数据结构

（C语言版）（第2版）

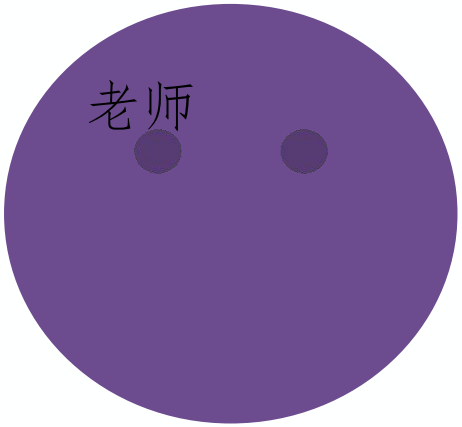
排序

选择排序

主讲教师：汪红松

教 学 内 容 Contents

一、简单选择排序
二、树形选择排序
三、堆排序

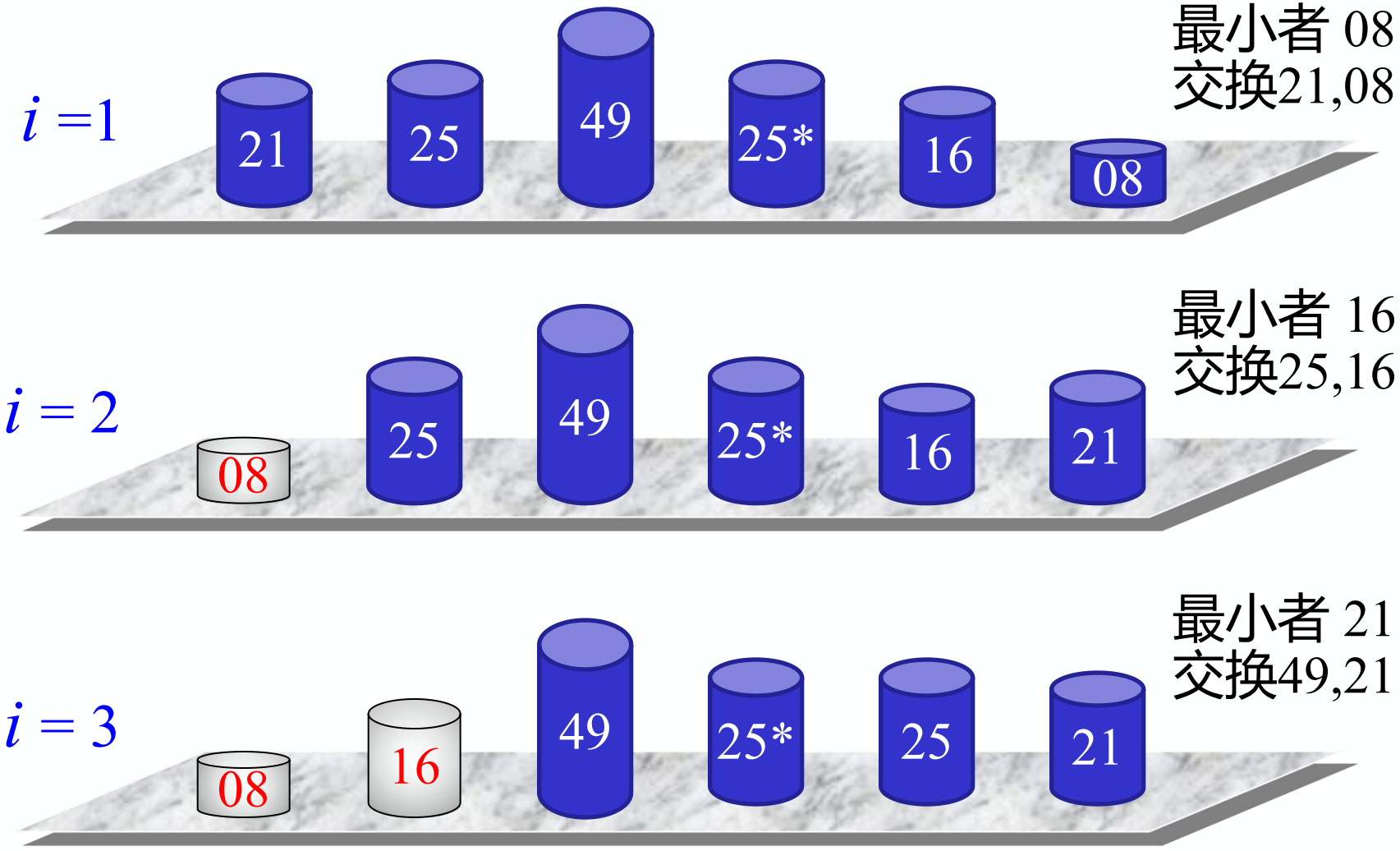


选择排序

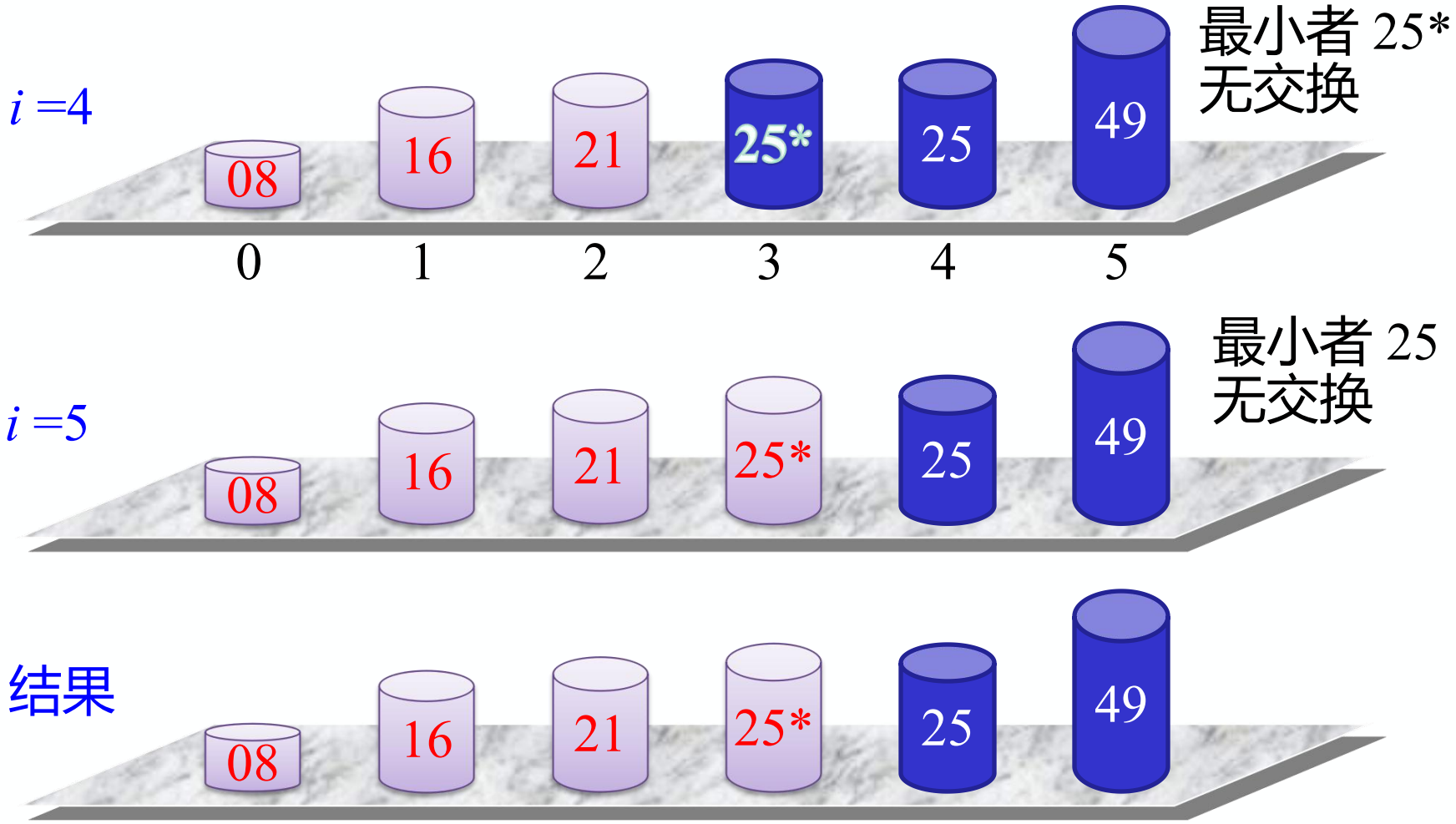
基本思想：

每一趟在后面 n-i +1个中选出关键码最小的对象, 作为有序序列的第 i 个记录。

一、简单选择排序


i =1
21
25
49
25*
16
08
最小者 08
交换21,08
i = 2
08
25
49
25*
16
21
最小者 16
交换25,16
i = 3
08
16
49
25*
25
21
最小者 21
交换49,21

一、简单选择排序



各趟排序后的结果

一、简单选择排序

```
void SelectSort(SqList &K)
 {
   for (i=1; i<L.length; ++i)
   { //在L.r[i..L.length] 中选择key最小的记录
      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[i]←→L.r[k];
   }
}
```

移动次数

最好情况：0

最坏情况：3(n-1)

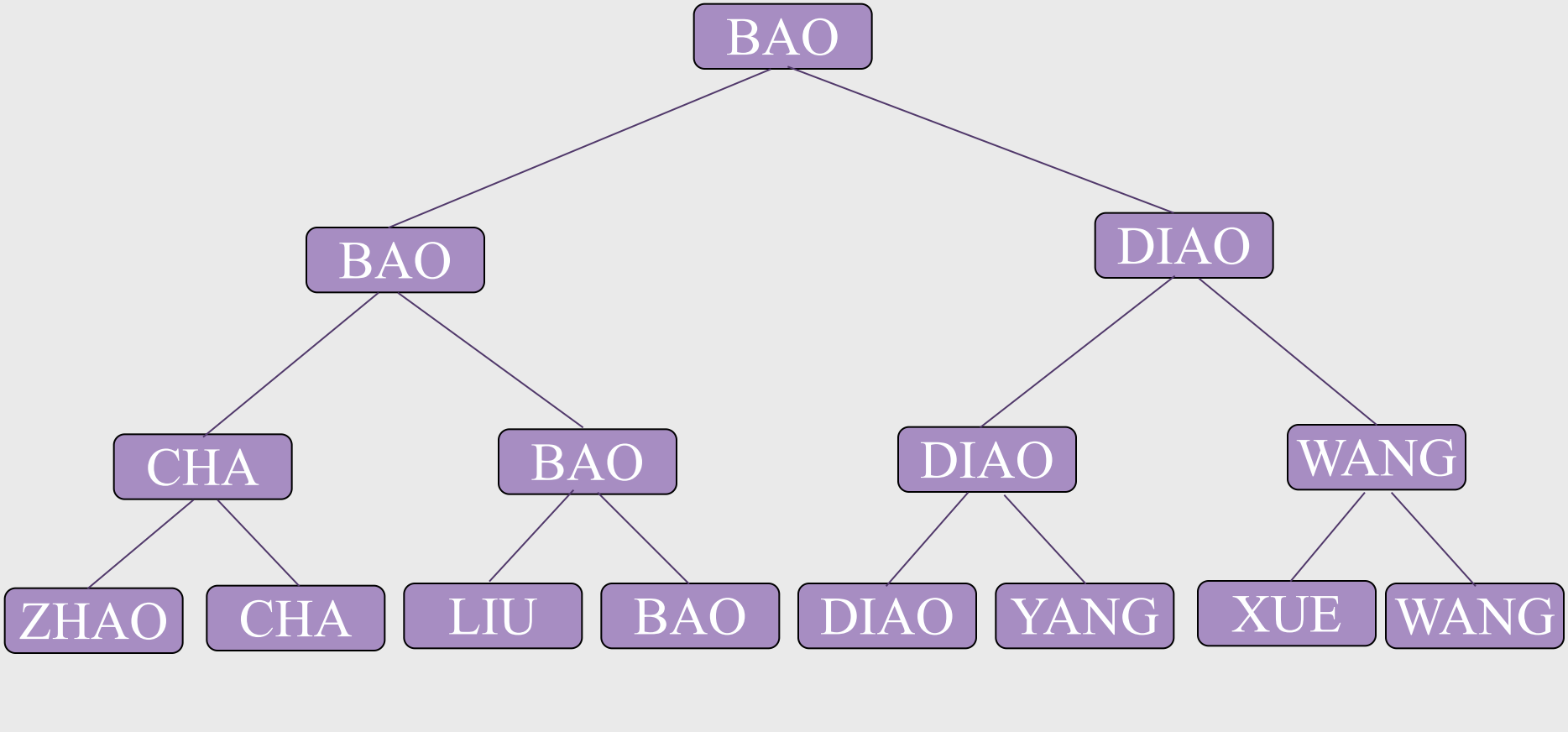
比较次数：$\sum_{i=1}^{n-1}(n-i)=\frac{1}{2}(n^2-n)$

