玩儿转数据结构

liuyubobobo

堆和优先队列

优先队列基础

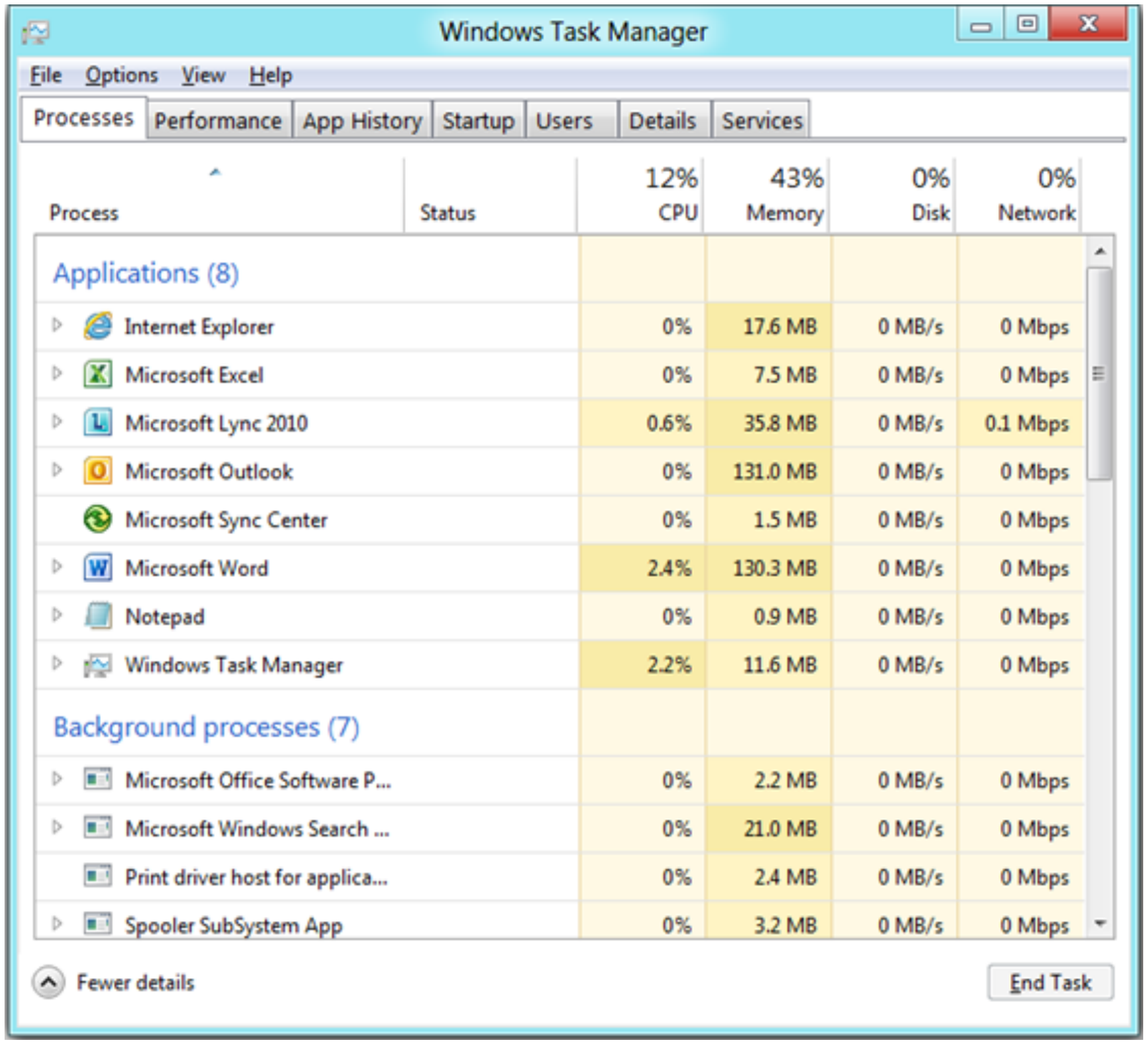
什么是优先队列?

