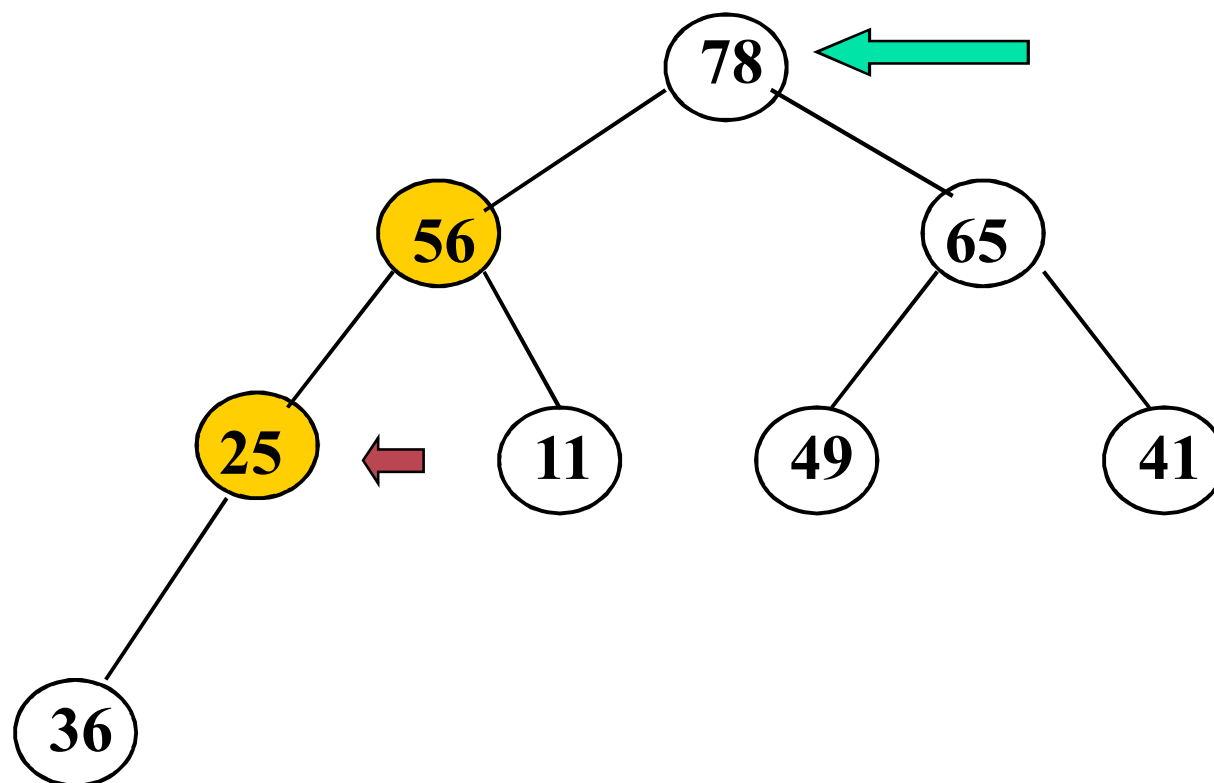
1	2	3	4	5	6	7	8
78	25	65	56	11	49	41	36



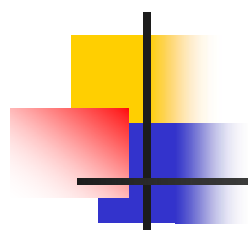
筛选法建堆过程演示



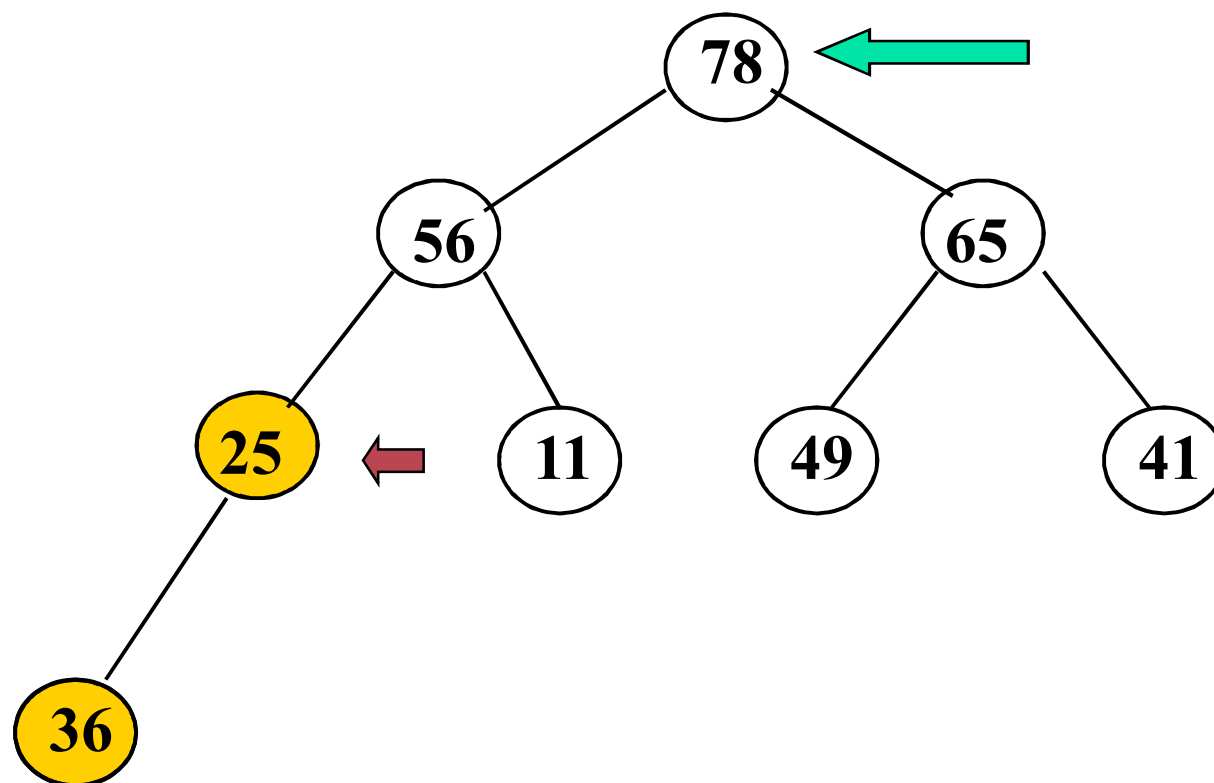
1	2	3	4	5	6	7	8
78	56	65	25	11	49	41	36



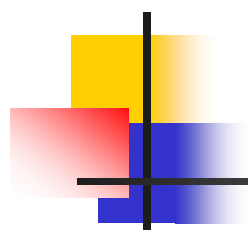
筛选法建堆过程演示



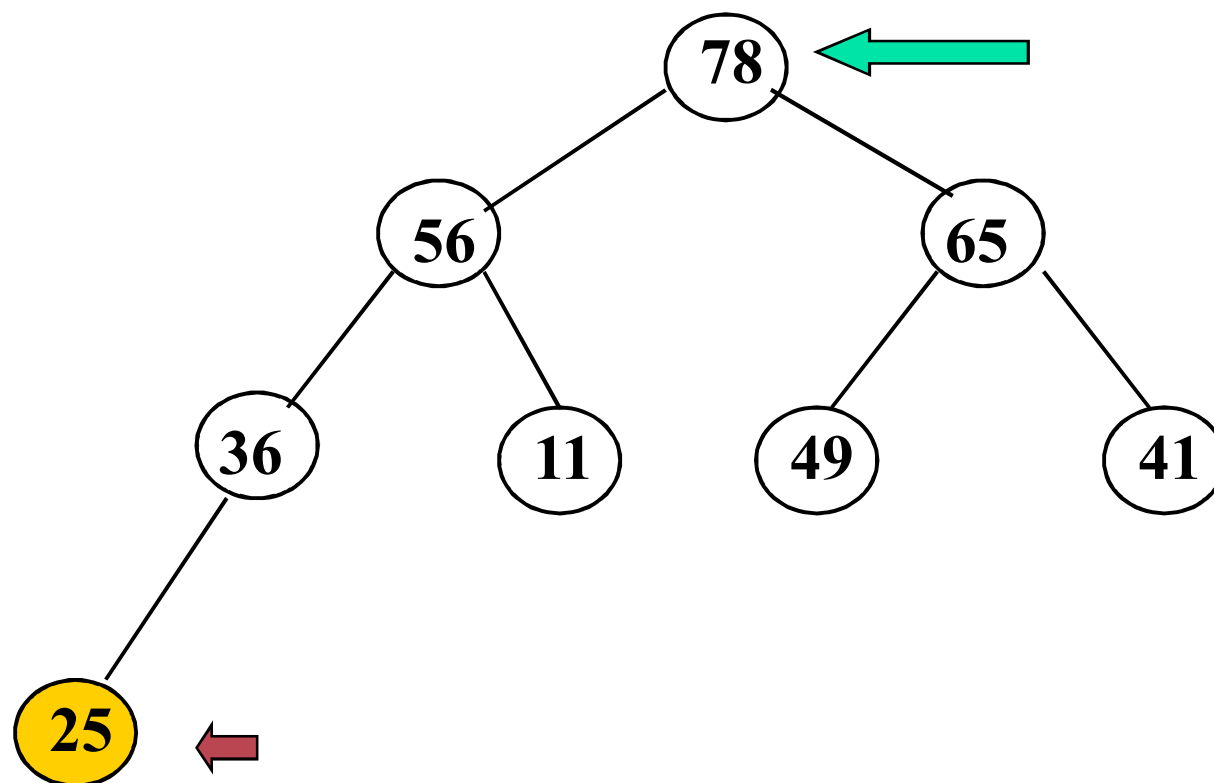
1	2	3	4	5	6	7	8
78	56	65	25	11	49	41	36



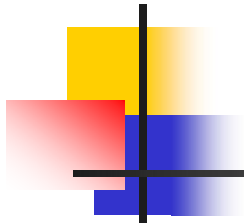
筛选法建堆过程演示



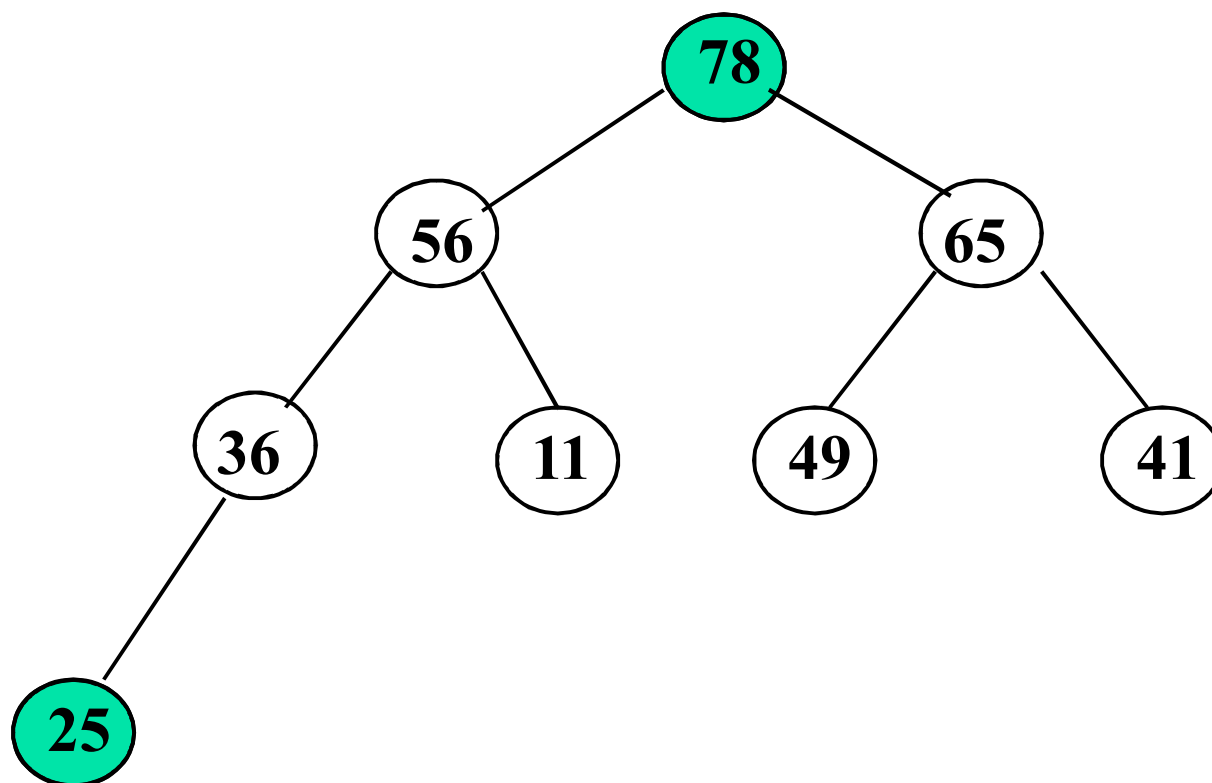
1	2	3	4	5	6	7	8
78	56	65	36	11	49	41	25



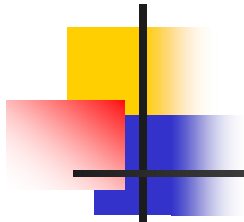
筛选法建堆过程演示



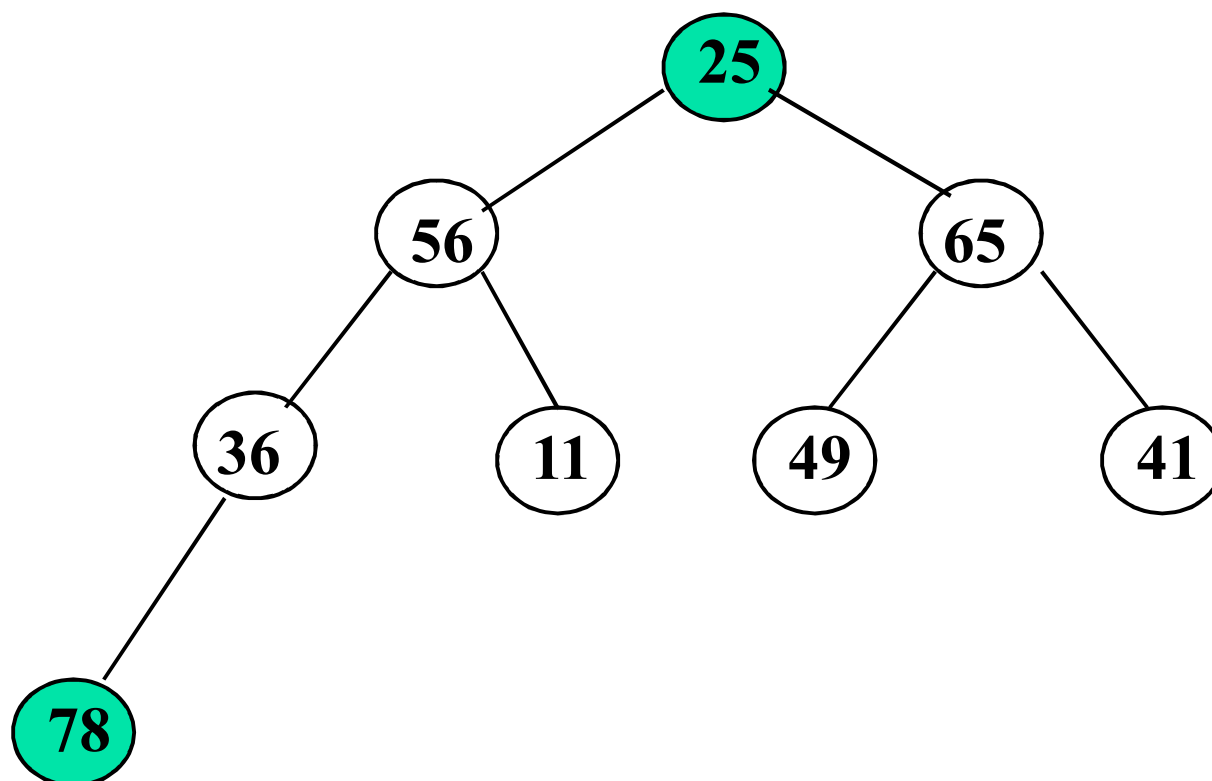
1	2	3	4	5	6	7	8
78	56	65	36	11	49	41	25



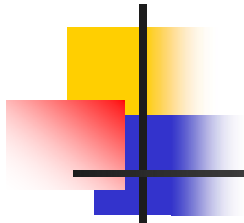
筛选法建的大顶堆



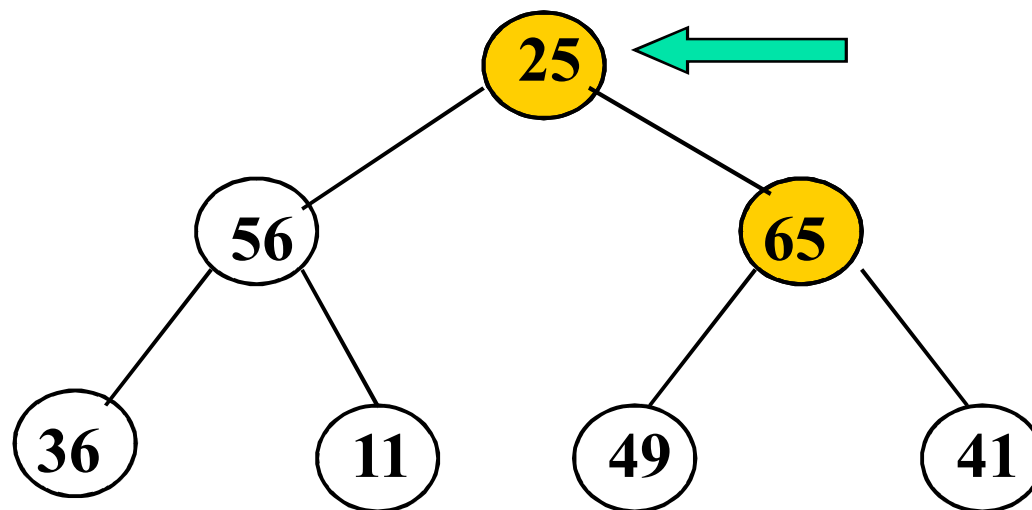
1	2	3	4	5	6	7	8
25	56	65	36	11	49	41	78



第一趟堆排序

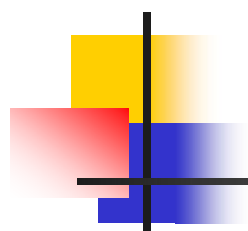


1	2	3	4	5	6	7	8
25	56	65	36	11	49	41	78

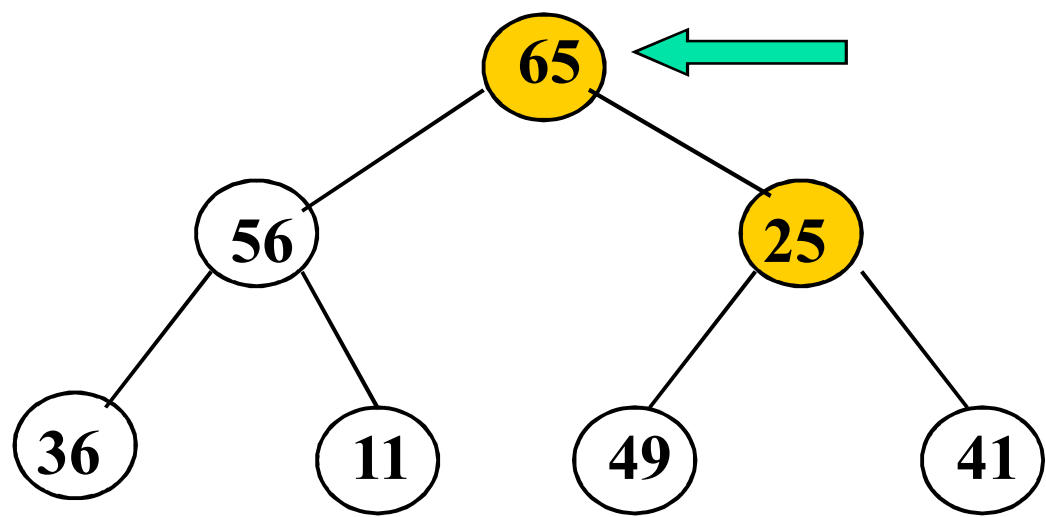


78

第一趟堆排序

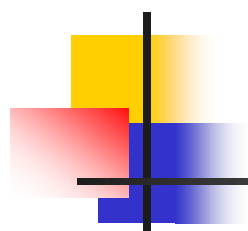


1	2	3	4	5	6	7	8
65	56	25	36	11	49	41	78

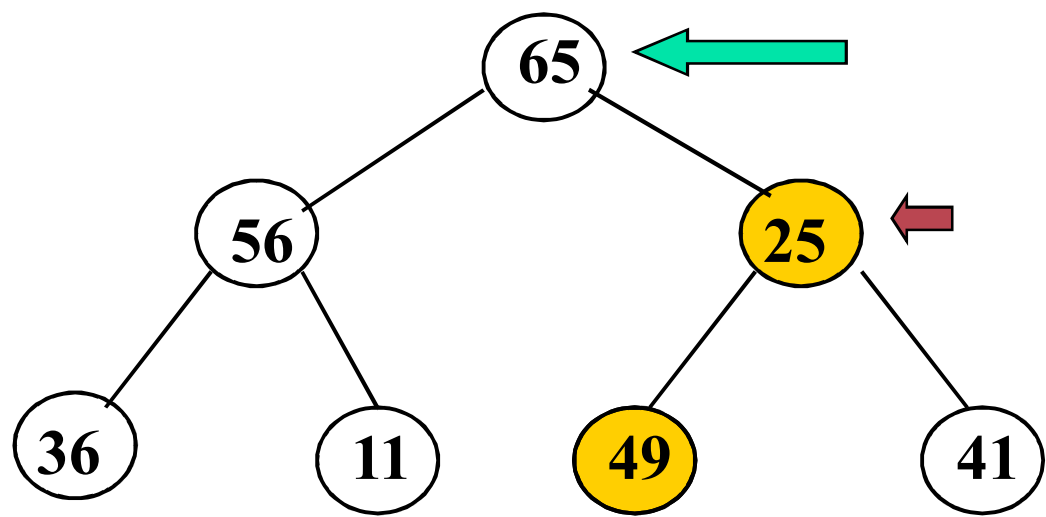


78

第一趟堆排序

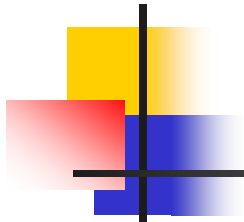


1	2	3	4	5	6	7	8
65	56	25	36	11	49	41	78

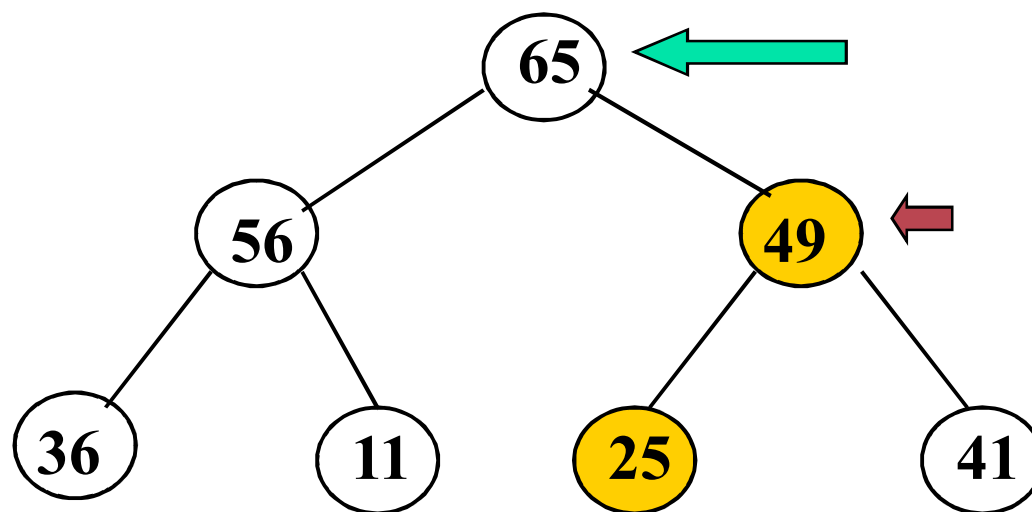


78

第一趟堆排序

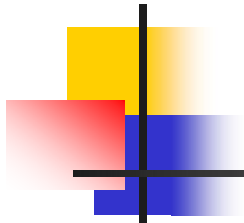


1	2	3	4	5	6	7	8
65	56	49	36	11	25	41	78

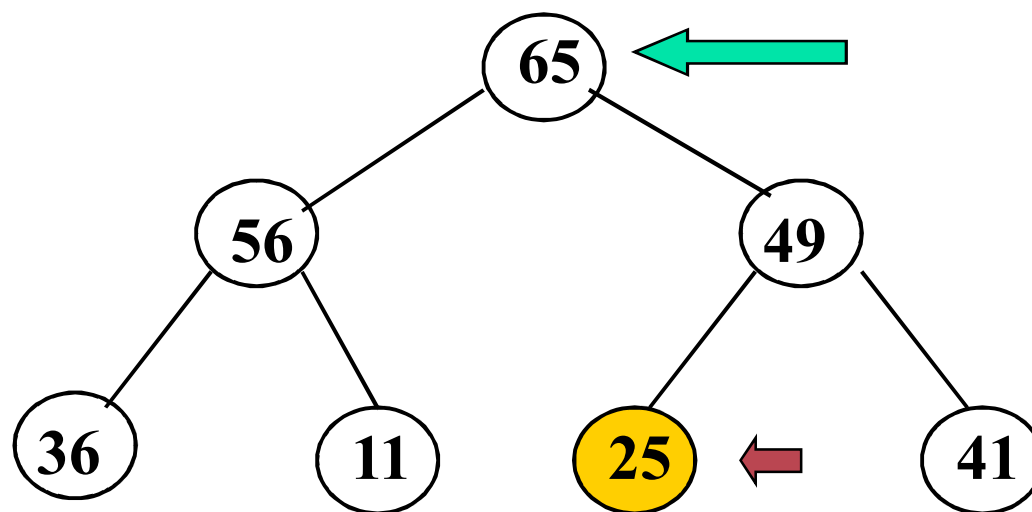


78

第一趟堆排序

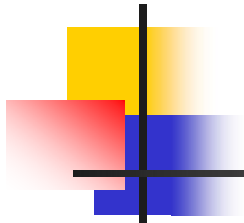


1	2	3	4	5	6	7	8
65	56	49	36	11	25	41	78

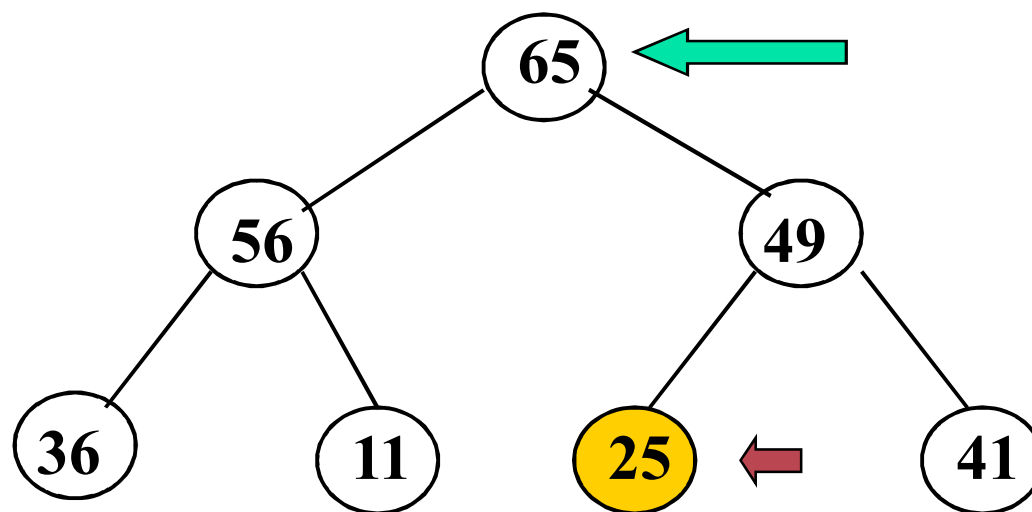