时间复杂度：$O(n^2)$

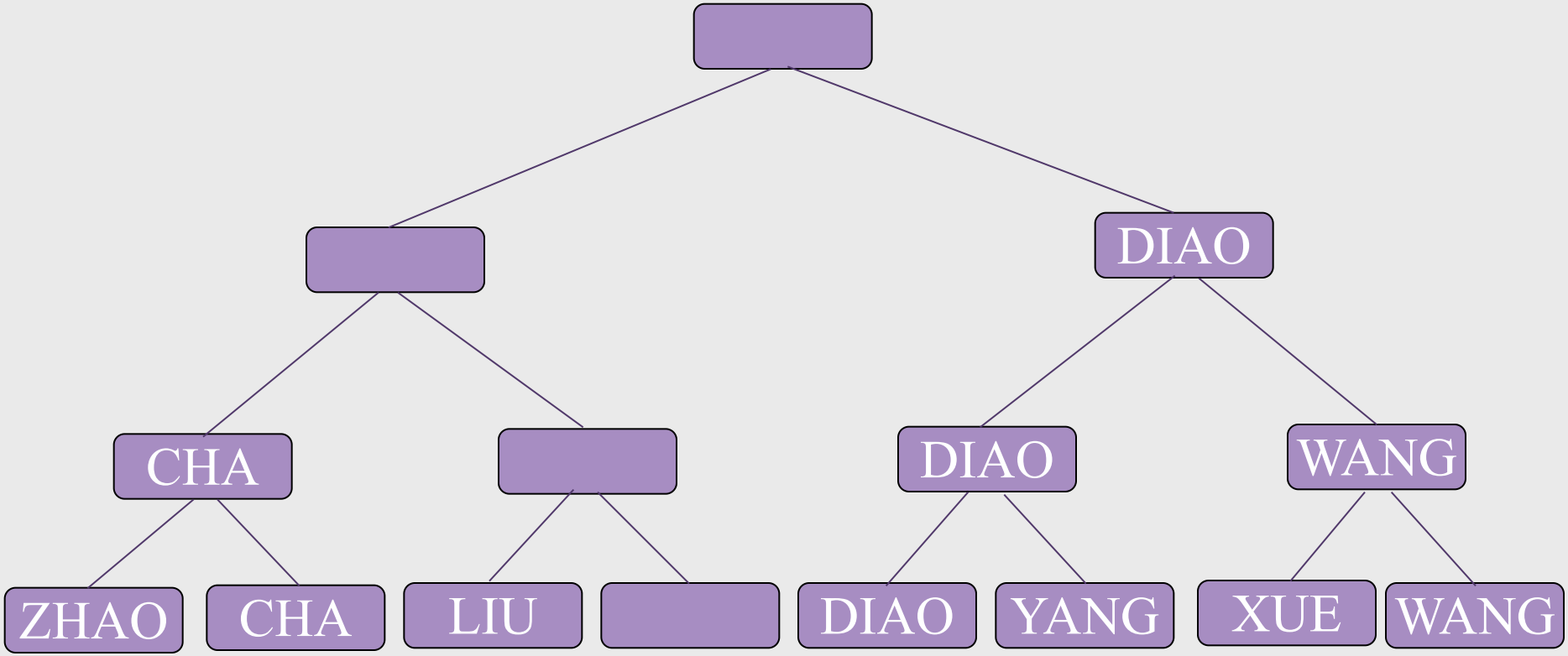
空间复杂度：$O(1)$

稳定

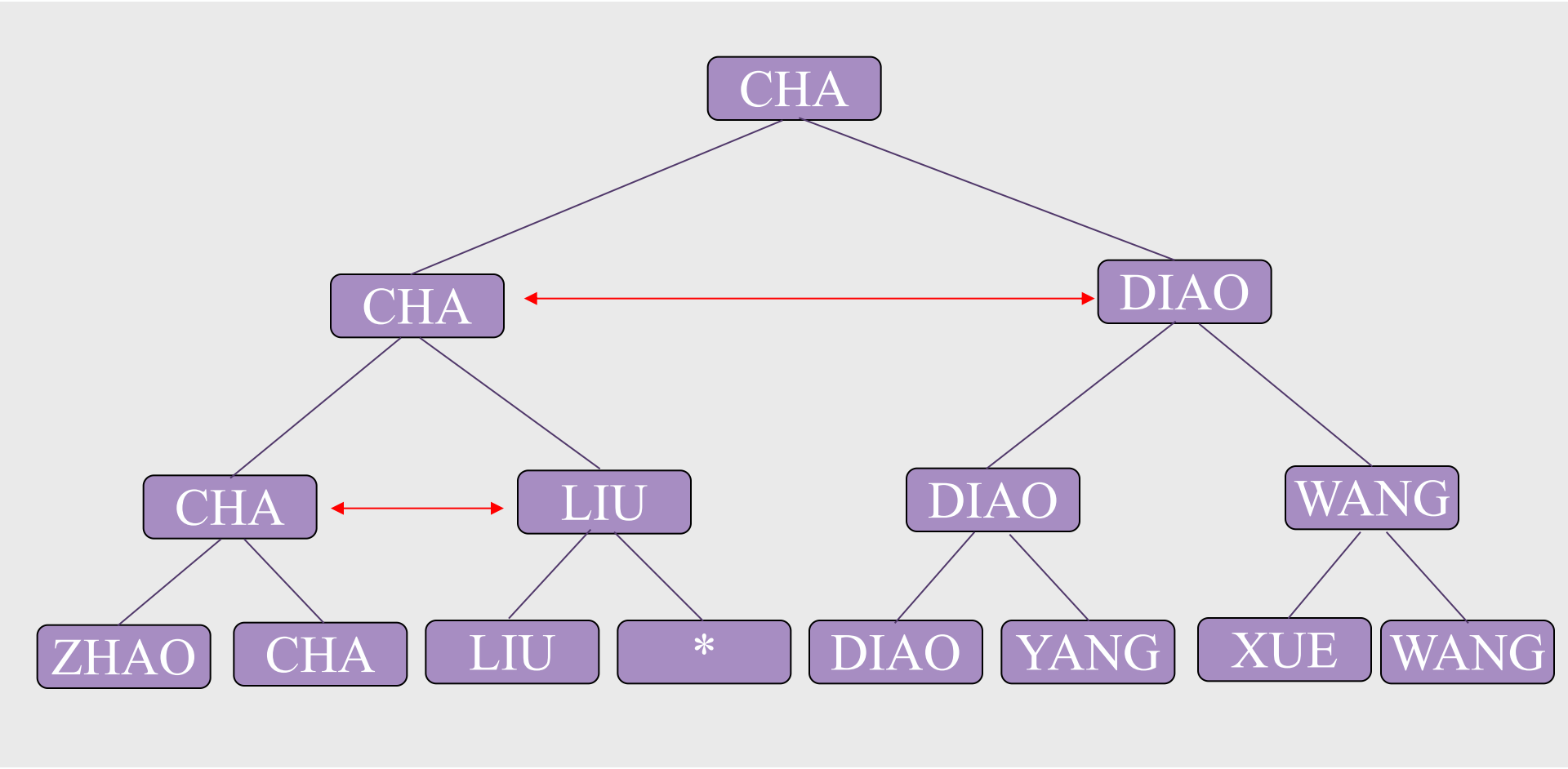
二、树形选择排序


BAO
BAO
DIAO
CHA
BAO
DIAO
WANG
ZHAO
CHA
LIU
BAO
DIAO
YANG
XUE
WANG

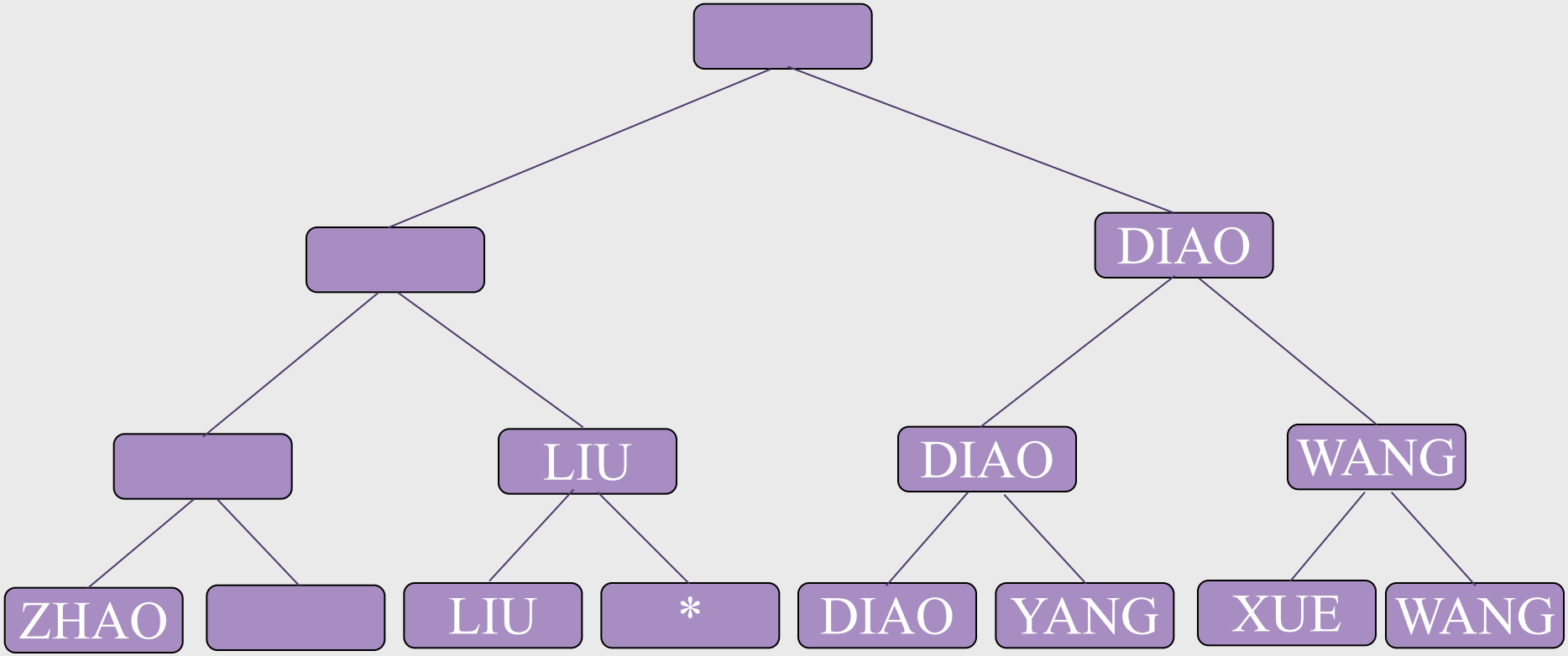
二、树形选择排序



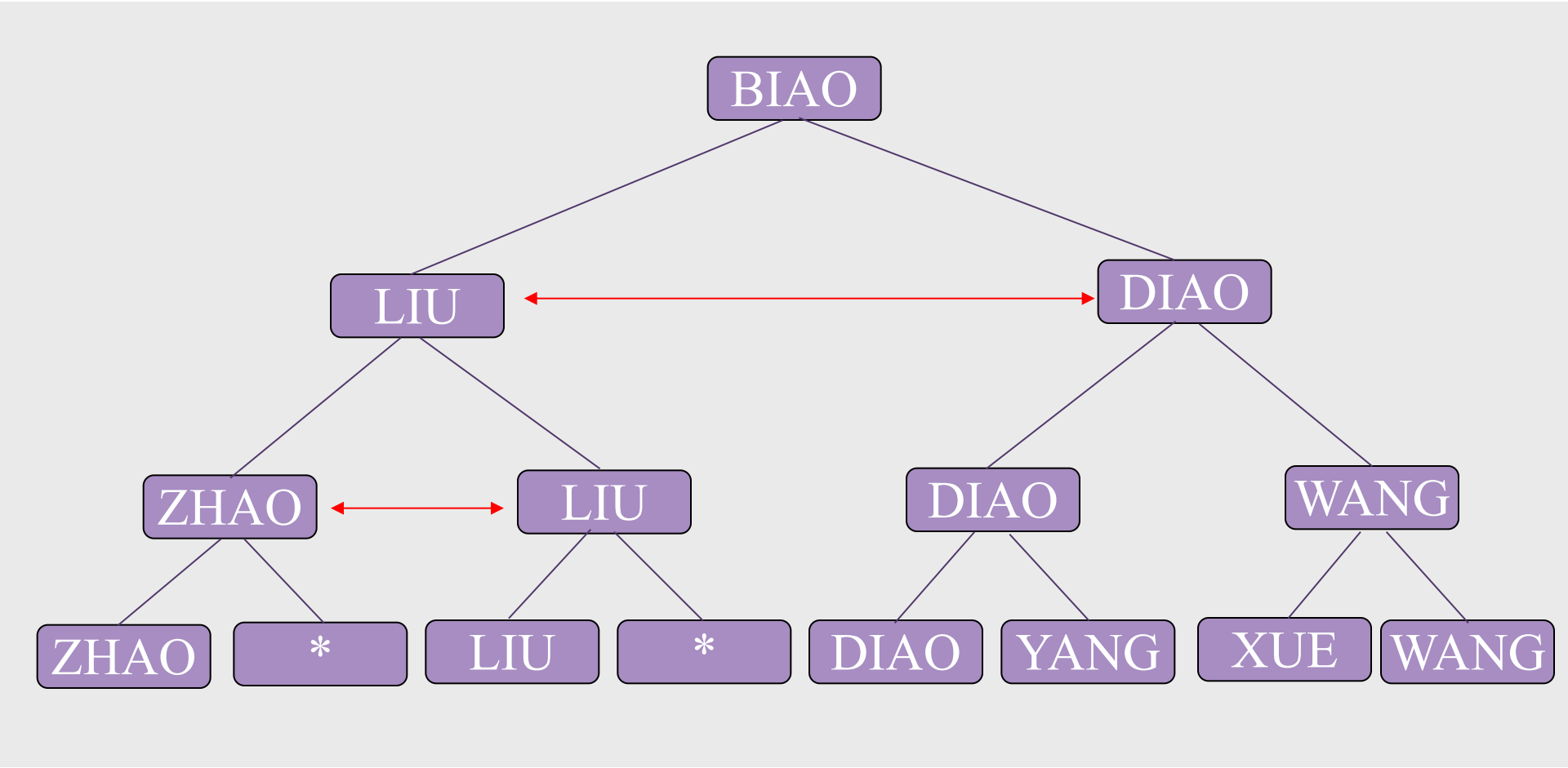
二、树形选择排序



二、树形选择排序

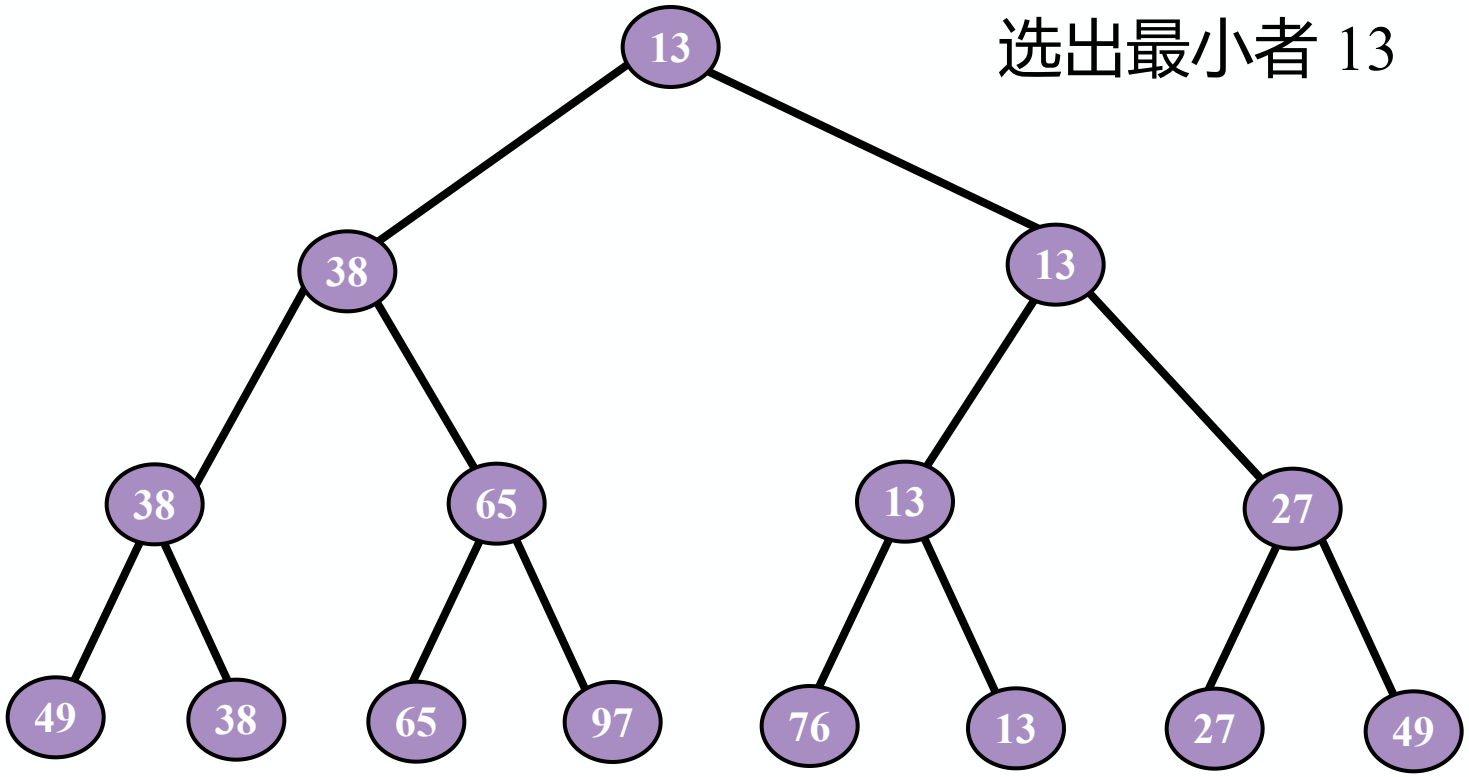


二、树形选择排序



二、树形选择排序

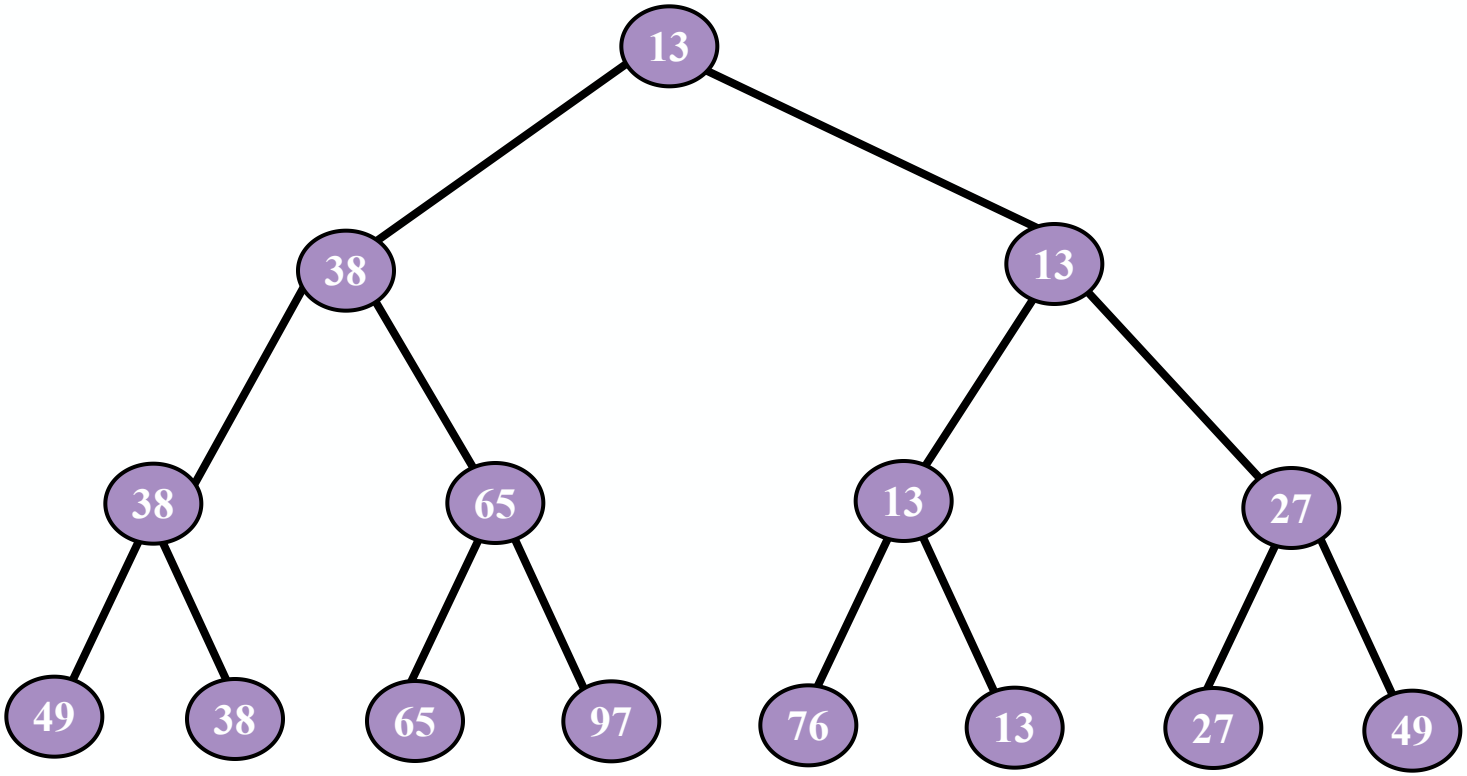
改进：简单选择排序没有利用上次选择的结果，是造成速度慢的重要原因。如果，能够加以改进，将会提高排序的速度。



