



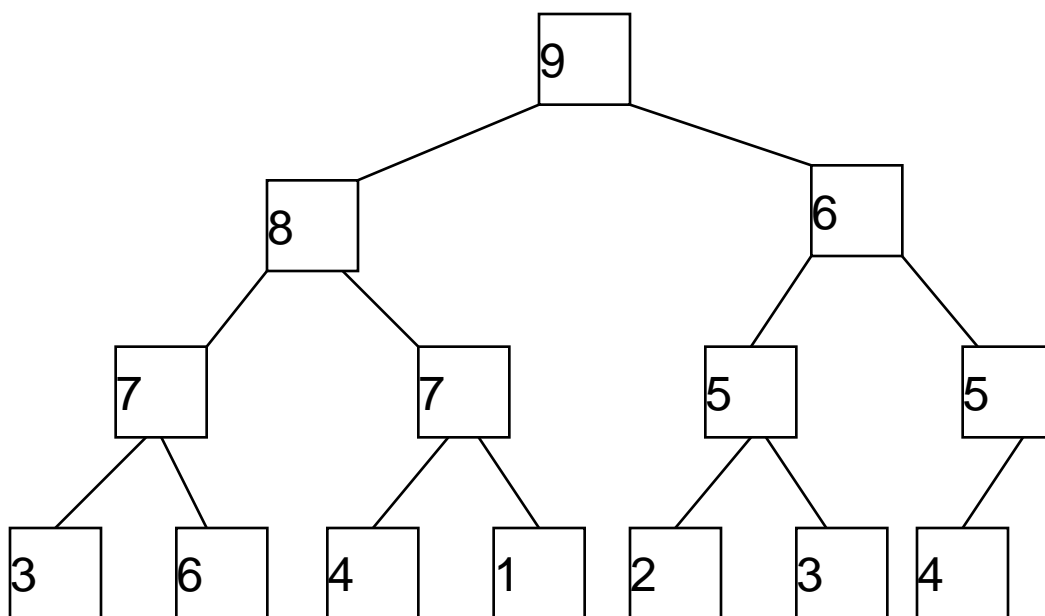
heap算法

瞿绍军
湖南师范大学



Heap算法

heap是一种特殊的元素组织方式，可以被视为二叉树
第一个元素总是最大
总是能够在对数时间那增加或移除一个元素



Heap算法

*heap算法

*make_heap()—将某区间内的元素转化为heap

*push_heap(beg, end)—[beg, end-1)原本就是heap, 将end之前的那个元素加入使区间[beg, end)]重新成为heap

*pop_heap(beg, end)—从区间[beg, end)取出第一个元素，放到最后位置，区间[beg, end-1)重新组成heap