78

第一趟堆排序

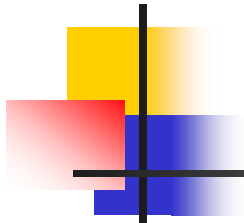


1	2	3	4	5	6	7	8
65	56	49	36	11	25	41	78

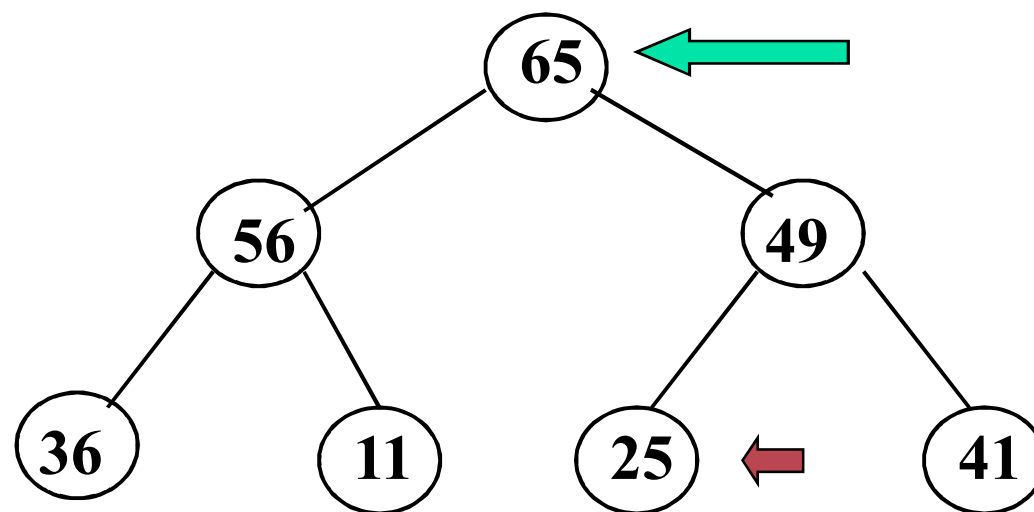


78

第一趟堆排序

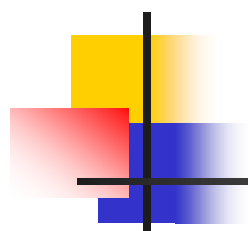


1	2	3	4	5	6	7	8
65	56	49	36	11	25	41	78

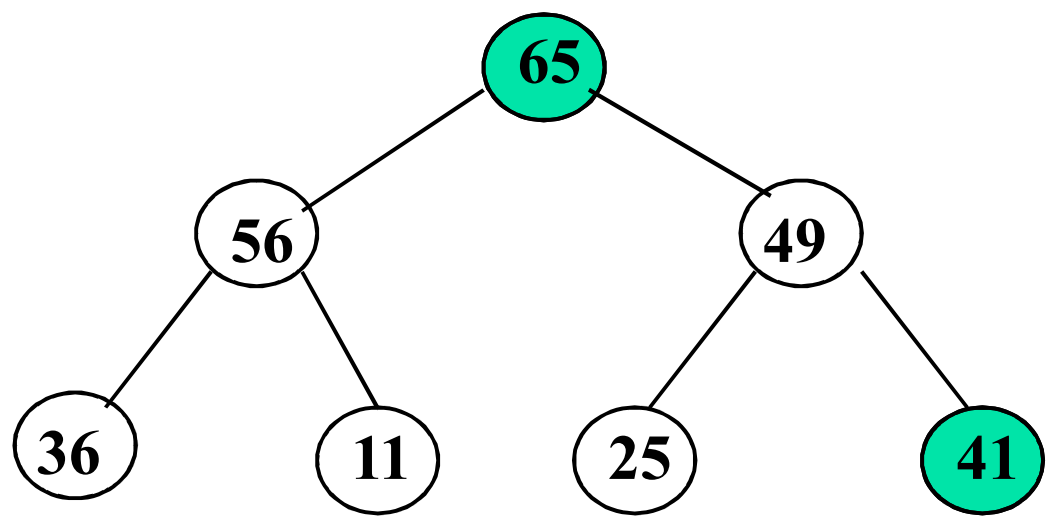


78

第一趟堆排序

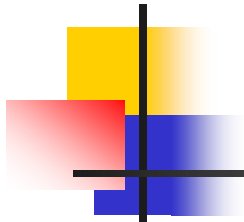


1	2	3	4	5	6	7	8
65	56	49	36	11	25	41	78

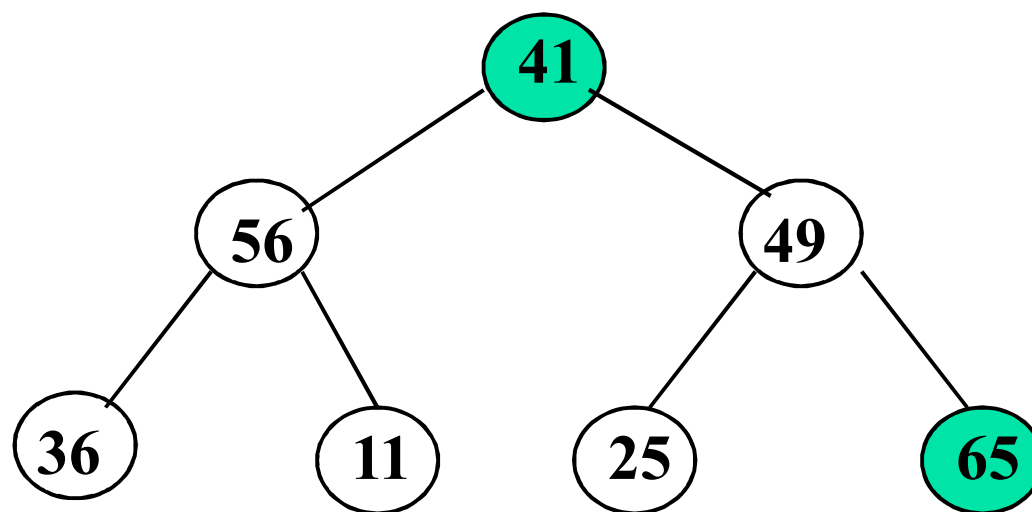


78

第二趟堆排序

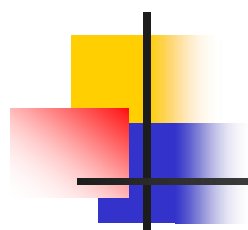


1	2	3	4	5	6	7	8
41	56	49	36	11	25	65	78

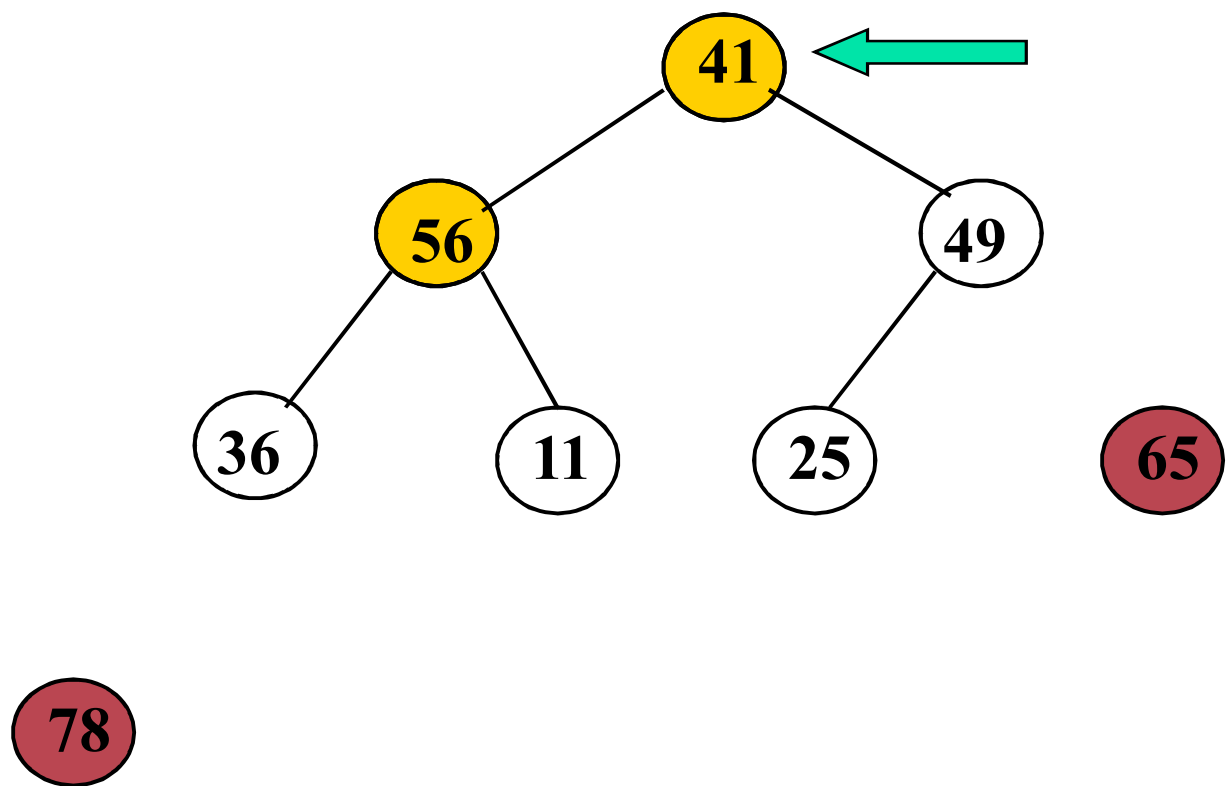


78

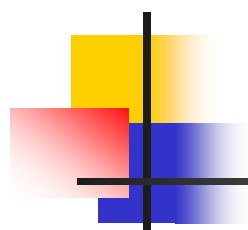
第二趟堆排序



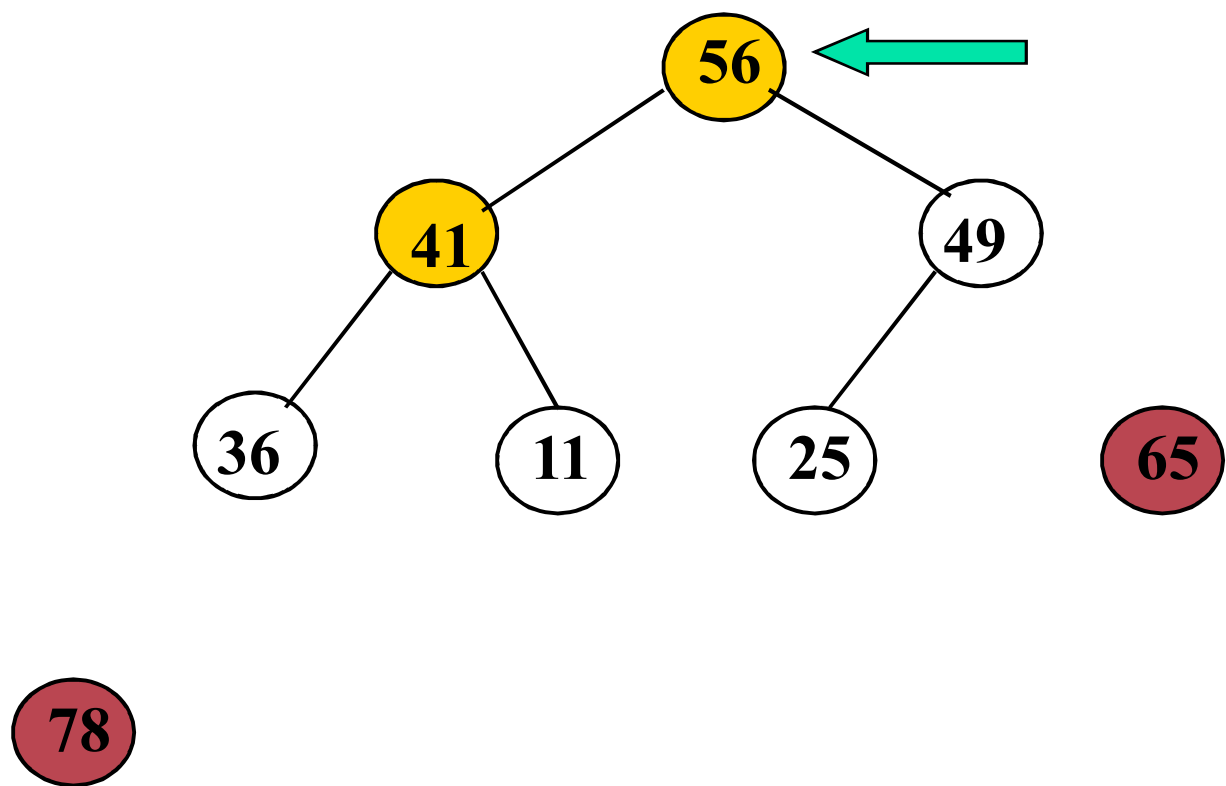
1	2	3	4	5	6	7	8
41	56	49	36	11	25	65	78



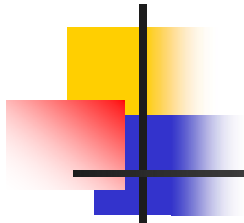
第二趟堆排序



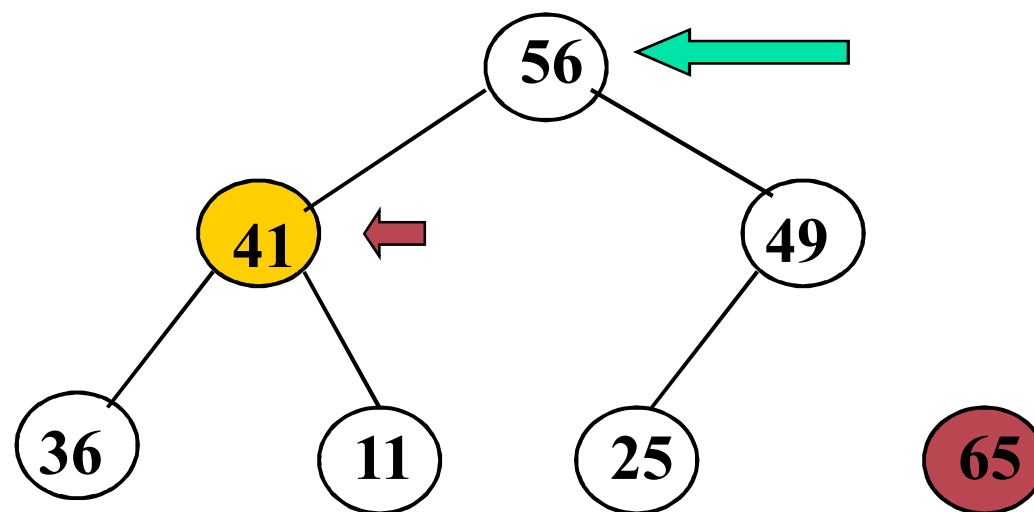
1	2	3	4	5	6	7	8
56	41	49	36	11	25	65	78



第二趟堆排序

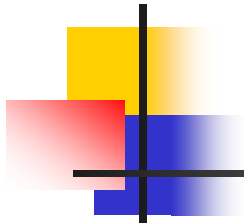


1	2	3	4	5	6	7	8
56	41	49	36	11	25	65	78

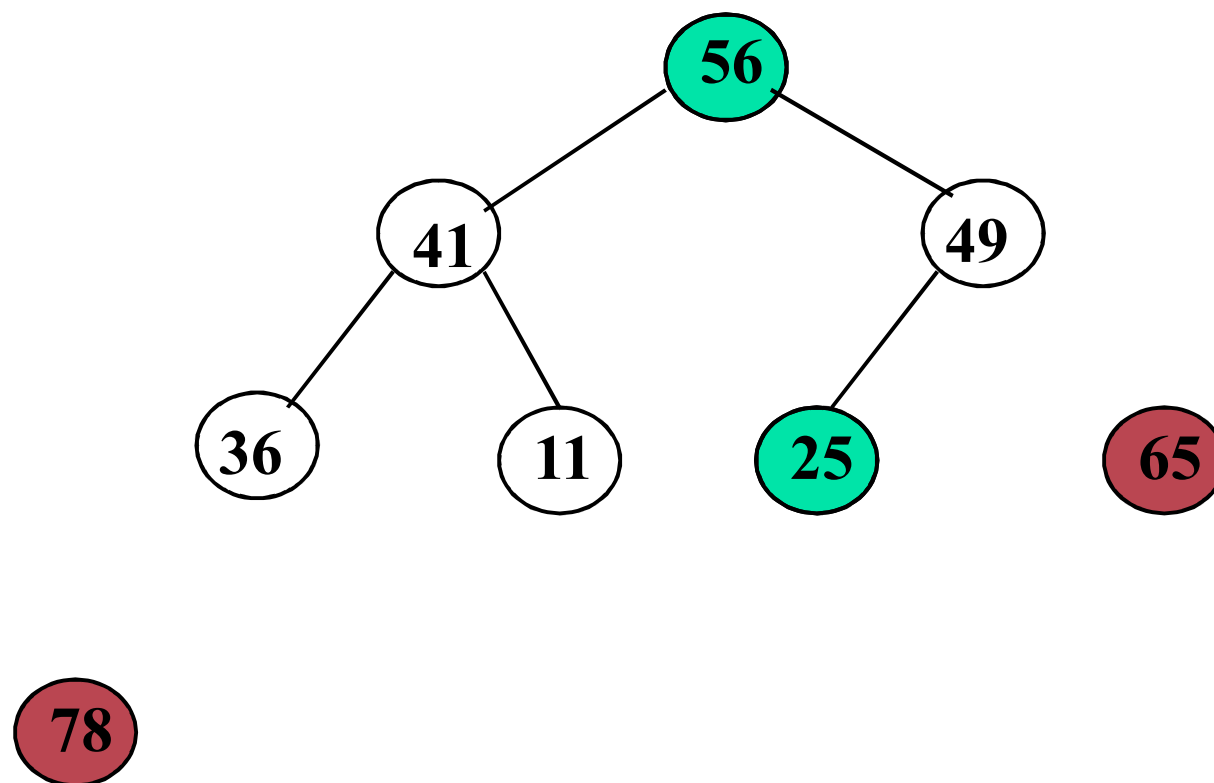


78

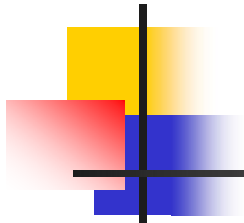
第二趟堆排序



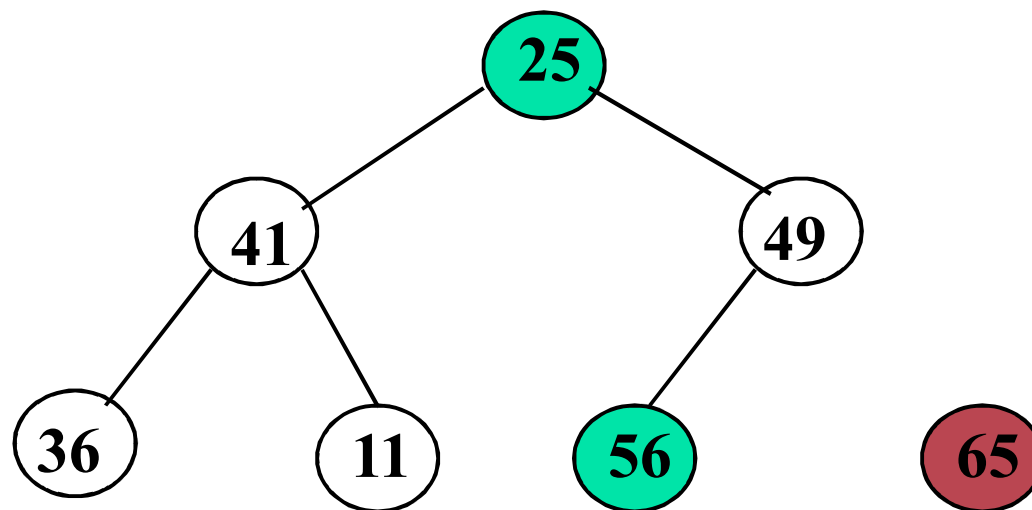
1	2	3	4	5	6	7	8
56	41	49	36	11	25	65	78



堆排序

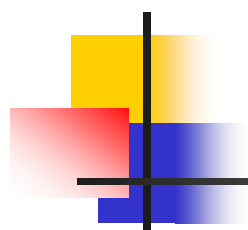


1	2	3	4	5	6	7	8
25	41	49	36	11	56	65	78

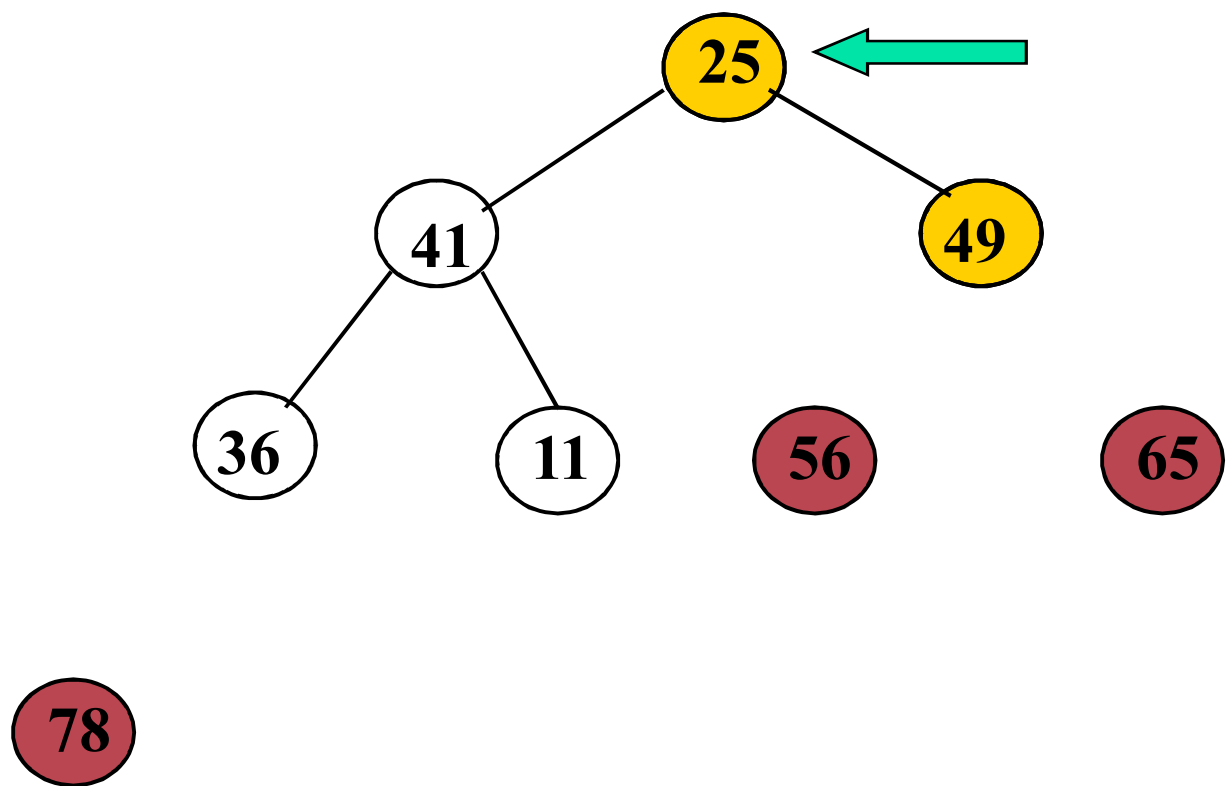


78

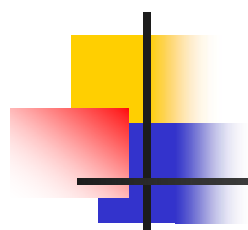
堆排序



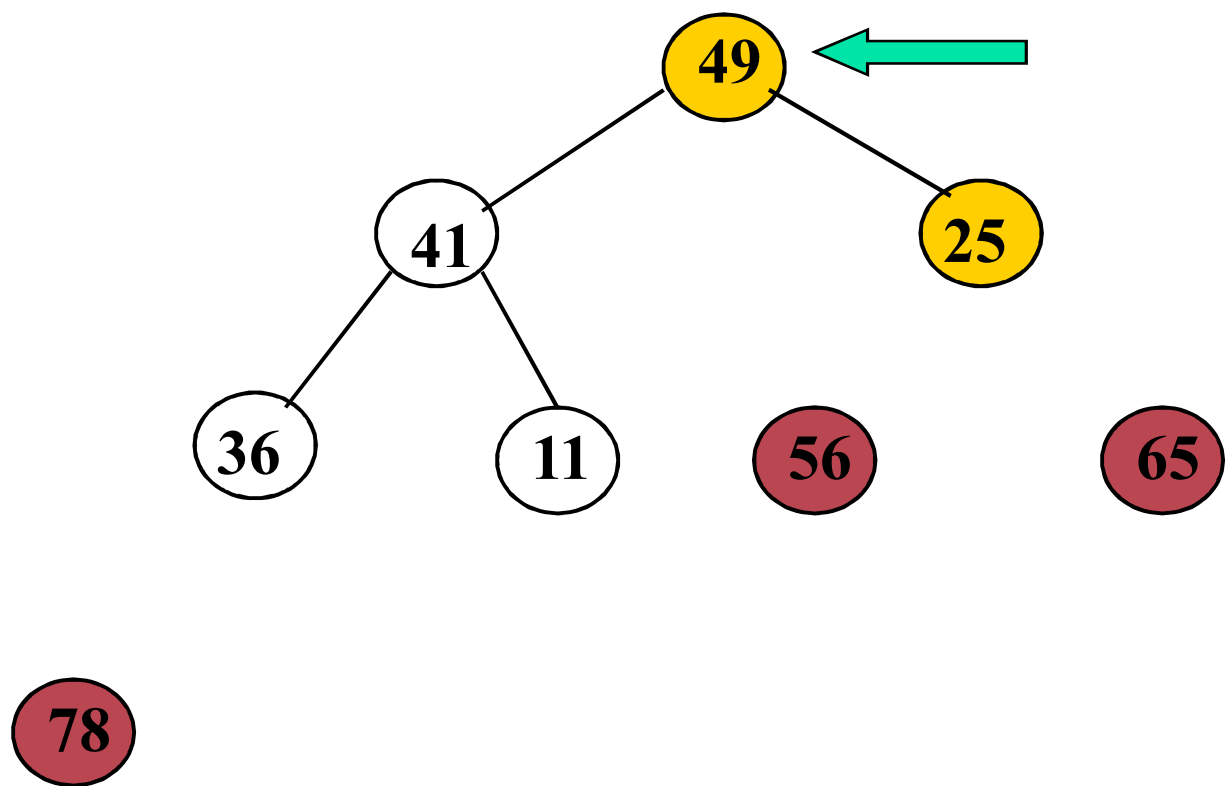
1	2	3	4	5	6	7	8
25	41	49	36	11	56	65	78



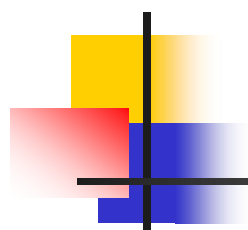
堆排序



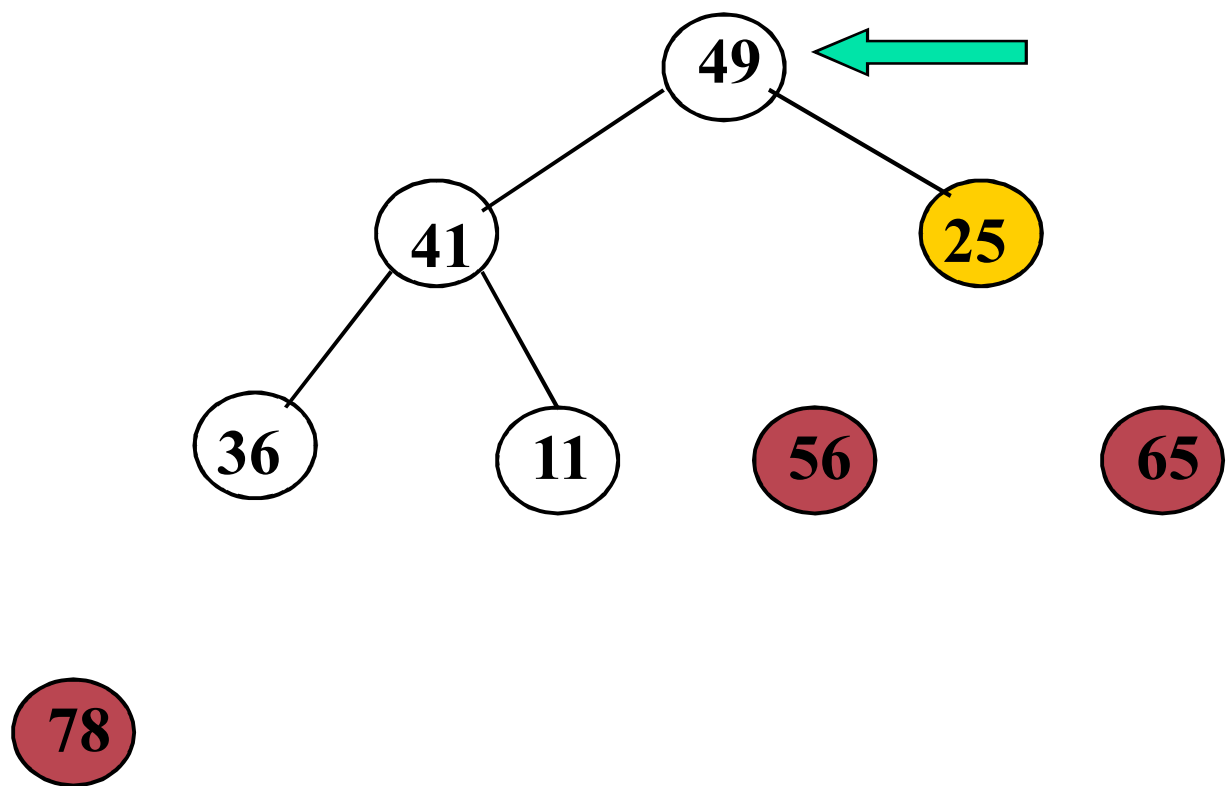