二、树形选择排序

改进：简单选择排序没有利用上次选择的结果，是造成速度满的重要原因。如果，能够加以改进，将会提高排序的速度。

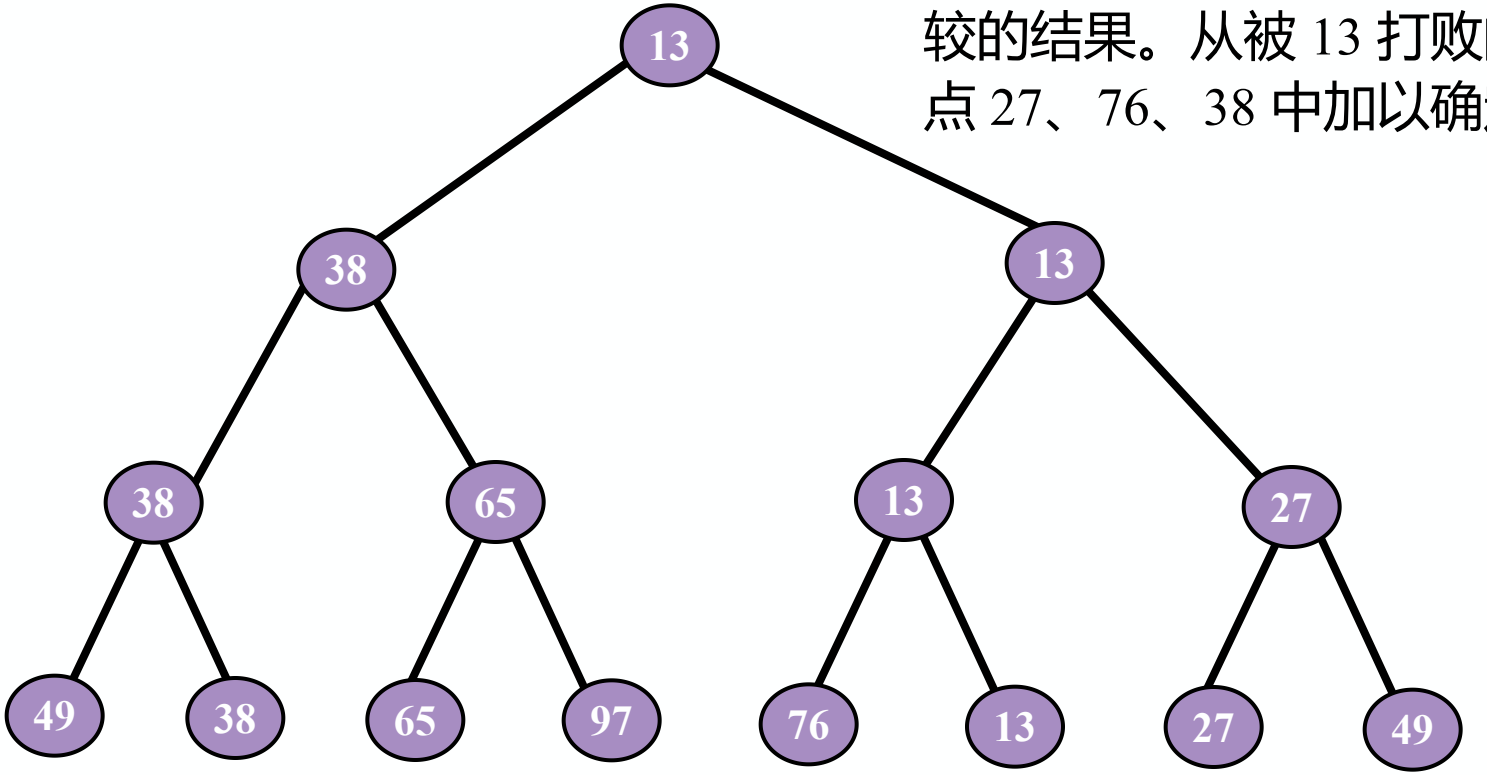
选出次最小者，应利用上次比较的结果。从被 13 打败的结点 27、76、38 中加以确定。



二、树形选择排序

改进：简单选择排序没有利用上次选择的结果，是造成速度满的重要原因。如果，能够加以改进，将会提高排序的速度。

选出次最小者，应利用上次比较的结果。从被 13 打败的结点 27、76、38 中加以确定。



三、堆排序

什么是堆？

n个元素的序列{k1,k2,…,kn}，当且仅当满足下列关系时，成为堆：

$$\begin{cases} k_i \le k_{2i} \\ k_i \le k_{2i+1} \end{cases} \text{或} \begin{cases} k_i \ge k_{2i} \\ k_i \ge k_{2i+1} \end{cases}$$

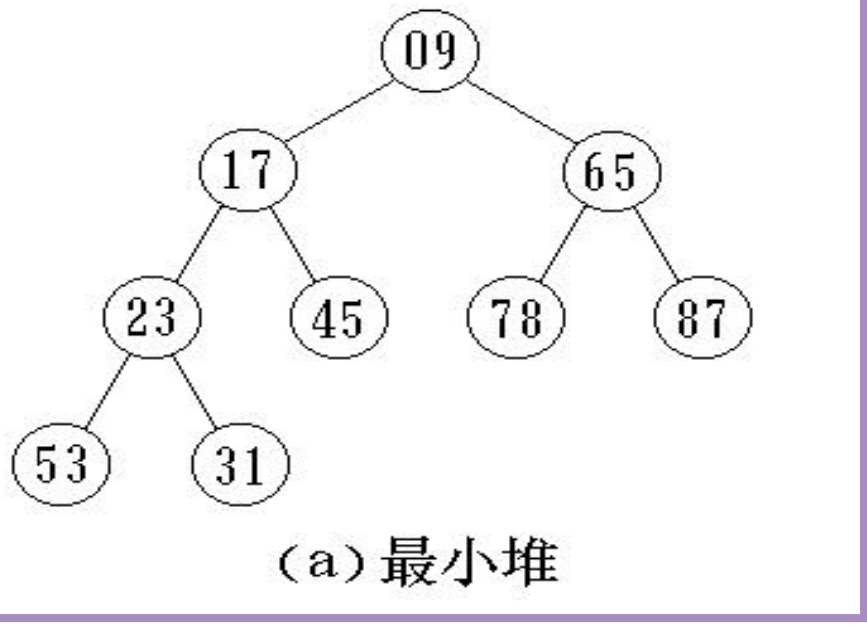
– 如果将序列看成一个完全二叉树，非终端结点的值均小于或大于左右子结点的值。

三、堆排序

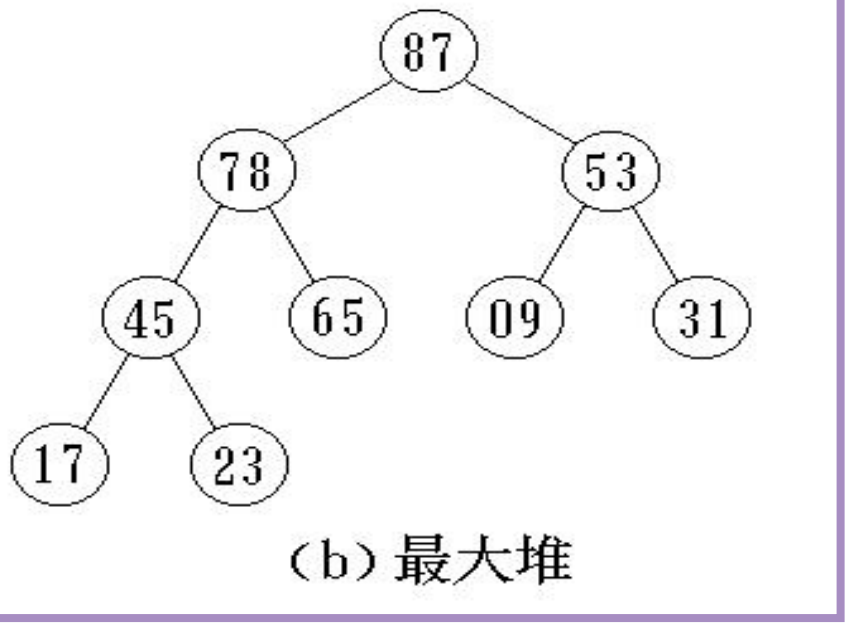
– 利用树的结构特征来描述堆，所以树只是作为堆的描述工具，堆实际是存放在线形空间中的。

(09,17,65,23,45,78,87,53,31)

(87,78,53,45,65,09,31,17,23)



(a) 最小堆

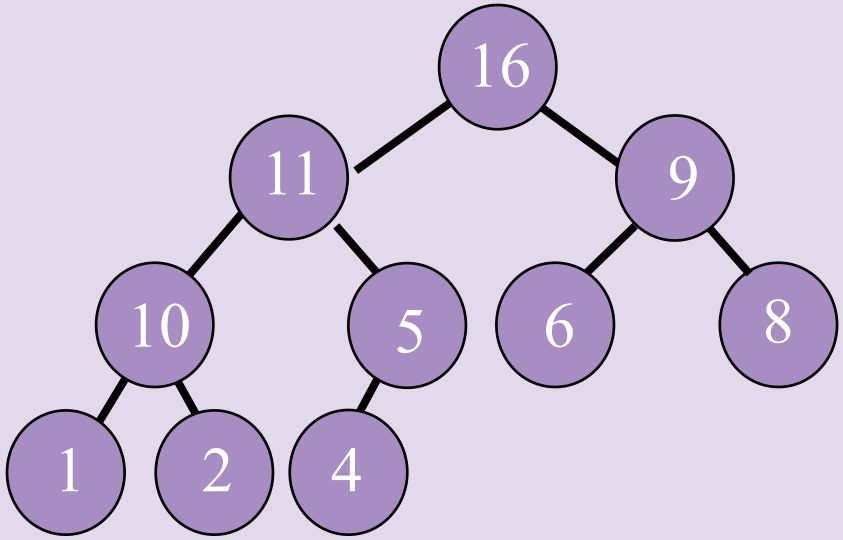


(b) 最大堆

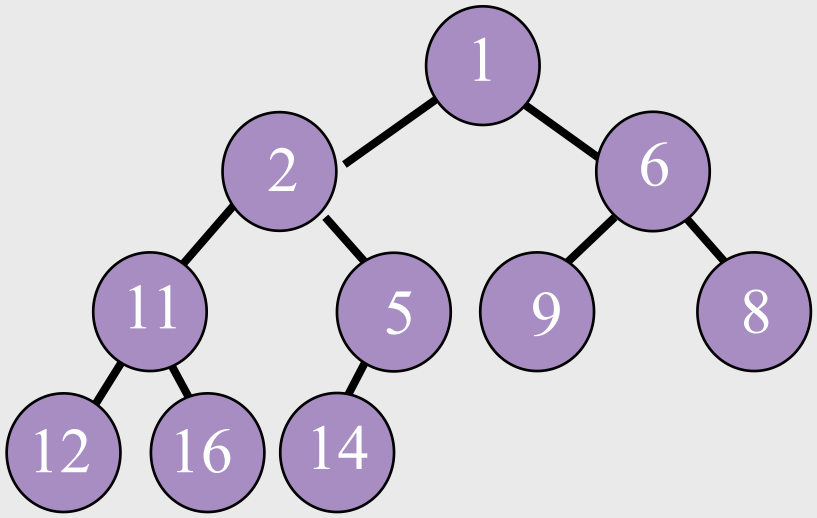
堆顶元素（根）为最小值或最大值

三、堆排序

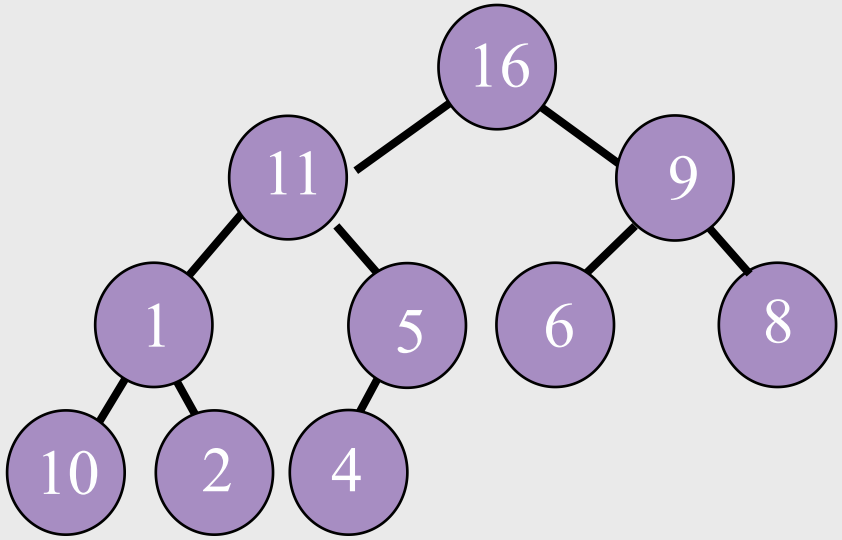
大根堆

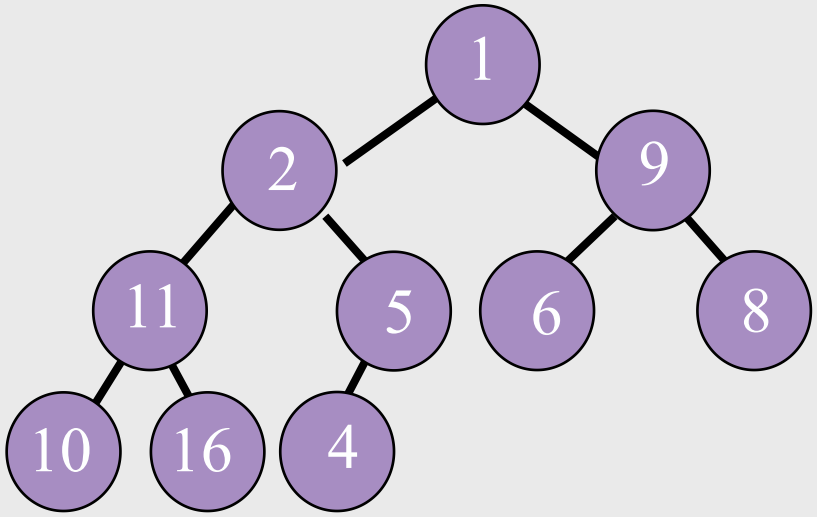


小根堆



三、堆排序

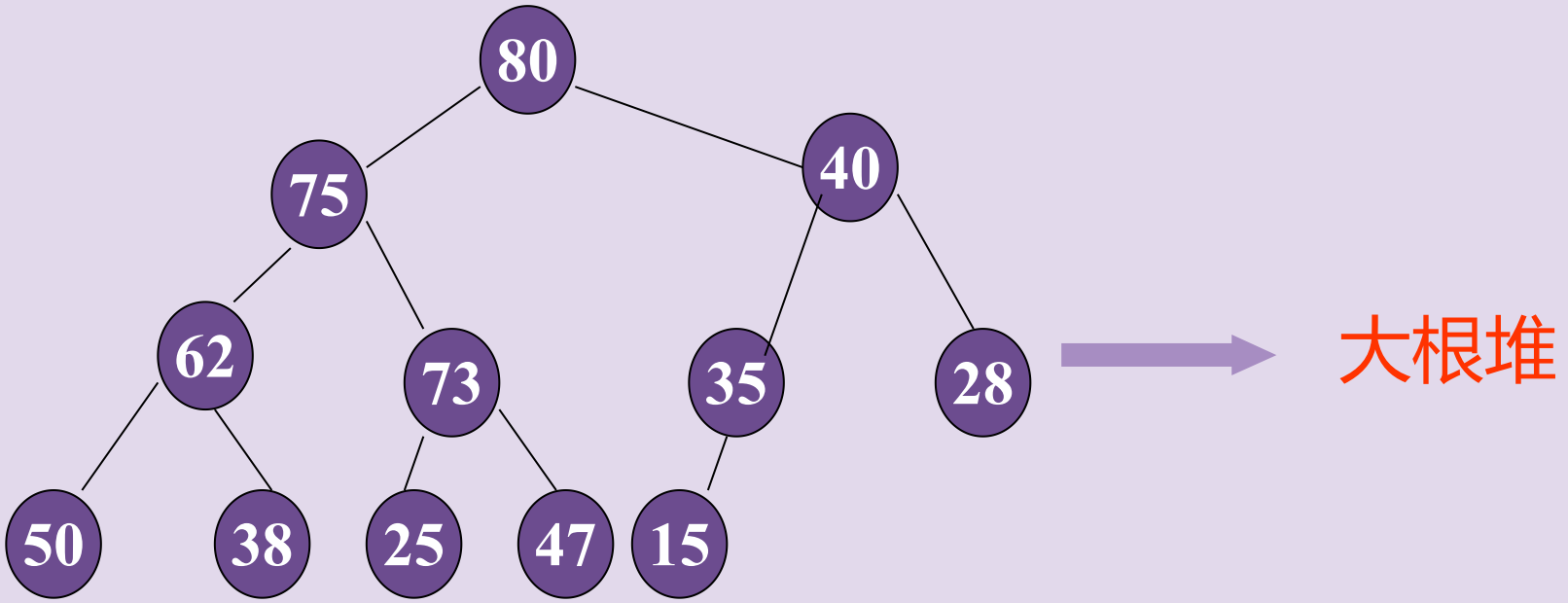




三、堆排序

判定(80,75,40,62,73,35,28,50,38,25,47,15)是否为堆

完全二叉树



三、堆排序

基本思想：

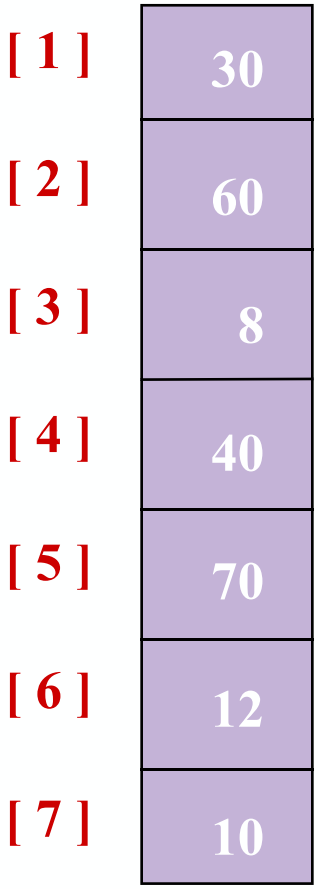
- ✓将无序序列建成一个堆
- ✓输出堆顶的最小（大）值
- ✓使剩余的n-1个元素又调整成一个堆，则可得到n个元素的次小值
- ✓重复执行，得到一个有序序列