*sort_heap()—将heap转换为一个有序集合

*可以用<运算符或op(elem1,elem2)作为排序准则

*示例: heap



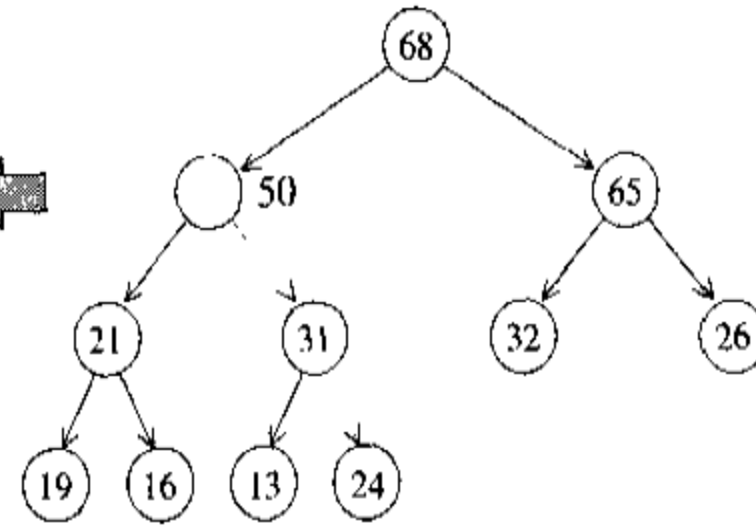
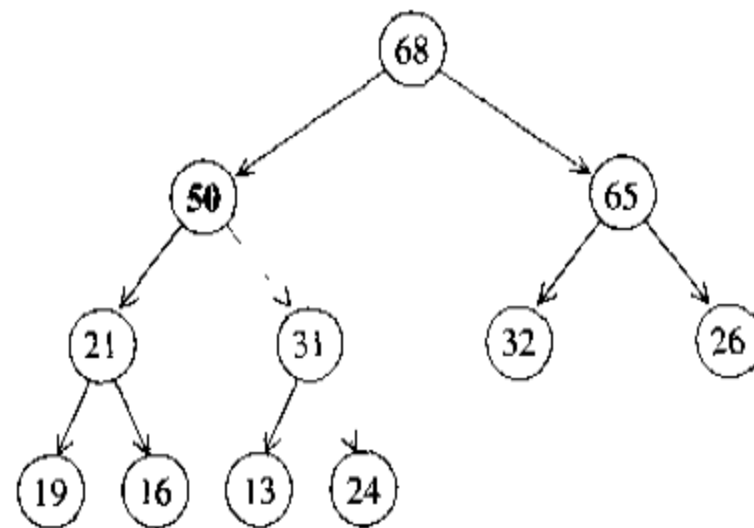
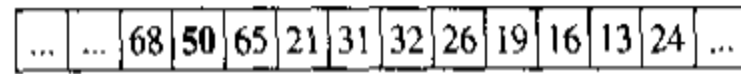
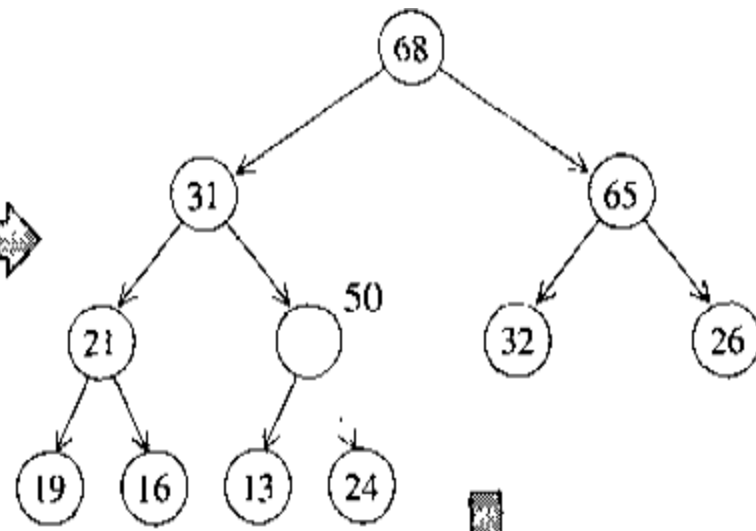
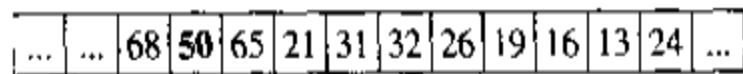
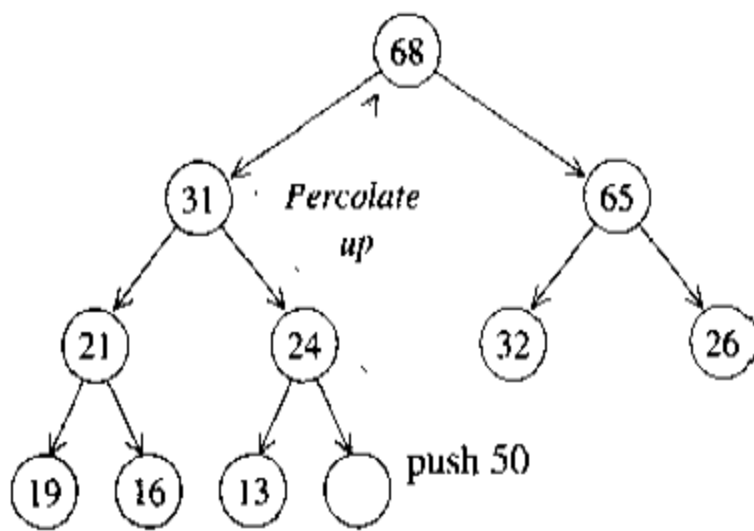
push_heap 算法