1	2	3	4	5	6	7	8
49	41	25	36	11	56	65	78



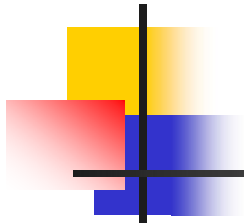
堆排序



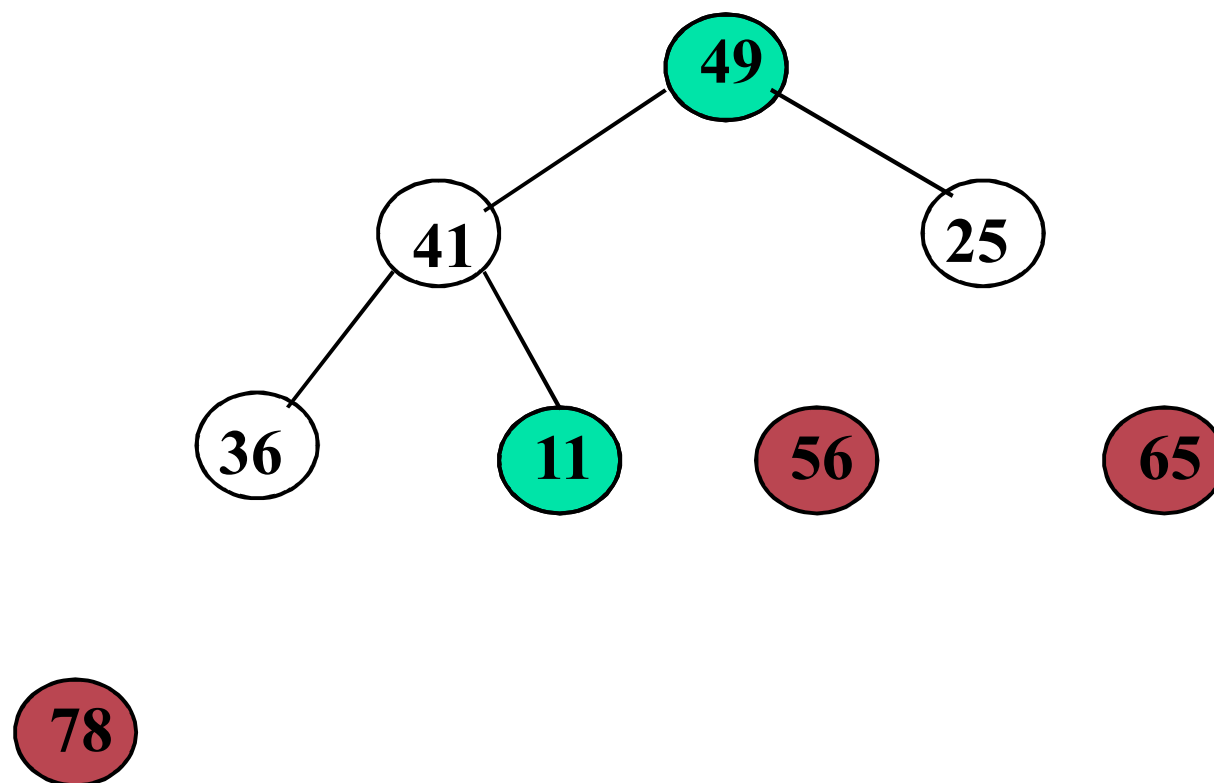
1	2	3	4	5	6	7	8
49	41	25	36	11	56	65	78



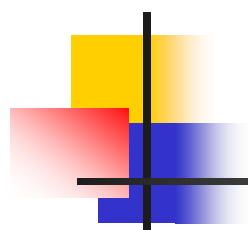
堆排序



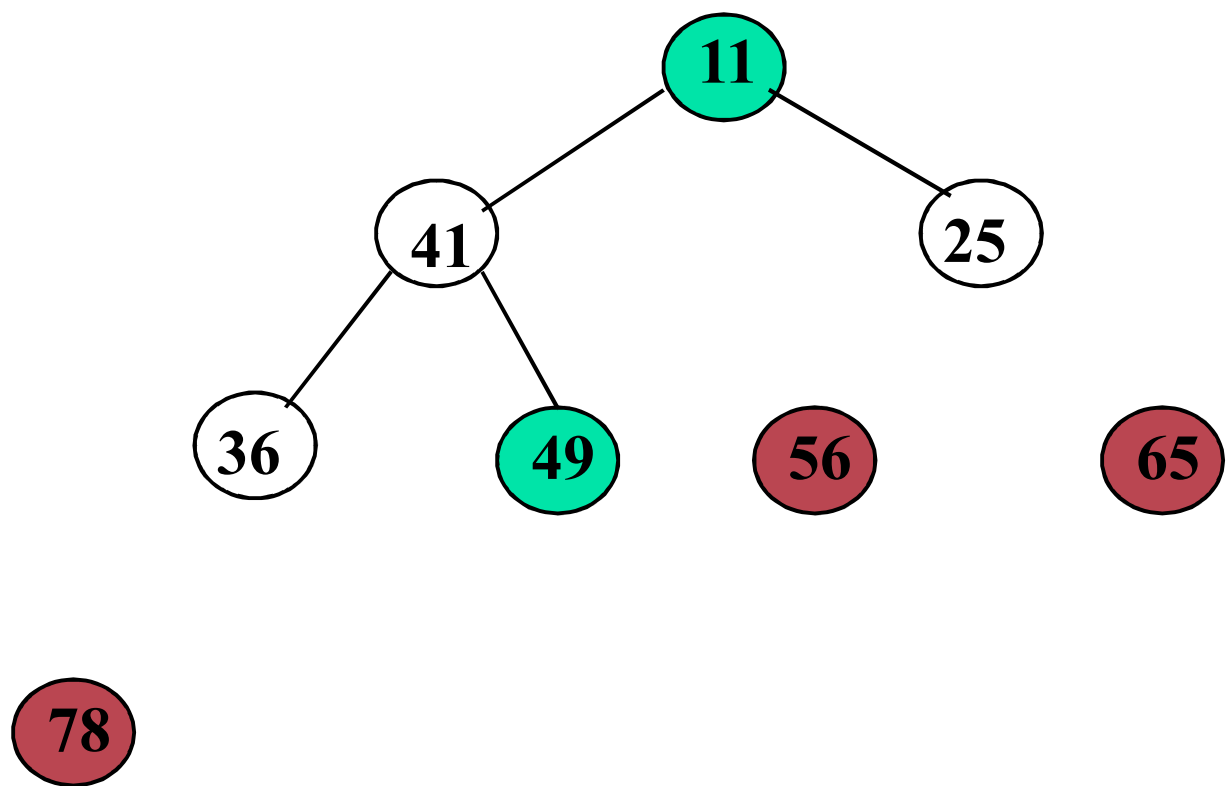
1	2	3	4	5	6	7	8
49	41	25	36	11	56	65	78



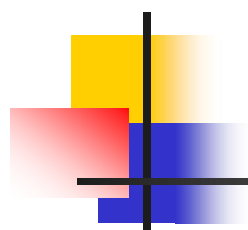
堆排序



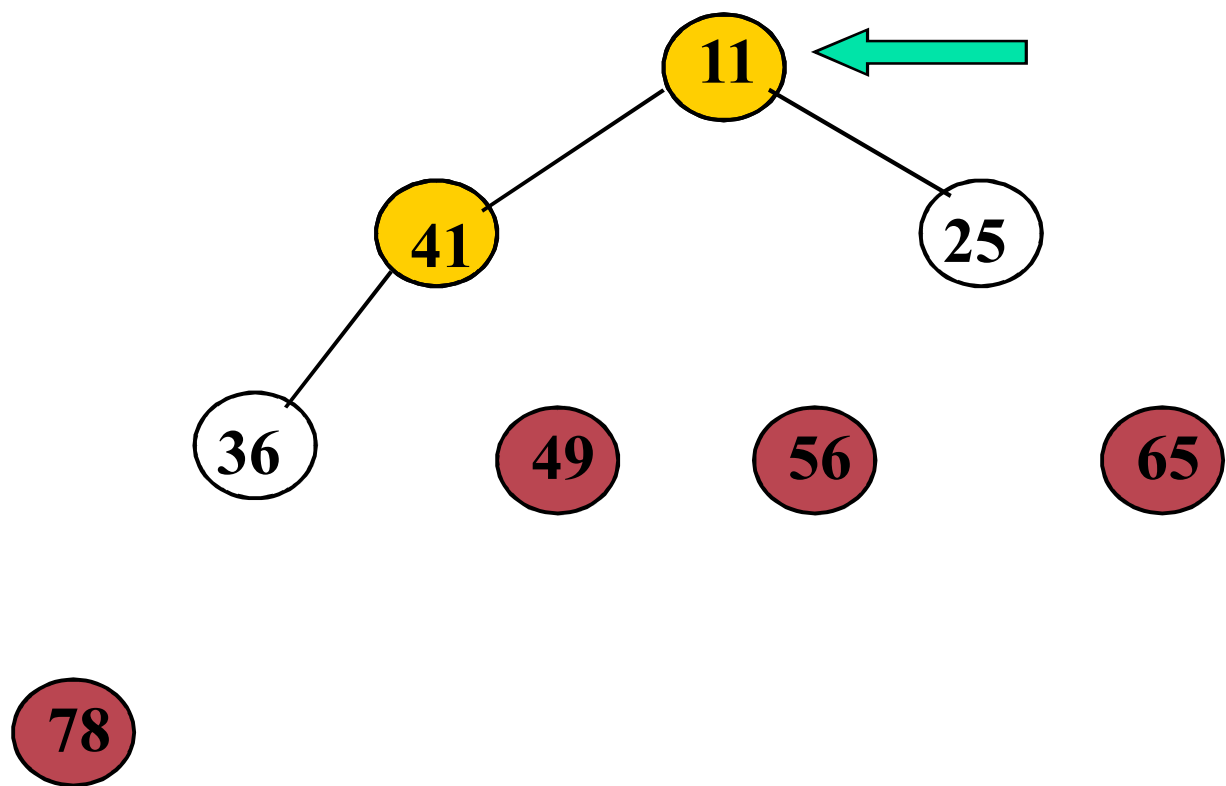
1	2	3	4	5	6	7	8
11	41	25	36	49	56	65	78



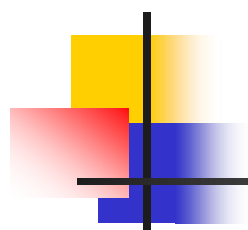
堆排序



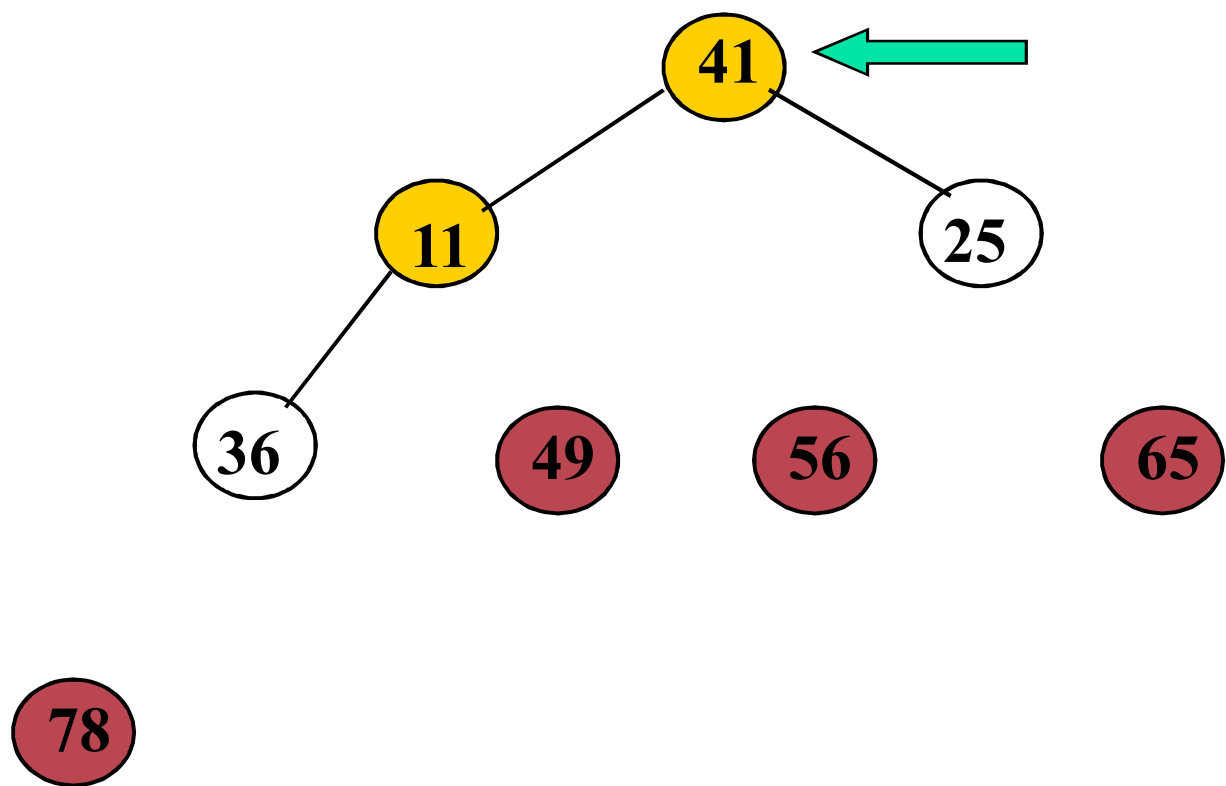
1	2	3	4	5	6	7	8
11	41	25	36	49	56	65	78



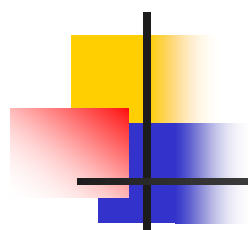
堆排序



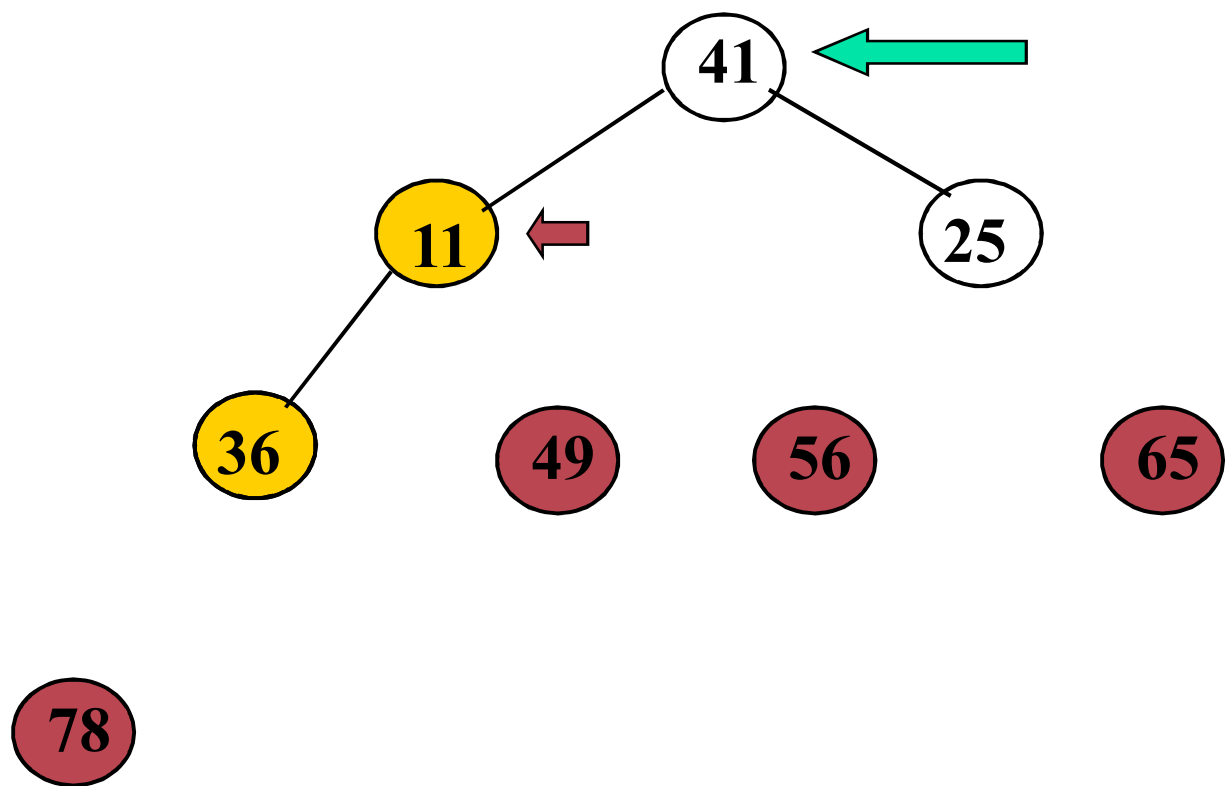
1	2	3	4	5	6	7	8
41	11	25	36	49	56	65	78



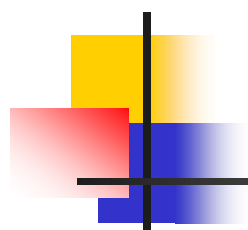
堆排序



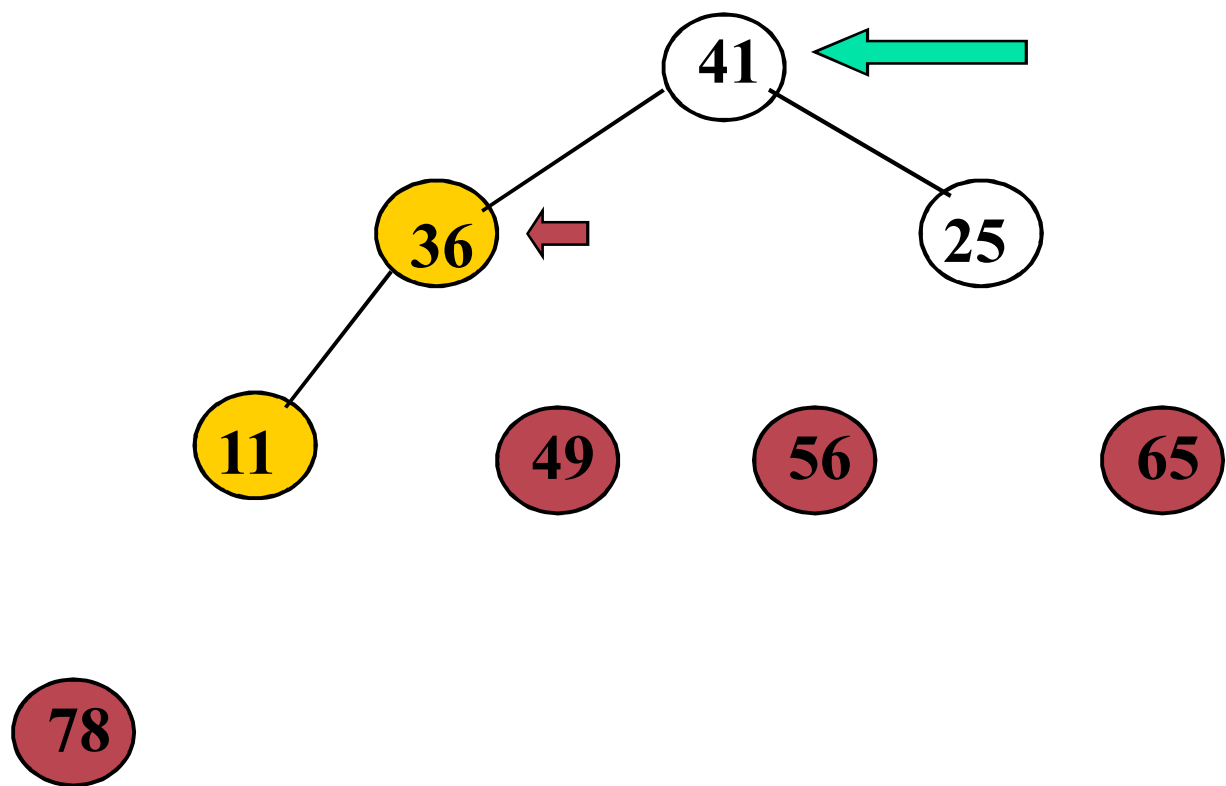
1	2	3	4	5	6	7	8
41	11	25	36	49	56	65	78



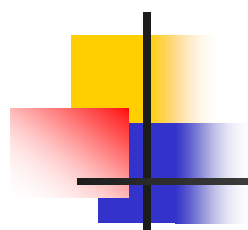
堆排序



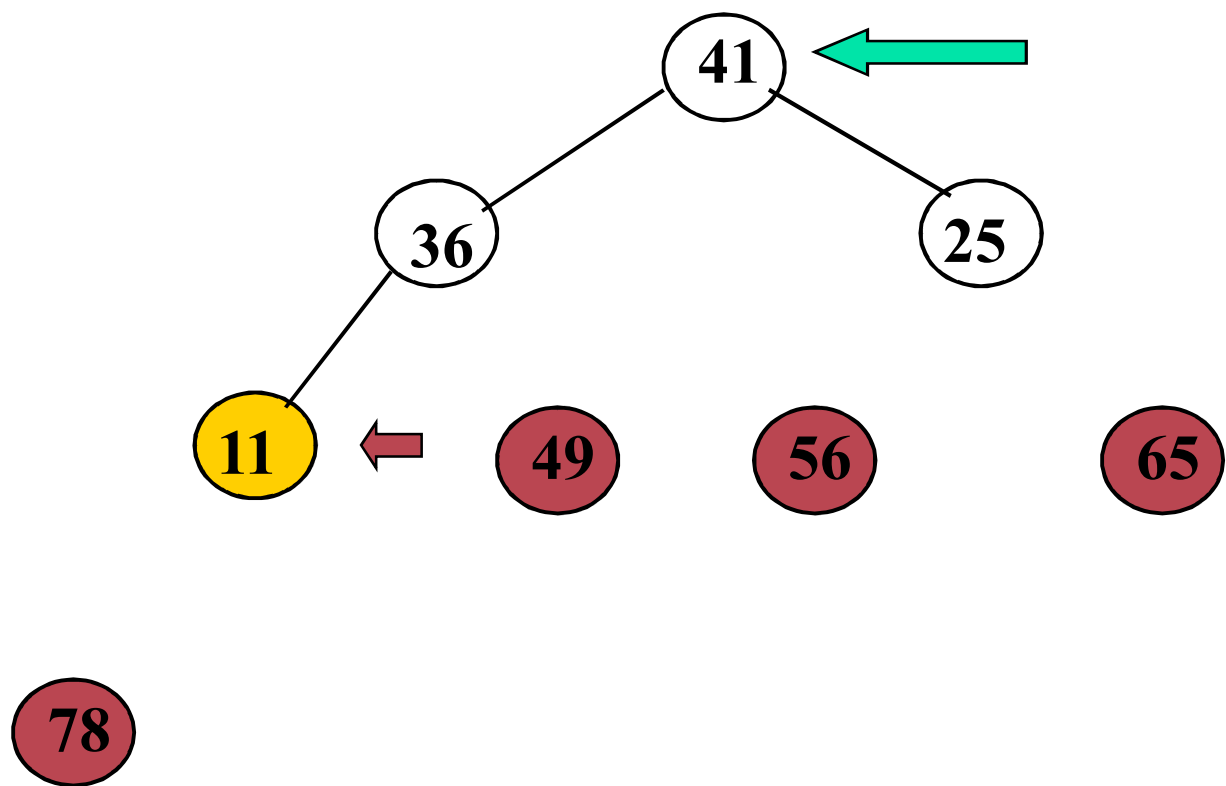
1	2	3	4	5	6	7	8
41	36	25	11	49	56	65	78



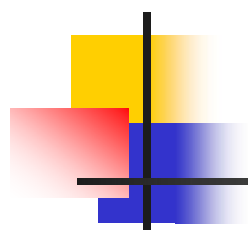
堆排序



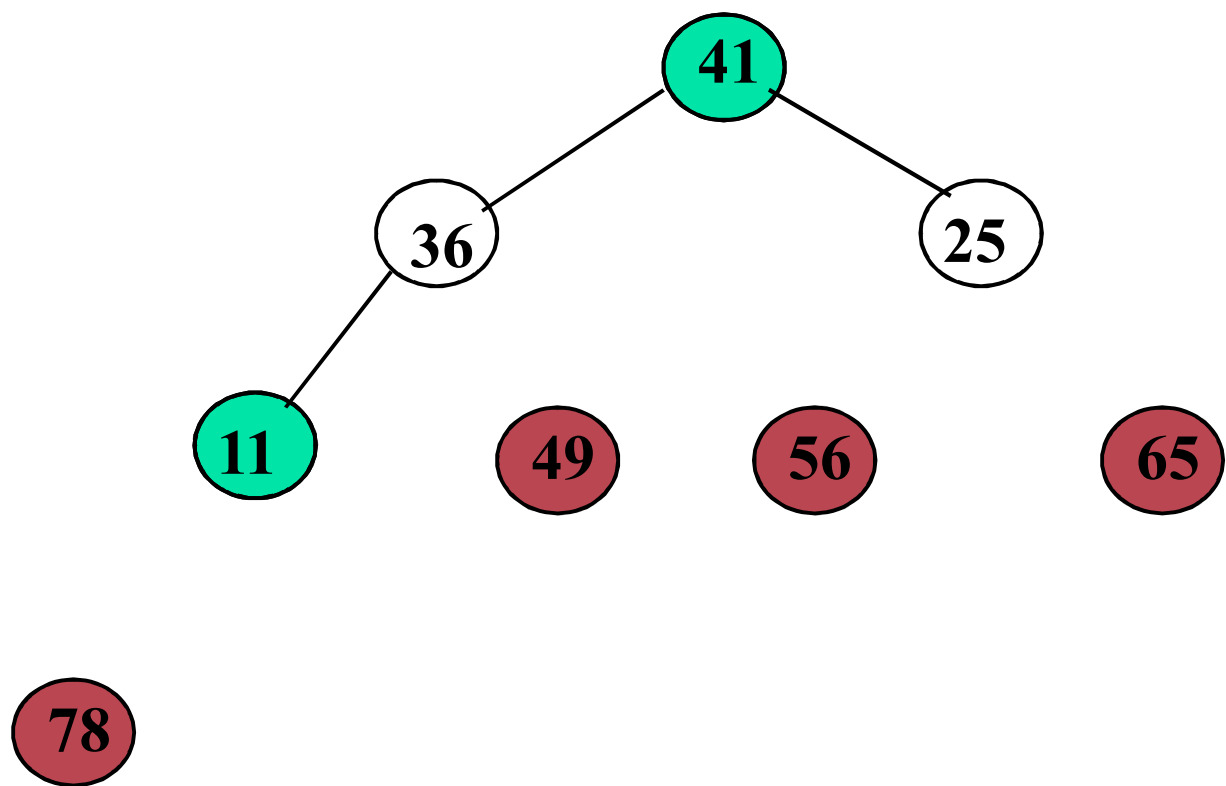
1	2	3	4	5	6	7	8
41	36	25	11	49	56	65	78



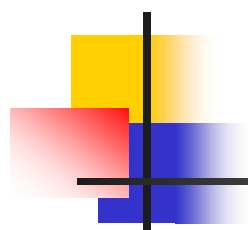
堆排序



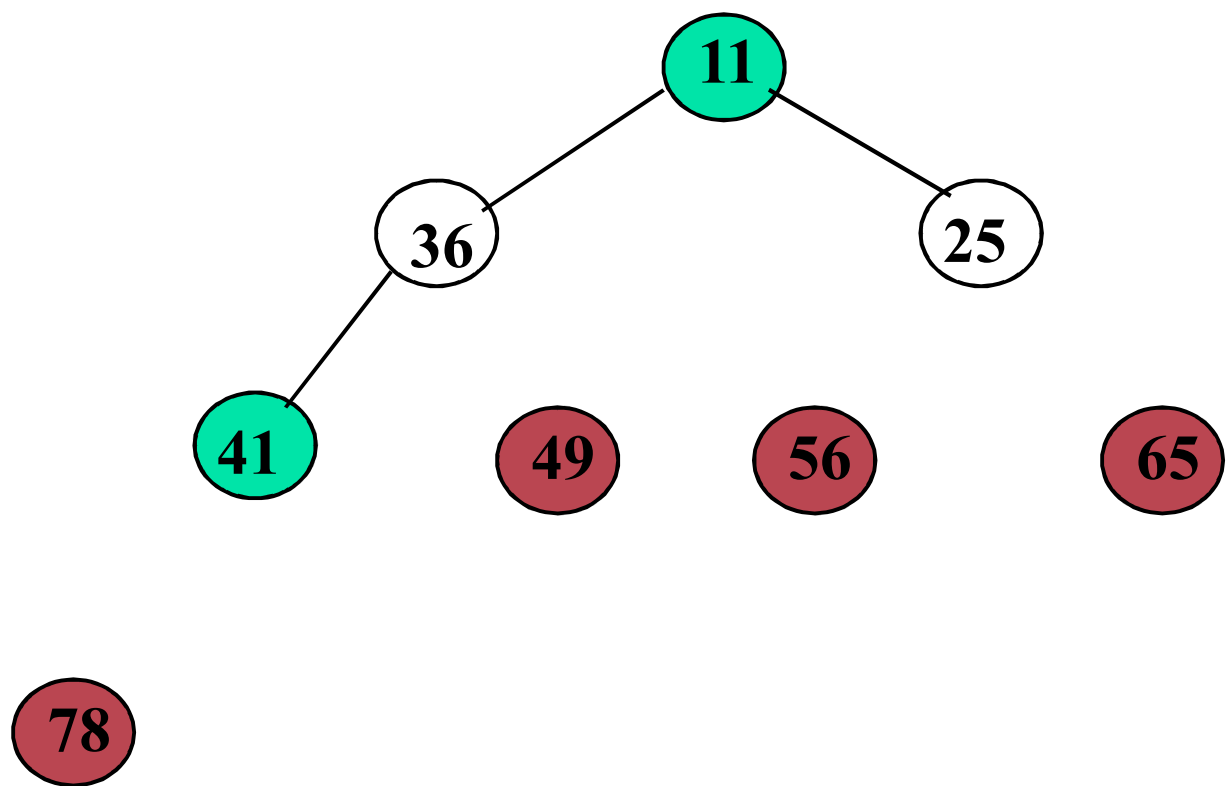
1	2	3	4	5	6	7	8
41	36	25	11	49	56	65	78



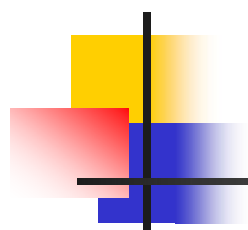
堆排序



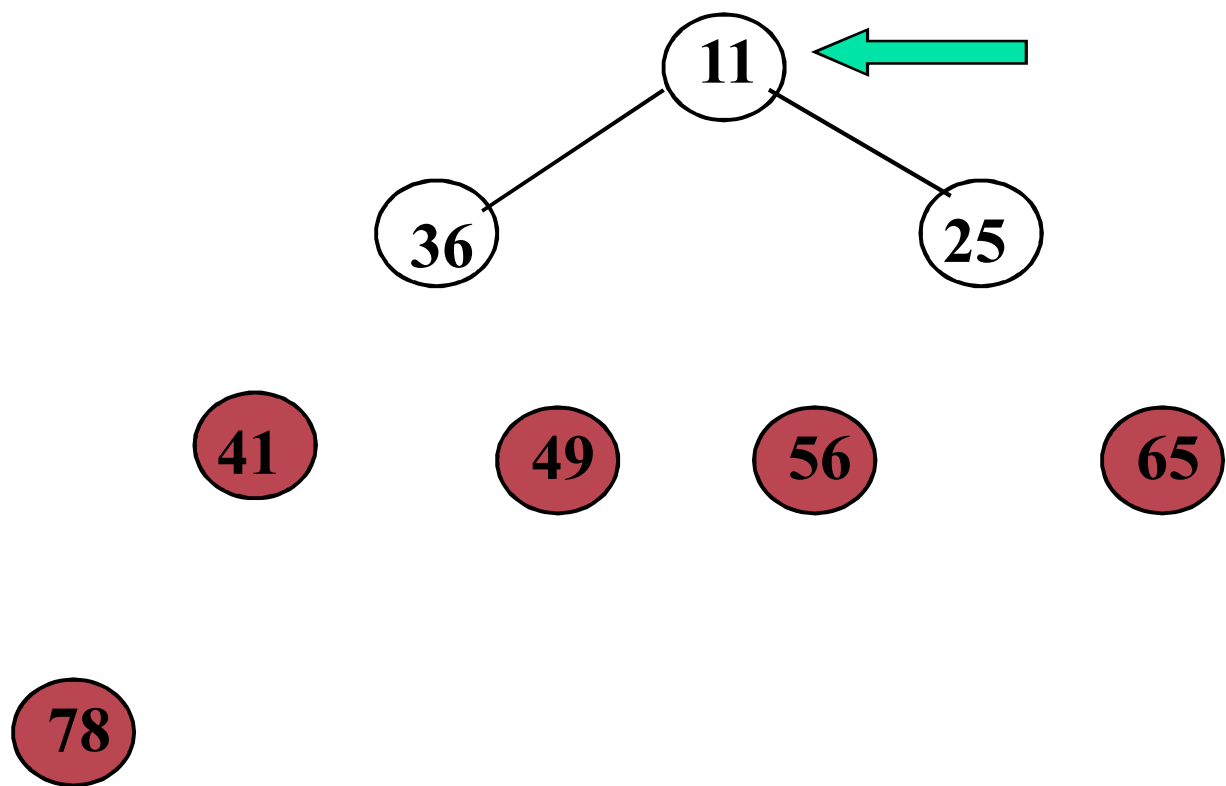
1	2	3	4	5	6	7	8
11	36	25	41	49	56	65	78



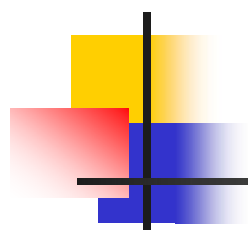
堆排序



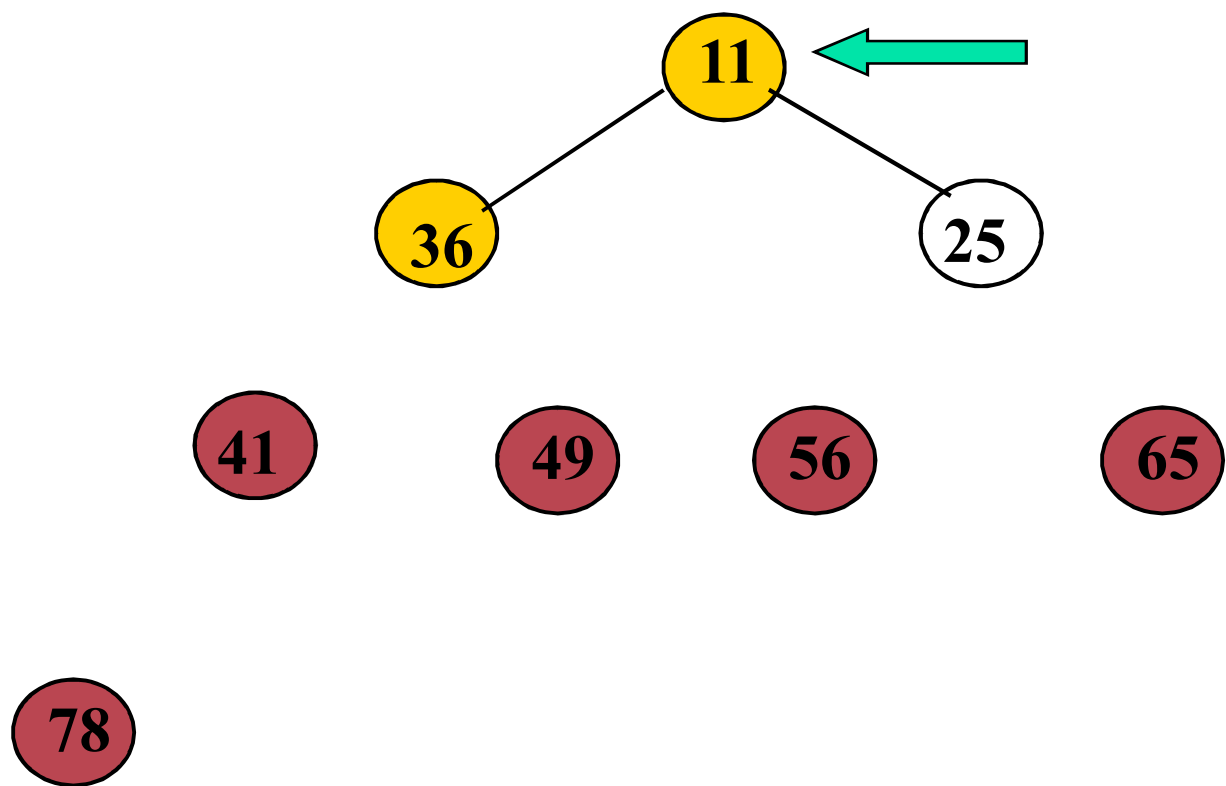