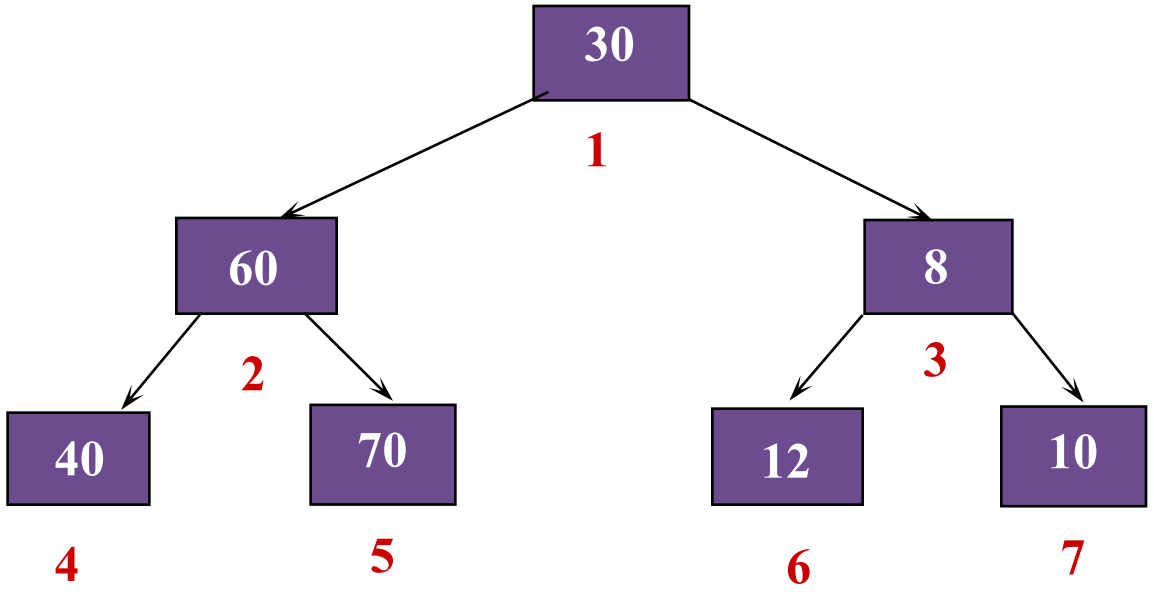
如何建？？

如何调整？？

三、堆排序

1.如何将无序序列建成堆



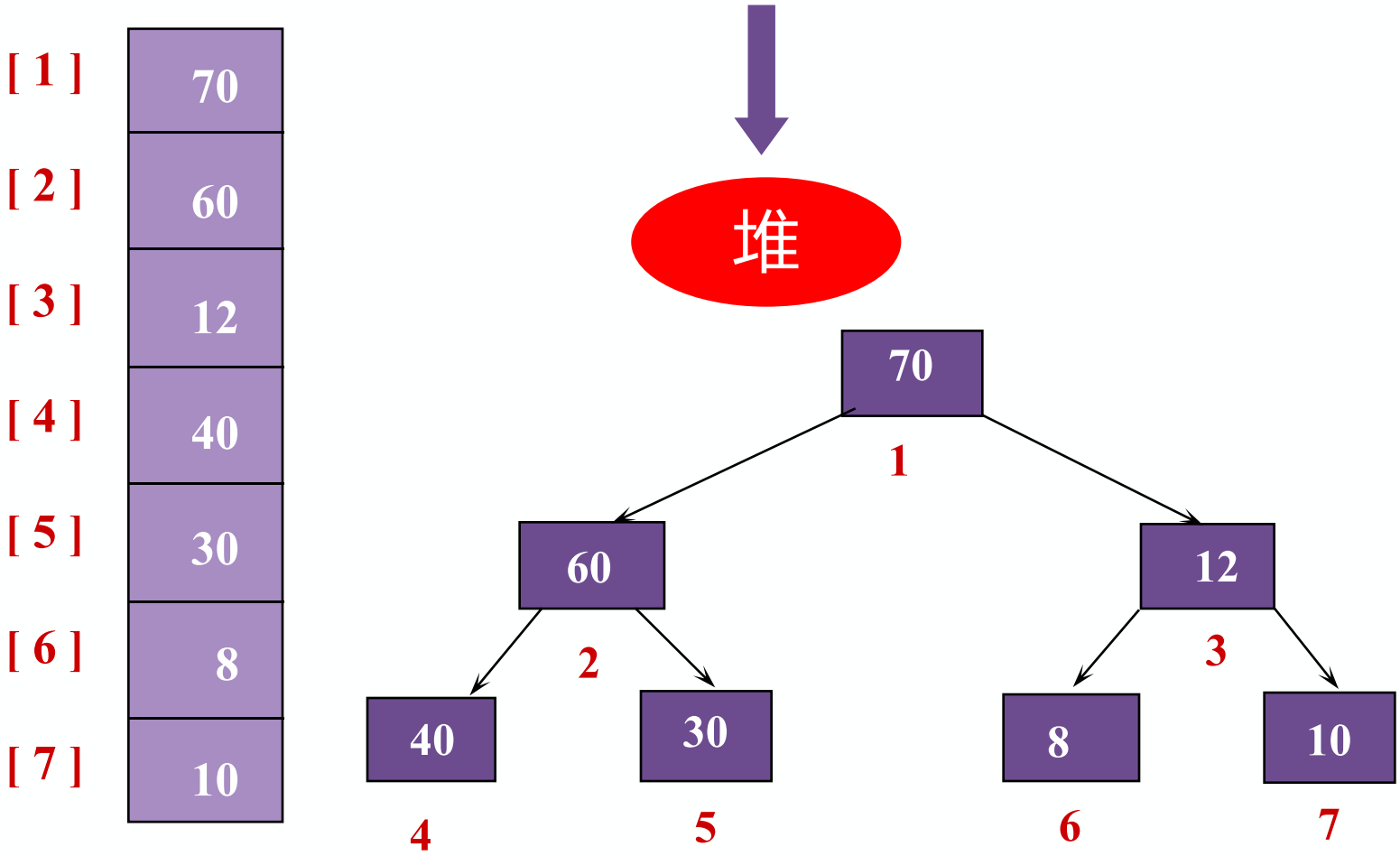


思考：有n 个结点的完全二叉树，
最后一个分支结点的标号是多少？ $\lfloor n/2 \rfloor$

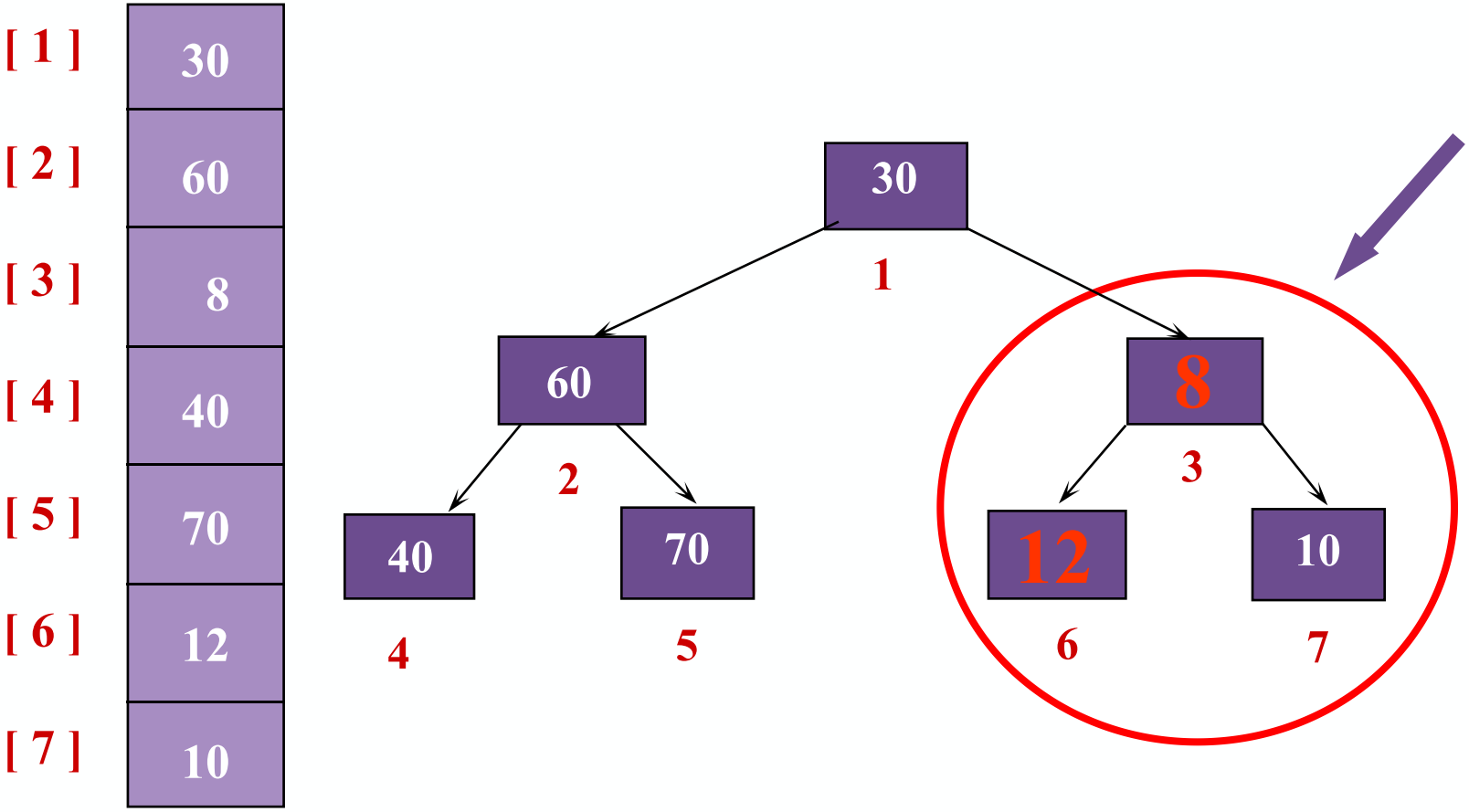
三、堆排序

1.如何将无序序列建成堆

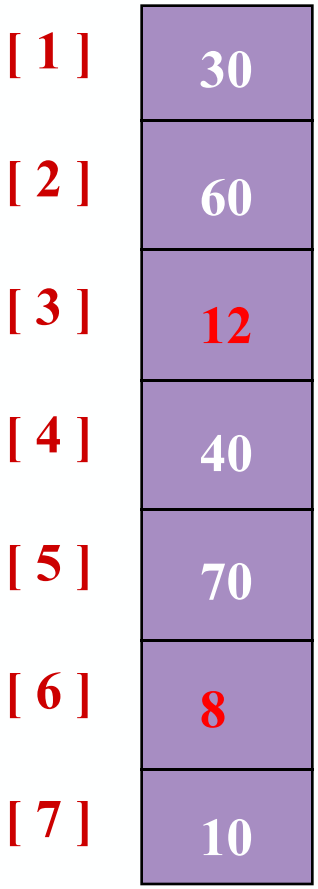
从第⌊n/2⌋个元素起，至第一个元素止，进行反复筛选

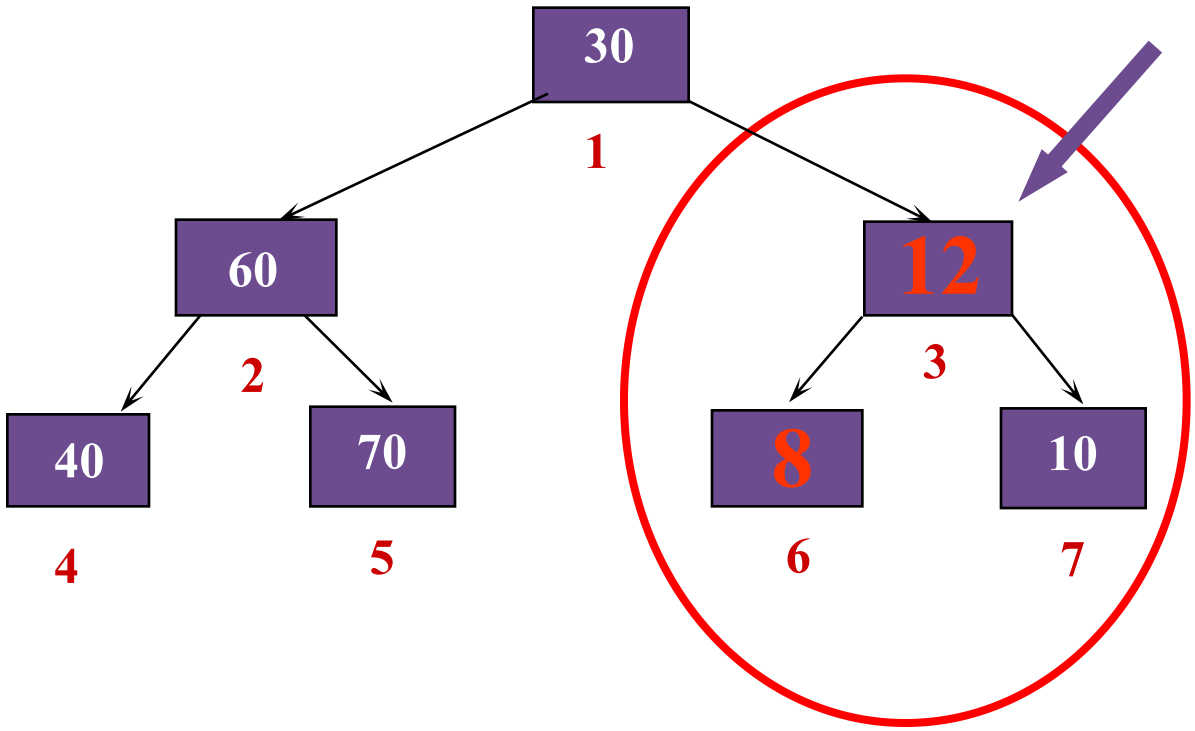