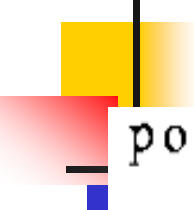
图 4-21 所示的是 `push_heap` 算法的实际操演情况。为了满足 **complete binary tree** 的条件，新加入的元素一定要放在最下一层作为叶节点，并填补在由左至右的第一个空格，也就是把新元素插入在底层 **vector** 的 `end()` 处。

...	...	68	31	65	21	24	32	26	19	16	13	50	...
-----	-----	----	----	----	----	----	----	----	----	----	----	----	-----

...	...	68	31	65	21	50	32	26	19	16	13	24	...
-----	-----	----	----	----	----	----	----	----	----	----	----	----	-----





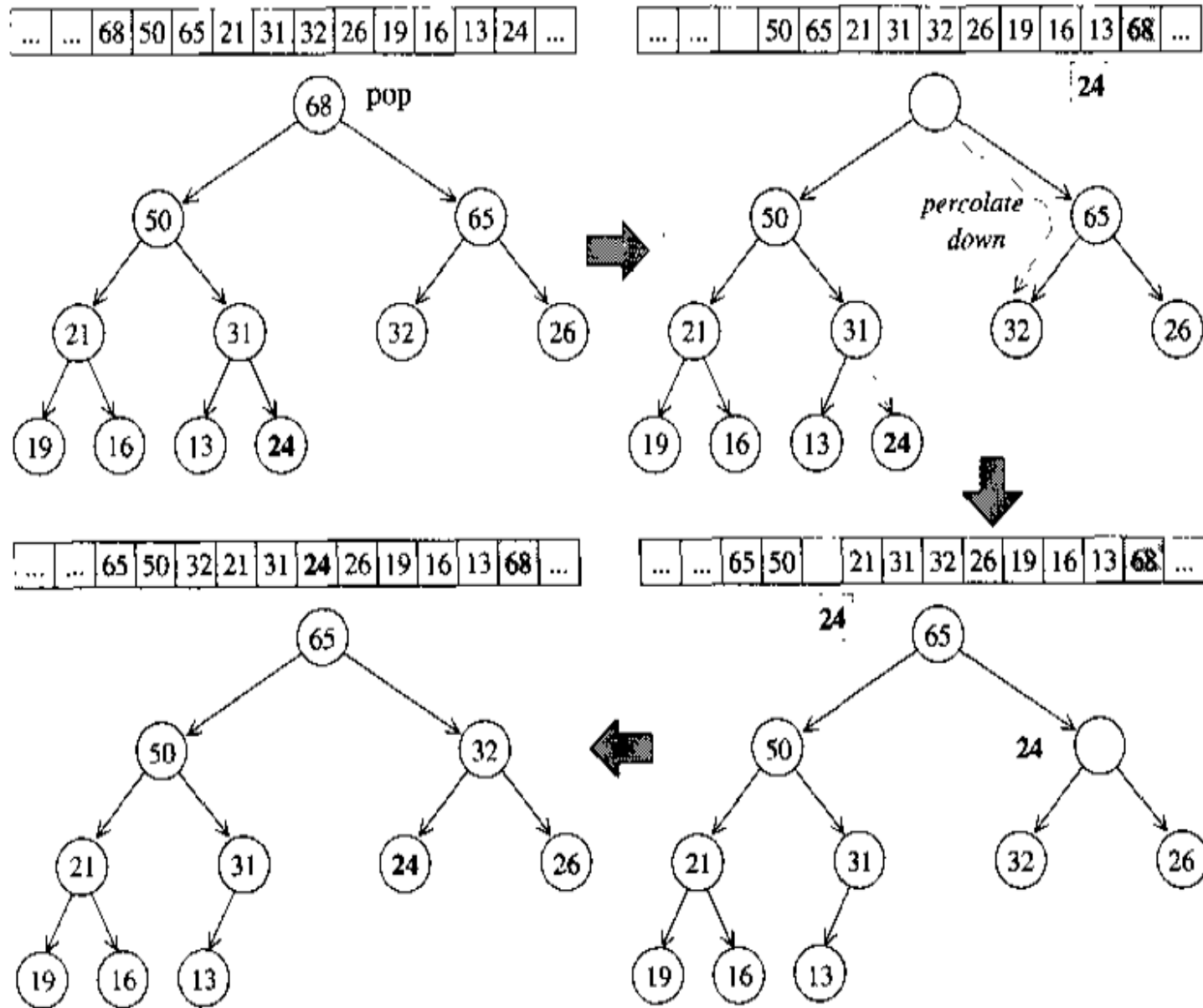


pop_heap 算法

图 4-22 所示的是 pop heap 算法的实际操演情况。既然身为 max-heap，最大值必然在根节点。pop 操作取走根节点（其实是移至底部容器 vector 的最后一个元素）之后，为了满足 complete binary tree 的条件，必须将最下一层最右边的叶节点拿掉，现在我们的任务是为这个被拿掉的节点找一个适当的位置。

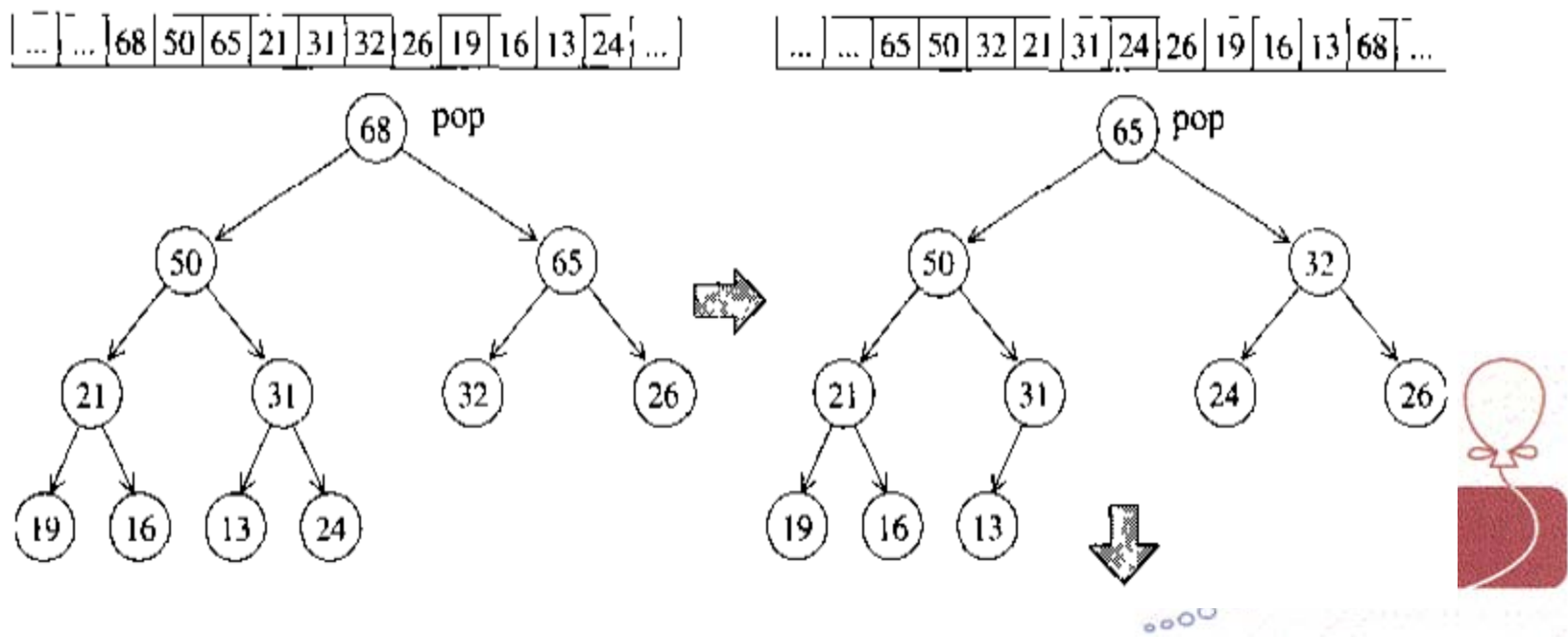
注意，pop_heap 之后，最大元素只是被置放于底部容器的最尾端，尚未被取走。如果要取其值，可使用底部容器（vector）所提供的 back() 操作函数。如果要移除它，可使用底部容器（vector）所提供的 pop_back() 操作函数。