1	2	3	4	5	6	7	8
11	36	25	41	49	56	65	78



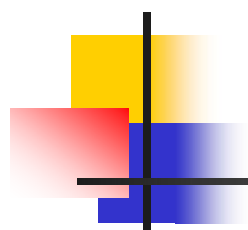
堆排序



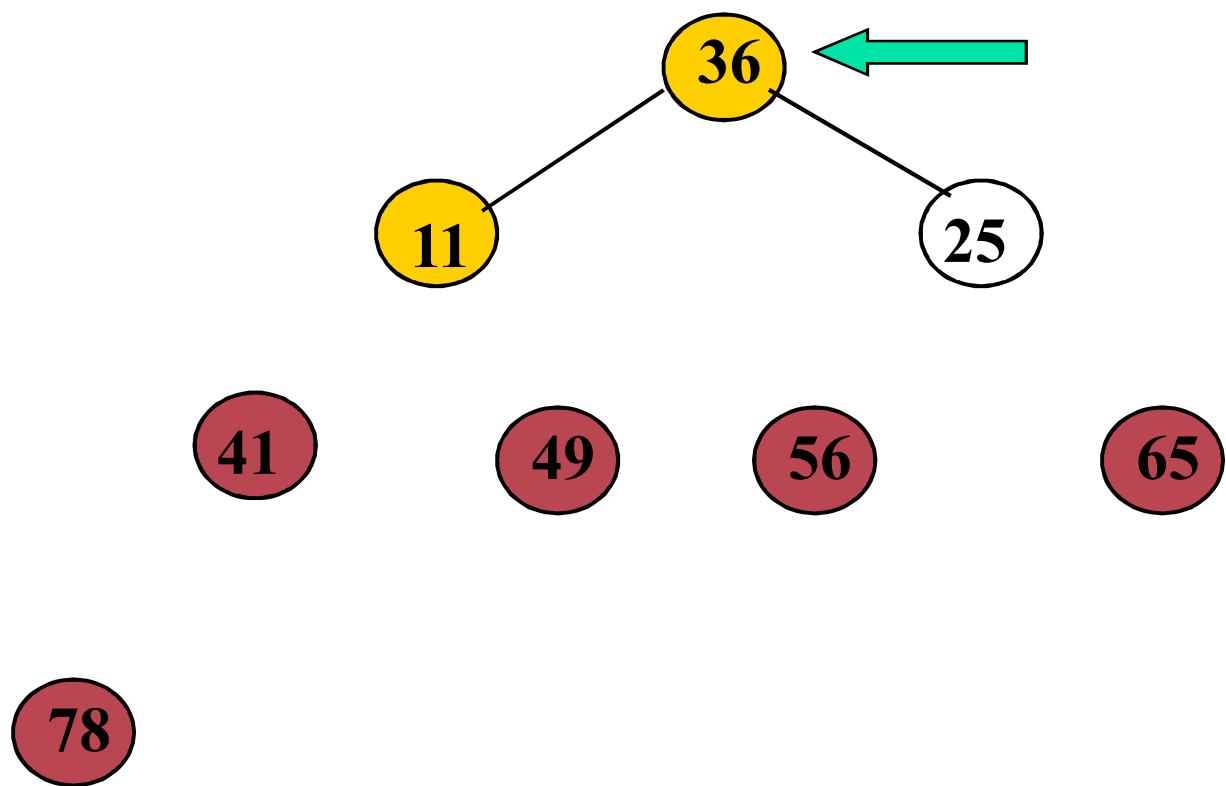
1	2	3	4	5	6	7	8
11	36	25	41	49	56	65	78



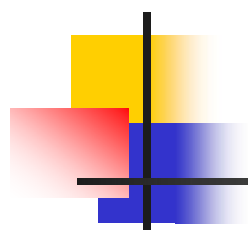
堆排序



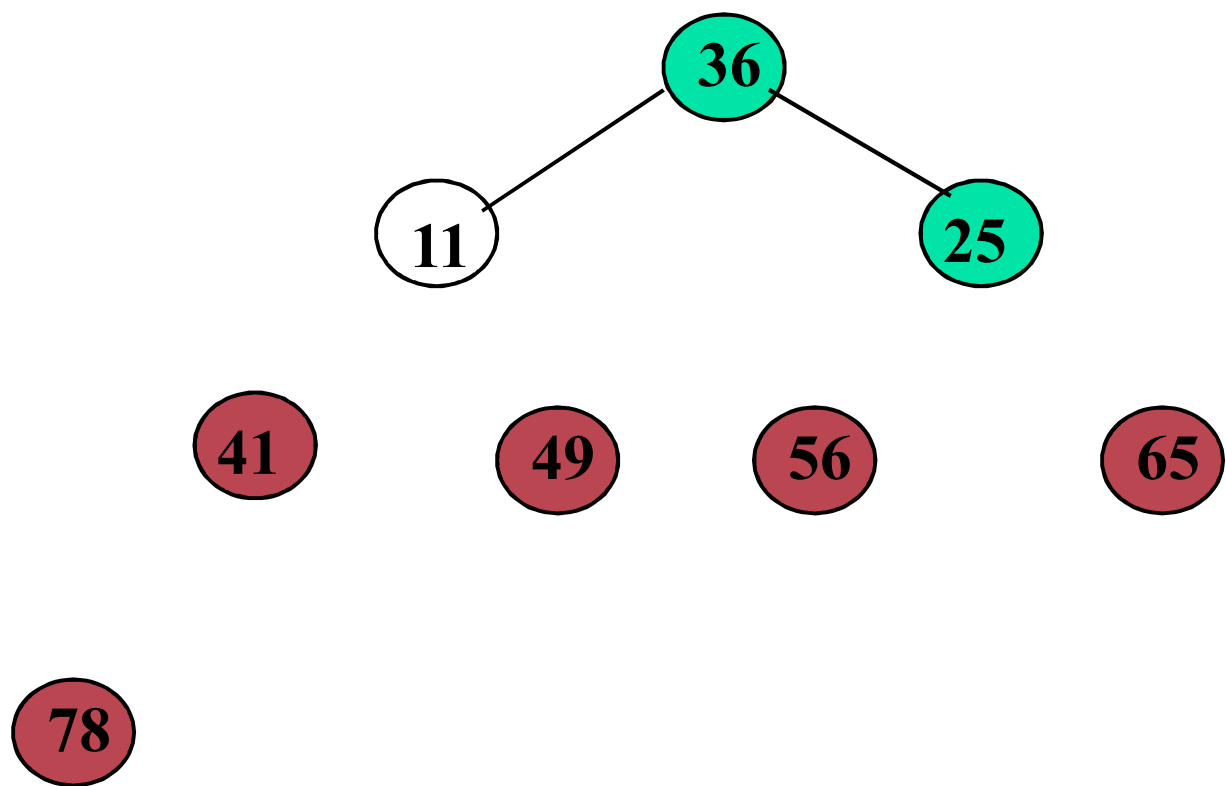
1	2	3	4	5	6	7	8
36	11	25	41	49	56	65	78



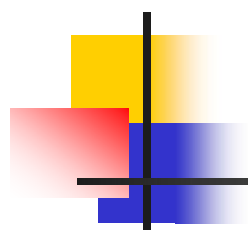
堆排序



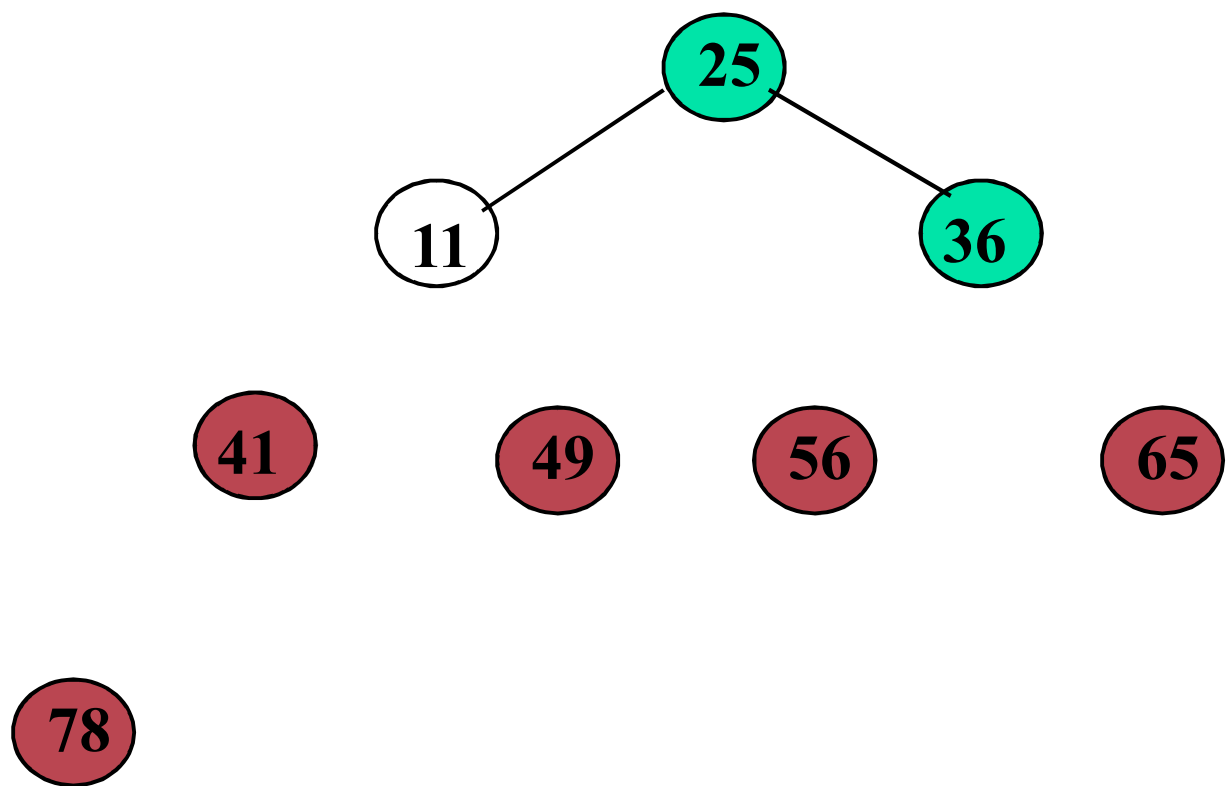
1	2	3	4	5	6	7	8
36	11	25	41	49	56	65	78



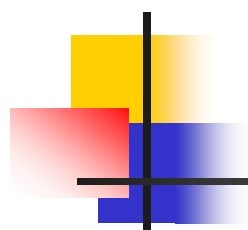
堆排序



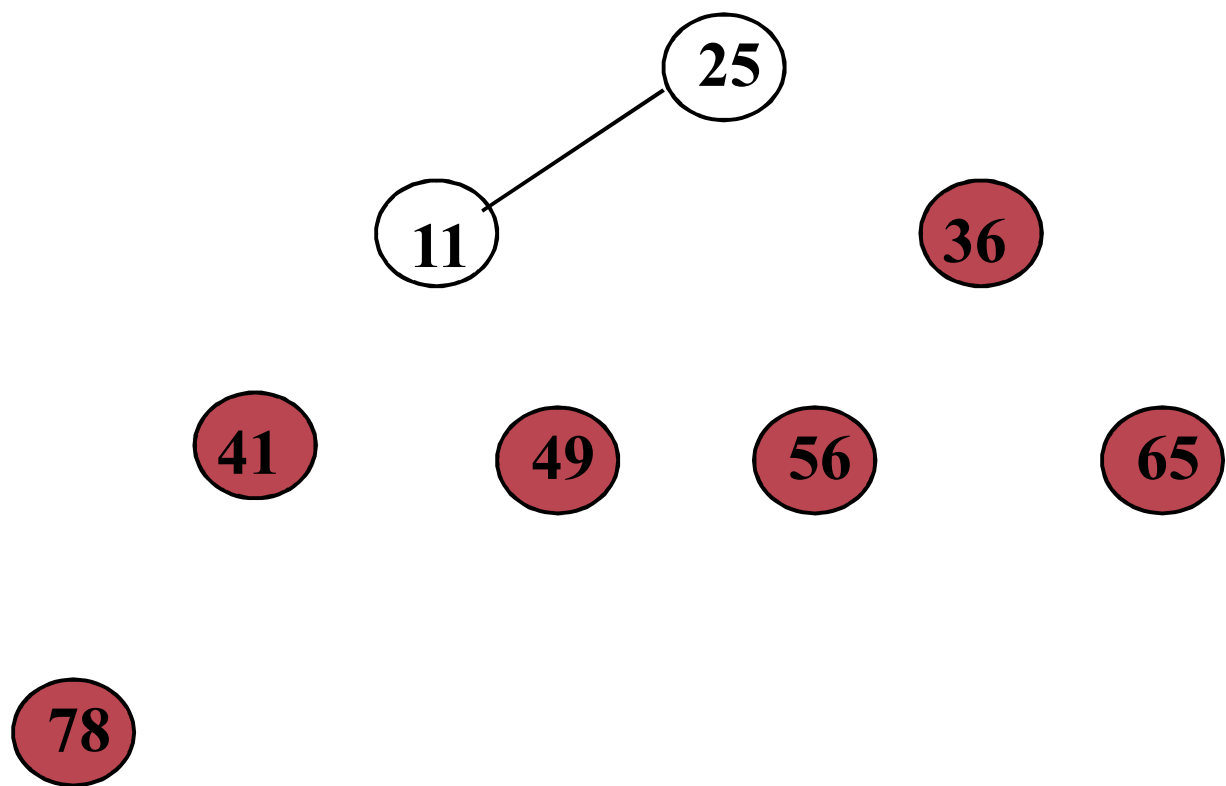
1	2	3	4	5	6	7	8
25	11	36	41	49	56	65	78



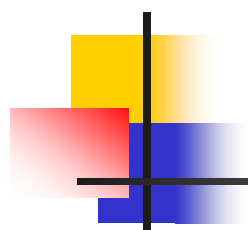
堆排序



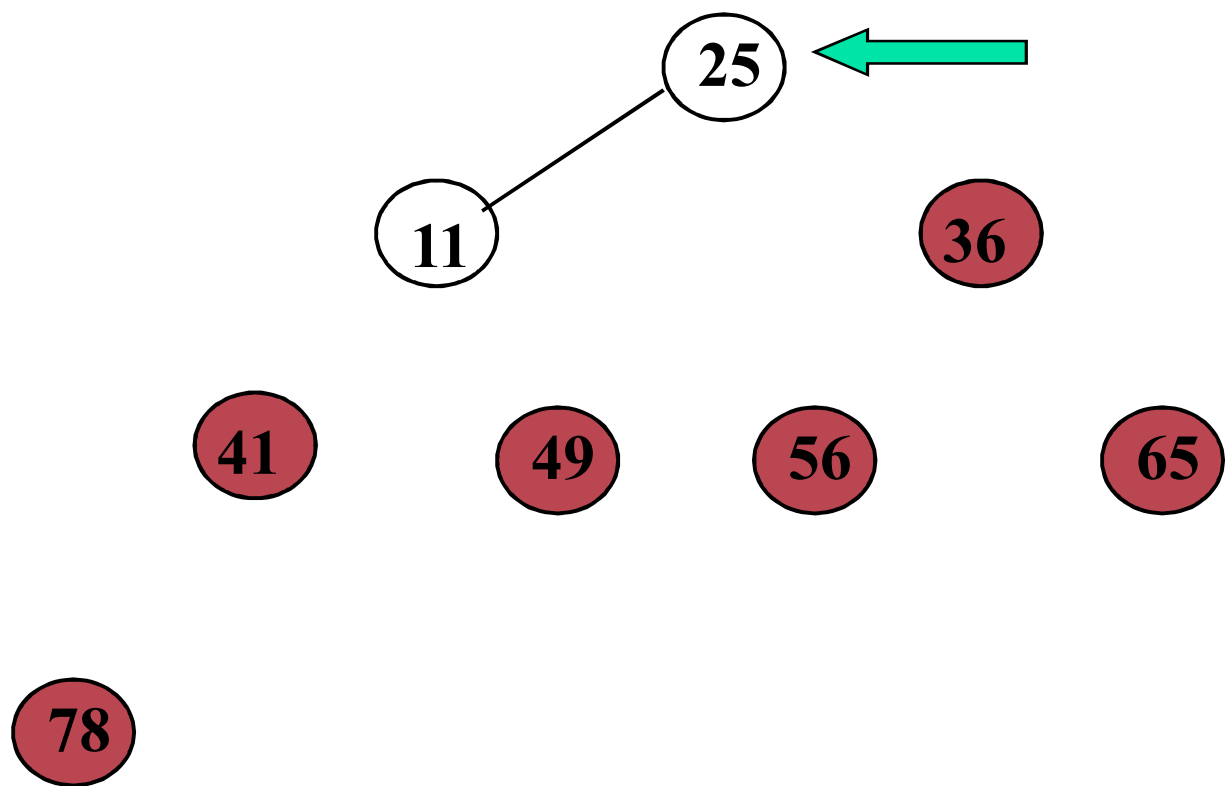
1	2	3	4	5	6	7	8
25	11	36	41	49	56	65	78



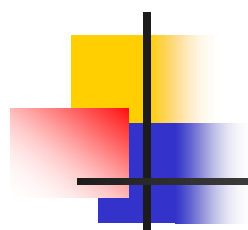
堆排序



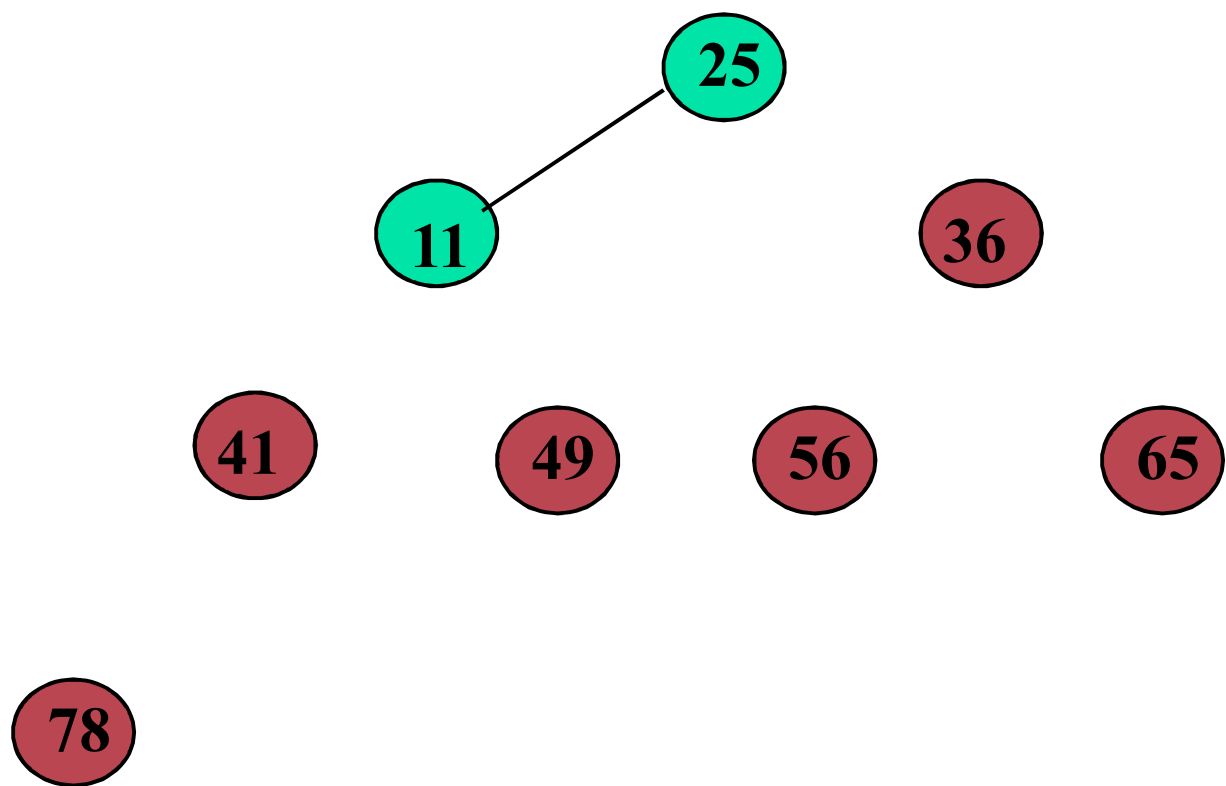
1	2	3	4	5	6	7	8
25	11	36	41	49	56	65	78



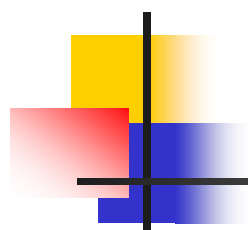
堆排序



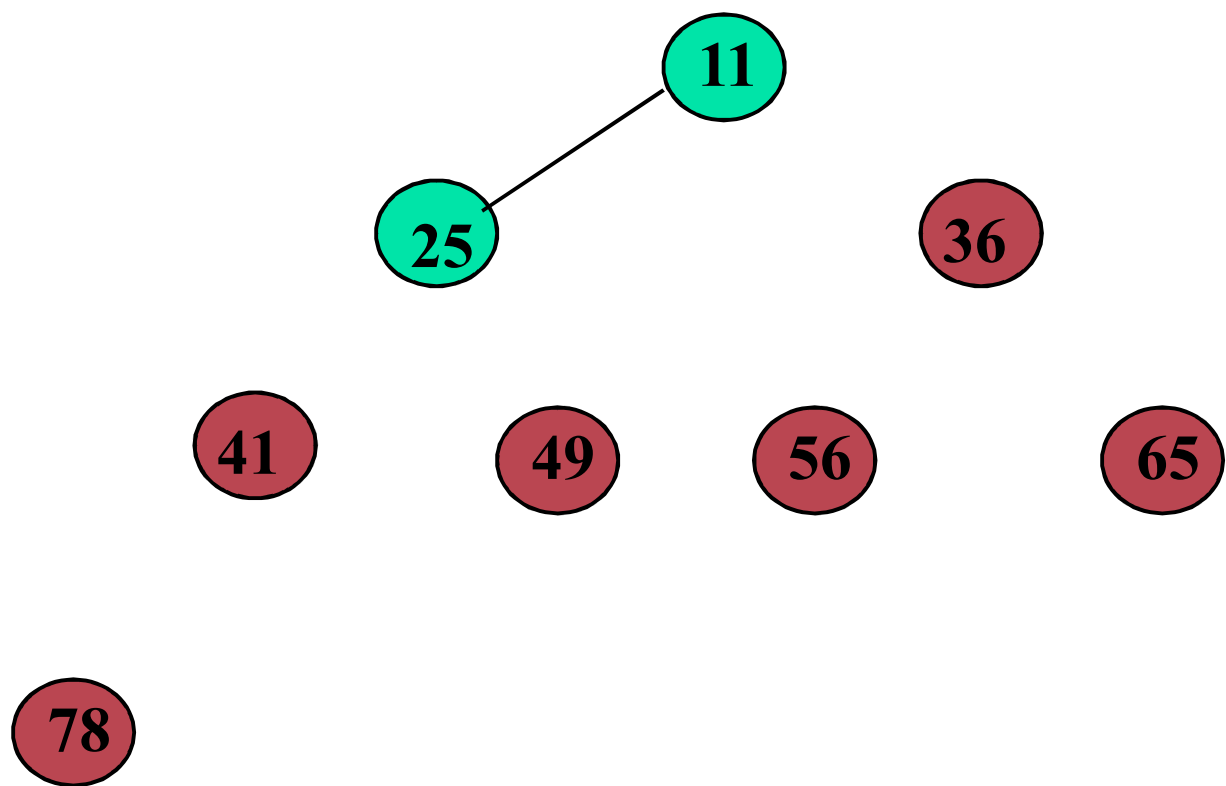
1	2	3	4	5	6	7	8
25	11	36	41	49	56	65	78



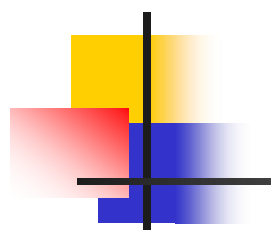
堆排序



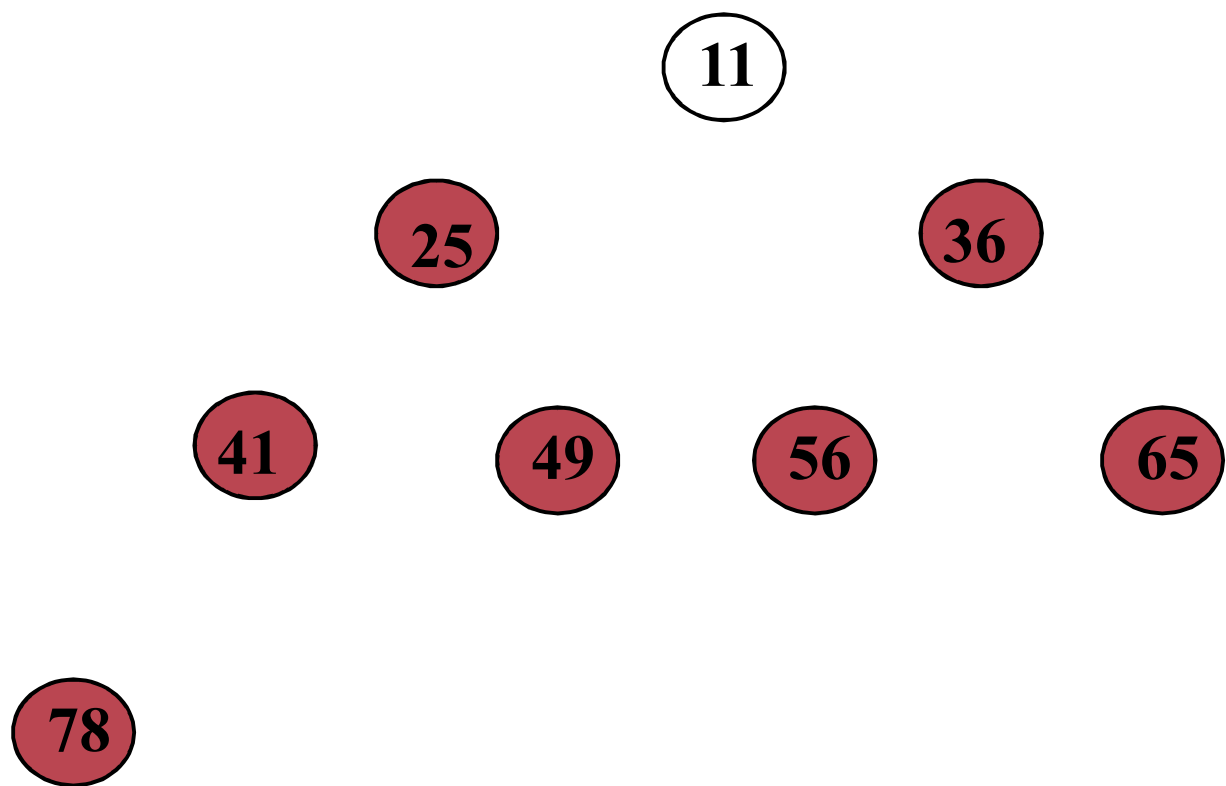
1	2	3	4	5	6	7	8
11	25	36	41	49	56	65	78



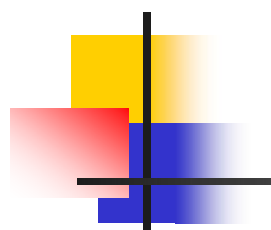
堆排序



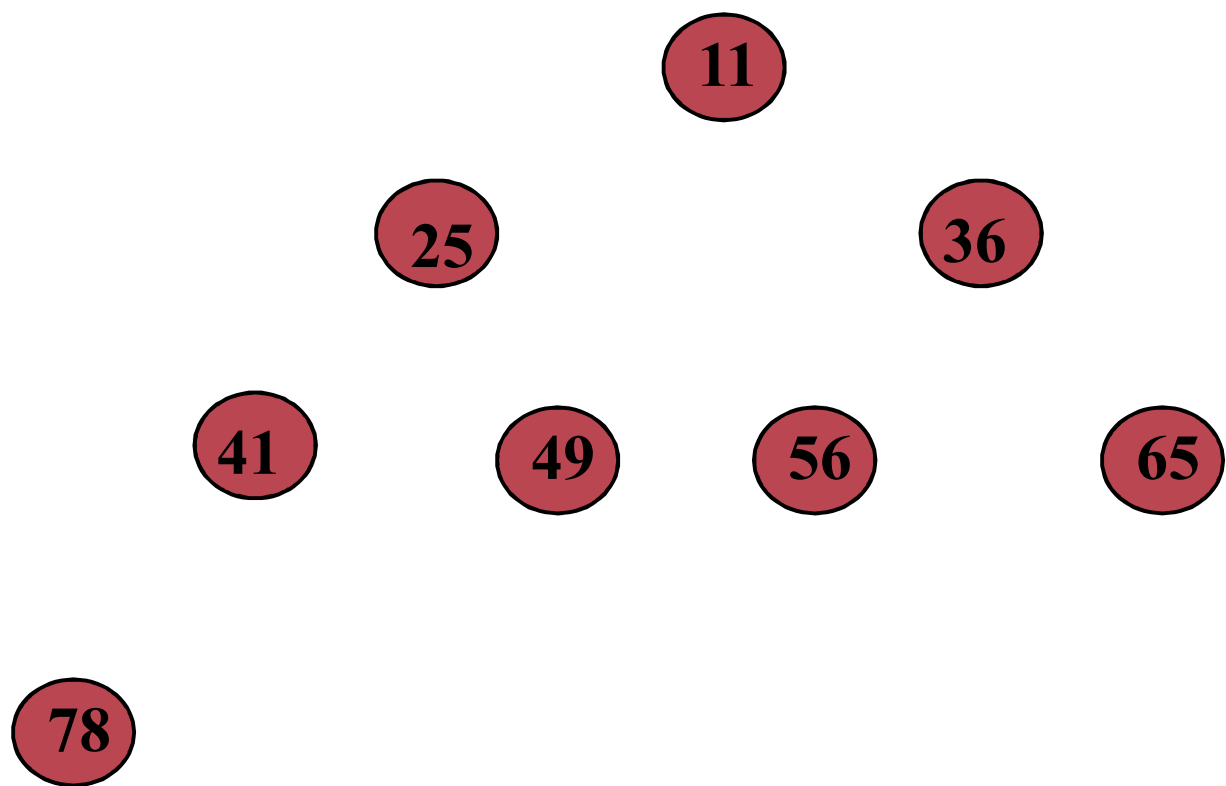
1	2	3	4	5	6	7	8
11	25	36	41	49	56	65	78



堆排序



1	2	3	4	5	6	7	8
11	25	36	41	49	56	65	78




堆排序

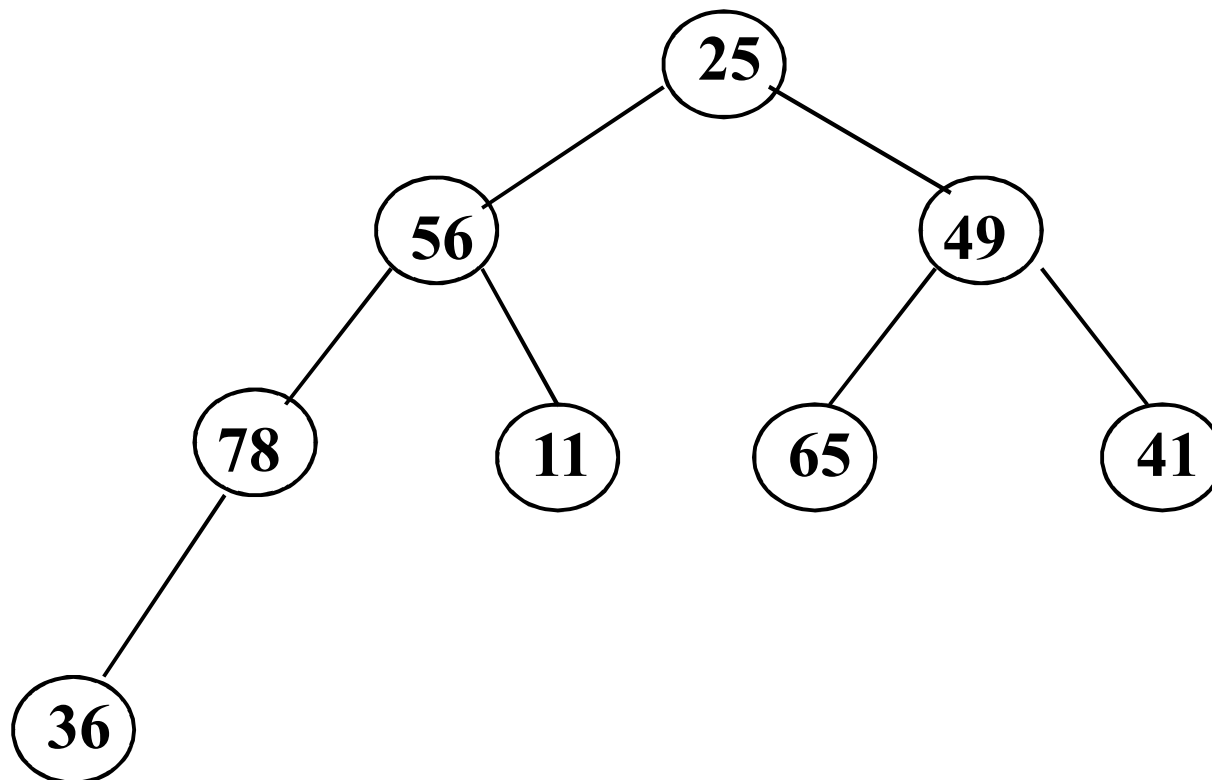


堆排序

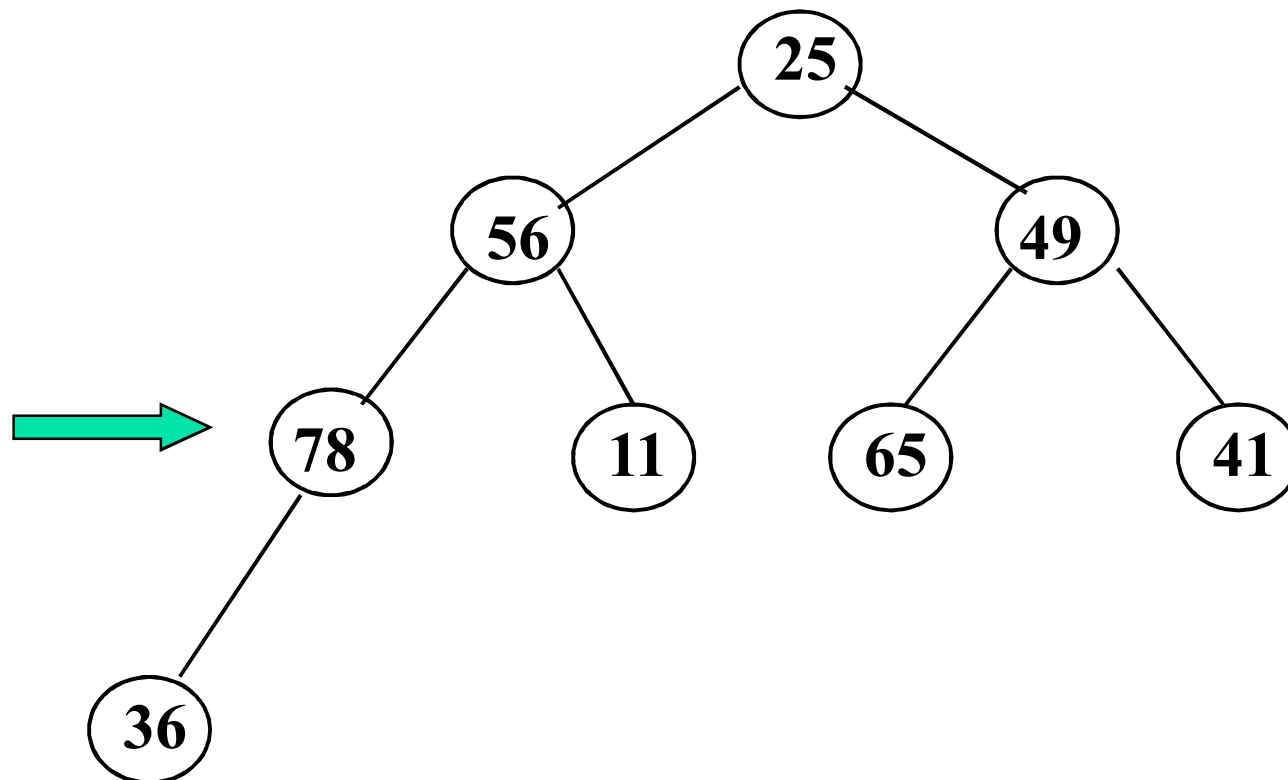
- 时间复杂度为 $O(n\log_2 n)$ 。
- 适用于待排序元素较多的情况。
- 一个额外的辅助空间, $O(1)$ 。
- 稳定性? -----不稳定