（1）无序序列建成堆 - 1

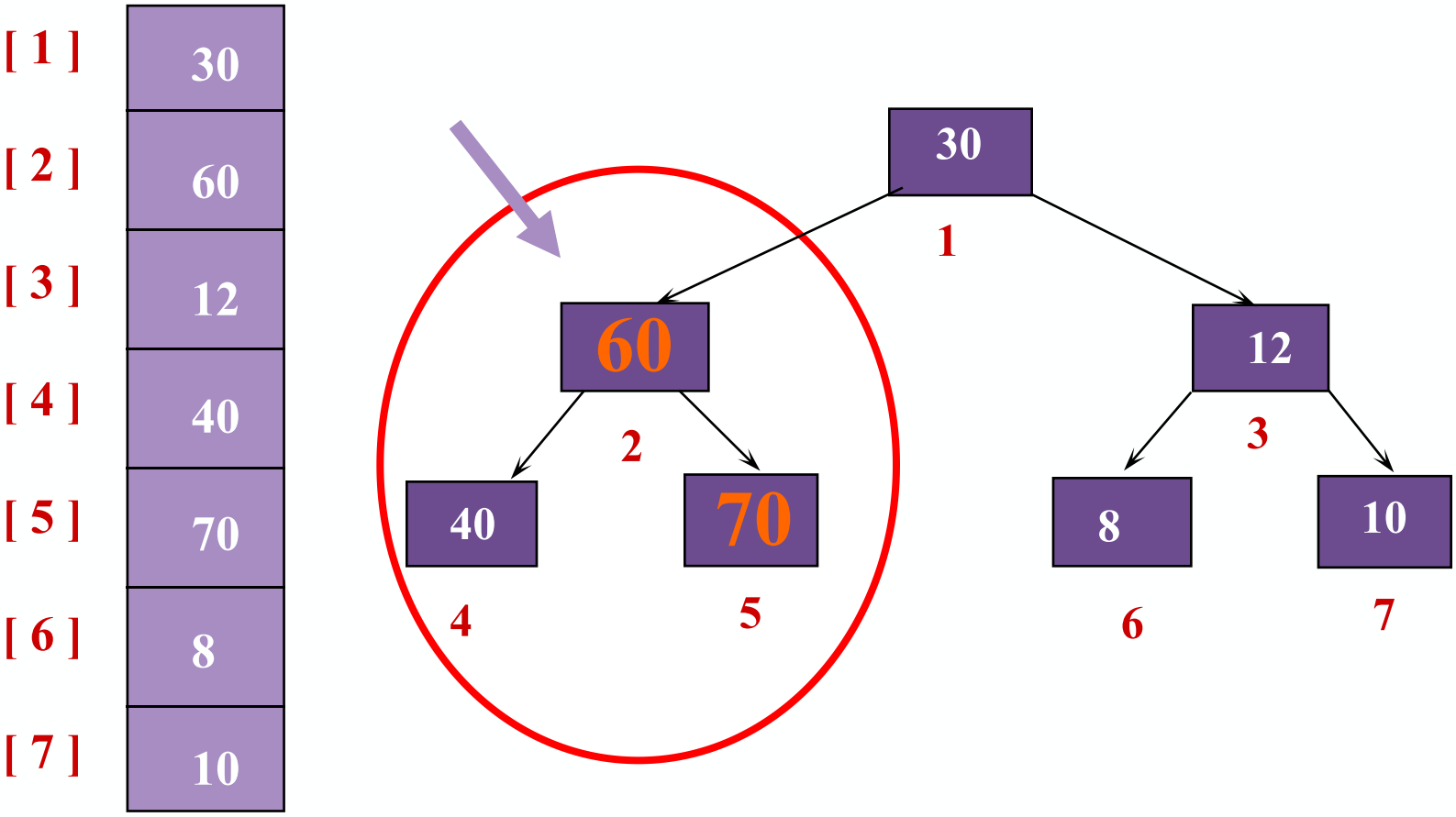


（1）无序序列建成堆 - 1

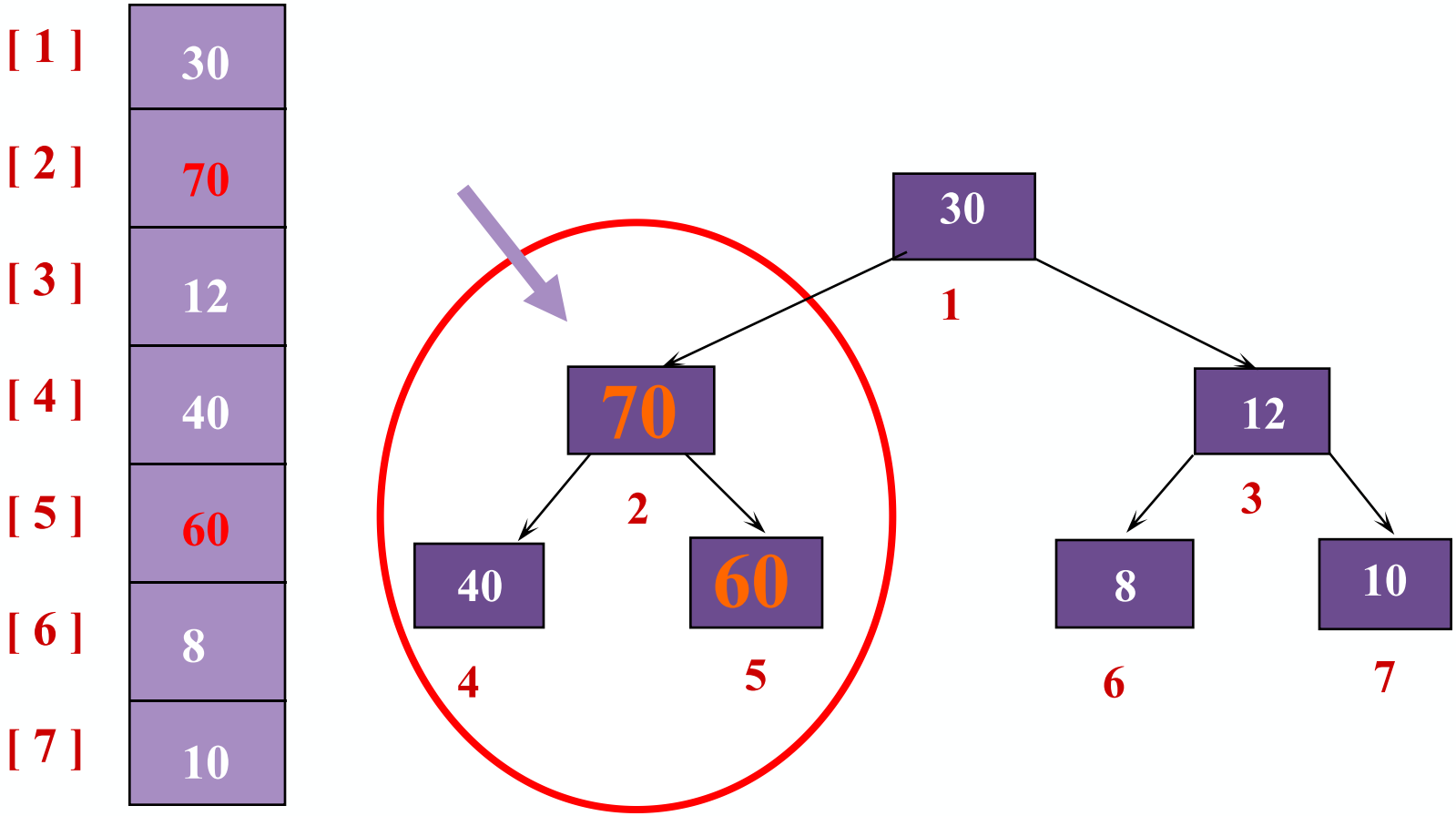




（2）无序序列建成堆 - 2

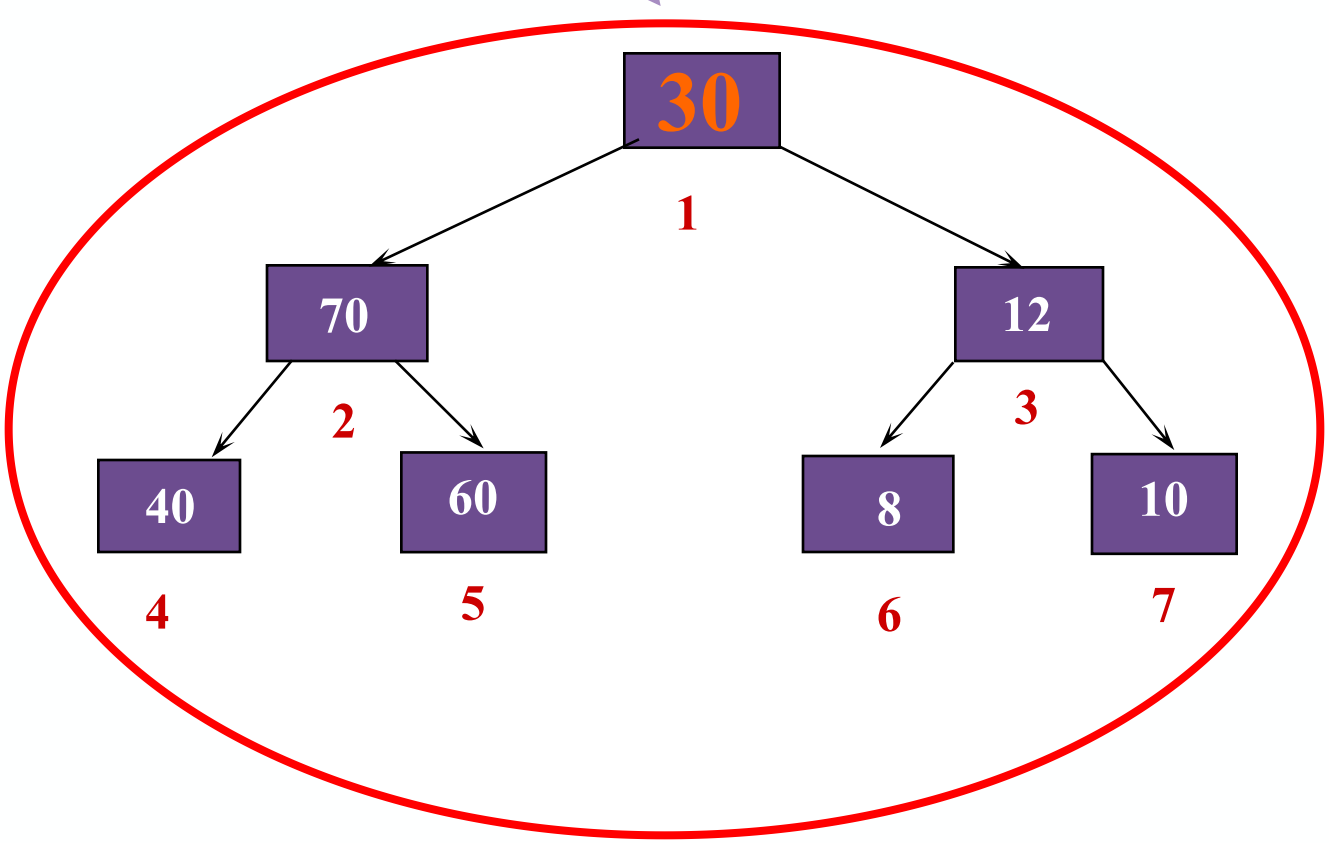


（2）无序序列建成堆 - 2

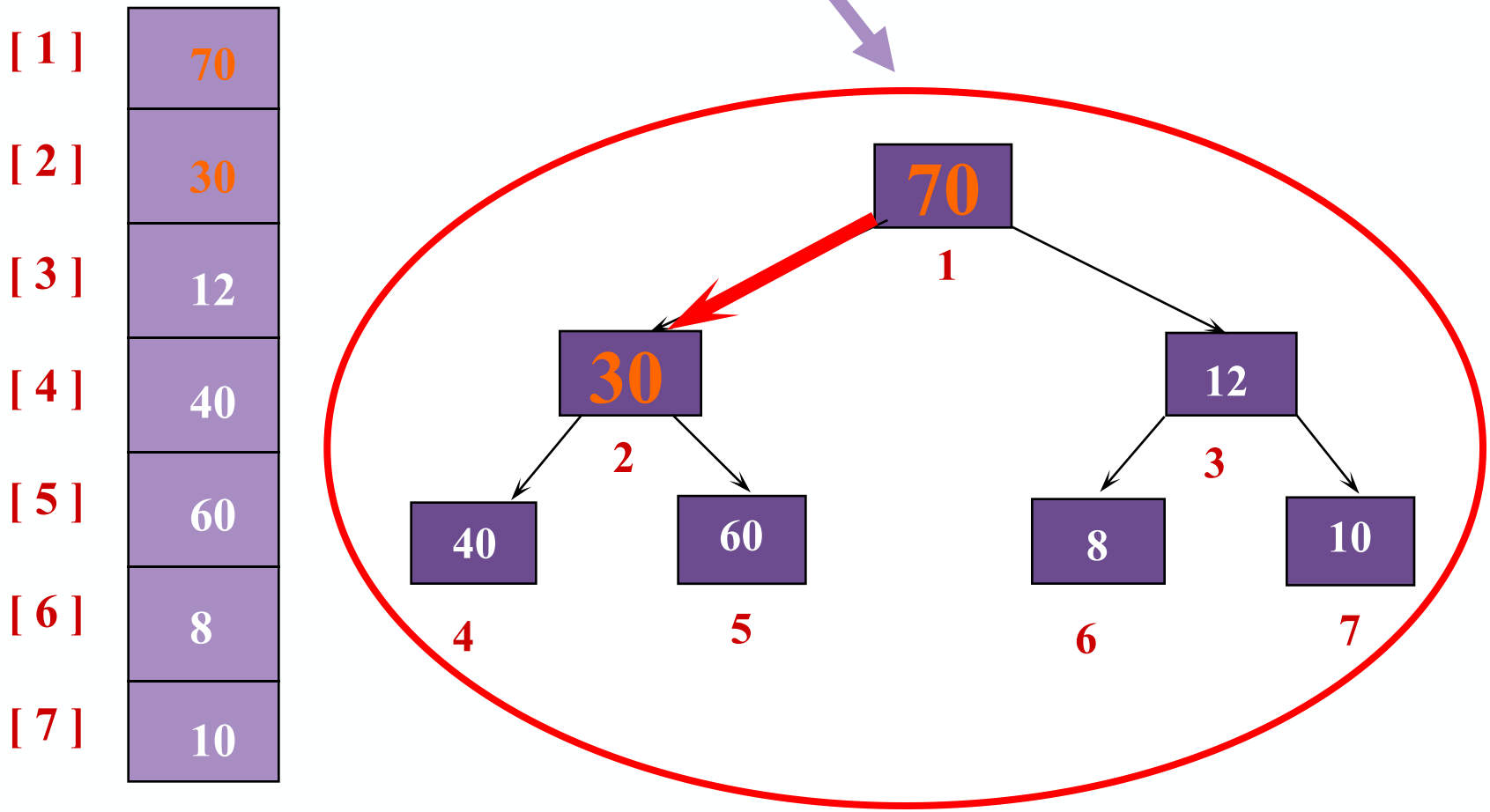


（3）无序序列建成堆 - 3

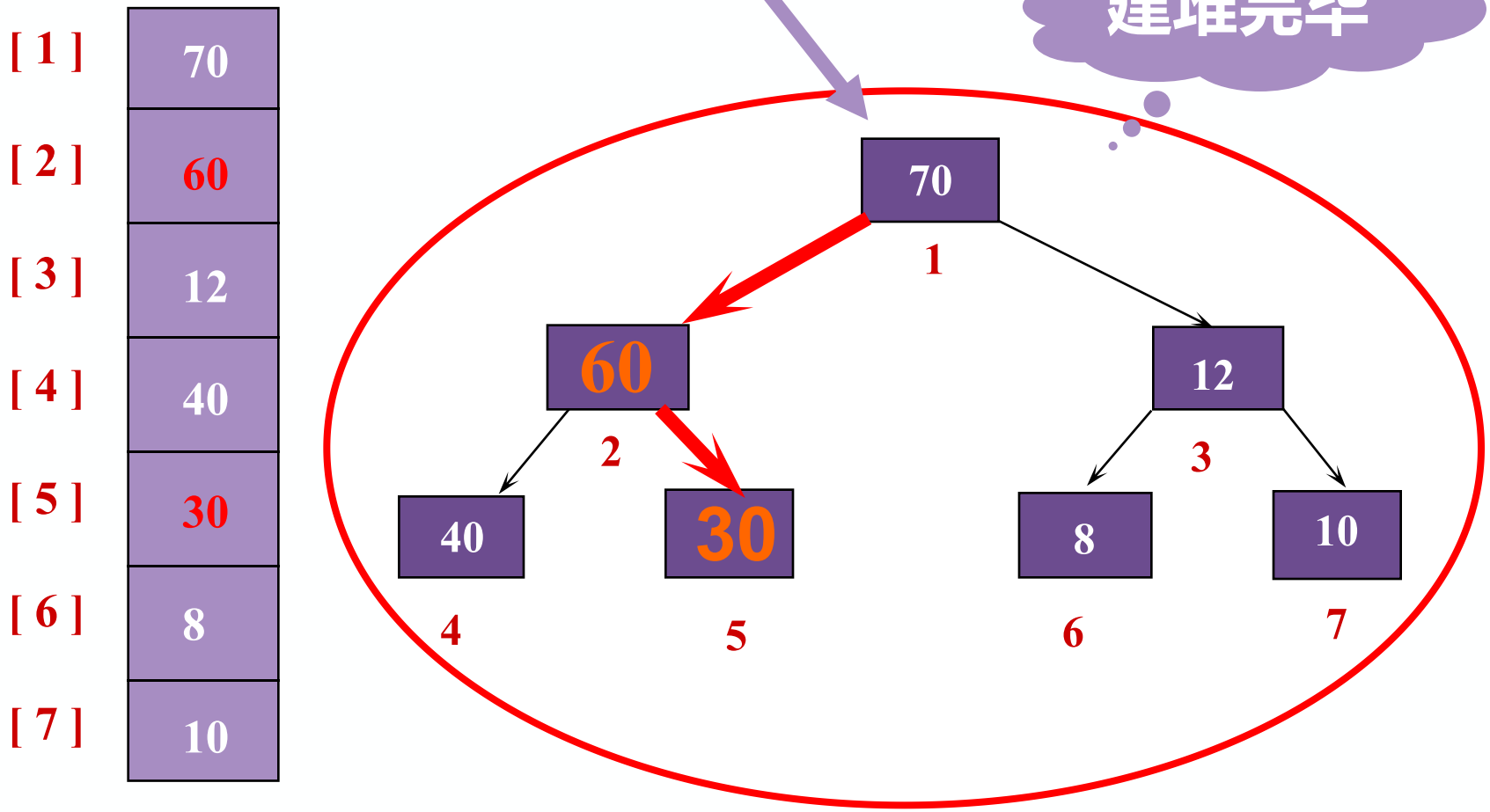
[1]	30
[2]	70
[3]	12
[4]	40
[5]	60
[6]	8
[7]	10