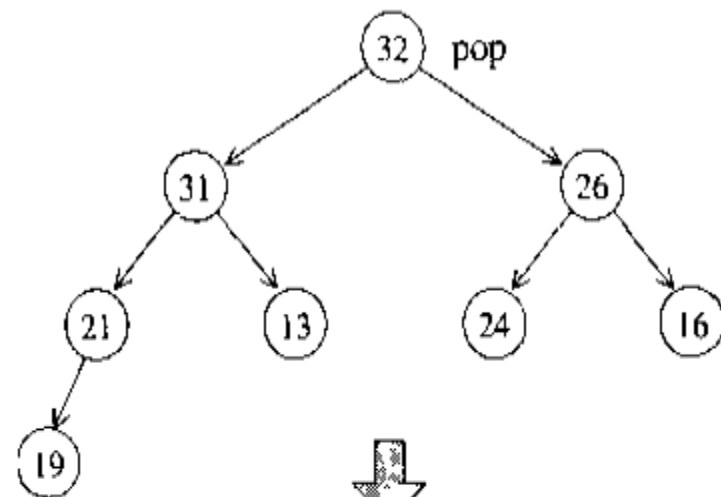
sort_heap 算法

既然每次 `pop_heap` 可获得 `heap` 中键值最大的元素，如果持续对整个 `heap` 做 `pop_heap` 操作，每次将操作范围从后向前缩减一个元素（因为 `pop_heap` 会把键值最大的元素放在底部容器的最尾端），当整个程序执行完毕时，我们便有了一个递增序列。图 4-23 所示的是 `sort_heap` 的实际操演情况。

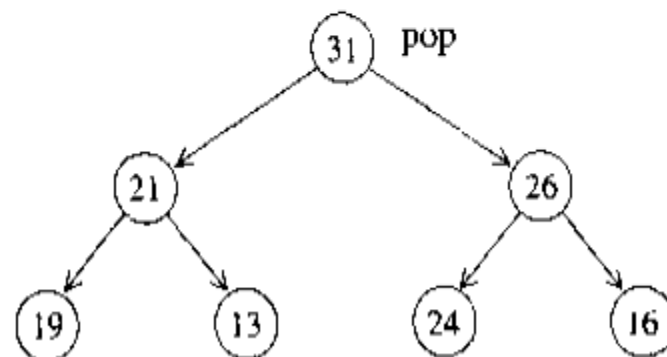




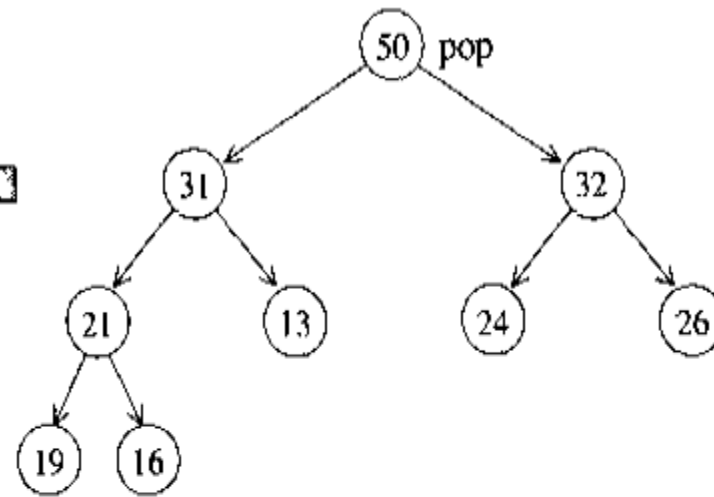
...	...	32	31	26	21	13	24	16	19	50	65	68	...
-----	-----	----	----	----	----	----	----	----	----	----	----	----	-----