普通队列：先进先出；后进后出

优先队列：出队顺序和入队顺序无关；和优先级相关

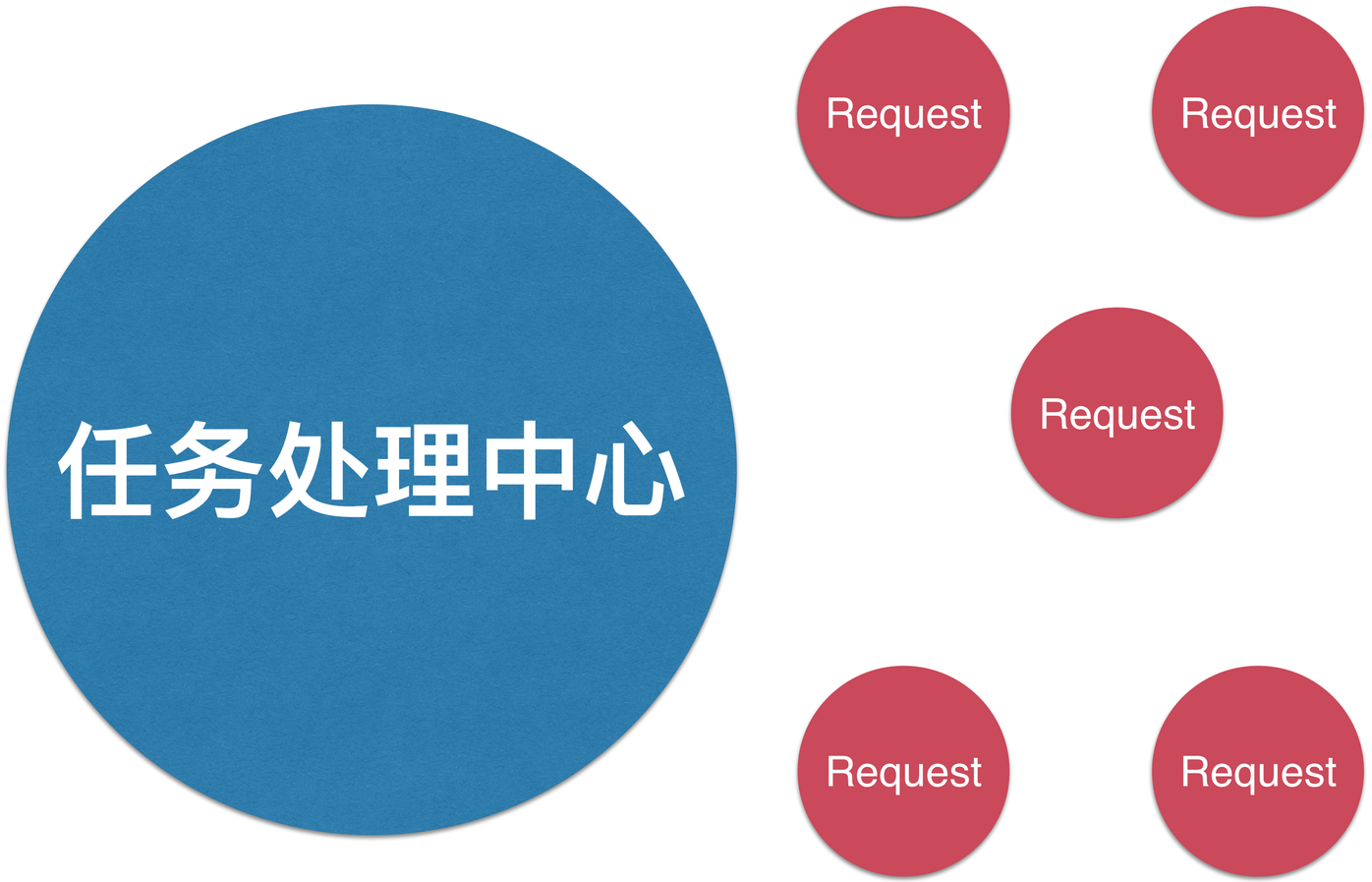
为什么使用优先队列?