（3）无序序列建成堆 - 3

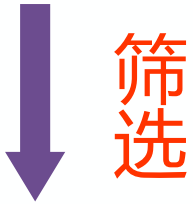


（3）无序序列建成堆 - 3

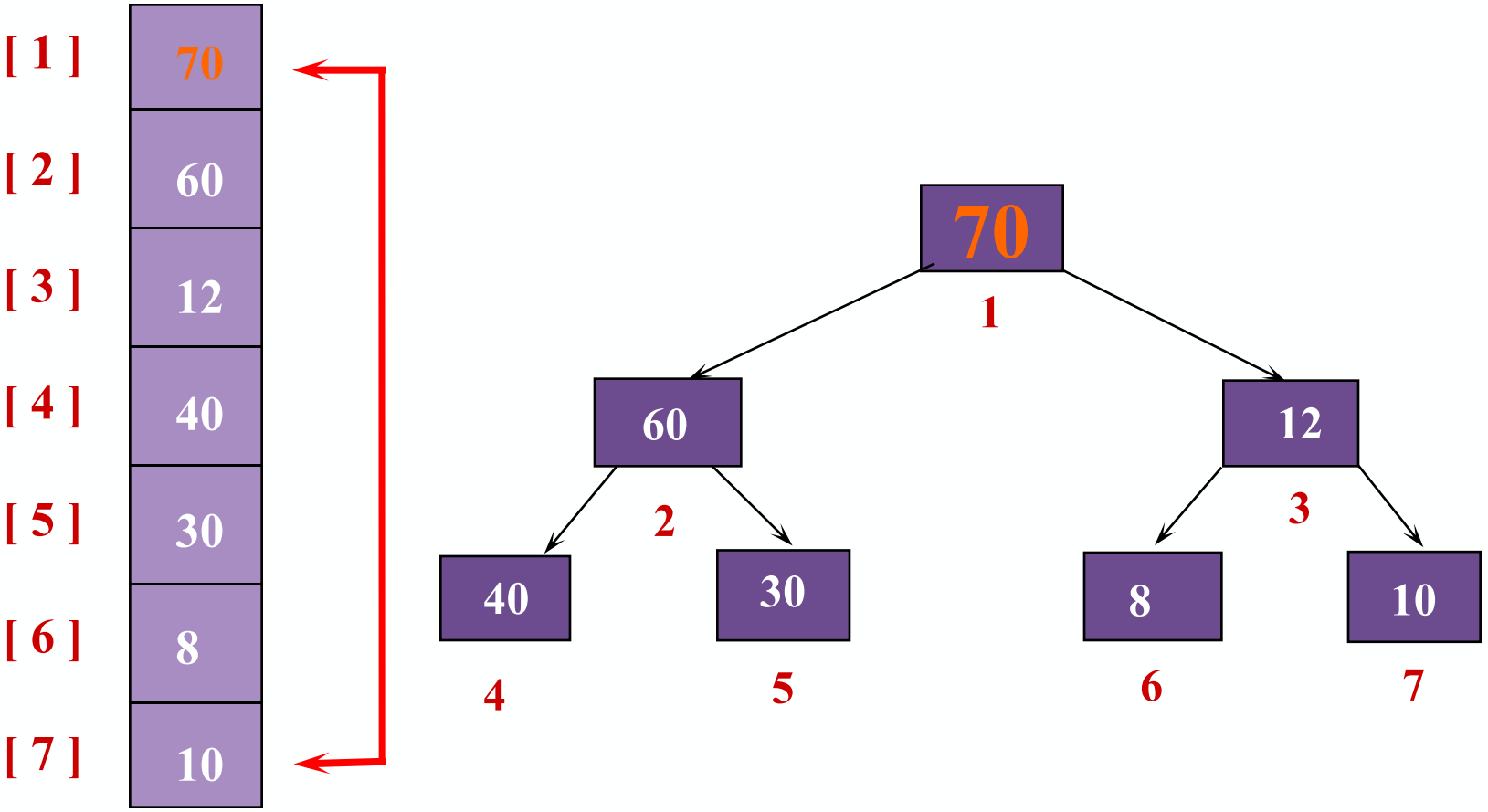


Back

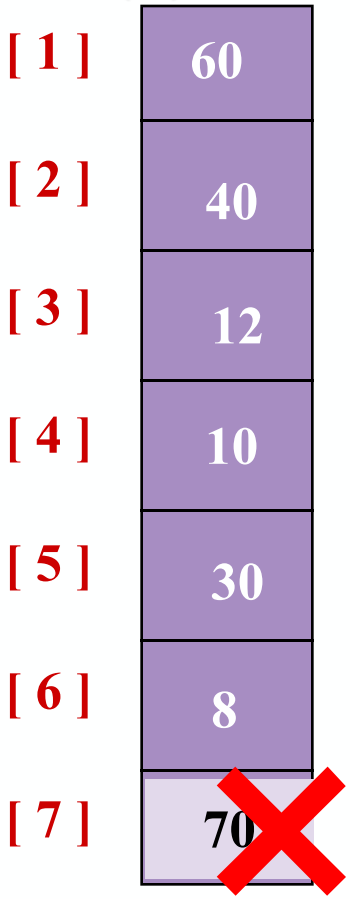
如何在输出堆顶元素后调整，使之成为新堆？

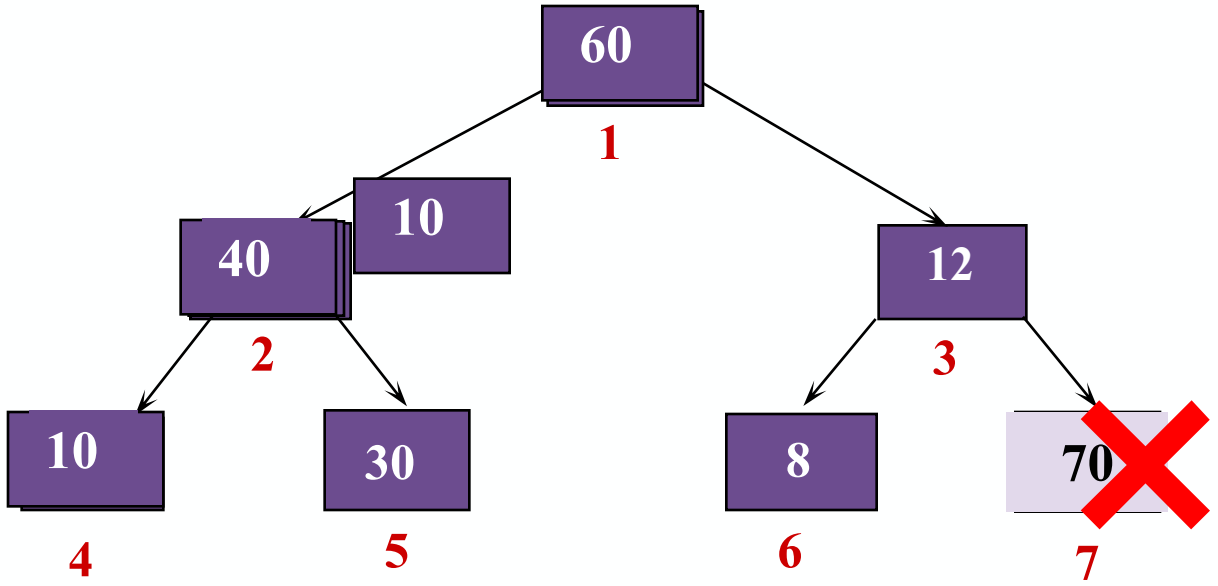


- ✓输出堆顶元素后，以堆中最后一个元素替代之；
- ✓将根结点与左、右子树根结点比较，并与小者交换；
- ✓重复直至叶子结点，得到新的堆。

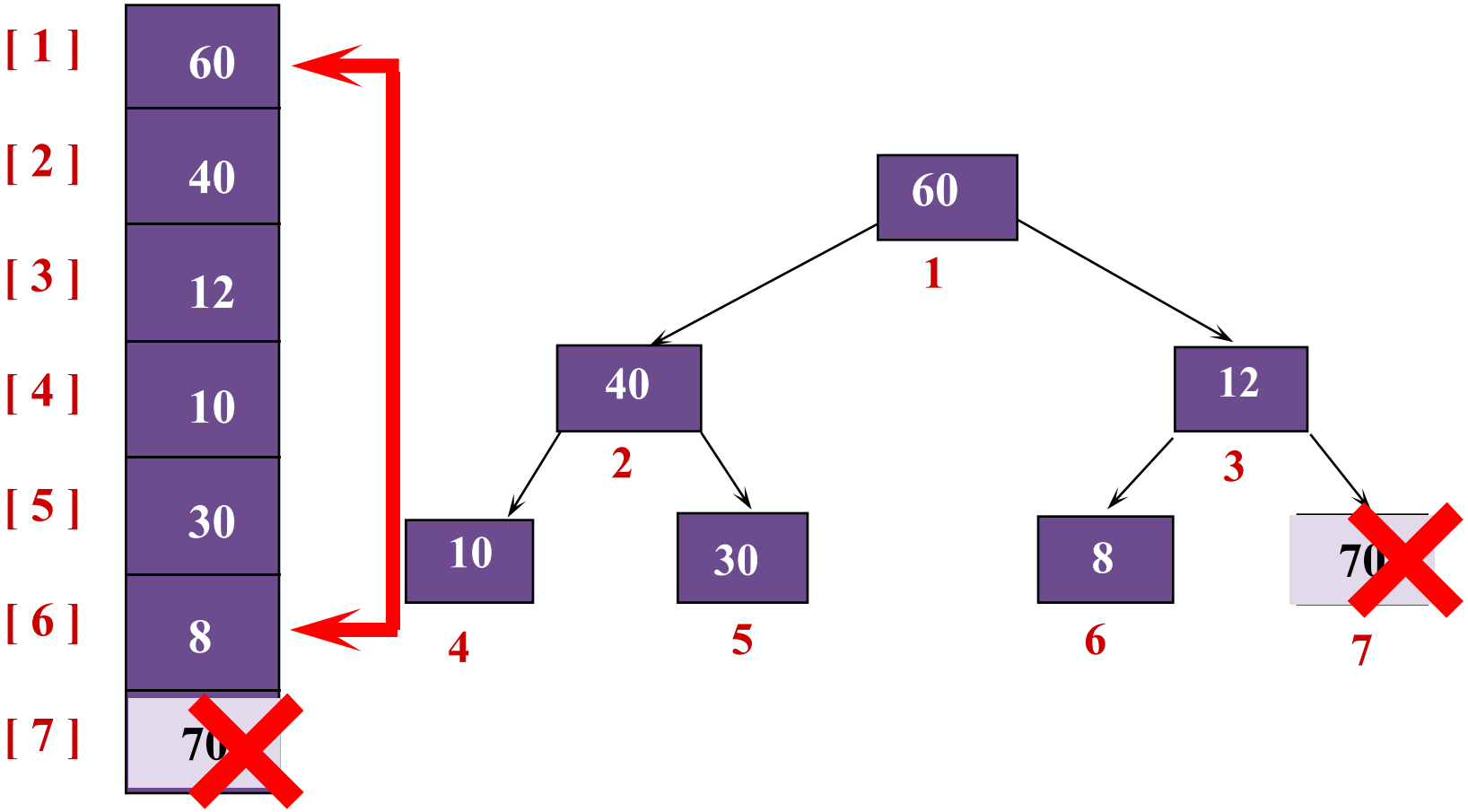

[1]
70
[2]
60
[3]
12
[4]
40
[5]
30
[6]
8
[7]
10
70
1
60
2
12
3
40
4
30
5
8
6
10
7