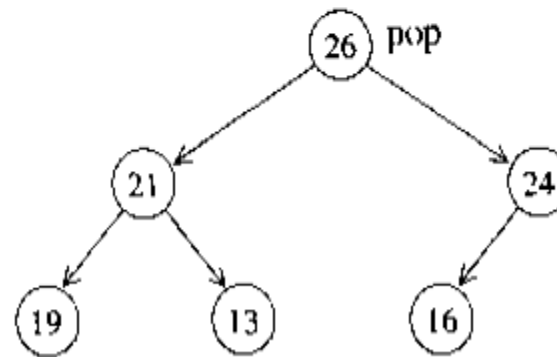
...	...	32	21	26	19	13	24	16	32	50	65	68	...
-----	-----	----	----	----	----	----	----	----	----	----	----	----	-----



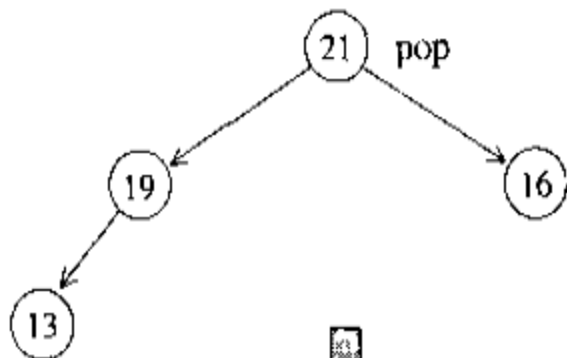
...	...	50	31	32	21	13	24	26	19	16	65	68	...
-----	-----	----	----	----	----	----	----	----	----	----	----	----	-----



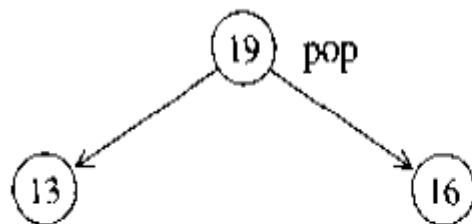
...	...	26	21	24	19	13	16	31	32	50	65	68	...
-----	-----	----	----	----	----	----	----	----	----	----	----	----	-----



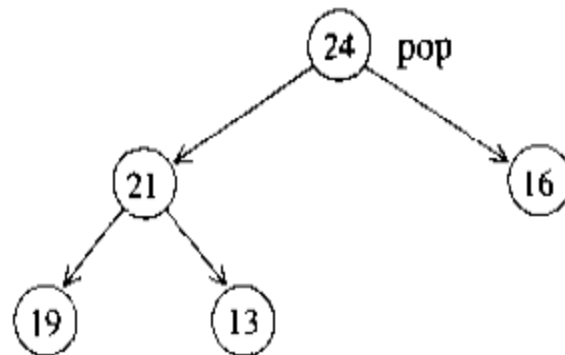
...	...	21	19	16	13	24	26	31	32	50	65	68	...
-----	-----	----	----	----	----	----	----	----	----	----	----	----	-----



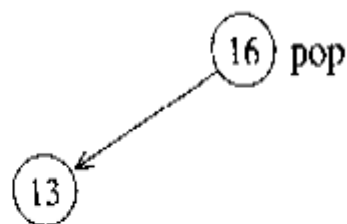
...	...	19	13	16	21	24	26	31	32	50	65	68	...
-----	-----	----	----	----	----	----	----	----	----	----	----	----	-----



...	...	24	21	16	19	13	26	31	32	50	65	68	...
-----	-----	----	----	----	----	----	----	----	----	----	----	----	-----



...	...	16	13	19	21	24	26	31	32	50	65	68	...
-----	-----	----	----	----	----	----	----	----	----	----	----	----	-----



...	...	13	16	19	21	24	26	31	32	50	65	68	...
-----	-----	----	----	----	----	----	----	----	----	----	----	----	-----