动态选择优先级最高的任务执行

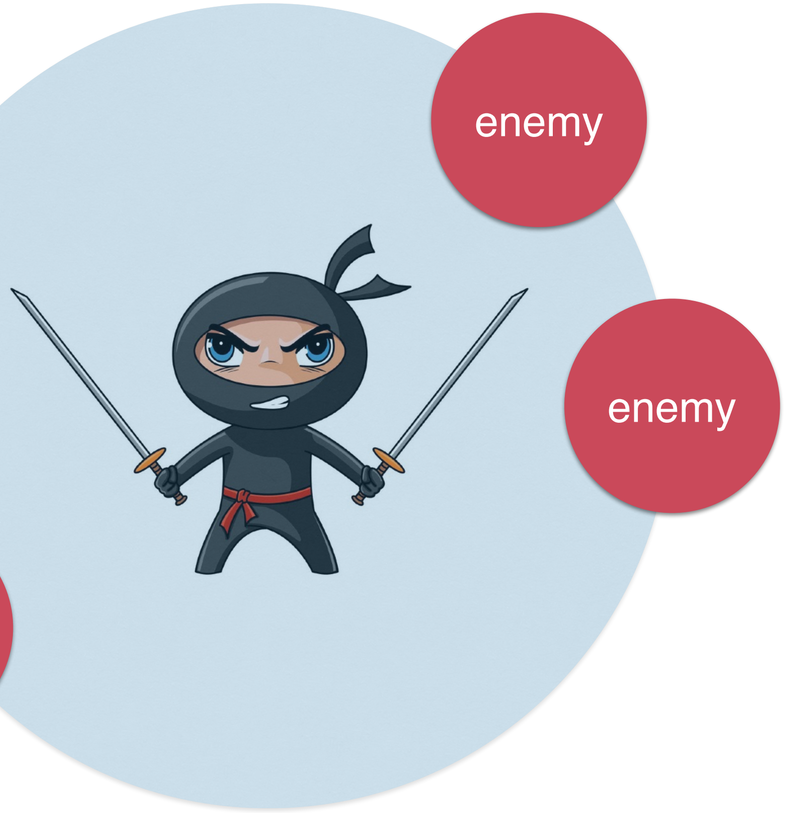


为什么使用优先队列?

关键词：动态



为什么使用优先队列?



enemy

enemy

enemy

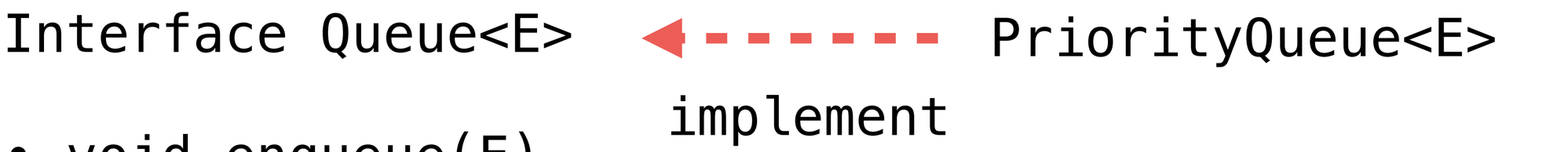
enemy

enemy

enemy

enemy

优先队列



- void enqueue(E)
- E dequeue()
- E getFront()
- int getSize()
- boolean isEmpty()