(1)堆的重新调整 - 1



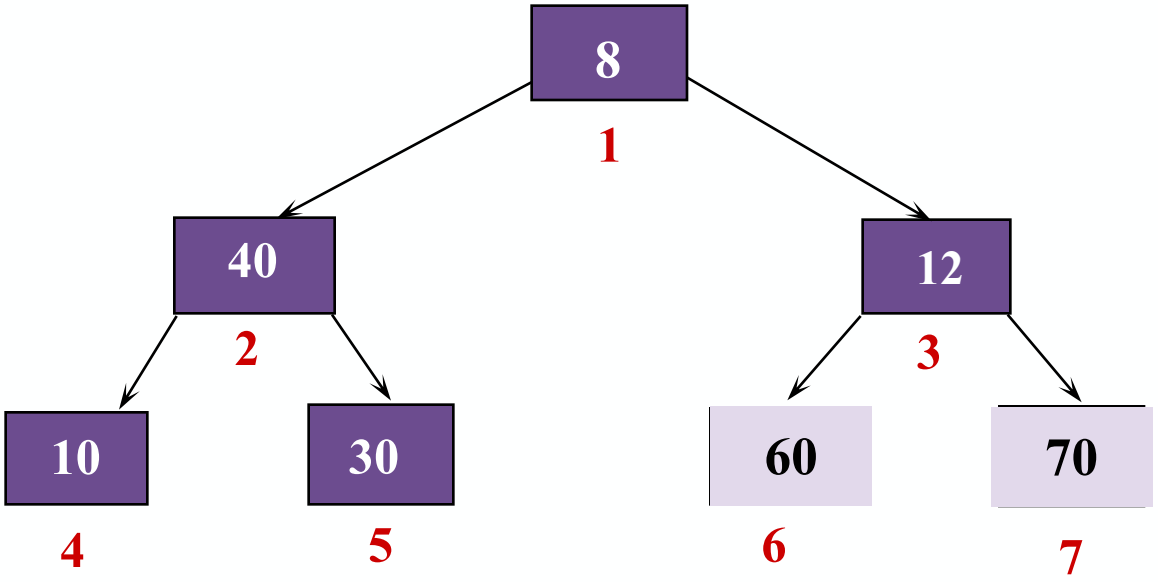


(2)堆的重新调整 - 2

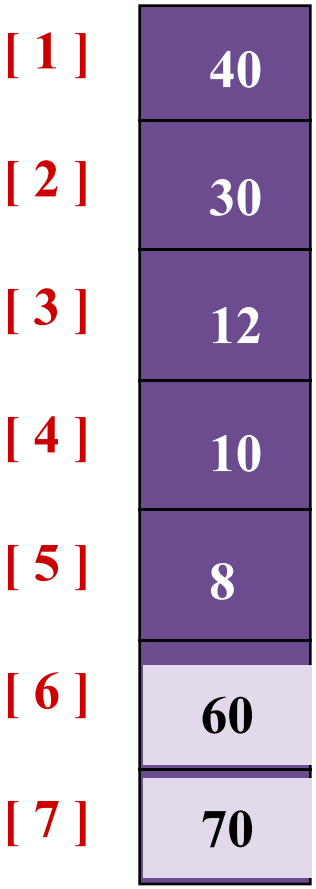


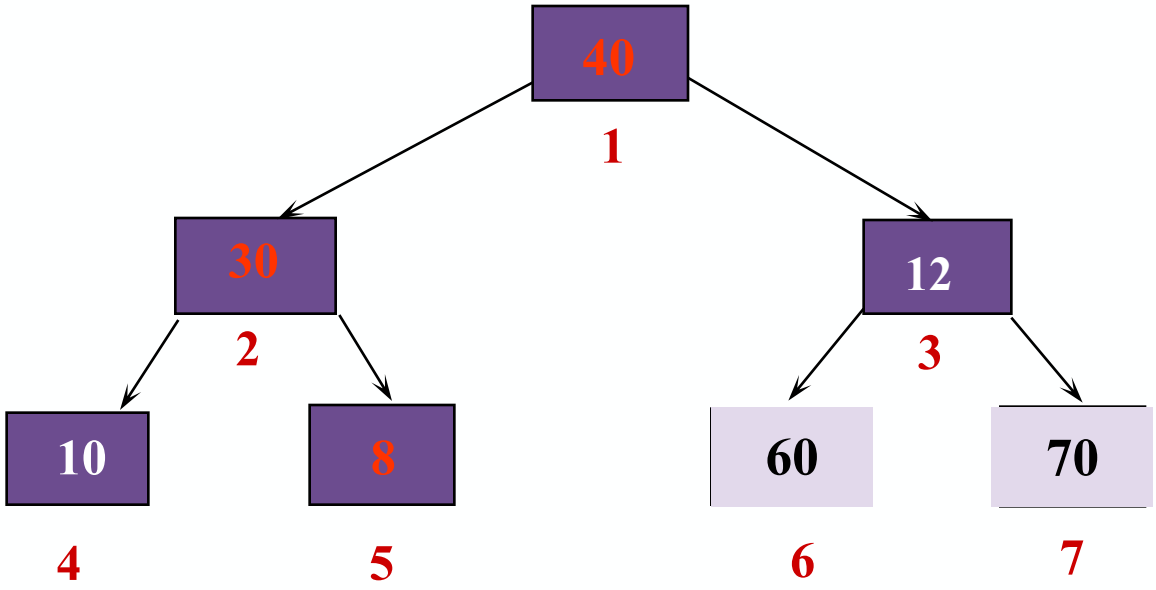
(2)堆的重新调整 - 2

[1]	8
[2]	40
[3]	12
[4]	10
[5]	30
[6]	60
[7]	70

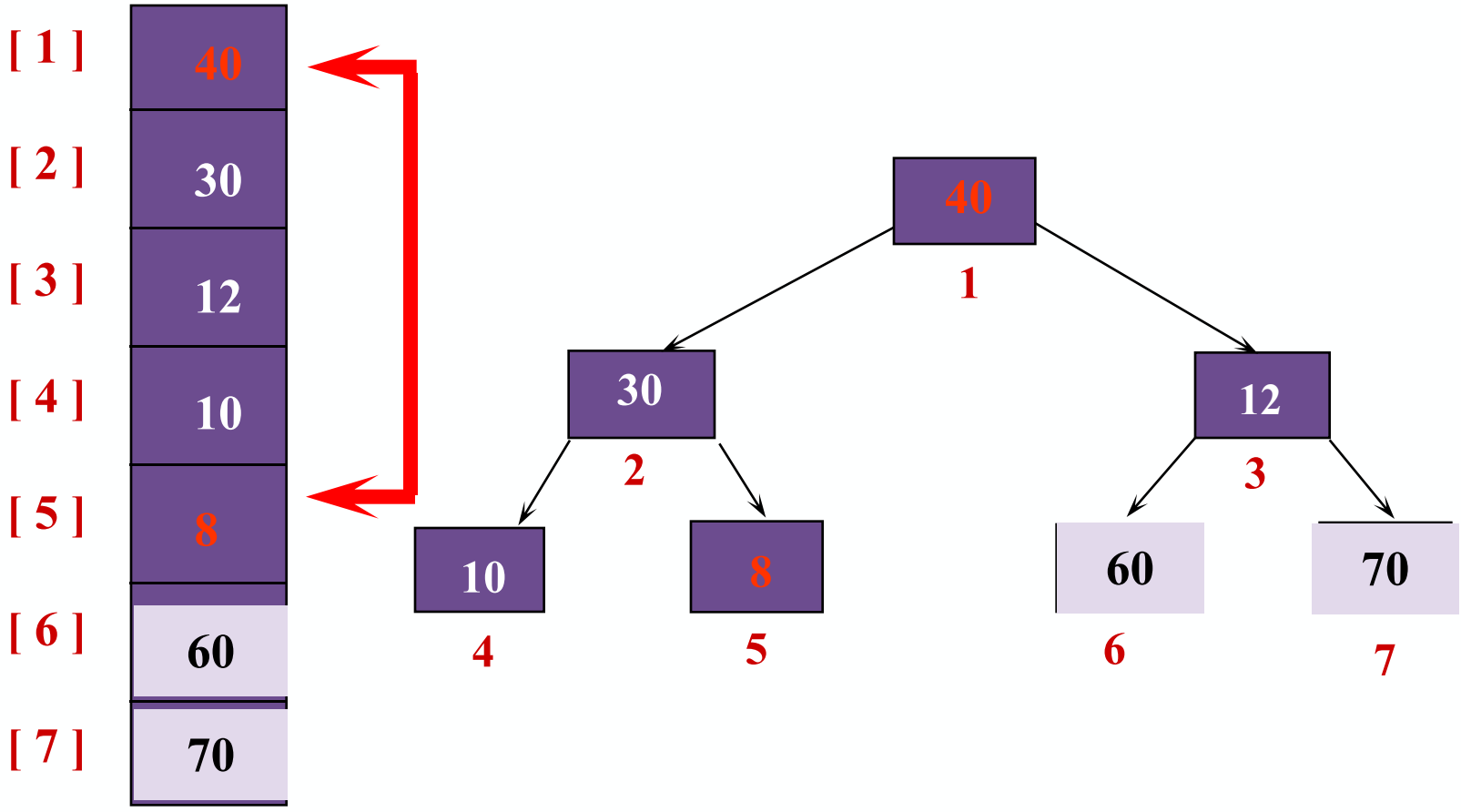


(2)堆的重新调整 - 2

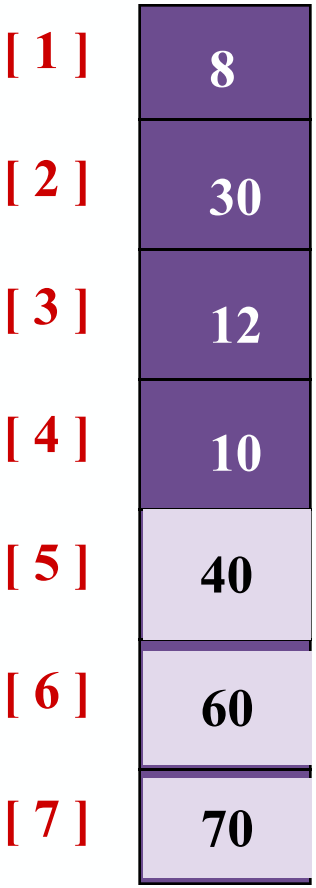


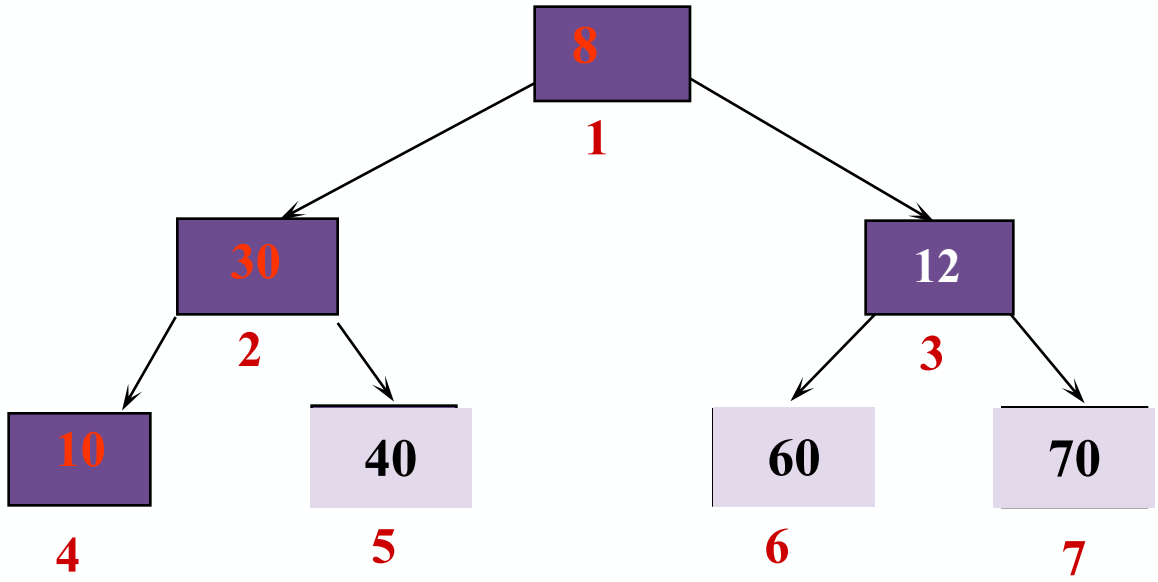


(3)堆的重新调整 - 3

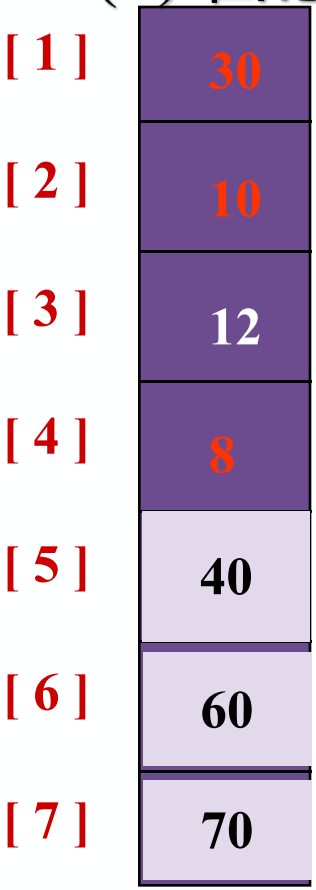


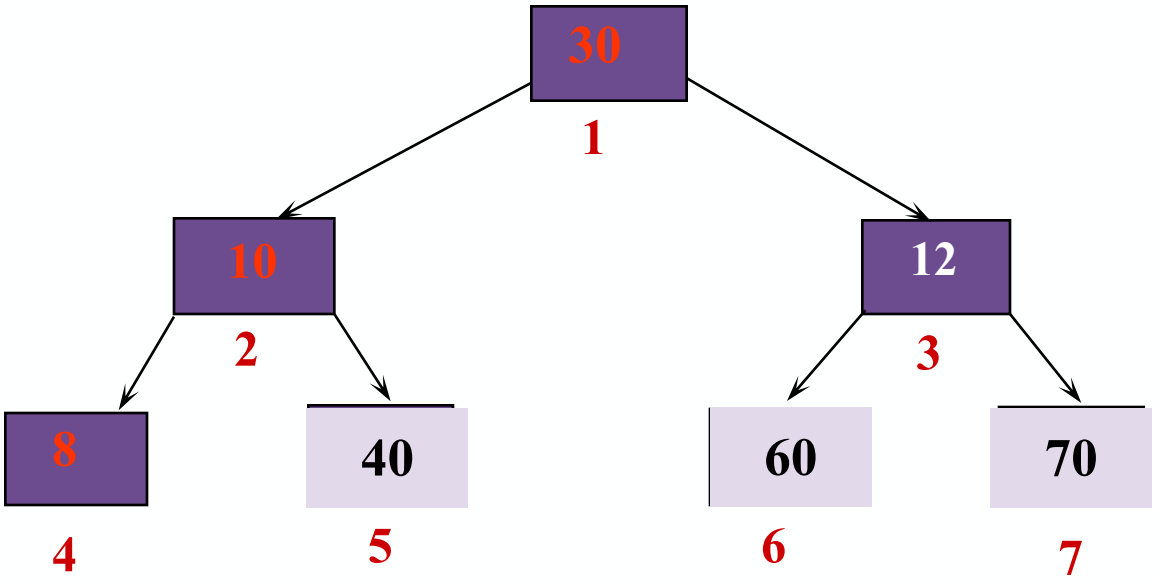
(3)堆的重新调整 - 3



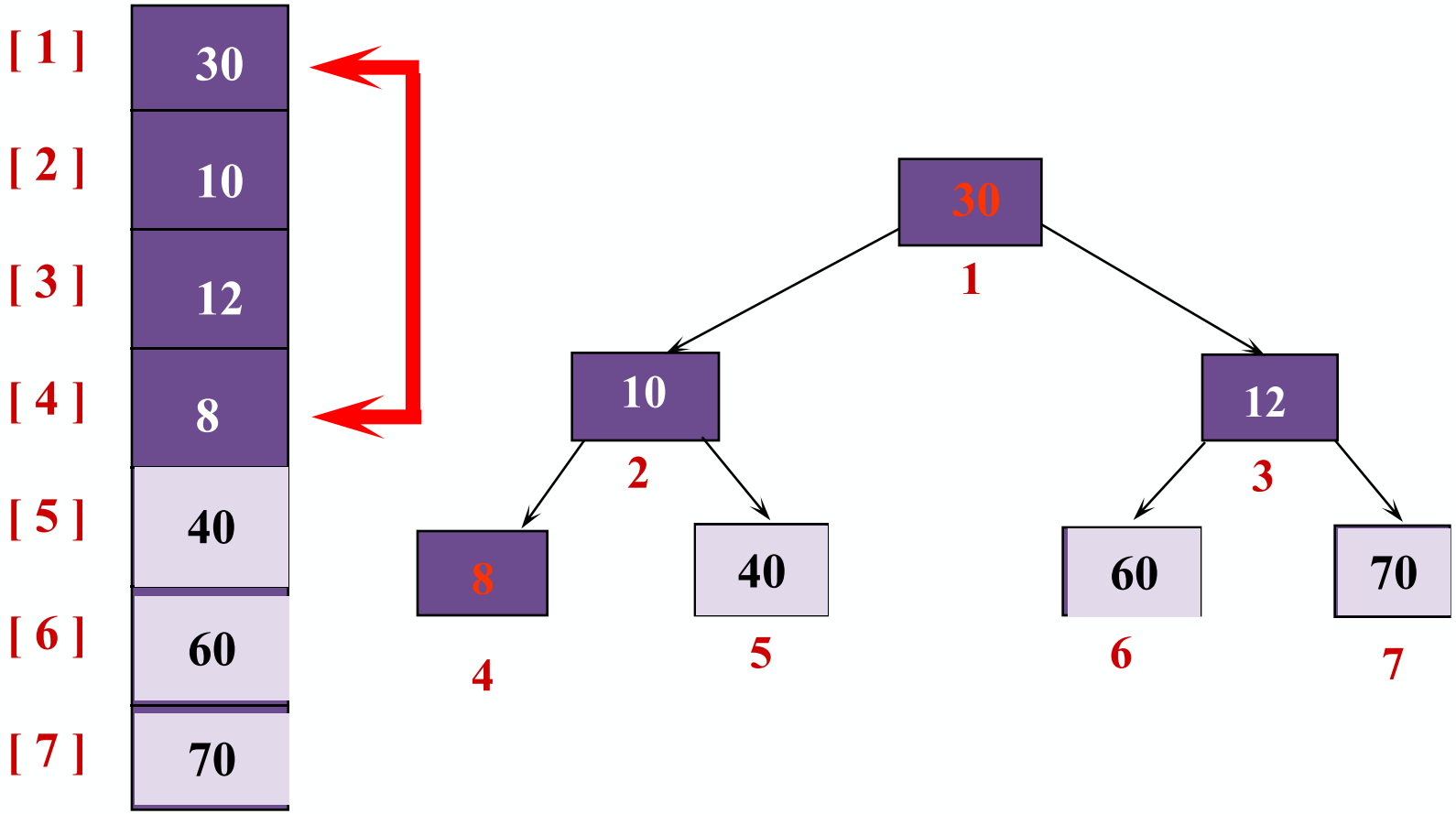


(3)堆的重新调整 - 3

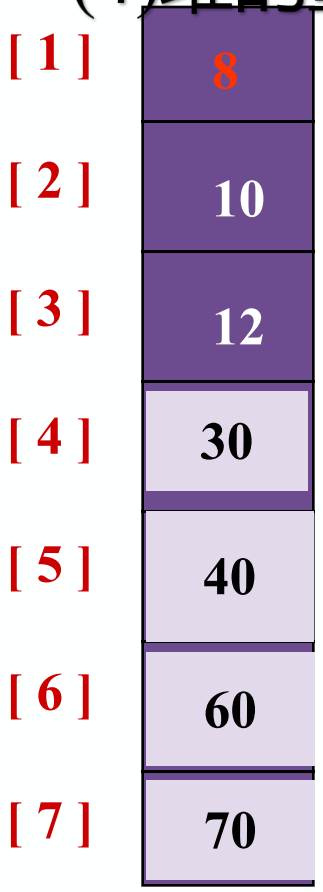


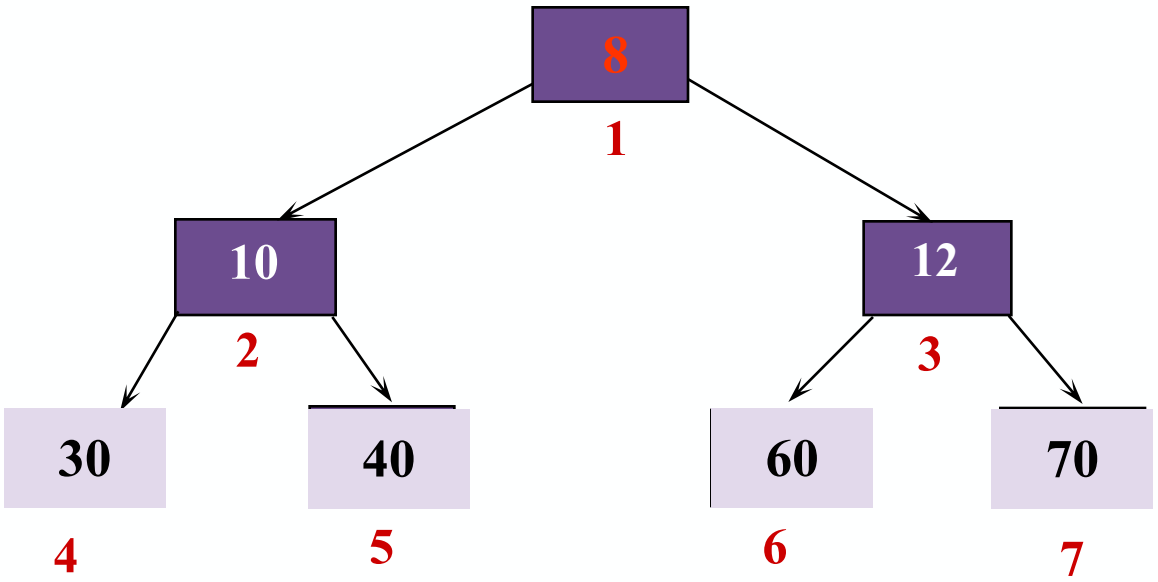


(4)堆的重新调整 - 4

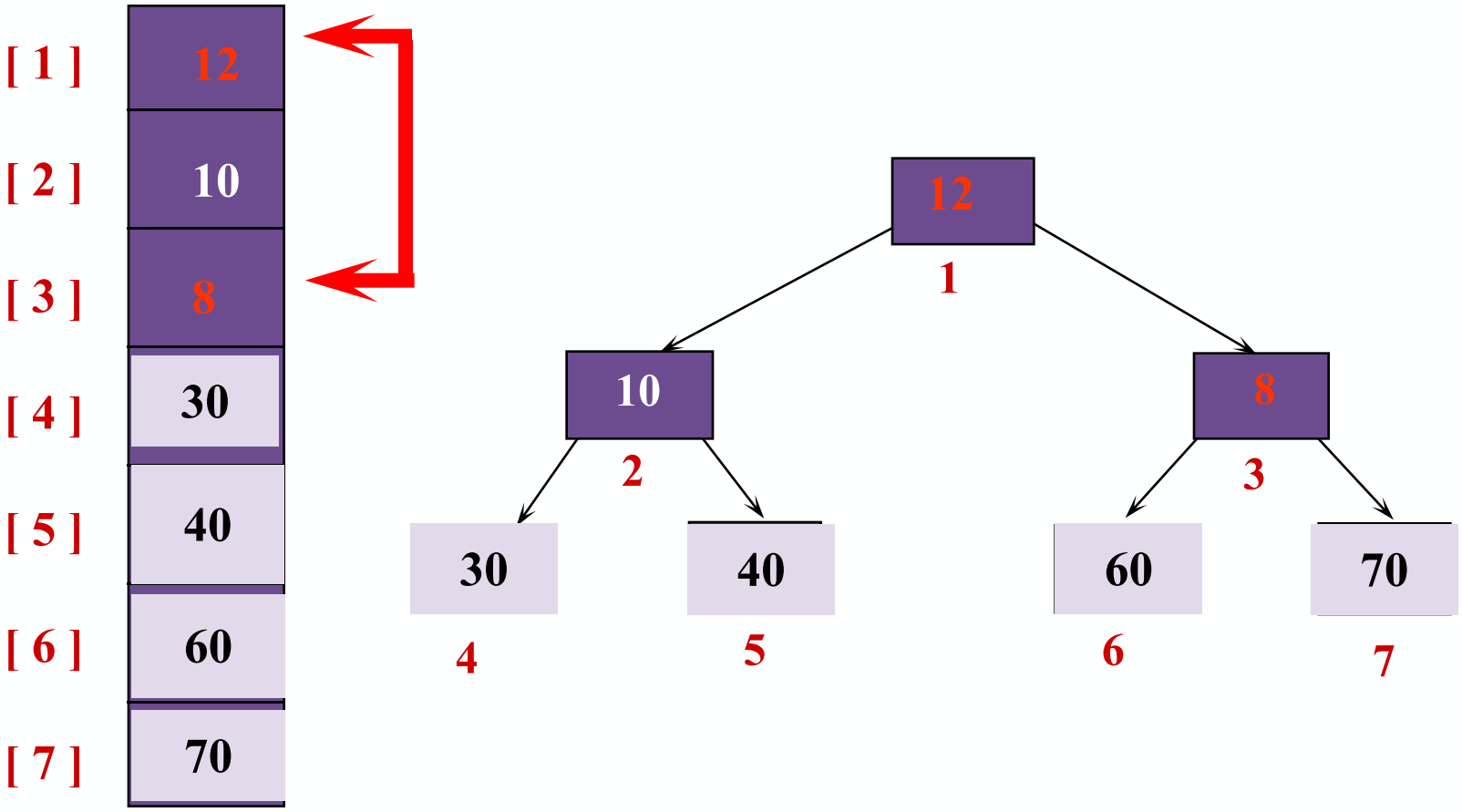