0	1	2	3	4	5	6	7	8
	25	56	49	78	11	65	41	36

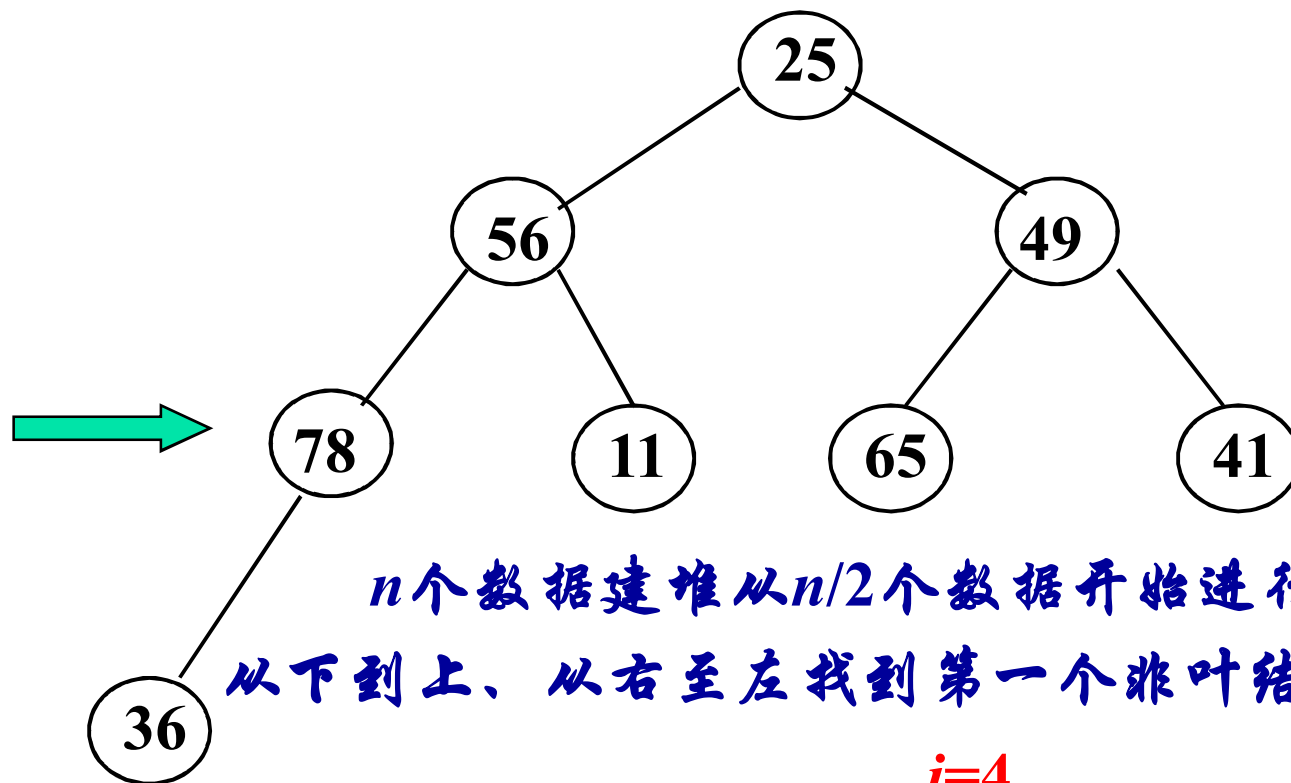


0	1	2	3	4	5	6	7	8
	25	56	49	78	11	65	41	36



若左、右孩子中最大的比第*i*个结点的值大则进行交换、调整；否则
该结点调整结束

0	1	2	3	4	5	6	7	8
78	25	56	49	78	11	65	41	36



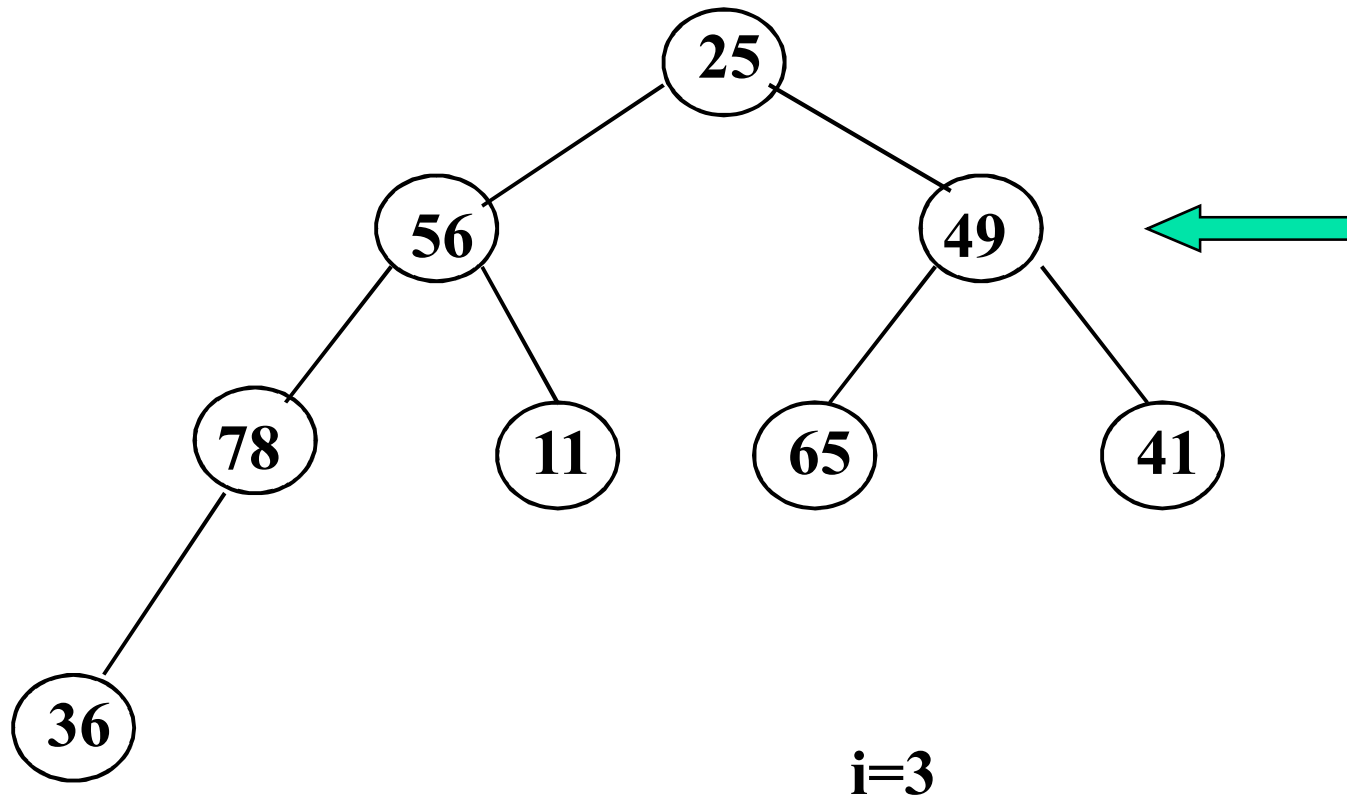
n 个数据建堆从 $n/2$ 个数据开始进行调整
从下到上、从右至左找到第一个非叶结点开始进行调整

$i=4$

$2i \leq n$ 有左孩子， $2i+1 \leq n$ 有右孩子
取左、右孩子中最大的与第*i*个结点比较

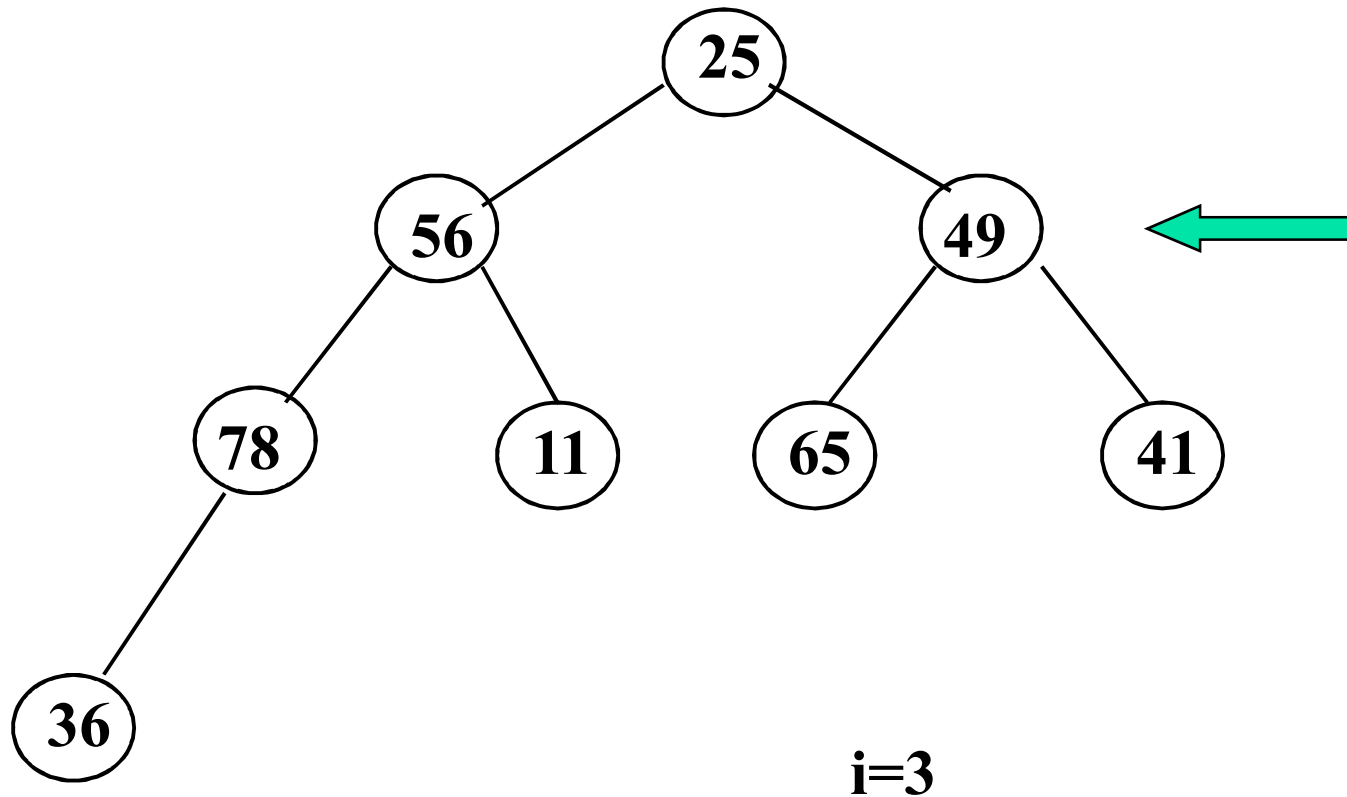
一个结点调整结束后，按照**下到上、从右至左**的顺序($i--$)找到下一个结点继续调整，直至根结点

0	1	2	3	4	5	6	7	8
	25	56	49	78	11	65	41	36



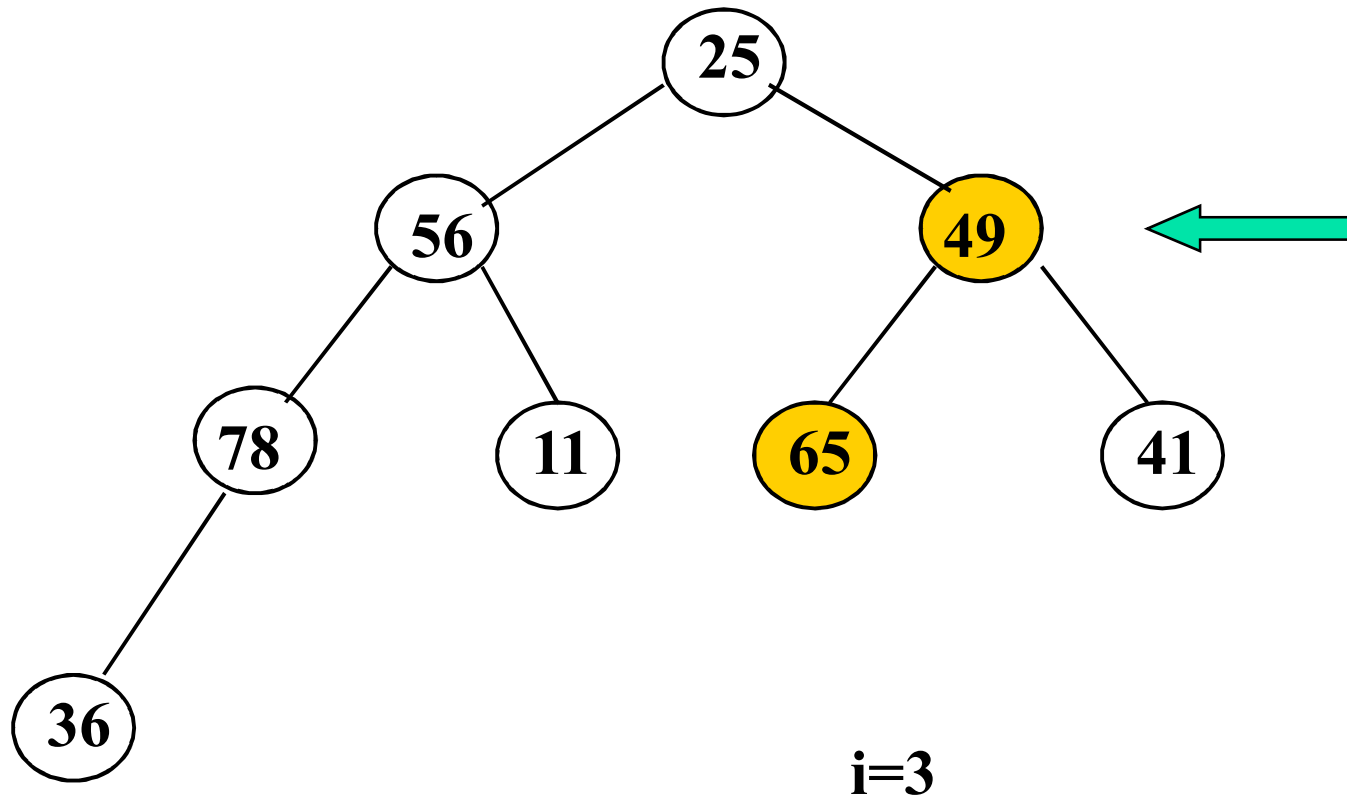
一个结点调整结束后，按照**下到上、从右至左**的顺序($i--$)找到下一个结点继续调整，直至根结点

0	1	2	3	4	5	6	7	8
49	25	56	49	78	11	65	41	36



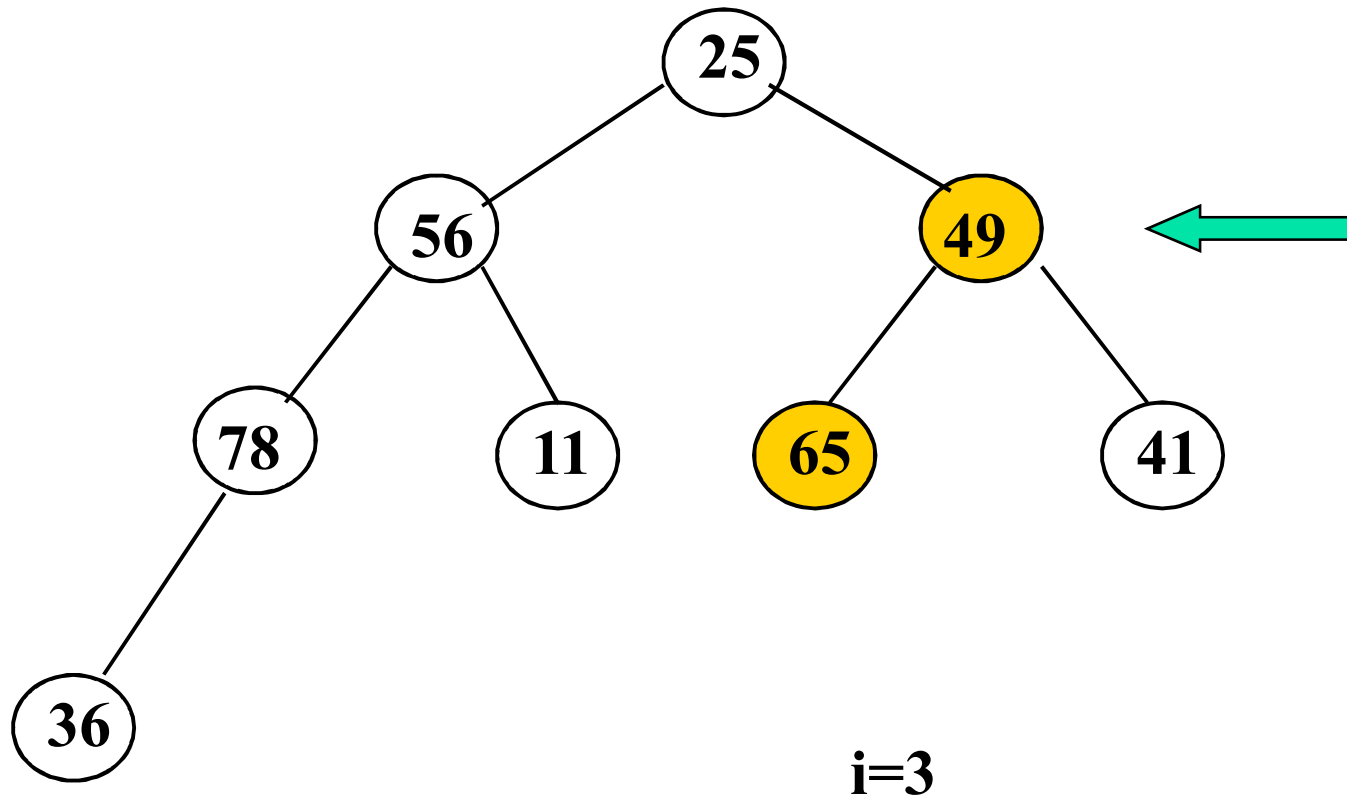
一个结点调整结束后，按照**下到上、从右至左**的顺序($i--$)找到下一个结点继续调整，直至根结点

0	1	2	3	4	5	6	7	8
49	25	56	49	78	11	65	41	36



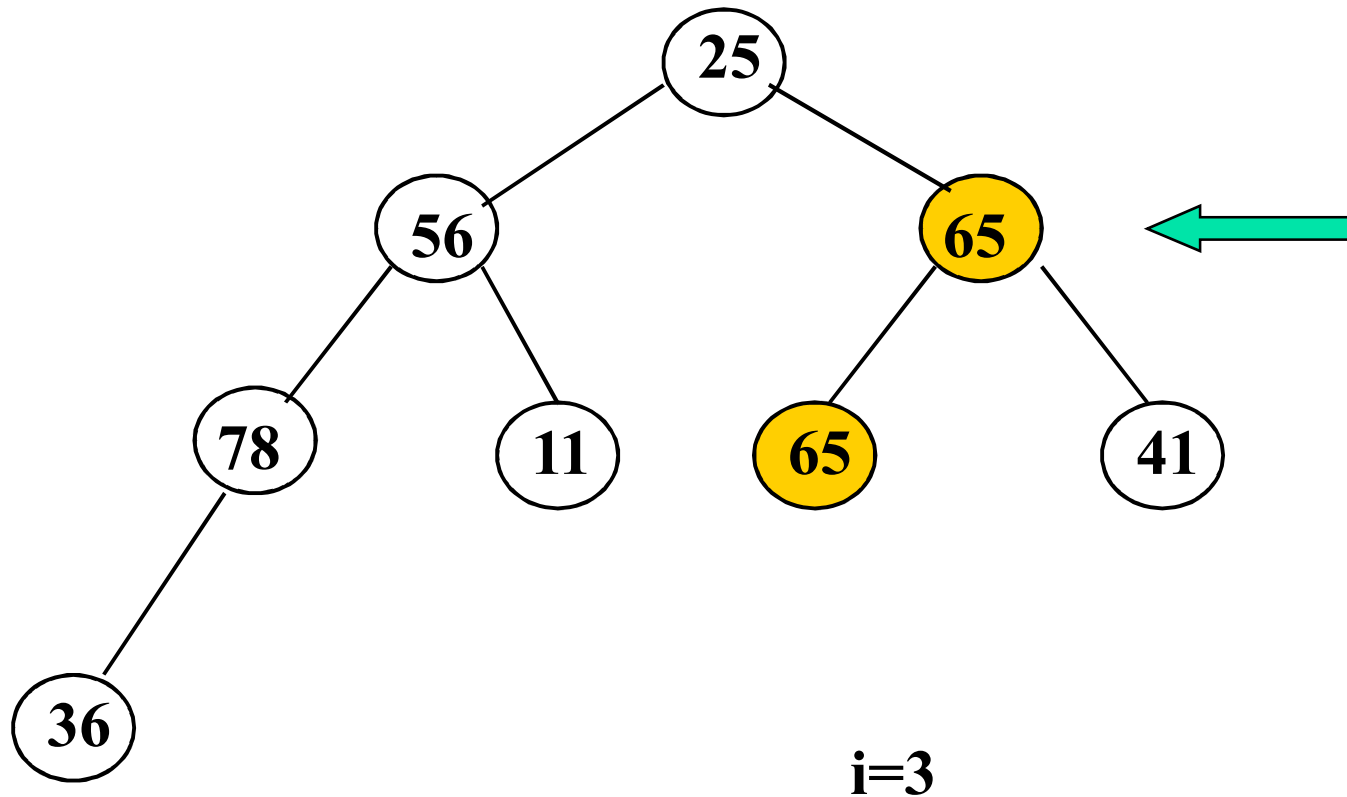
一个结点调整结束后，按照**下到上、从右至左**的顺序($i--$)找到下一个结点继续调整，直至根结点

0	1	2	3	4	5	6	7	8
49	25	56	49	78	11	65	41	36

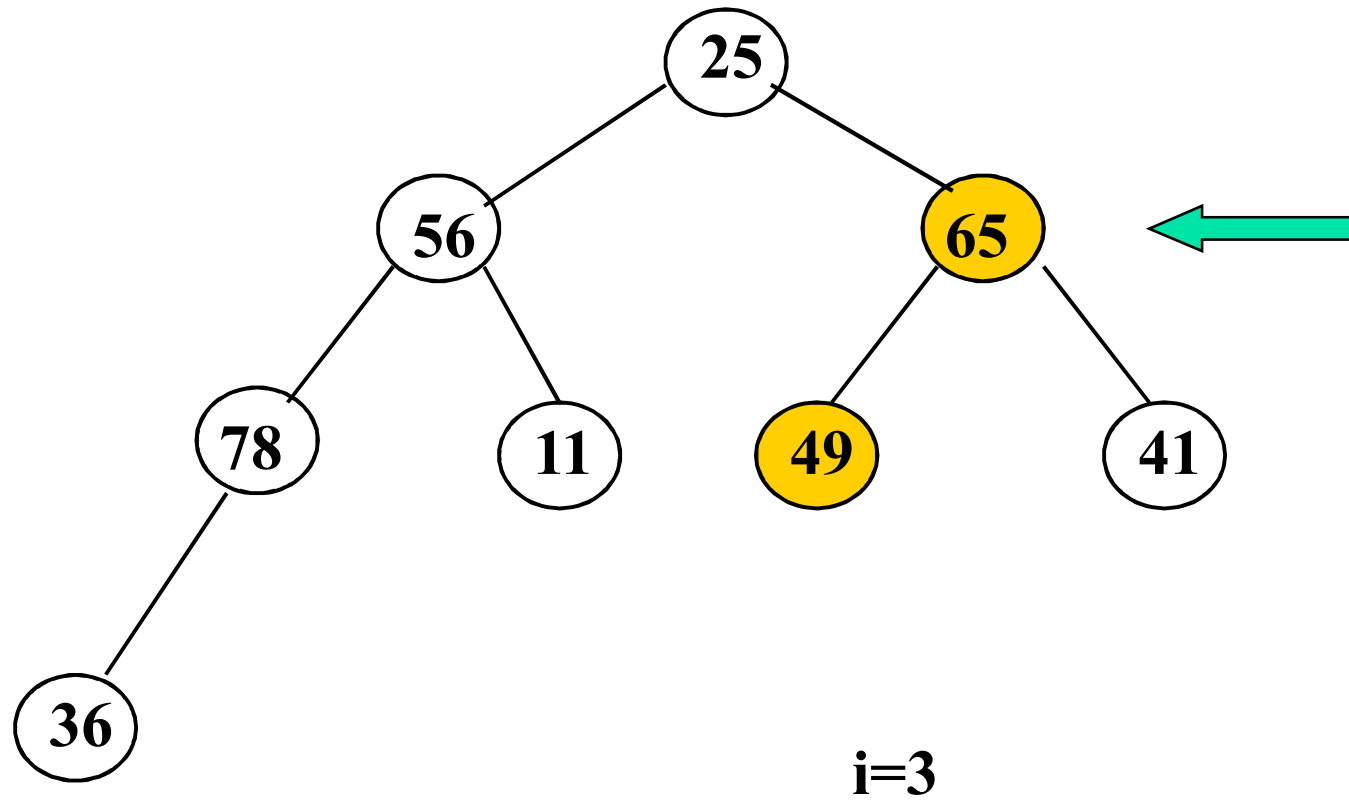


一个结点调整结束后，按照**下到上、从右至左**的顺序($i--$)找到下一个结点继续调整，直至根结点

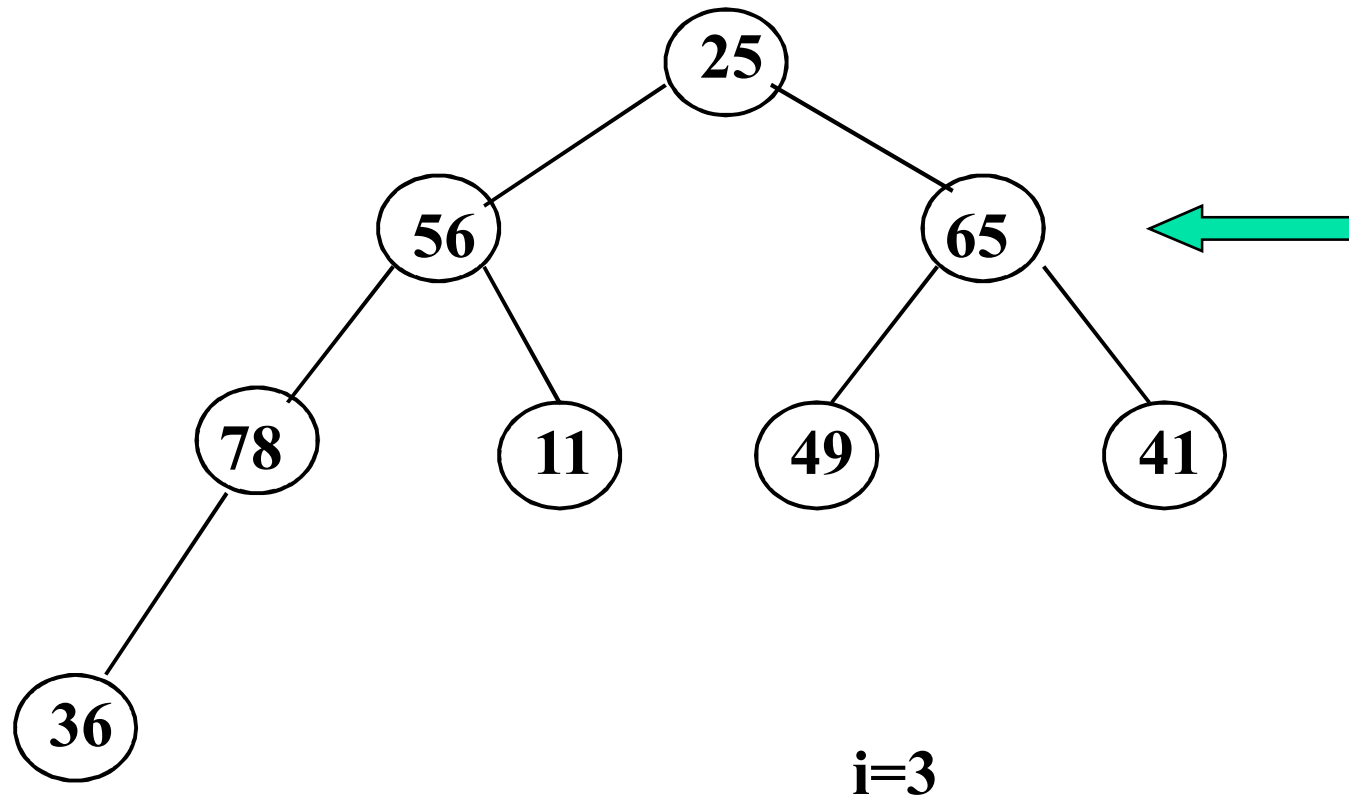
0	1	2	3	4	5	6	7	8
49	25	56	65	78	11	65	41	36



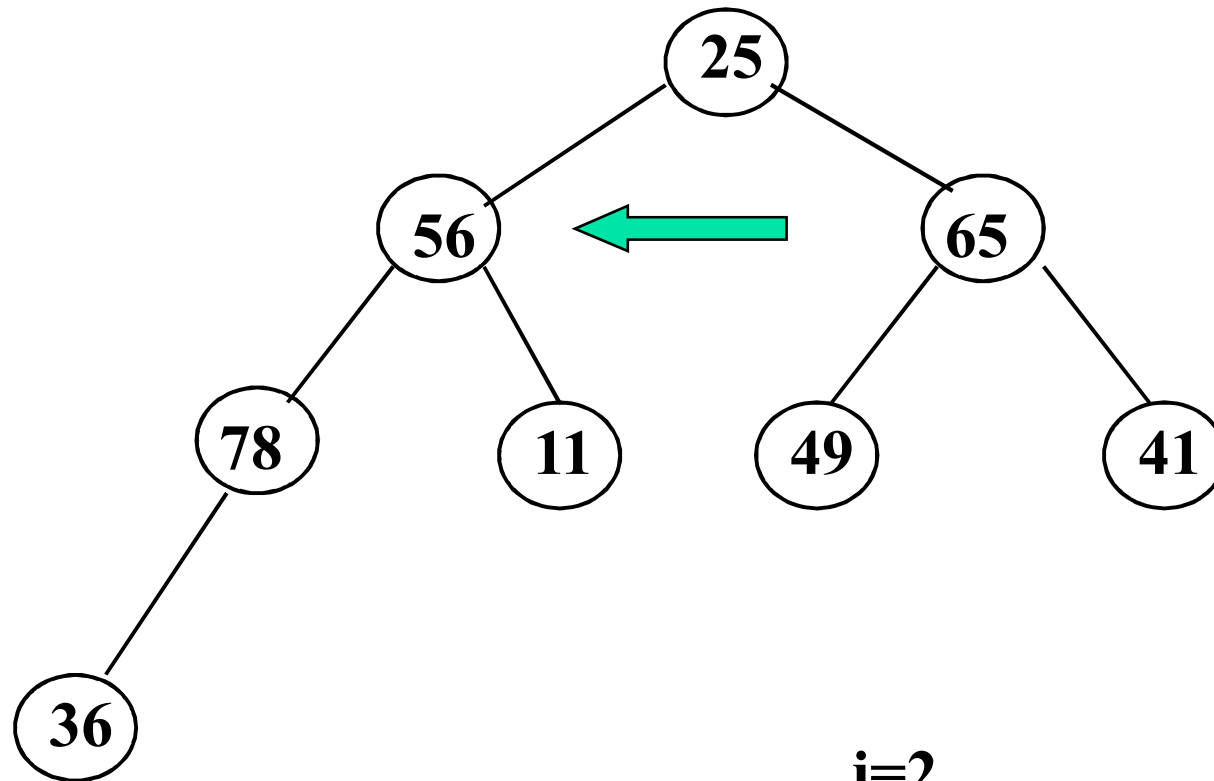
0	1	2	3	4	5	6	7	8
49	25	56	65	78	11	49	41	36



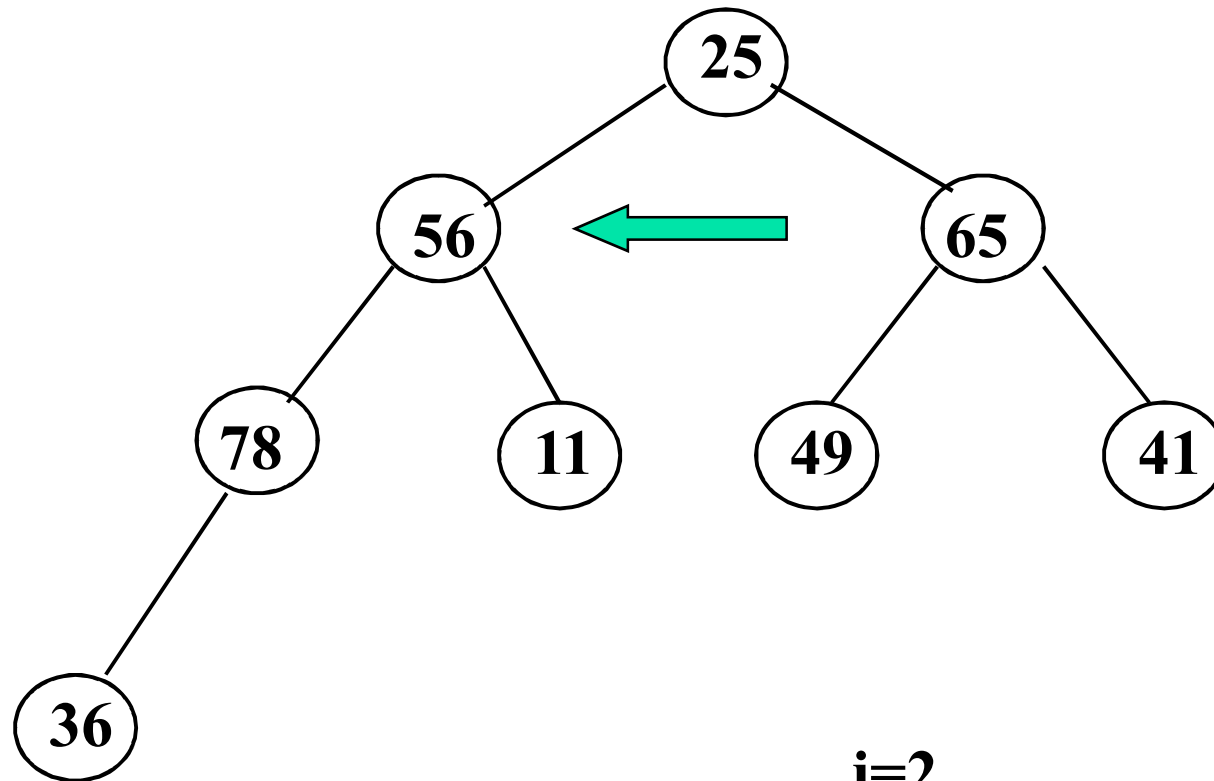
0	1	2	3	4	5	6	7	8
49	25	56	65	78	11	49	41	36



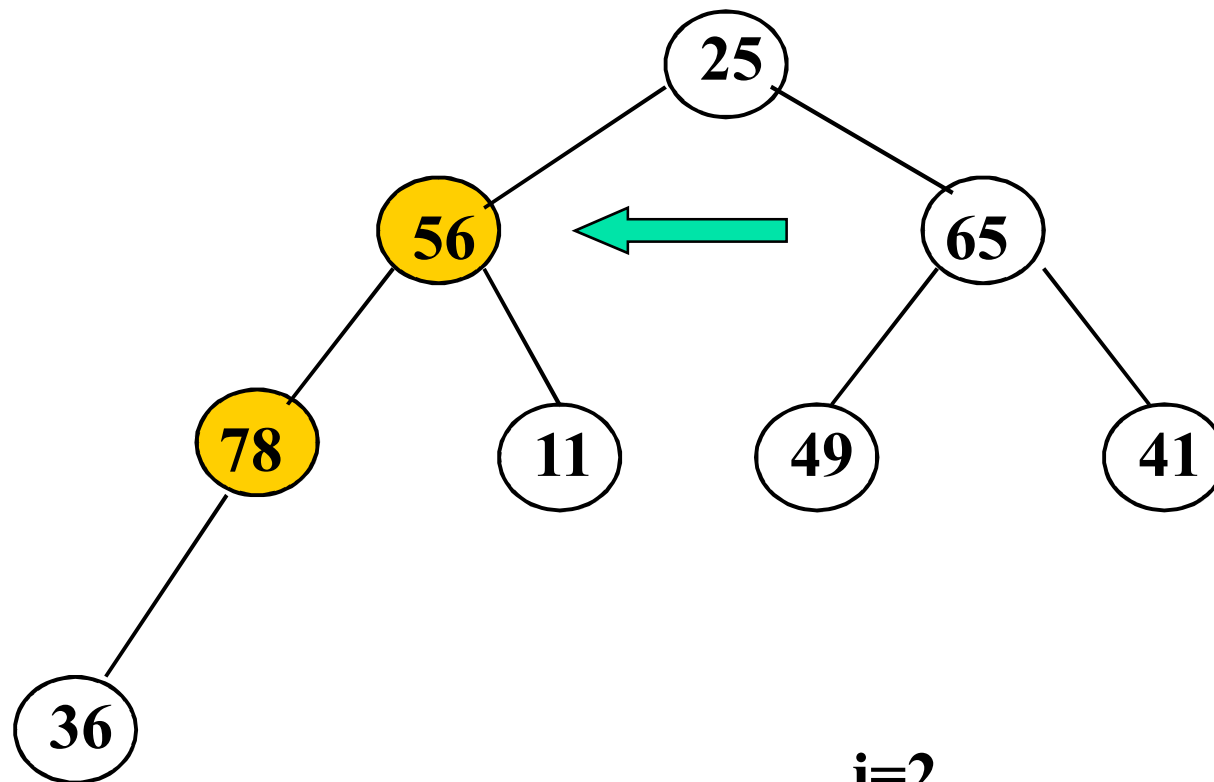
0	1	2	3	4	5	6	7	8
49	25	56	65	78	11	49	41	36



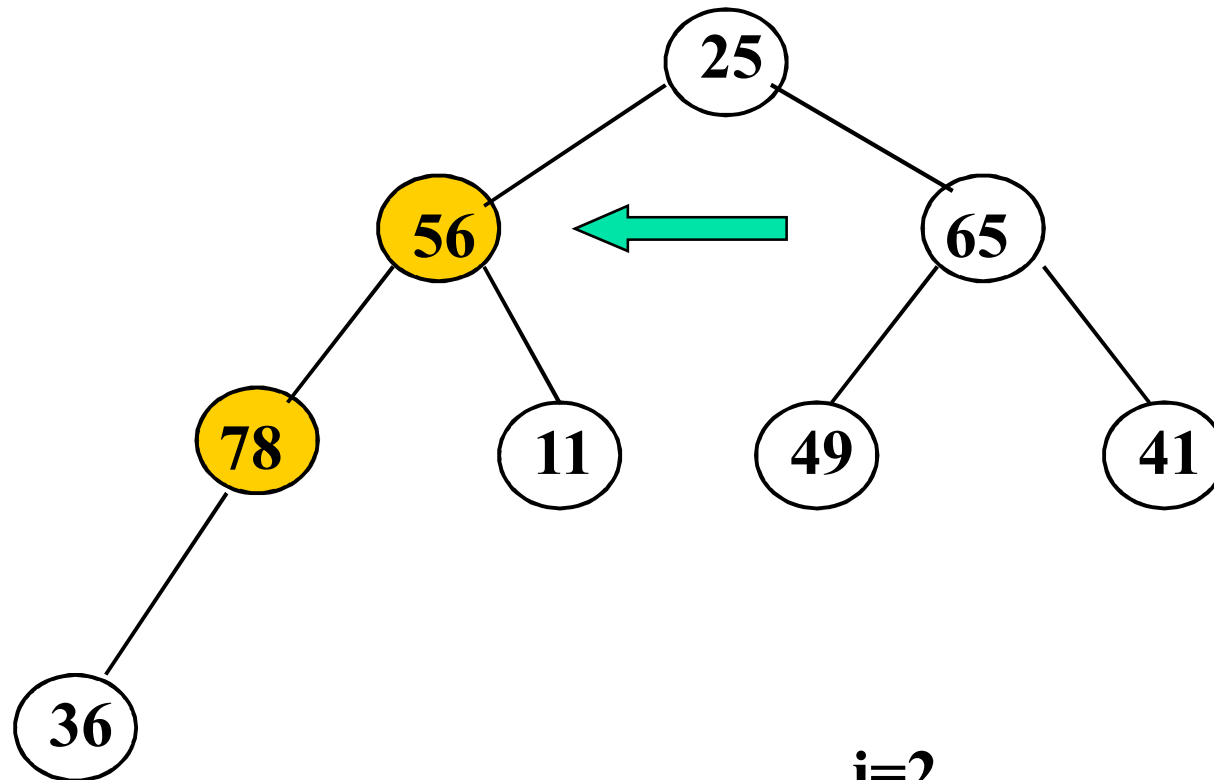
0	1	2	3	4	5	6	7	8
56	25	56	65	78	11	49	41	36



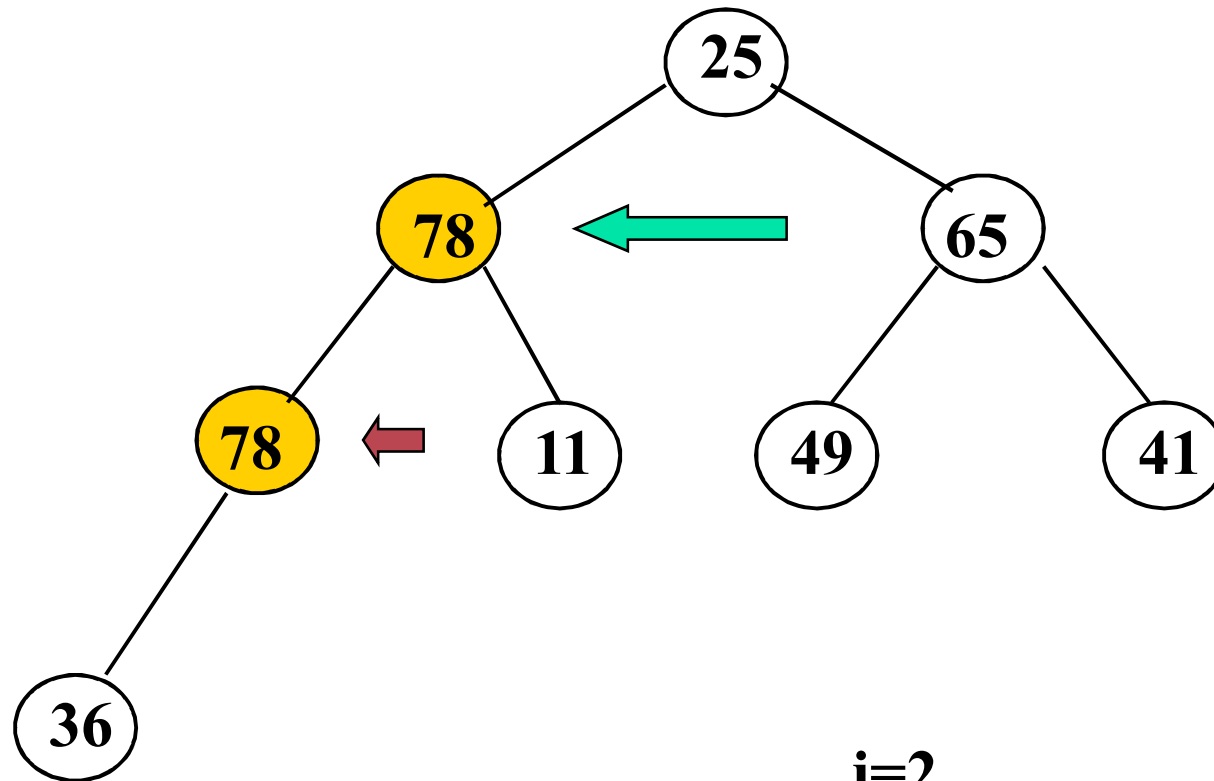
0	1	2	3	4	5	6	7	8
56	25	56	65	78	11	49	41	36



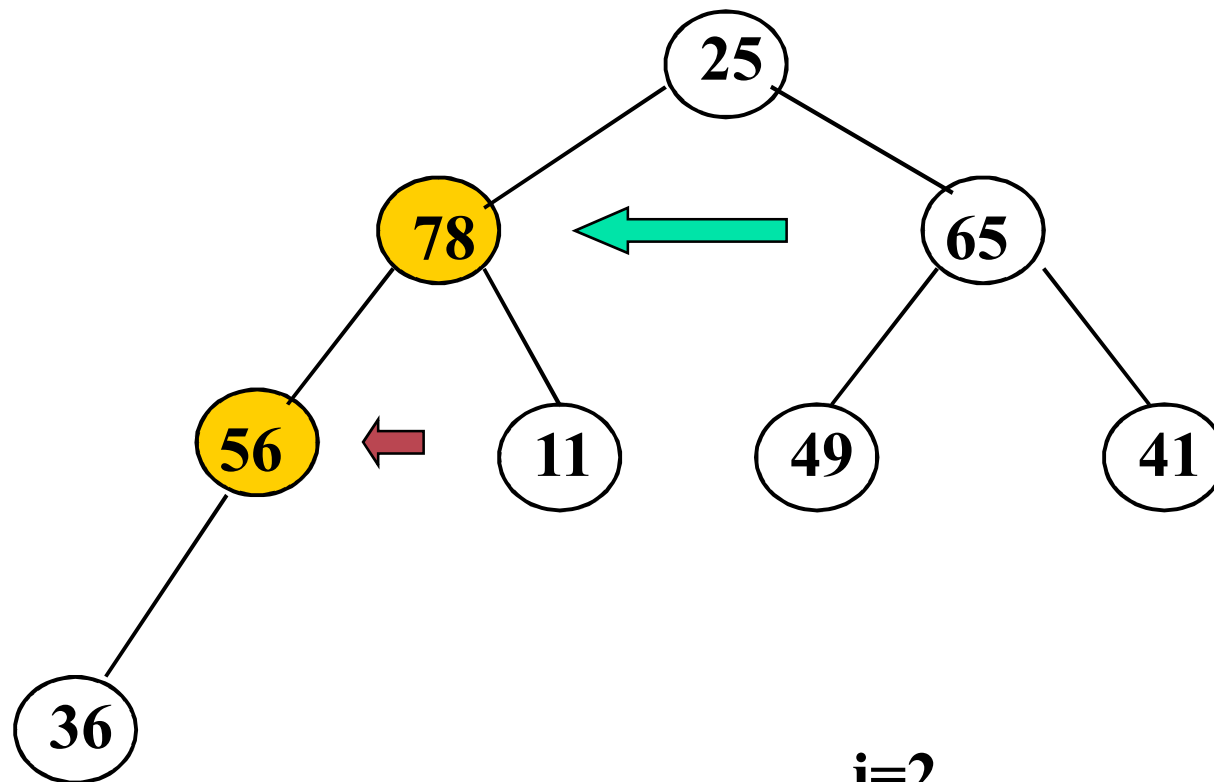
0	1	2	3	4	5	6	7	8
56	25	56	65	78	11	49	41	36



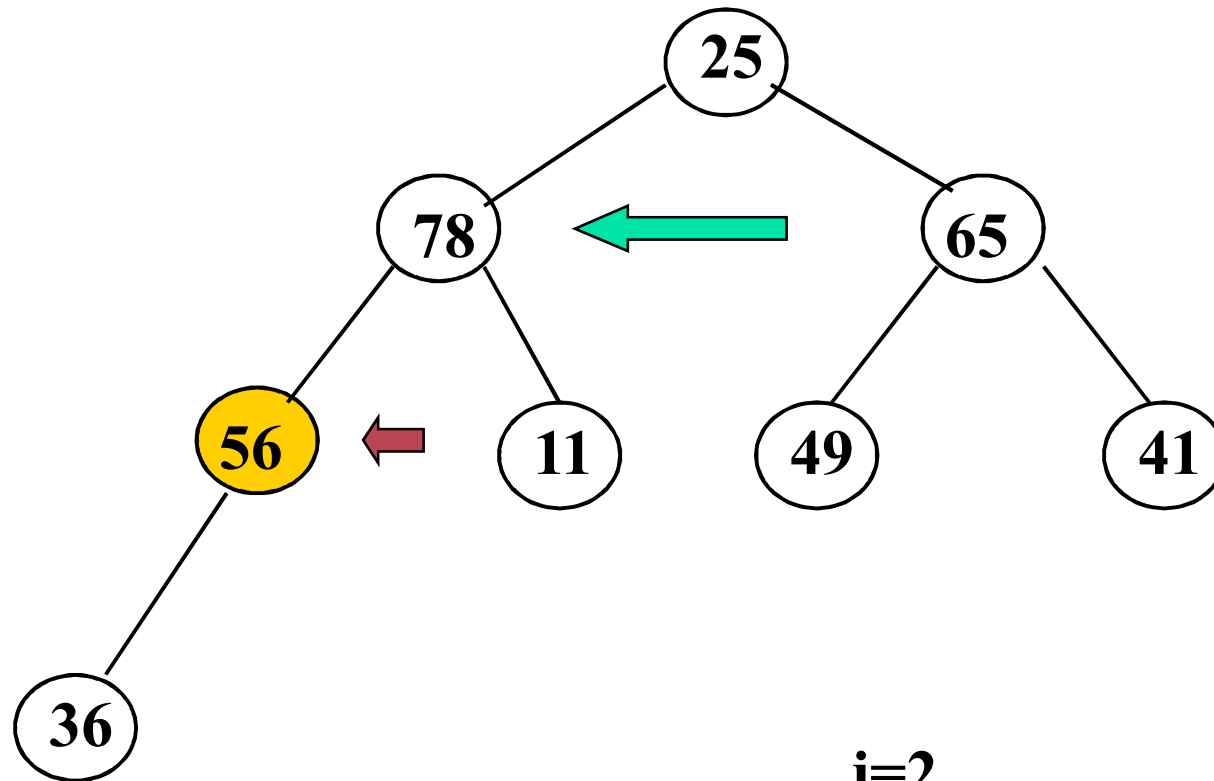
0	1	2	3	4	5	6	7	8
56	25	78	65	78	11	49	41	36

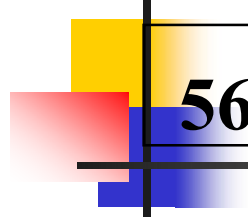


0	1	2	3	4	5	6	7	8
56	25	78	65	56	11	49	41	36

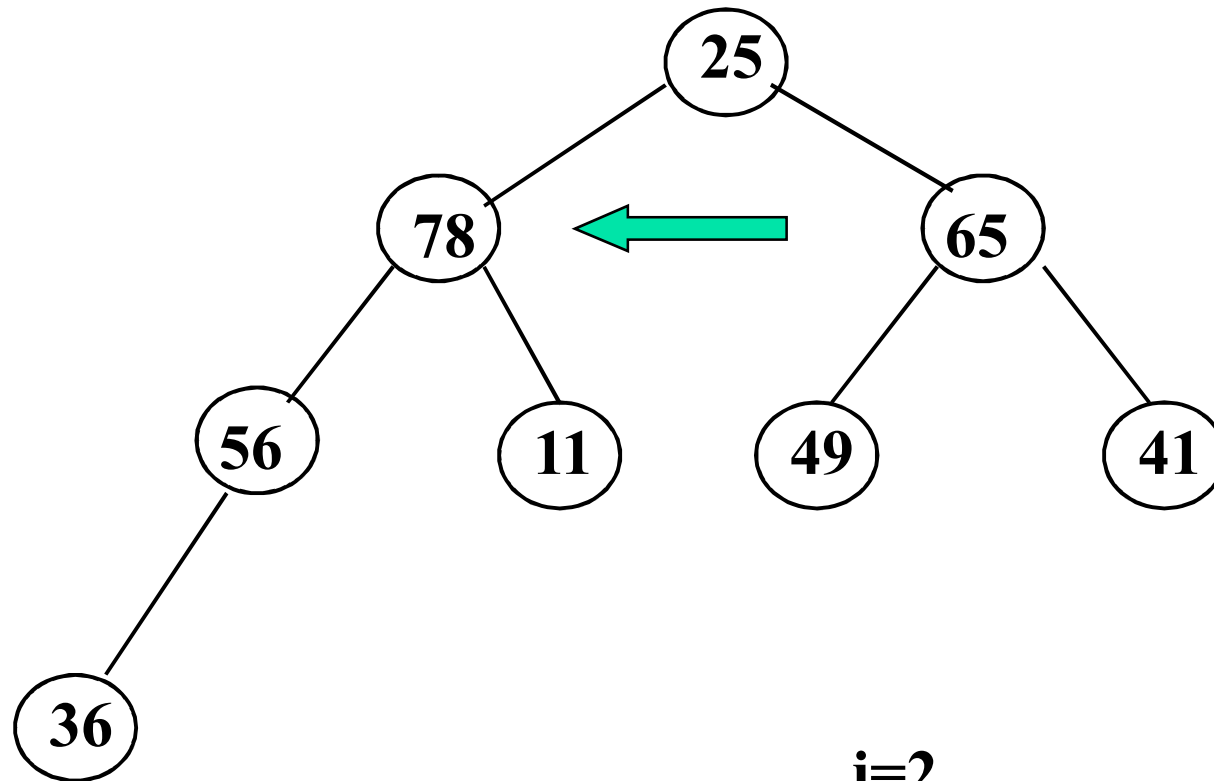


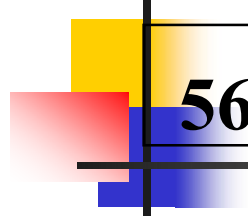
0	1	2	3	4	5	6	7	8
56	25	78	65	56	11	49	41	36



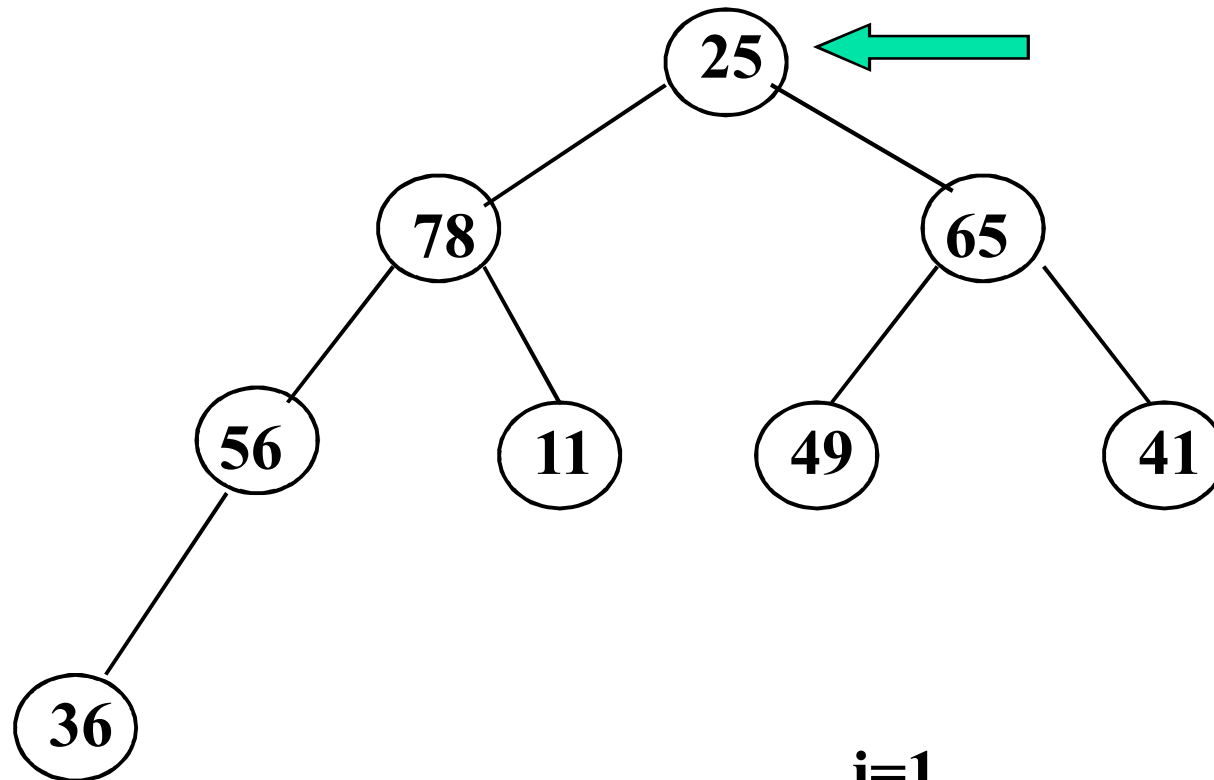


0	1	2	3	4	5	6	7	8
56	25	78	65	56	11	49	41	36

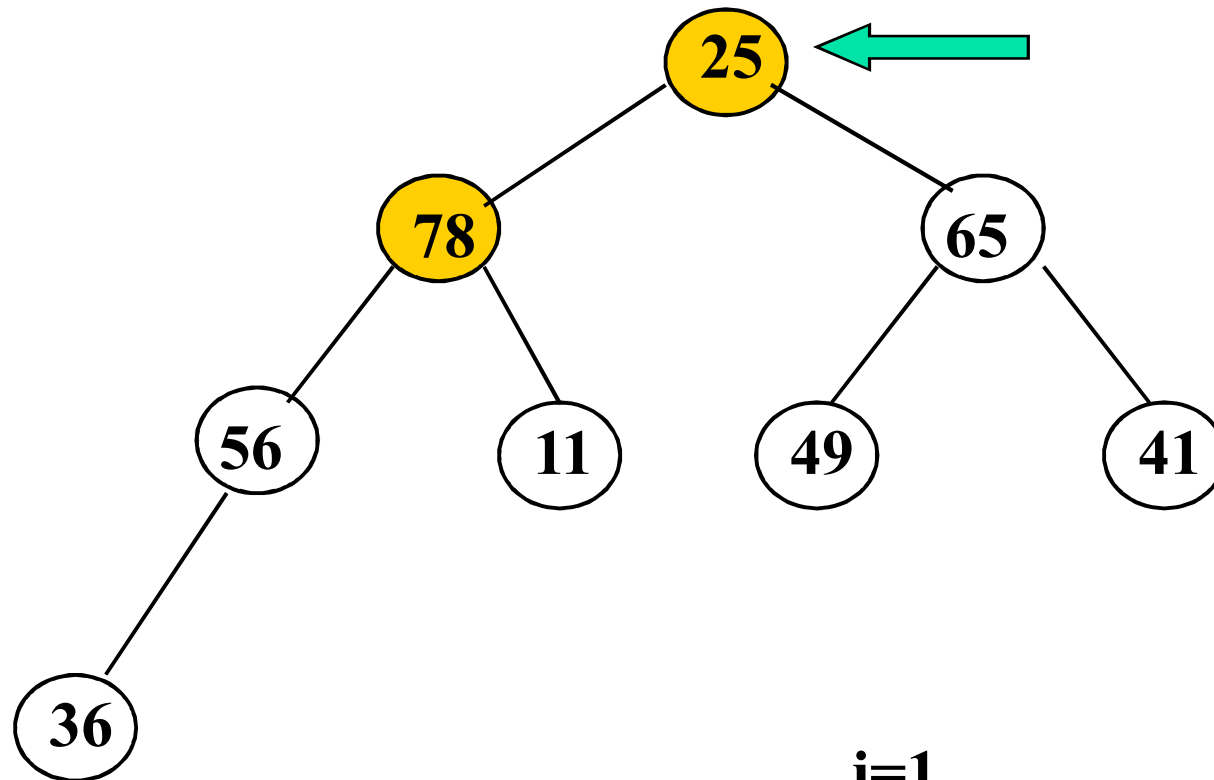




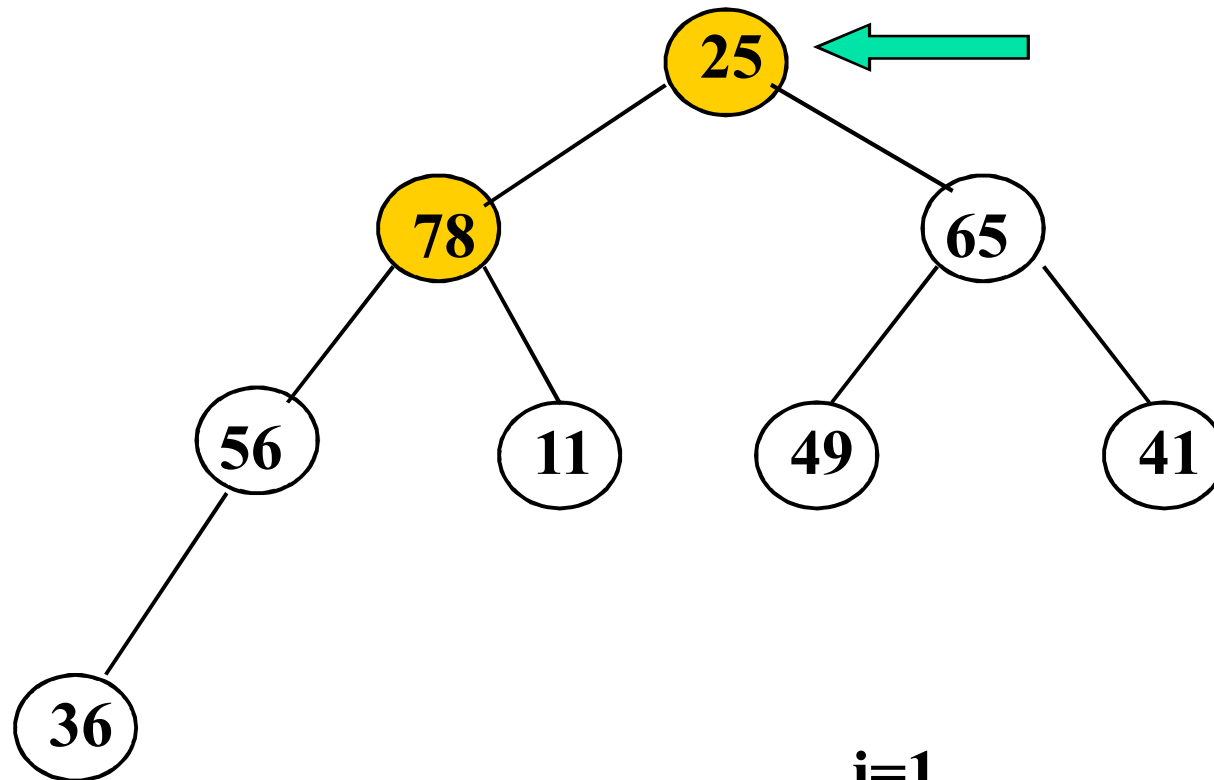
0	1	2	3	4	5	6	7	8
56	25	78	65	56	11	49	41	36