(4)堆的重新调整 - 4

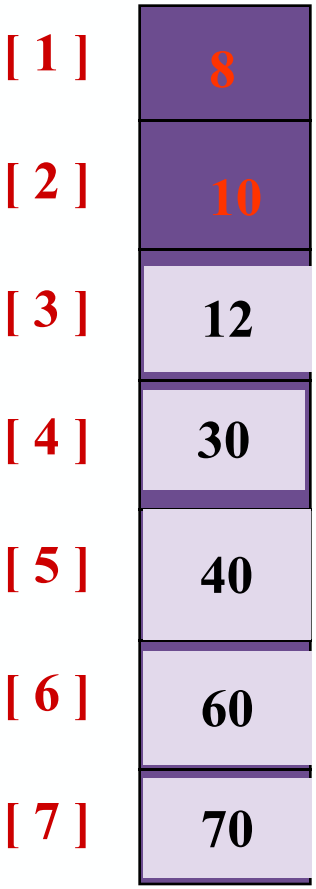


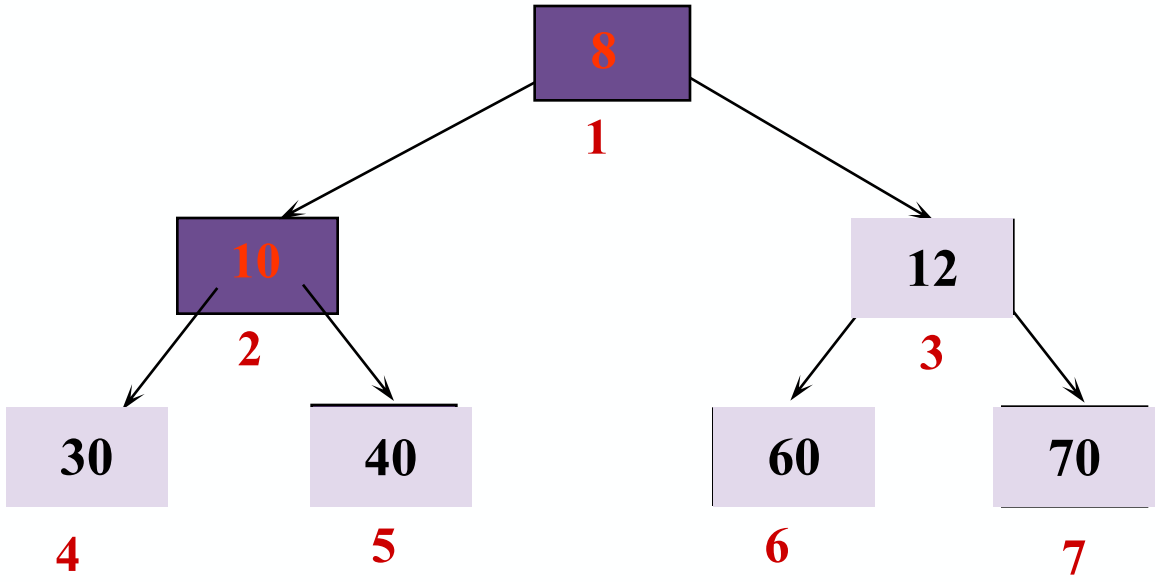


（5）堆的重新调整 - 5

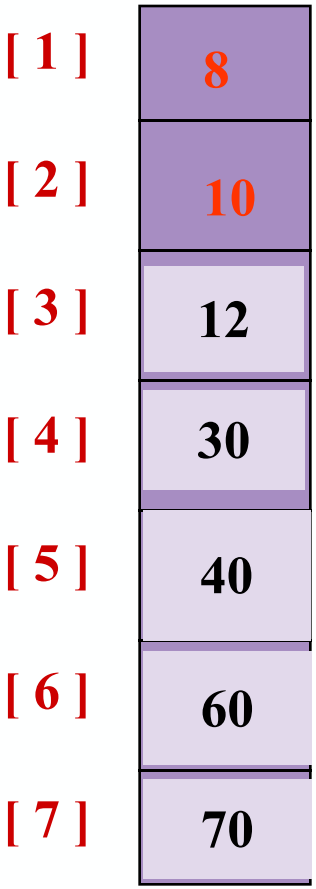


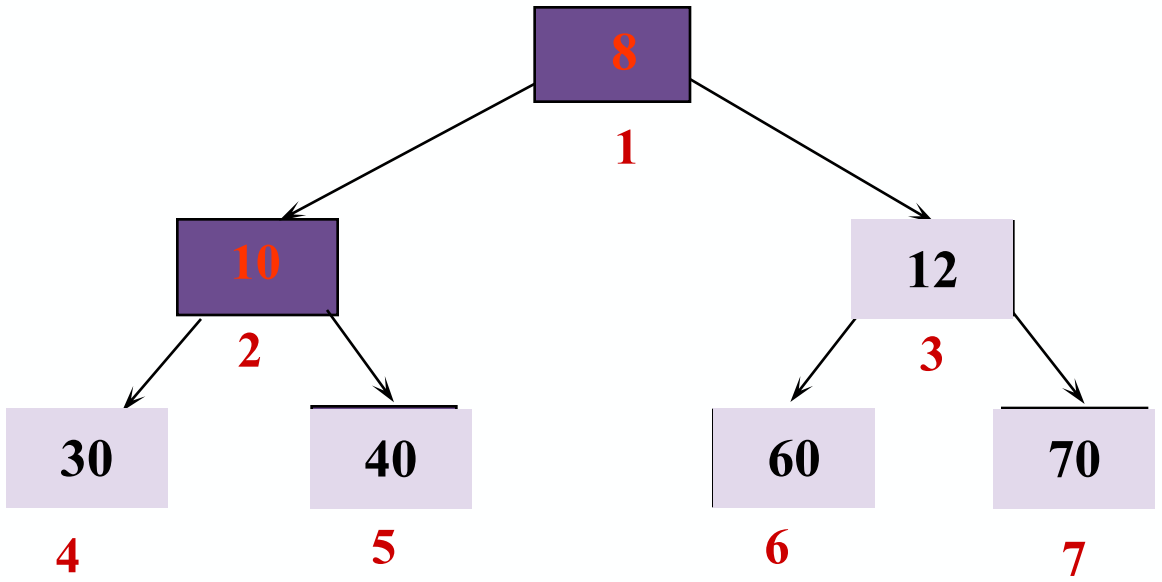
（5）堆的重新调整 - 5



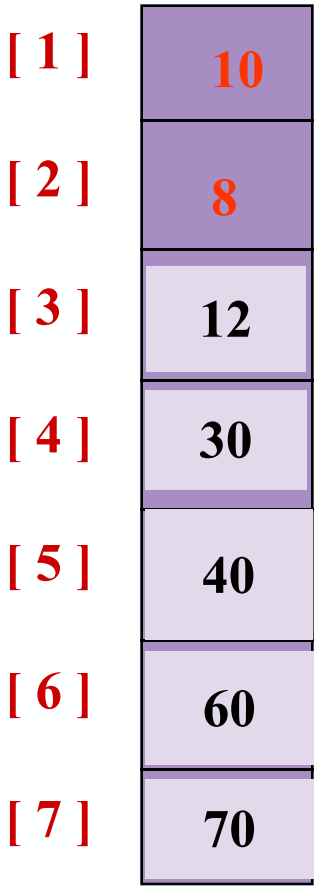


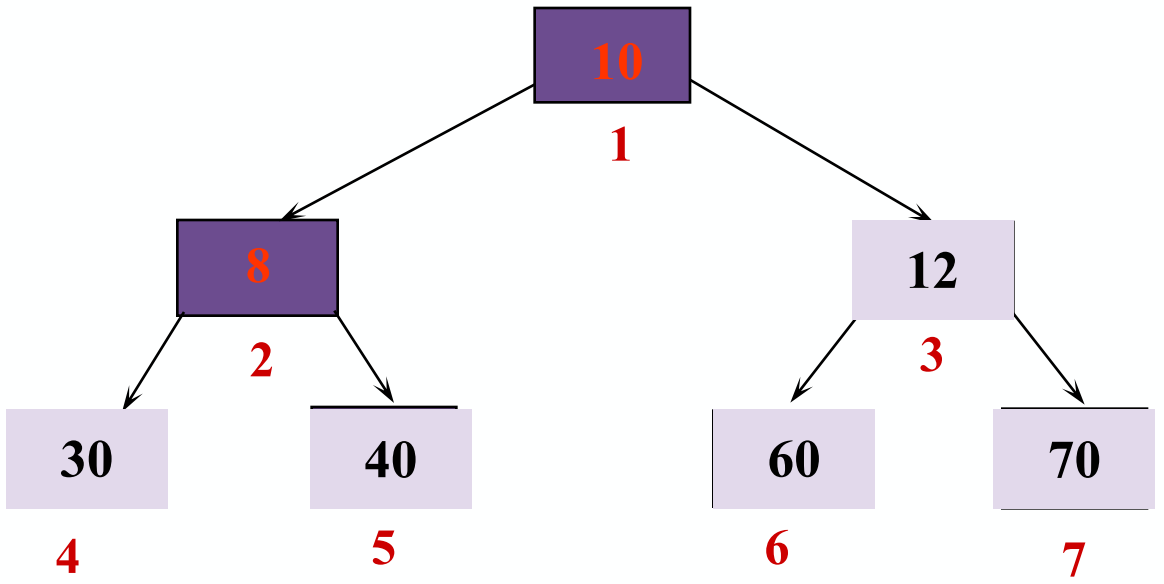
（5）堆的重新调整 - 5



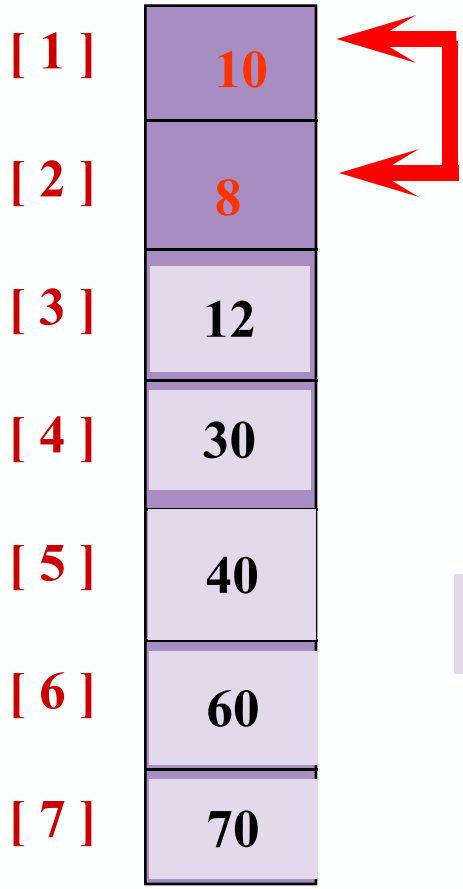


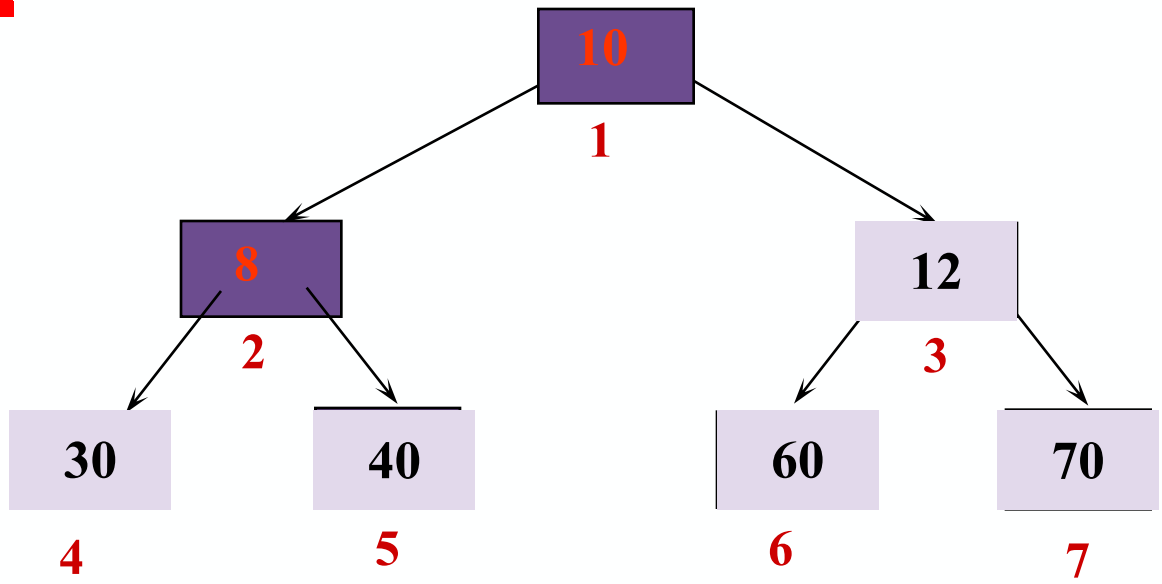
（5）堆的重新调整 - 5



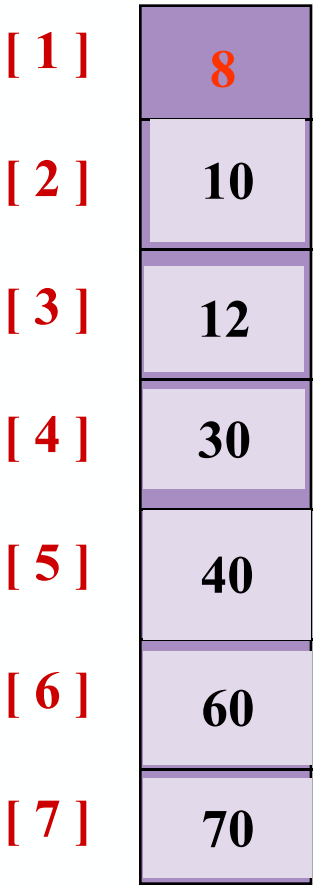


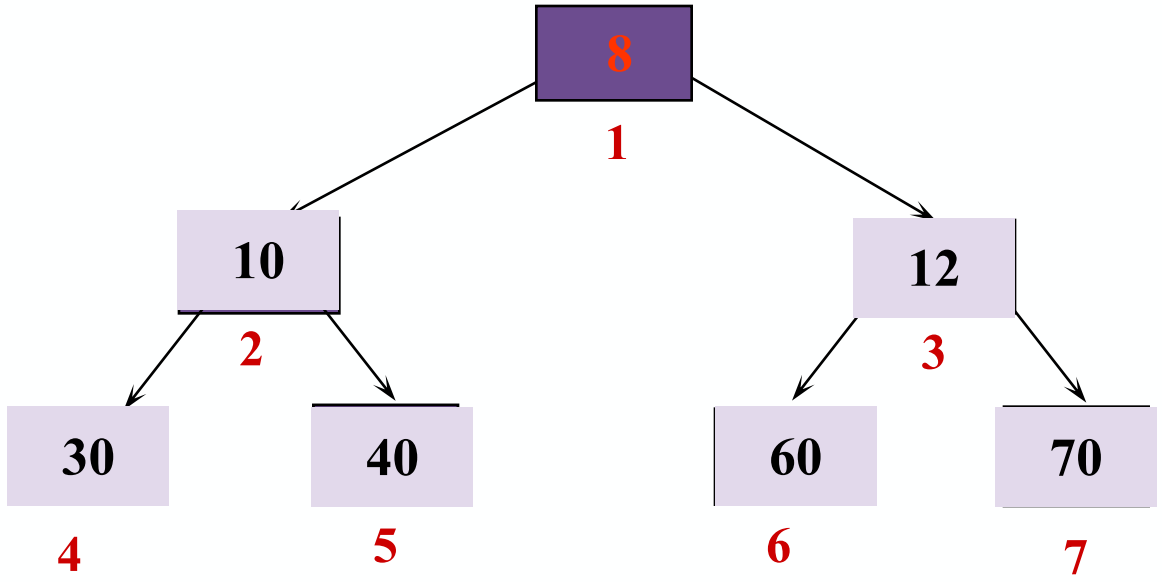
（6）堆的重新调整 - 6





（6）堆的重新调整 - 6





三、堆排序

3.算法分析

时间效率：$O(n\log_2 n)$

空间效率：O（1）

稳 定 性：不稳定

适用于n 较大的情况

小结

- 简单选择排序方法
- 堆的建立和堆的调整方法以及堆排序的过程