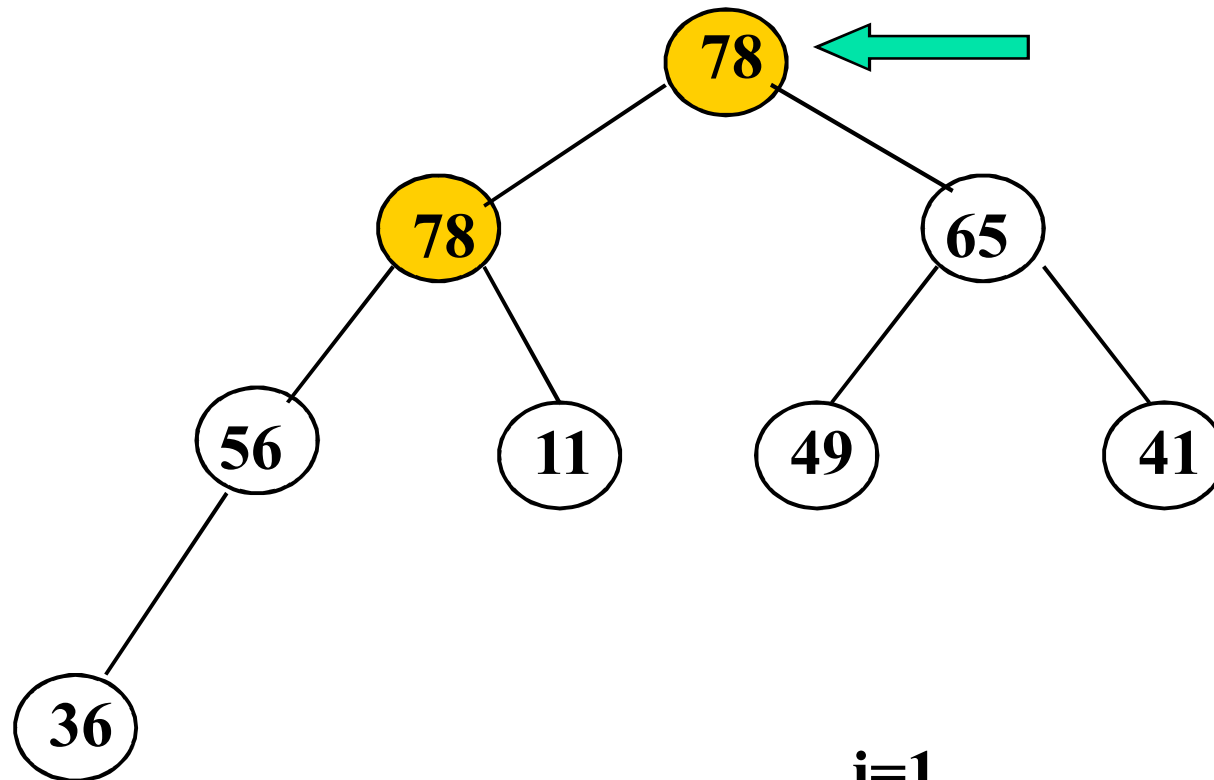
0	1	2	3	4	5	6	7	8
25	25	78	65	56	11	49	41	36



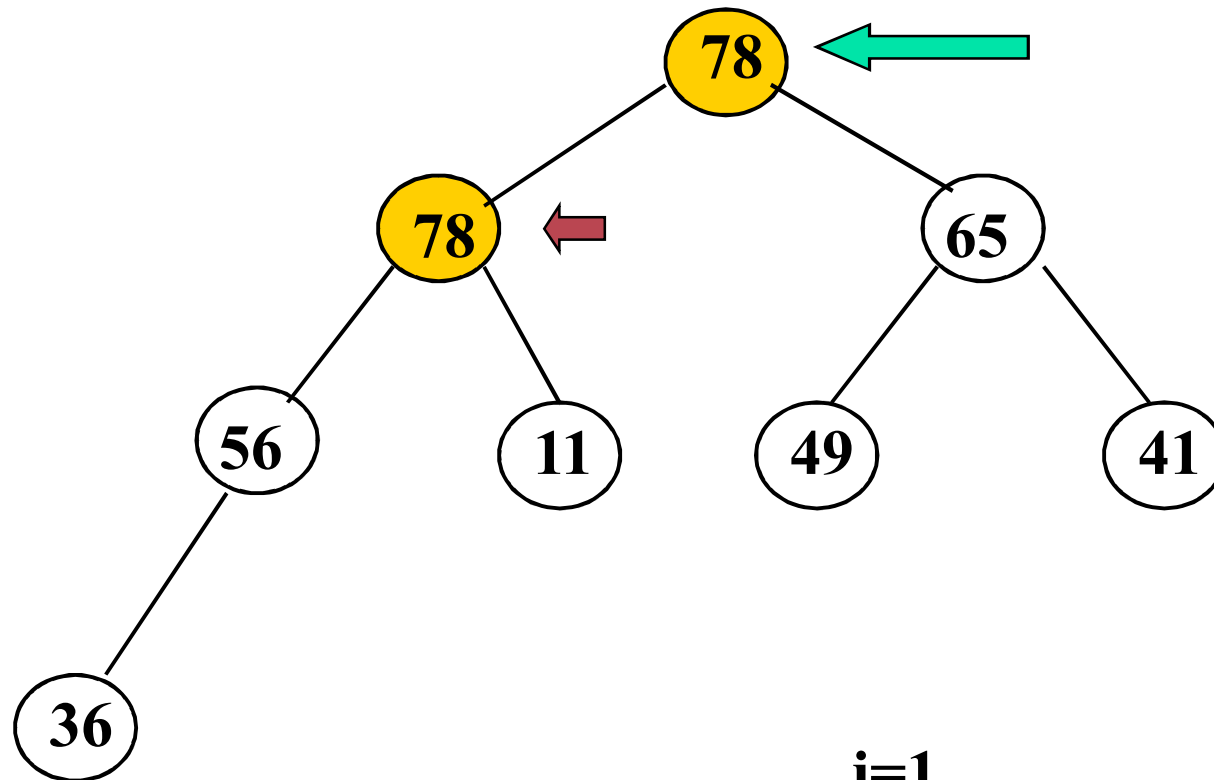
0	1	2	3	4	5	6	7	8
25	25	78	65	56	11	49	41	36



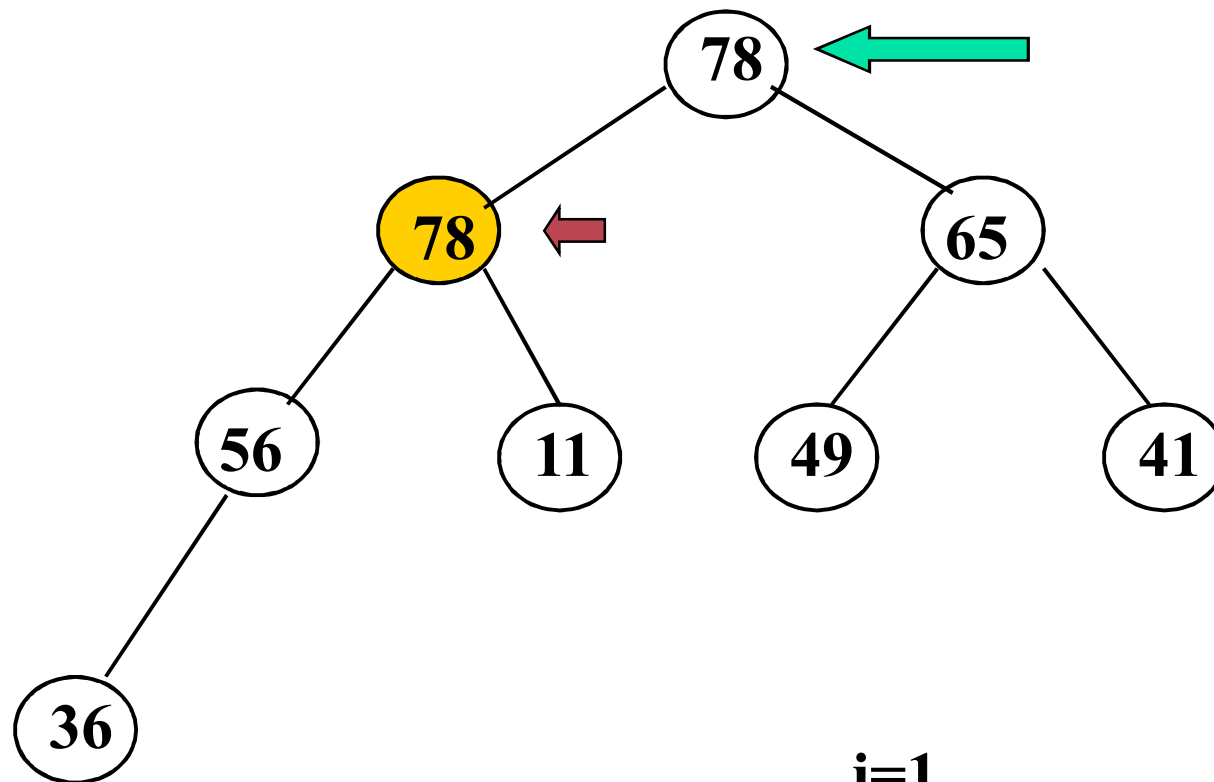
0	1	2	3	4	5	6	7	8
25	78	78	65	56	11	49	41	36



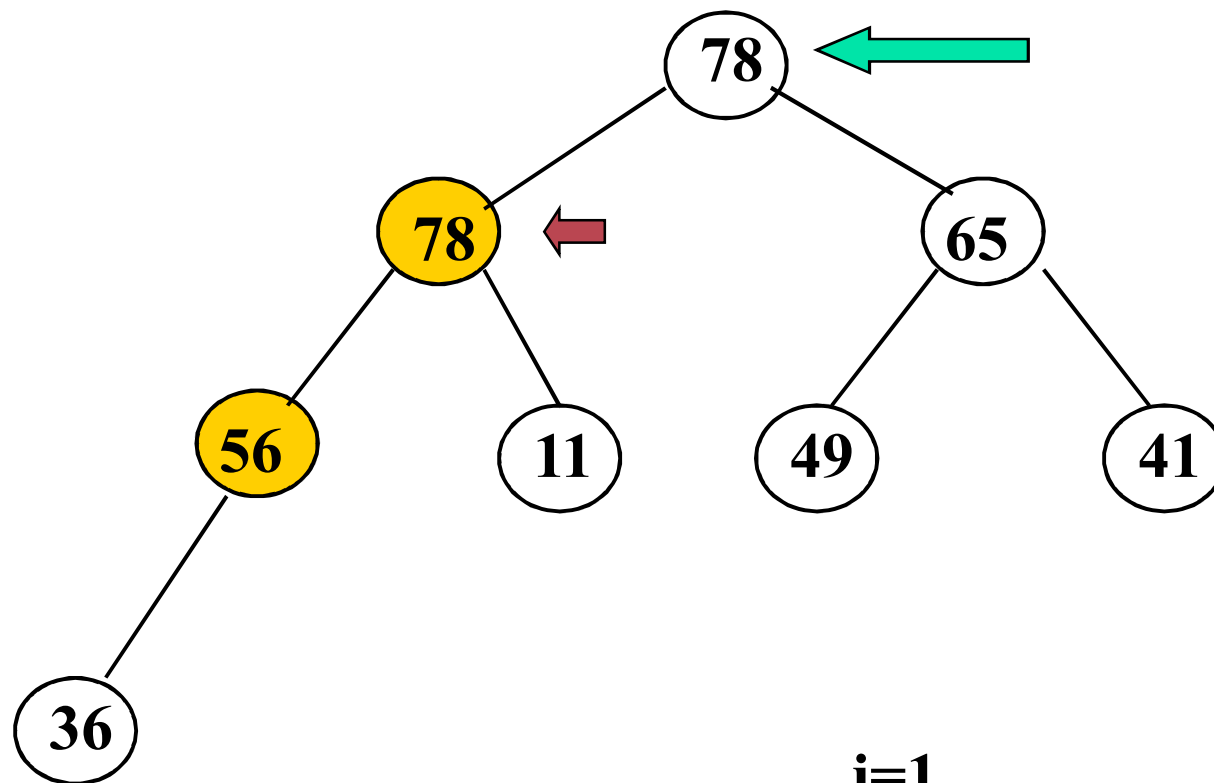
0	1	2	3	4	5	6	7	8
25	78	78	65	56	11	49	41	36



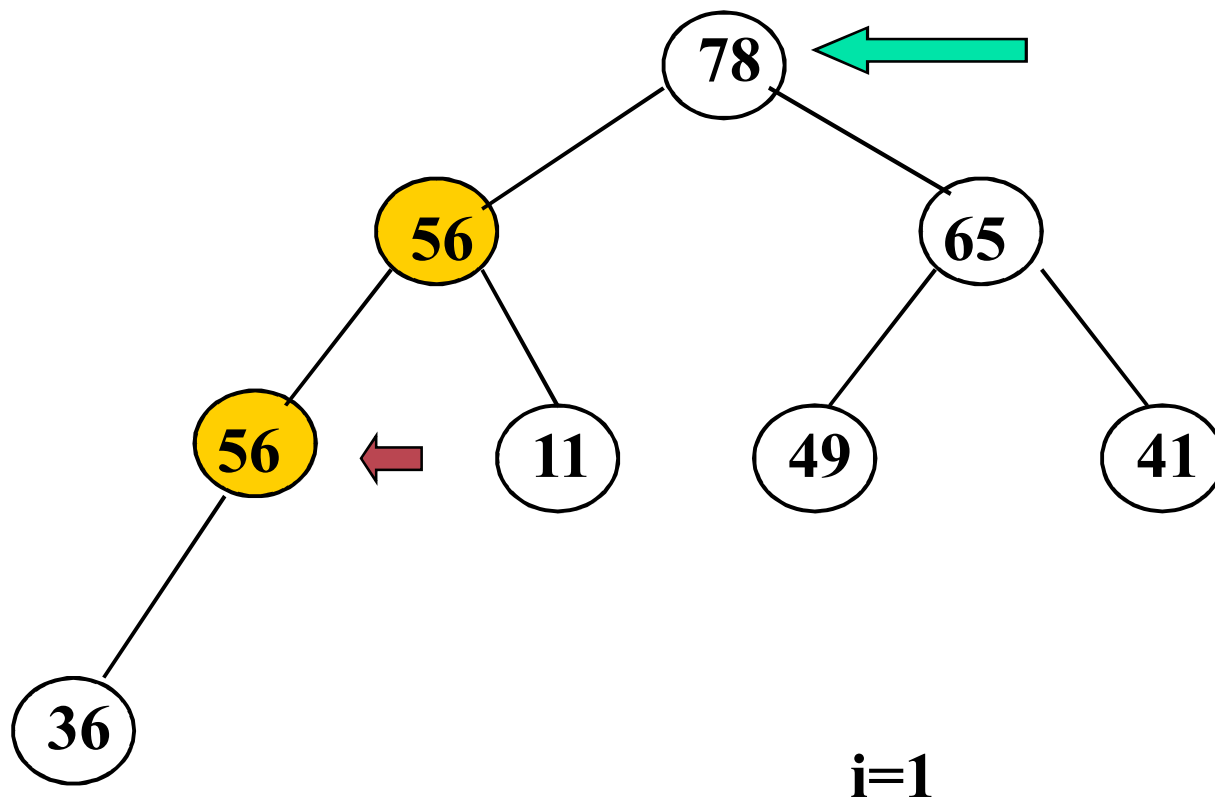
0	1	2	3	4	5	6	7	8
25	78	78	65	56	11	49	41	36



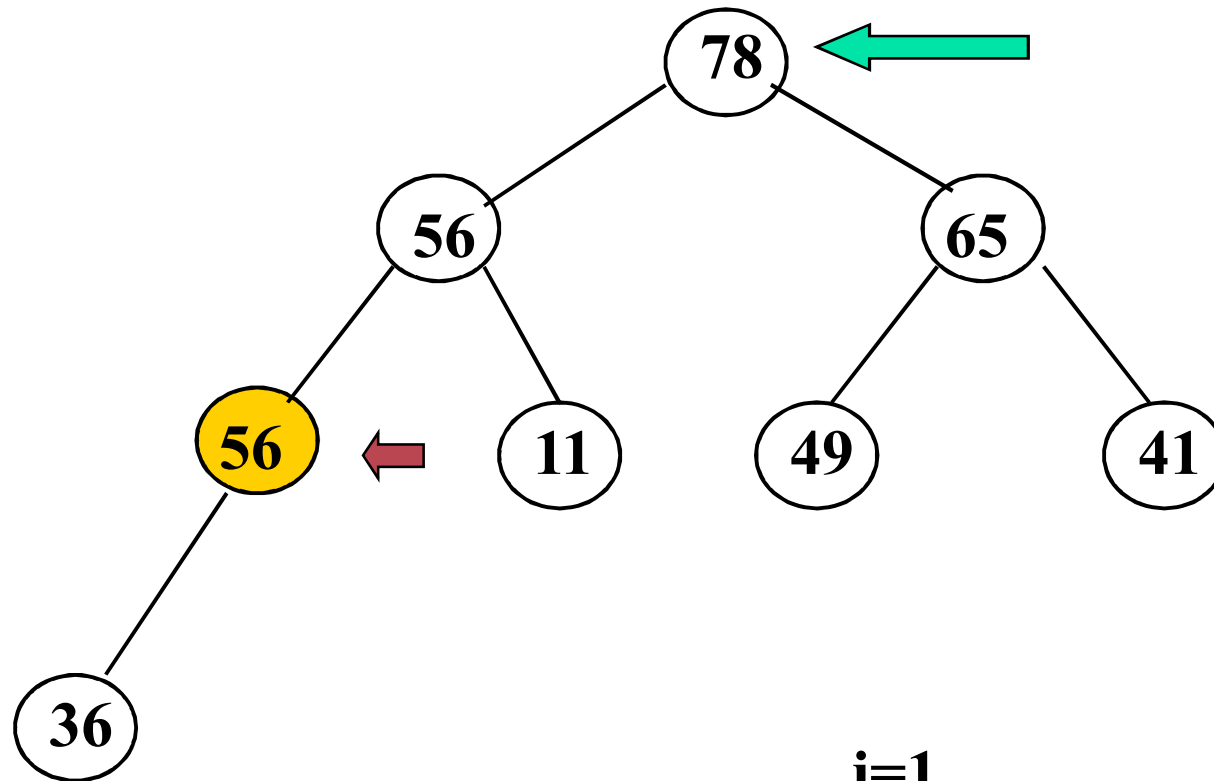
0	1	2	3	4	5	6	7	8
25	78	78	65	56	11	49	41	36



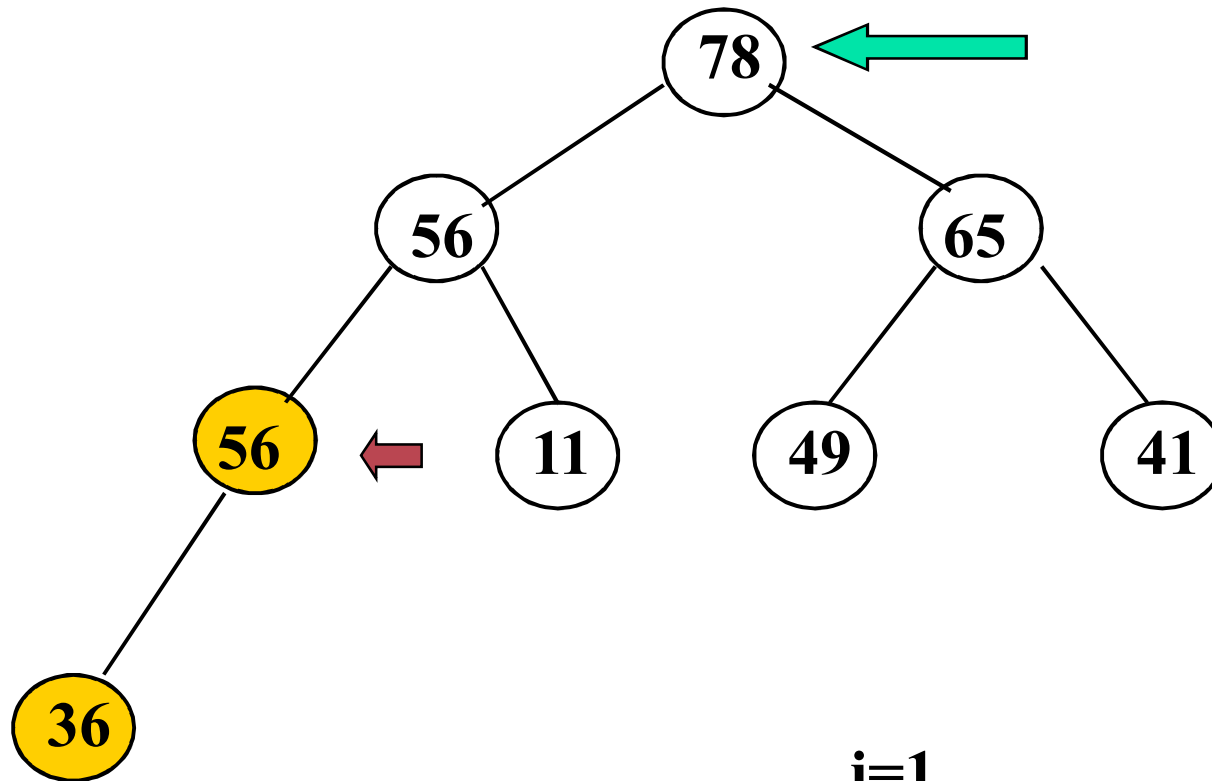
	1	2	3	4	5	6	7	8
	25	78	56	65	56	11	49	41
	36							



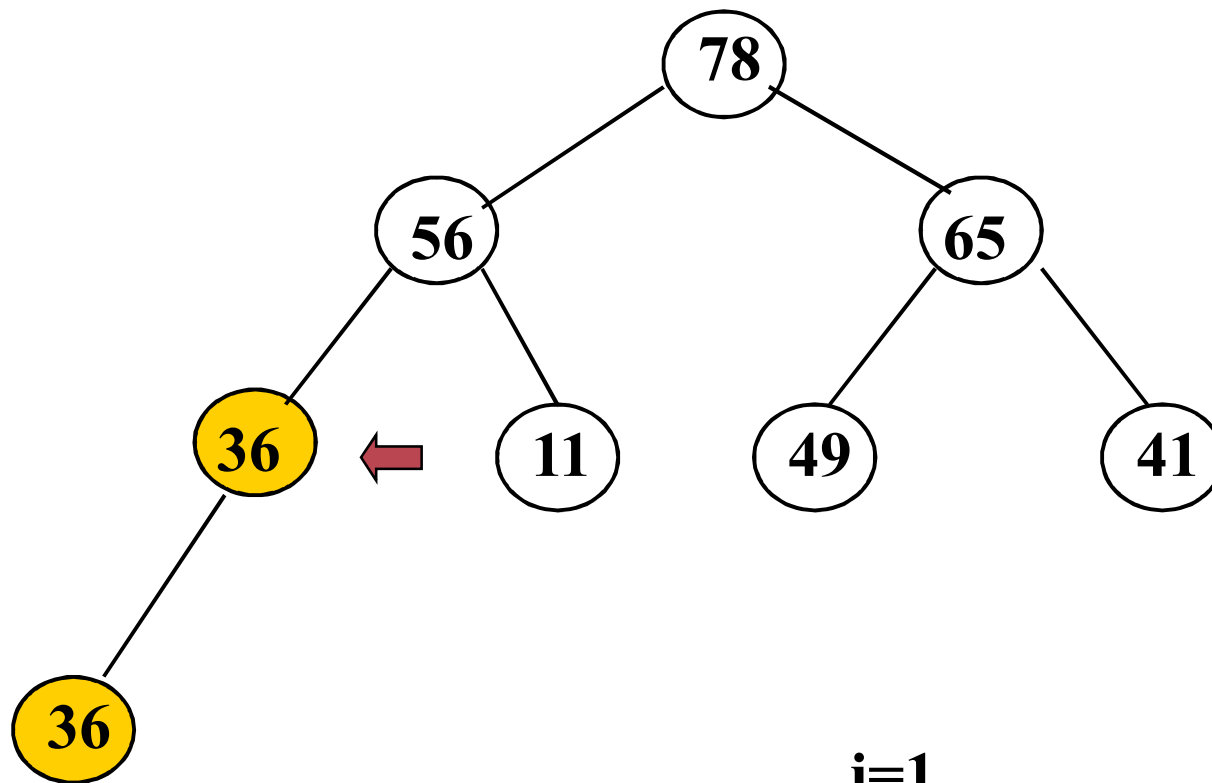
0	1	2	3	4	5	6	7	8
25	78	56	65	56	11	49	41	36



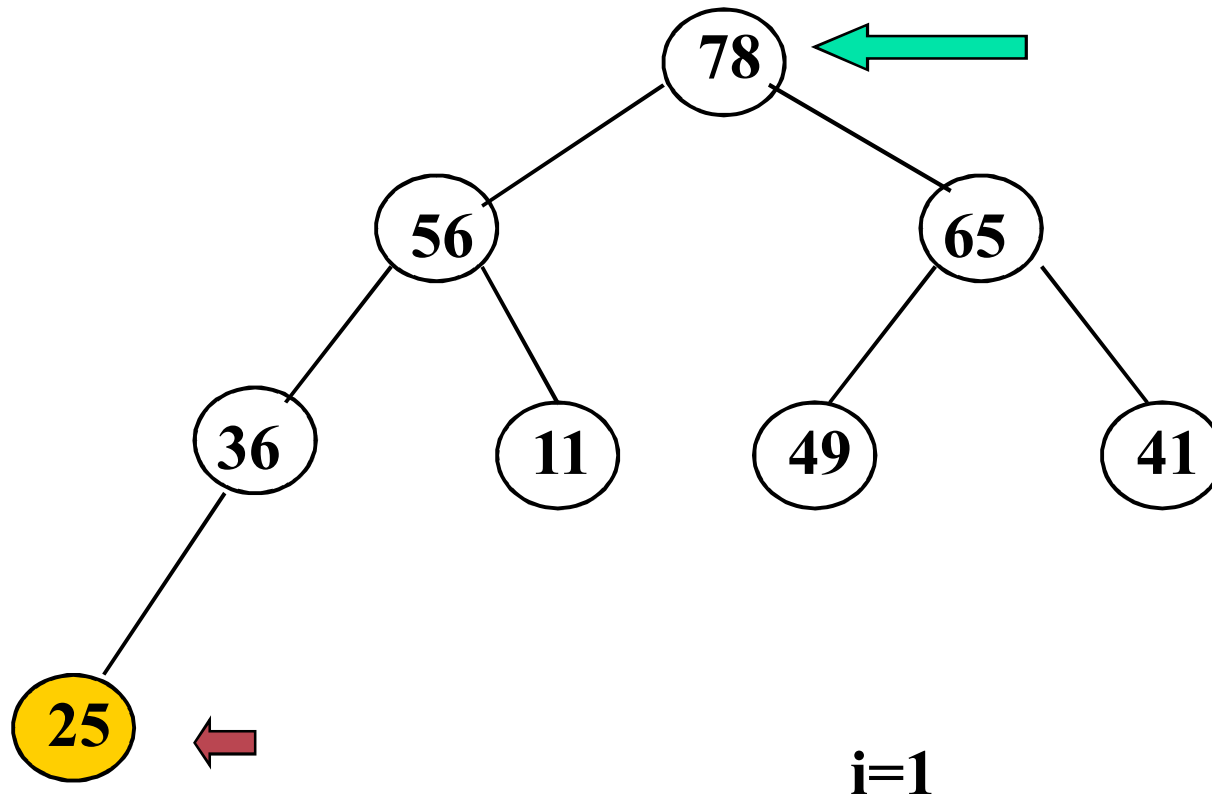
0	1	2	3	4	5	6	7	8
25	78	56	65	56	11	49	41	36

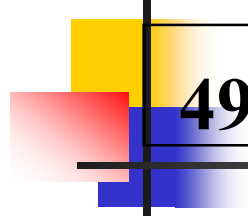


0	1	2	3	4	5	6	7	8
25	78	56	65	36	11	49	41	36

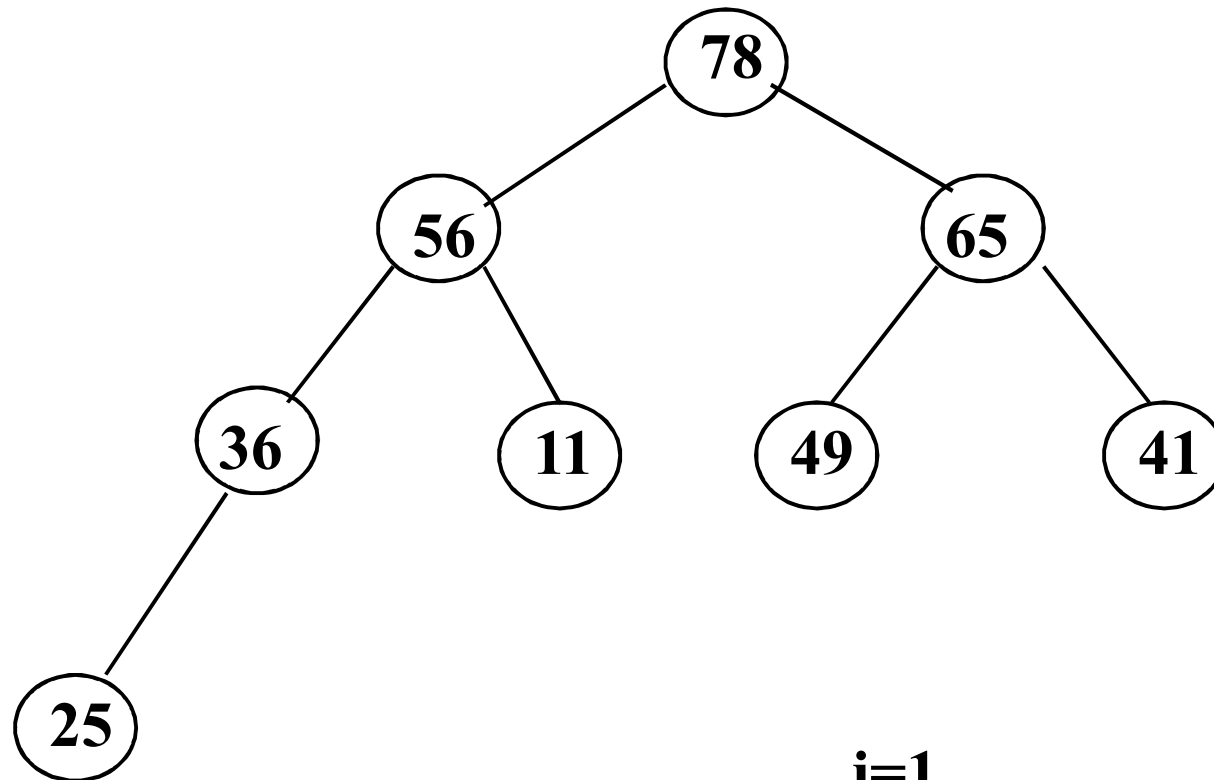


0	1	2	3	4	5	6	7	8
25	78	56	65	36	11	49	41	25





0	1	2	3	4	5	6	7	8
49	78	56	65	36	11	49	41	25



i=1



```
void HeapAdjust(SqList &L,int s,int m)
```

堆调整算法

```
{ int j;  
    L.r[0]=L.r[s];  
    for (j=2*s,j<=m;j=j*2 )  
    { if(j<m&&L.r[j].key< L.r[j+1].key) ++j;  
      if(L.r[0].key>=L.r[j].key) break;  
      L.r[s]=L.r[j];  
      s=j;  
    }  
    L.r[s]=L.r[0];  
}
```


void HeapSort(SqList L)//堆排序算法

{ int i,j,k;

for (i=L.Length/2;i>0;--i)

HeapAdjust(L,i,L.Length);

for(i=L.Length;i>1;--i)

{ L.r[0]=L.r[i];

L.r[i]=L.r[1];

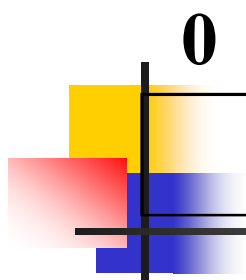
L.r[1]=L.r[0];

HeapAdjust(L,1,i-1);

}

}


HeapAdjust(L,i,L.Length);



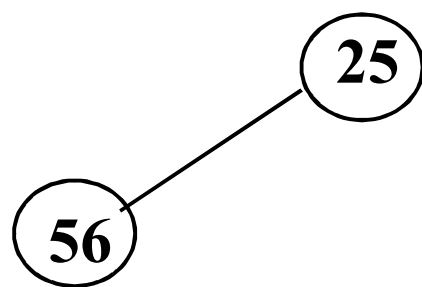
0	1	2	3	4	5	6	7	8
	25	56	49	78	11	65	41	36

25


插入法建堆



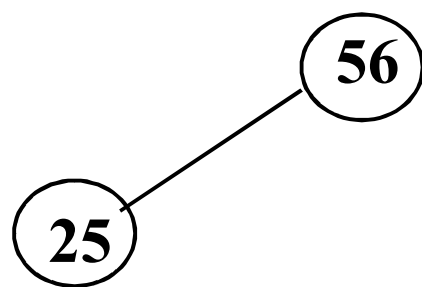
0	1	2	3	4	5	6	7	8
	25	56	49	78	11	65	41	36




插入法建堆



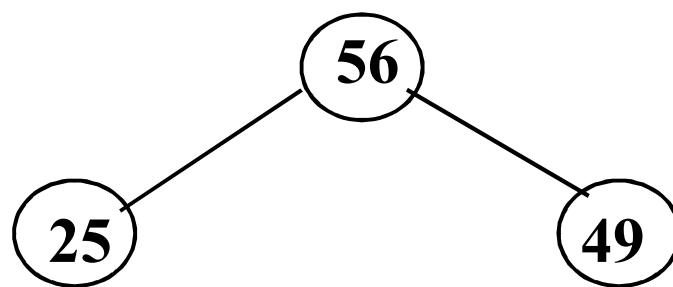
0	1	2	3	4	5	6	7	8
	56	25	49	78	11	65	41	36




插入法建堆



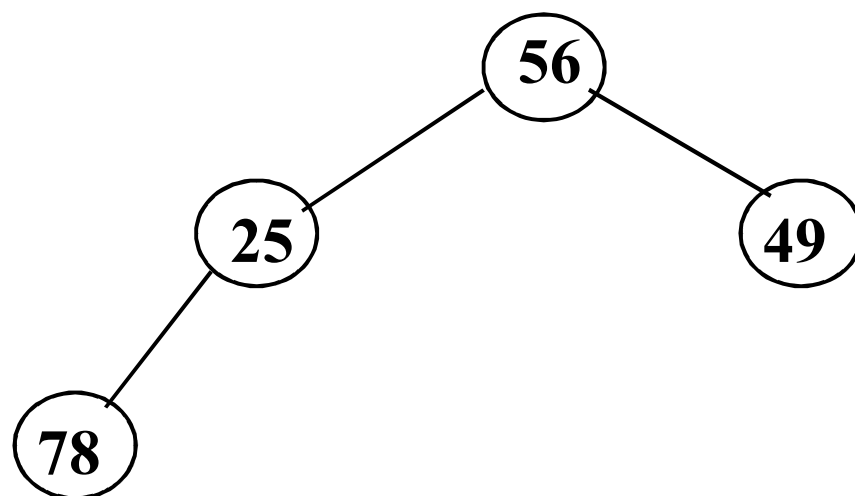
0	1	2	3	4	5	6	7	8
	56	25	49	78	11	65	41	36




插入法建堆



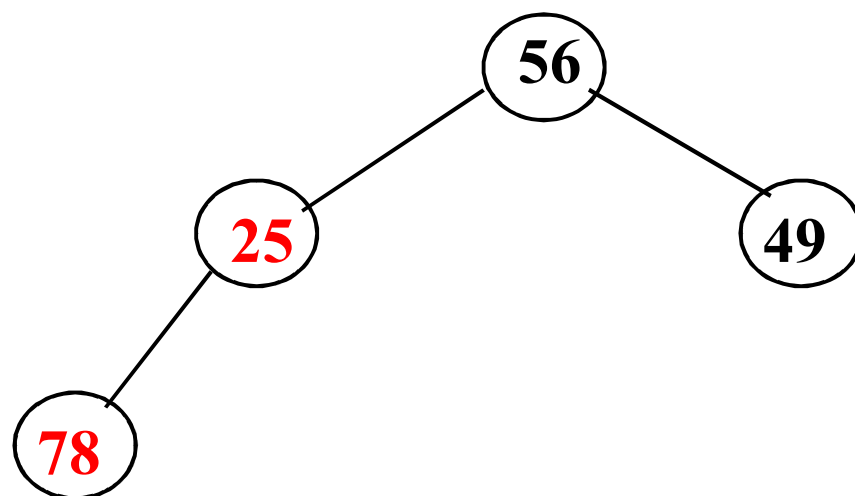
0	1	2	3	4	5	6	7	8
	56	25	49	78	11	65	41	36




插入法建堆



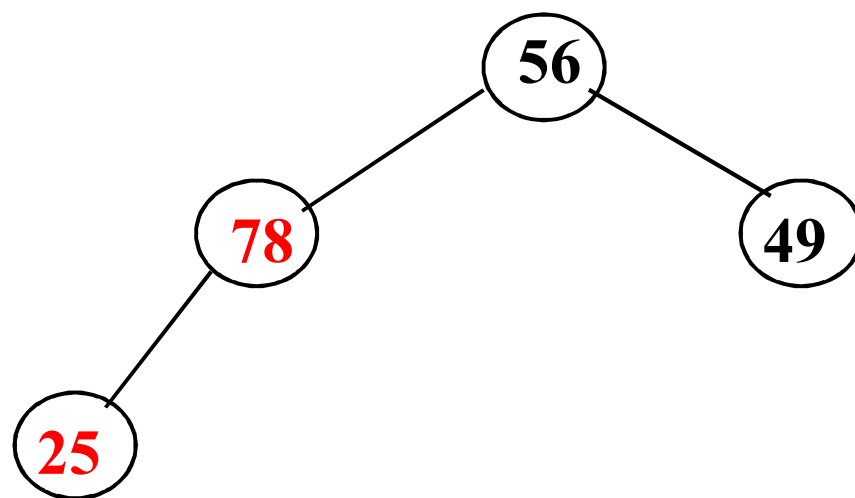
0	1	2	3	4	5	6	7	8
	56	25	49	78	11	65	41	36




插入法建堆



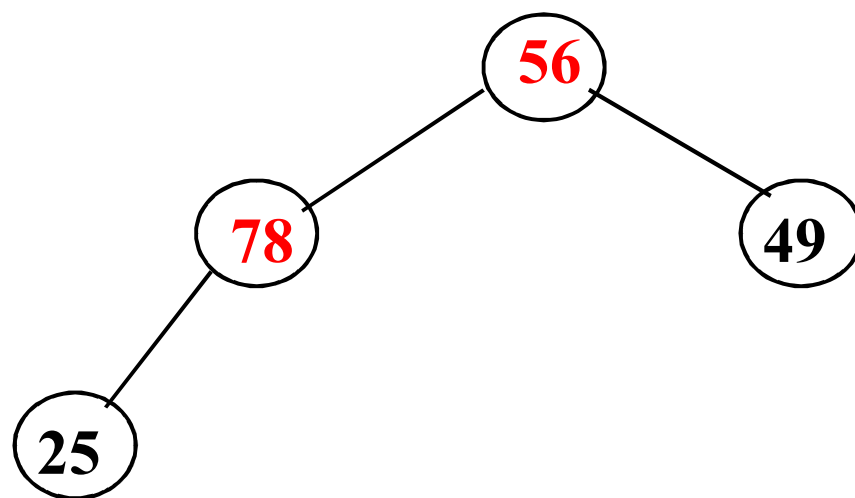
0	1	2	3	4	5	6	7	8
	56	78	49	25	11	65	41	36




插入法建堆



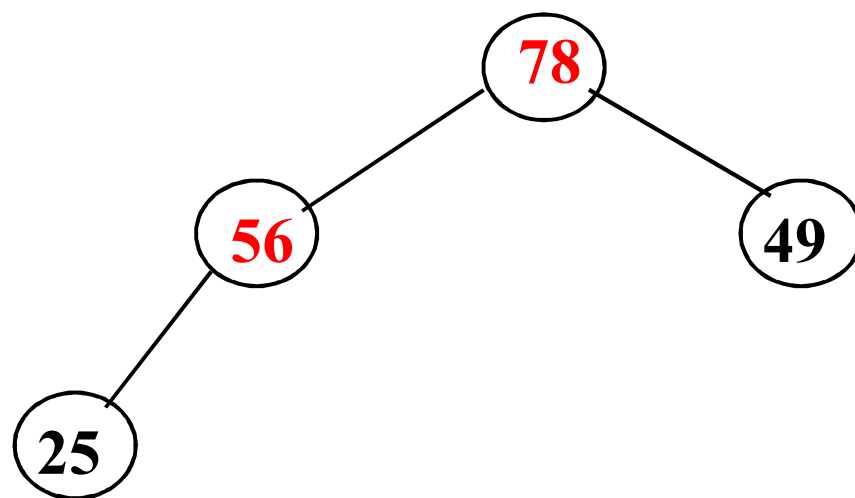
0	1	2	3	4	5	6	7	8
	56	78	49	25	11	65	41	36



插入法建堆

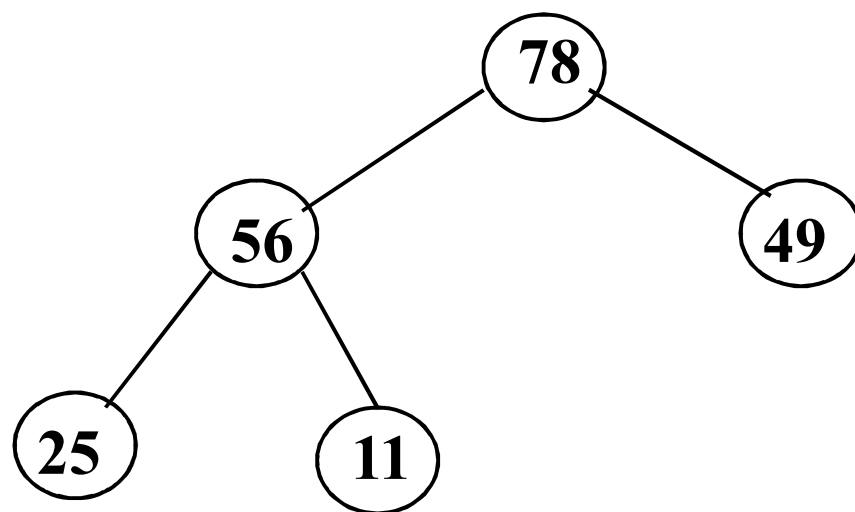


0	1	2	3	4	5	6	7	8
	78	56	49	25	11	65	41	36




插入法建堆

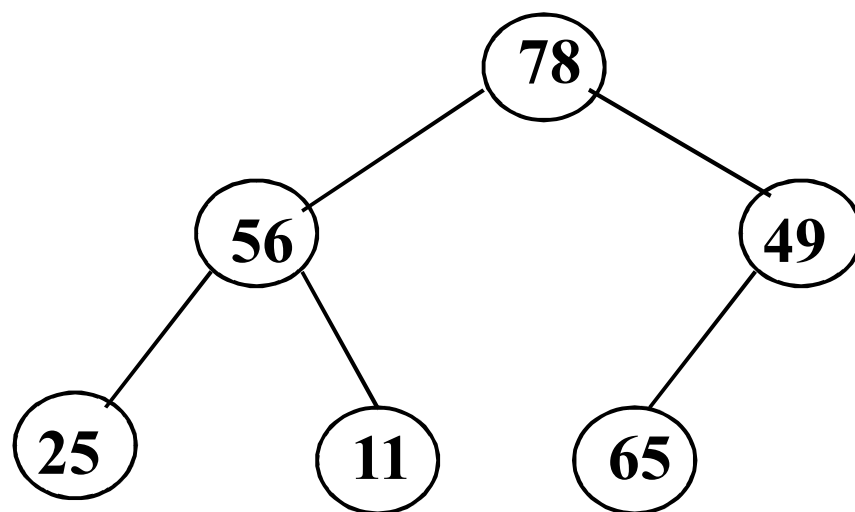
0	1	2	3	4	5	6	7	8
	78	56	49	25	11	65	41	36




插入法建堆



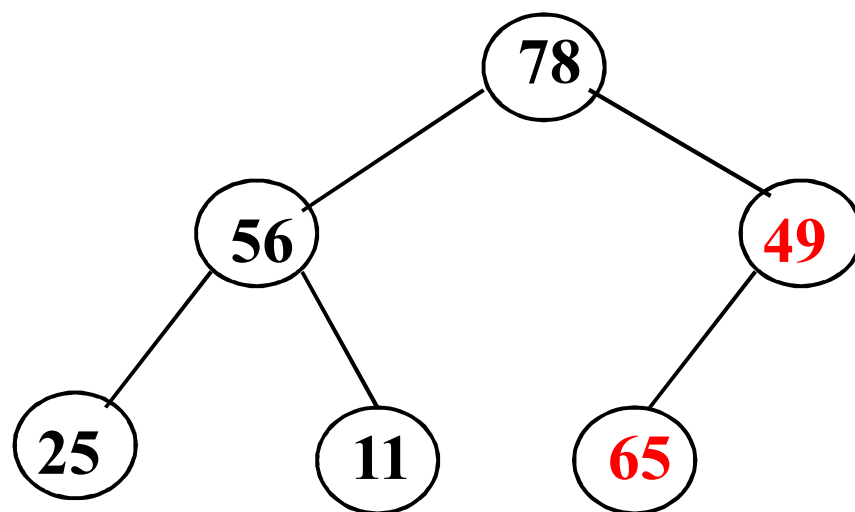
0	1	2	3	4	5	6	7	8
	78	56	49	25	11	65	41	36




插入法建堆



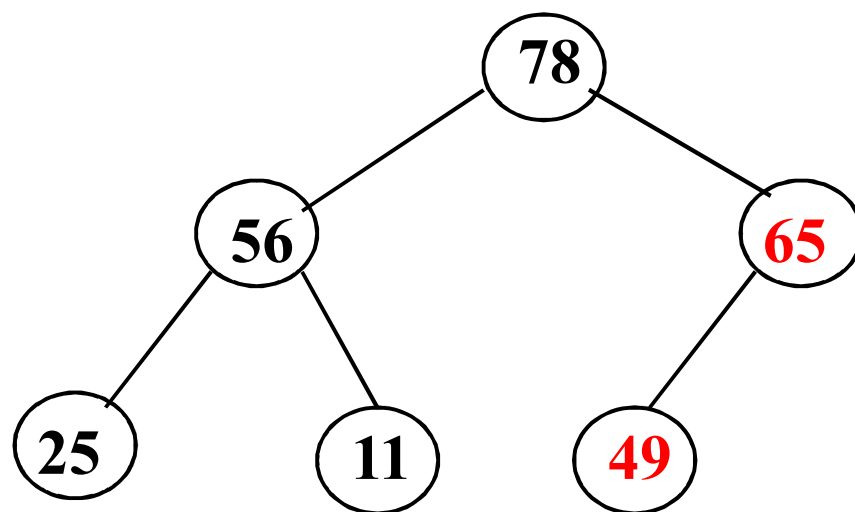
0	1	2	3	4	5	6	7	8
	78	56	49	25	11	65	41	36




插入法建堆



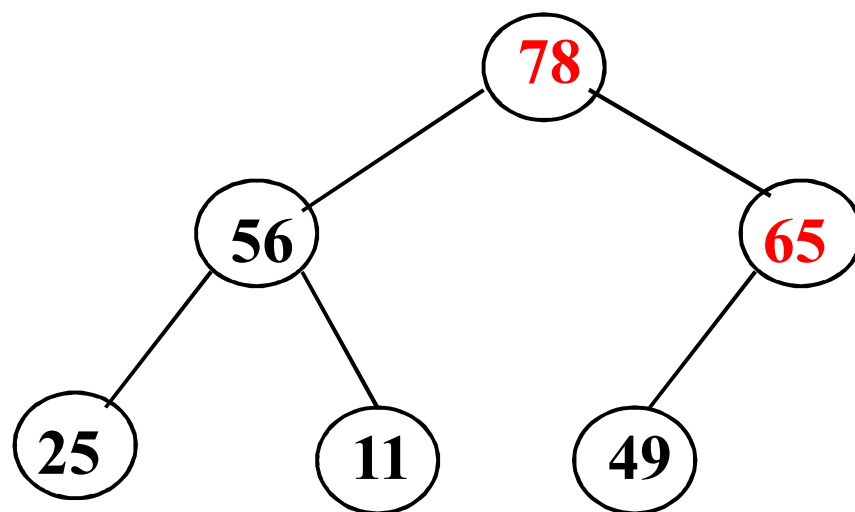
0	1	2	3	4	5	6	7	8
	78	56	65	25	11	49	41	36




插入法建堆



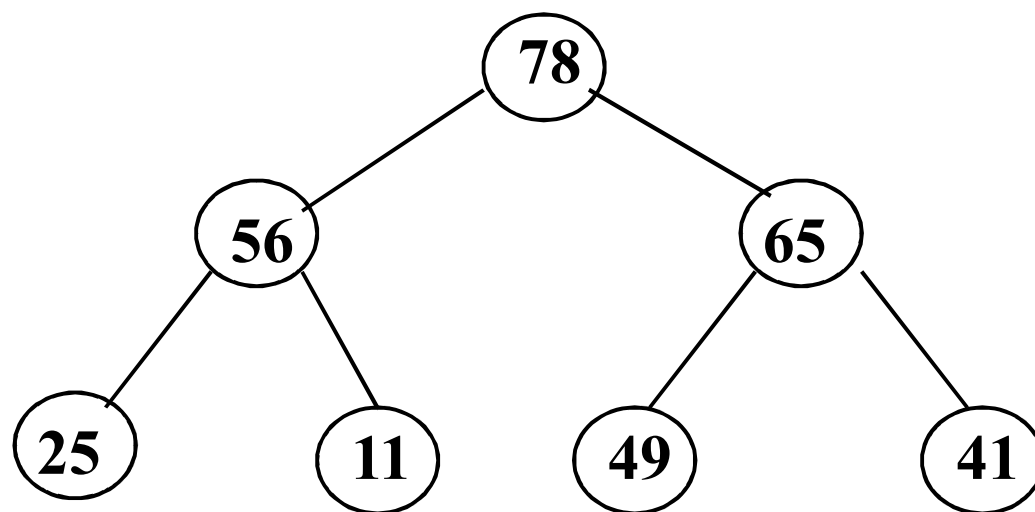
0	1	2	3	4	5	6	7	8
	78	56	65	25	11	49	41	36



插入法建堆

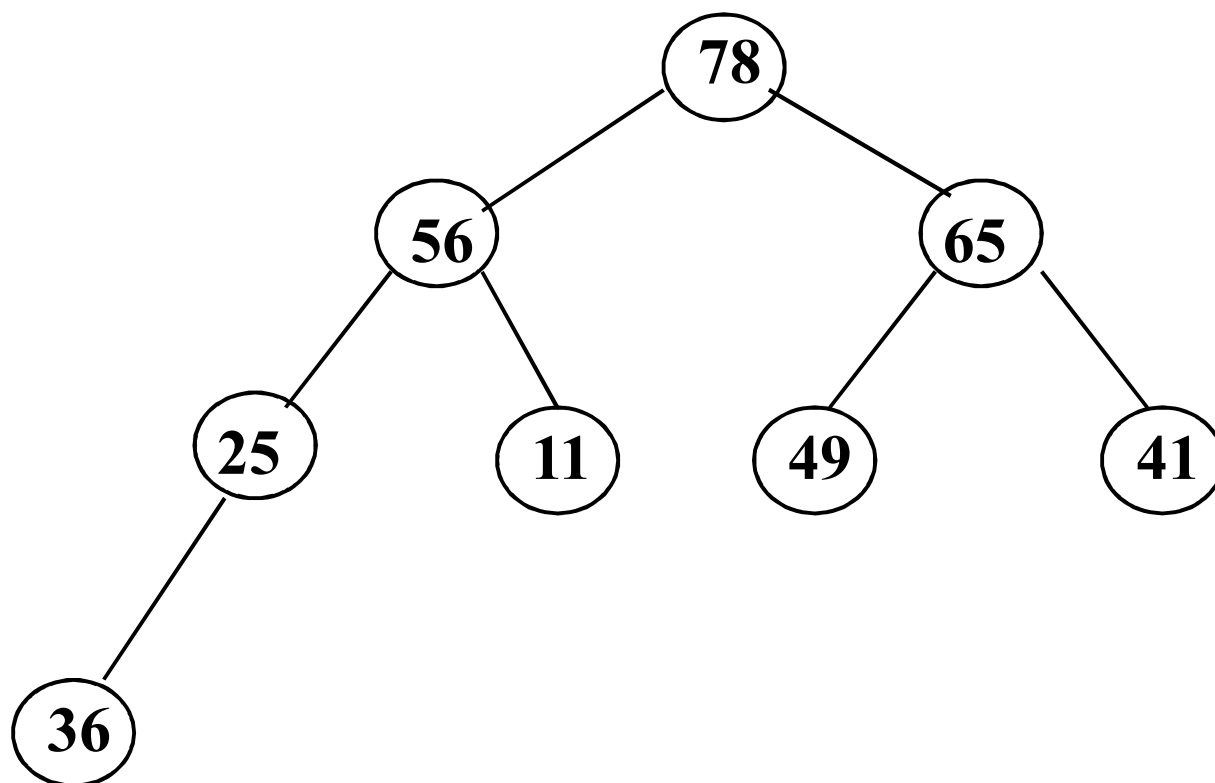


0	1	2	3	4	5	6	7	8
	78	56	65	25	11	49	41	36



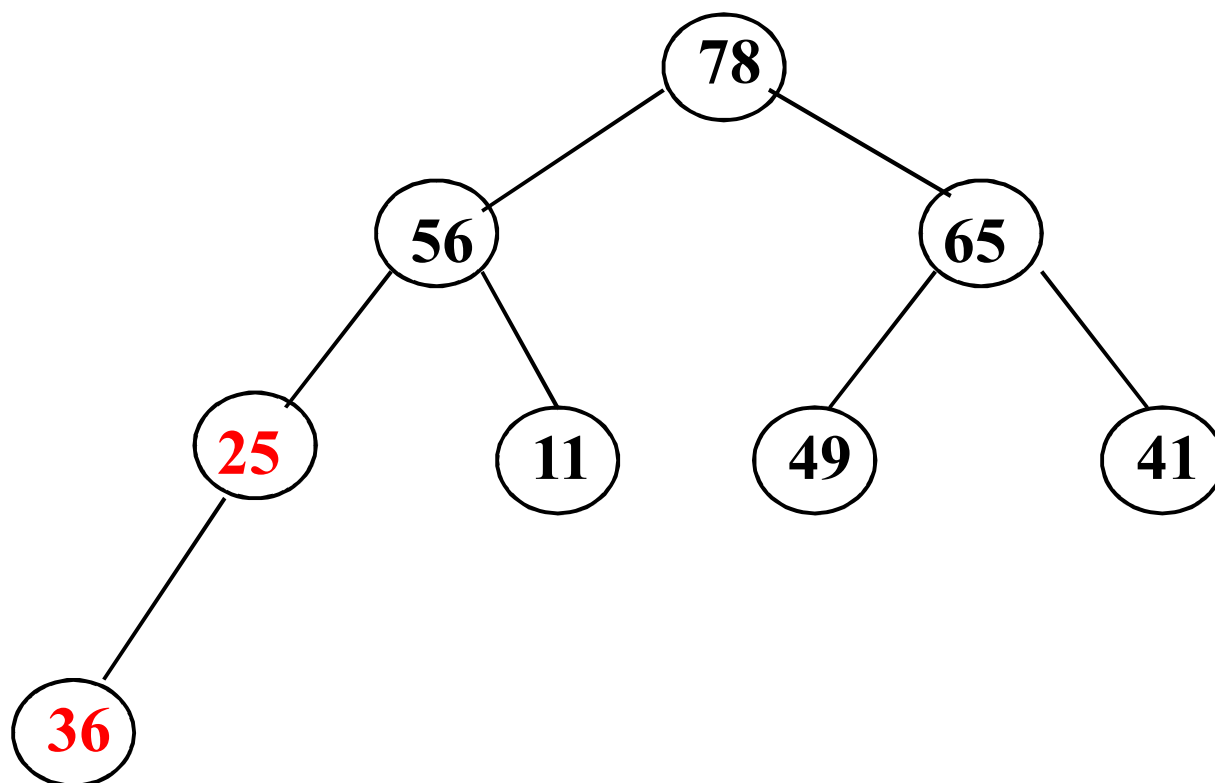
插入法建堆

0	1	2	3	4	5	6	7	8
	78	56	65	25	11	49	41	36



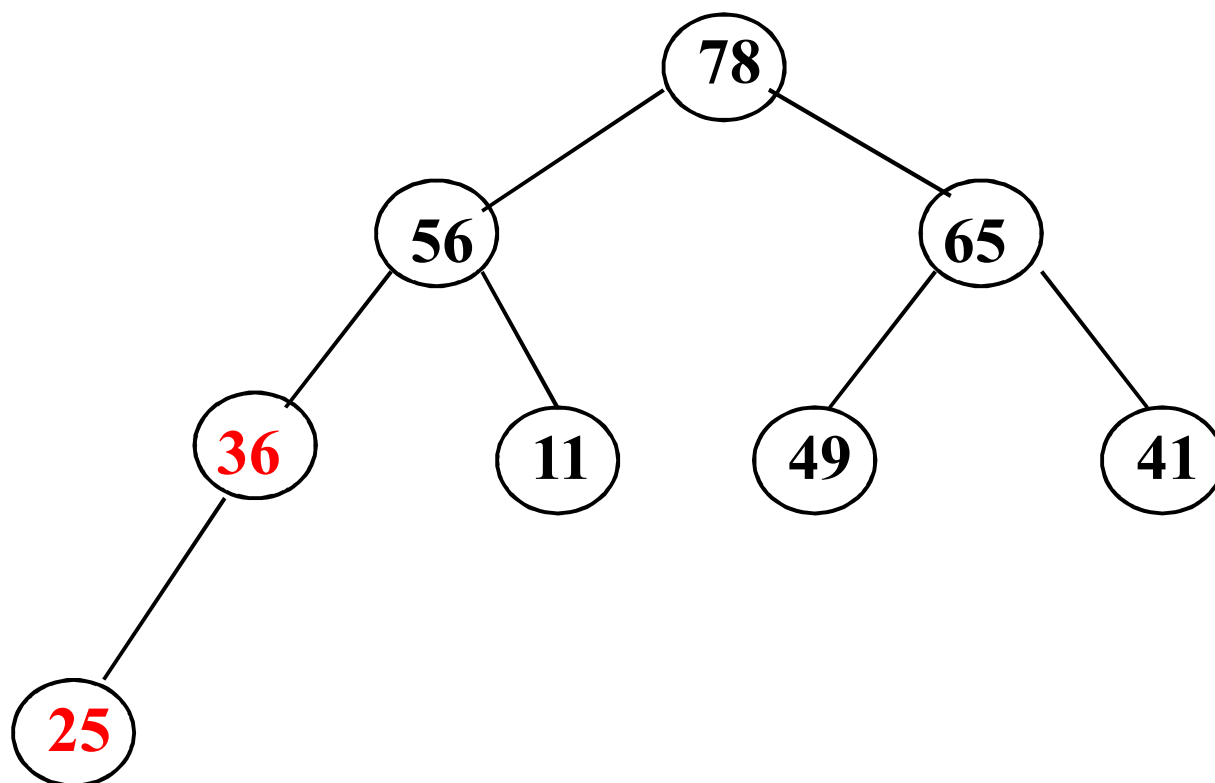
插入法建堆

0	1	2	3	4	5	6	7	8
	78	56	65	25	11	49	41	36



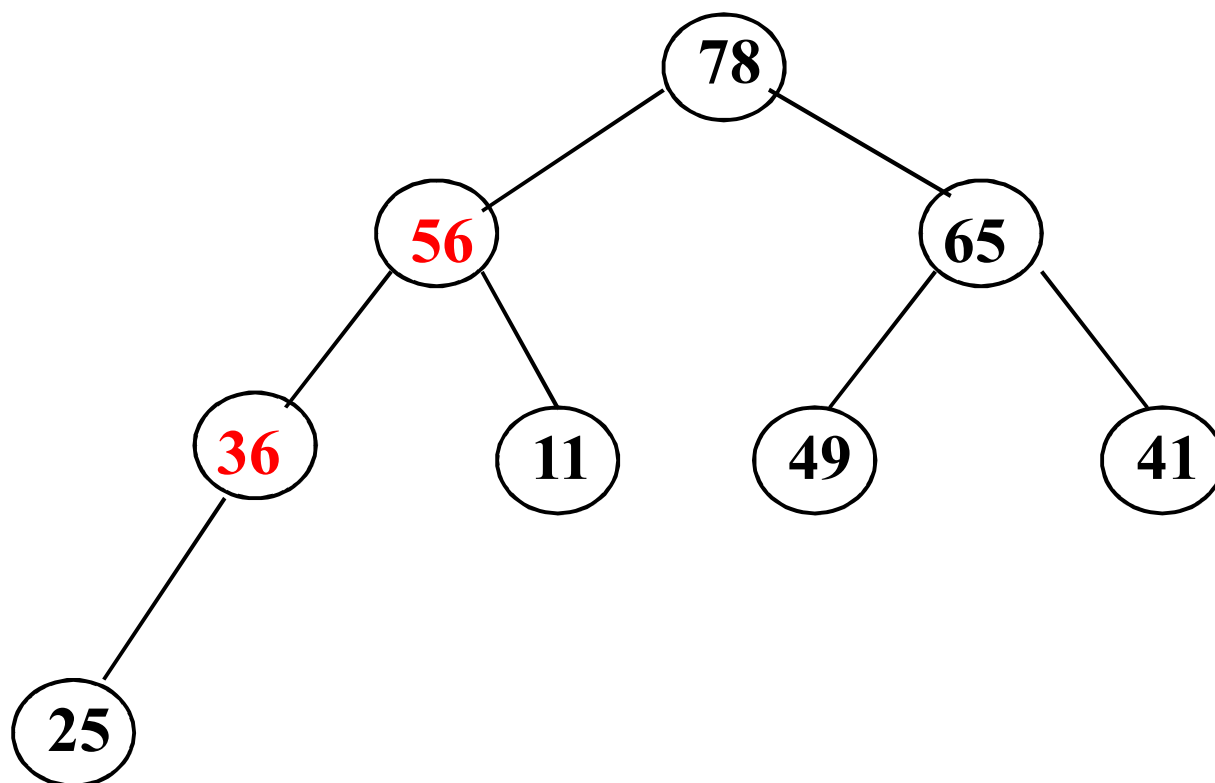
插入法建堆

0	1	2	3	4	5	6	7	8
	78	56	65	36	11	49	41	25



插入法建堆

0	1	2	3	4	5	6	7	8
	78	56	65	36	11	49	41	25



插入法建堆