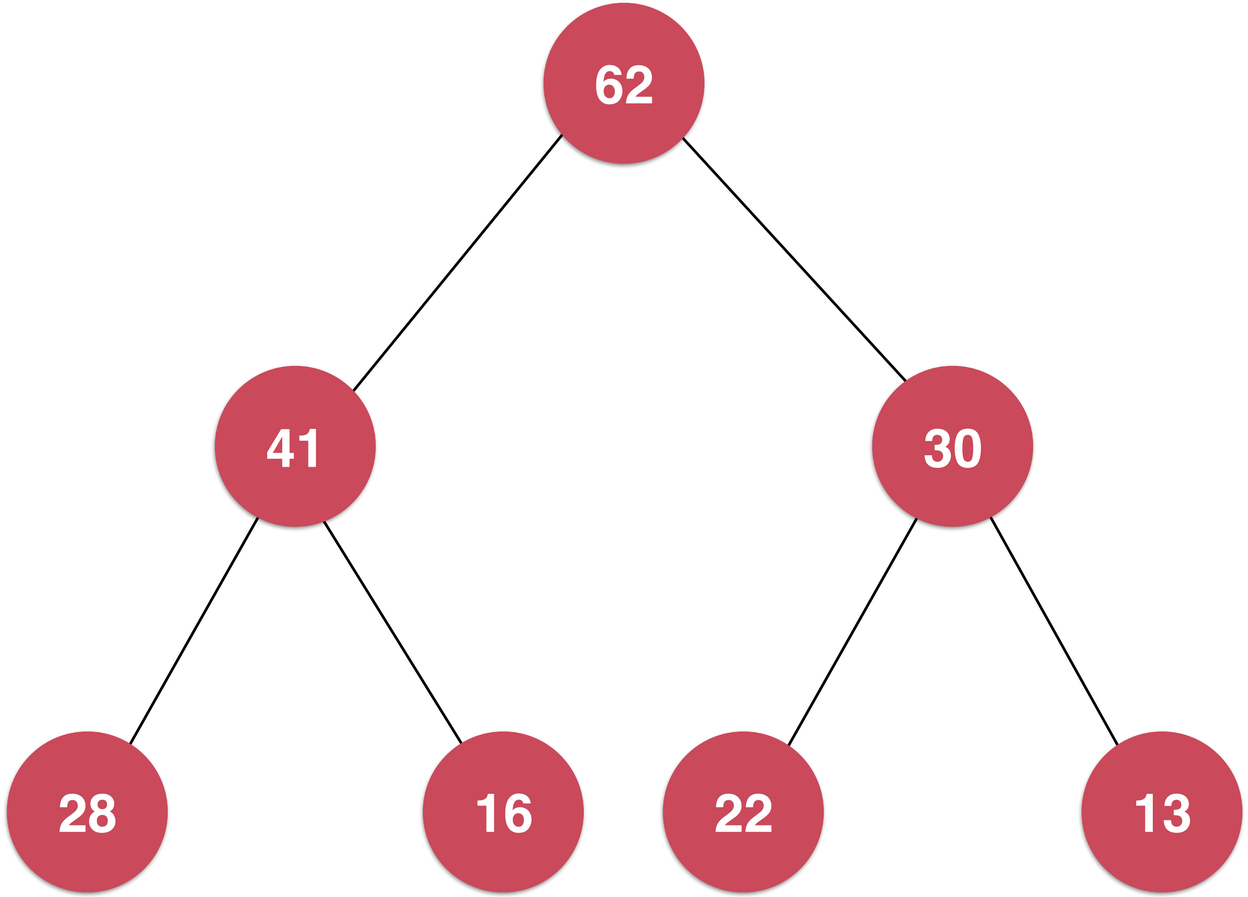
可以使用不同的底层实现

优先队列

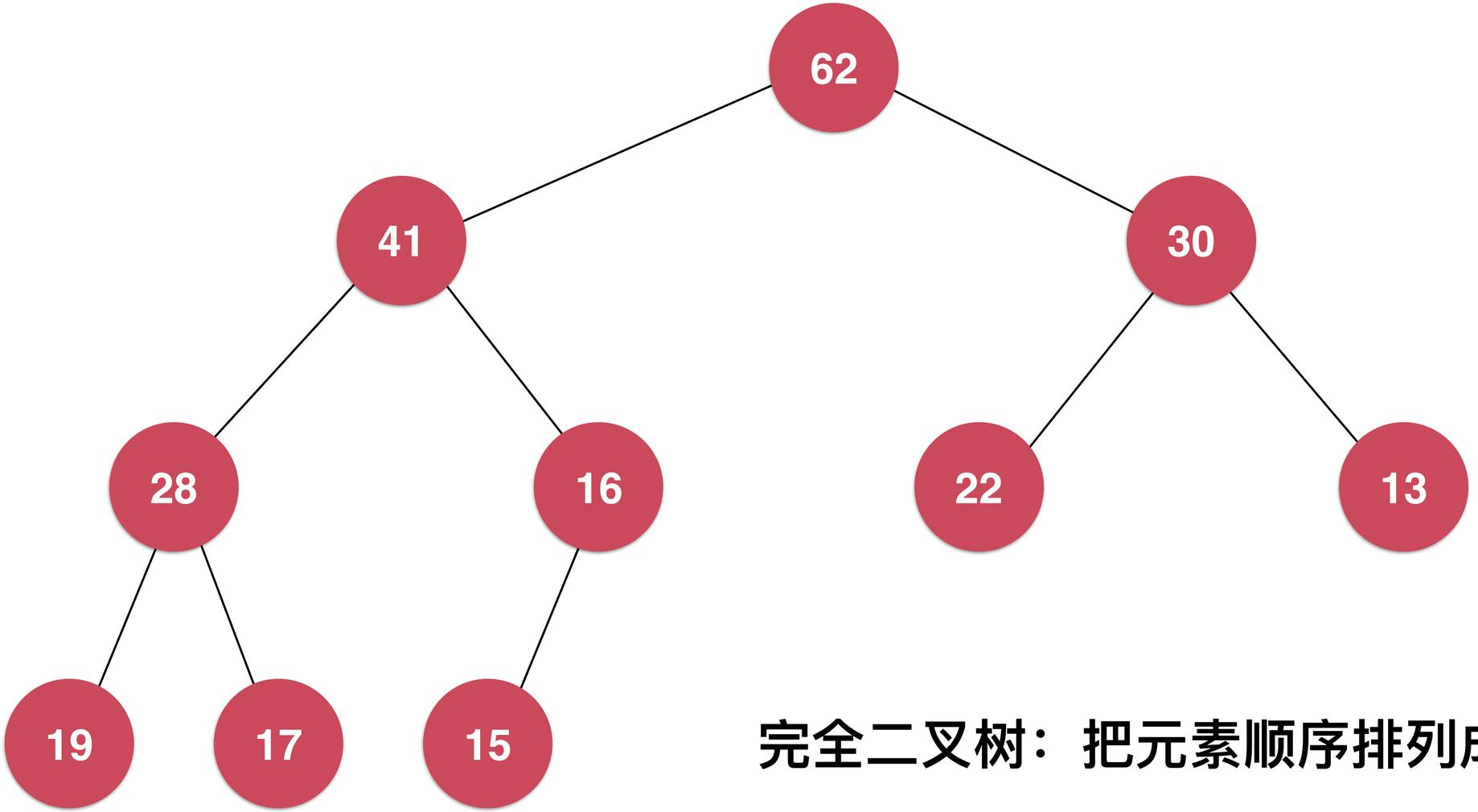
	入队	出队（拿出最大元素）
普通线性结构	O(1)	O(n)
顺序线性结构	O(n)	O(1)
堆	O(logn)	O(logn)

堆的基本结构

二叉堆 Binary Heap

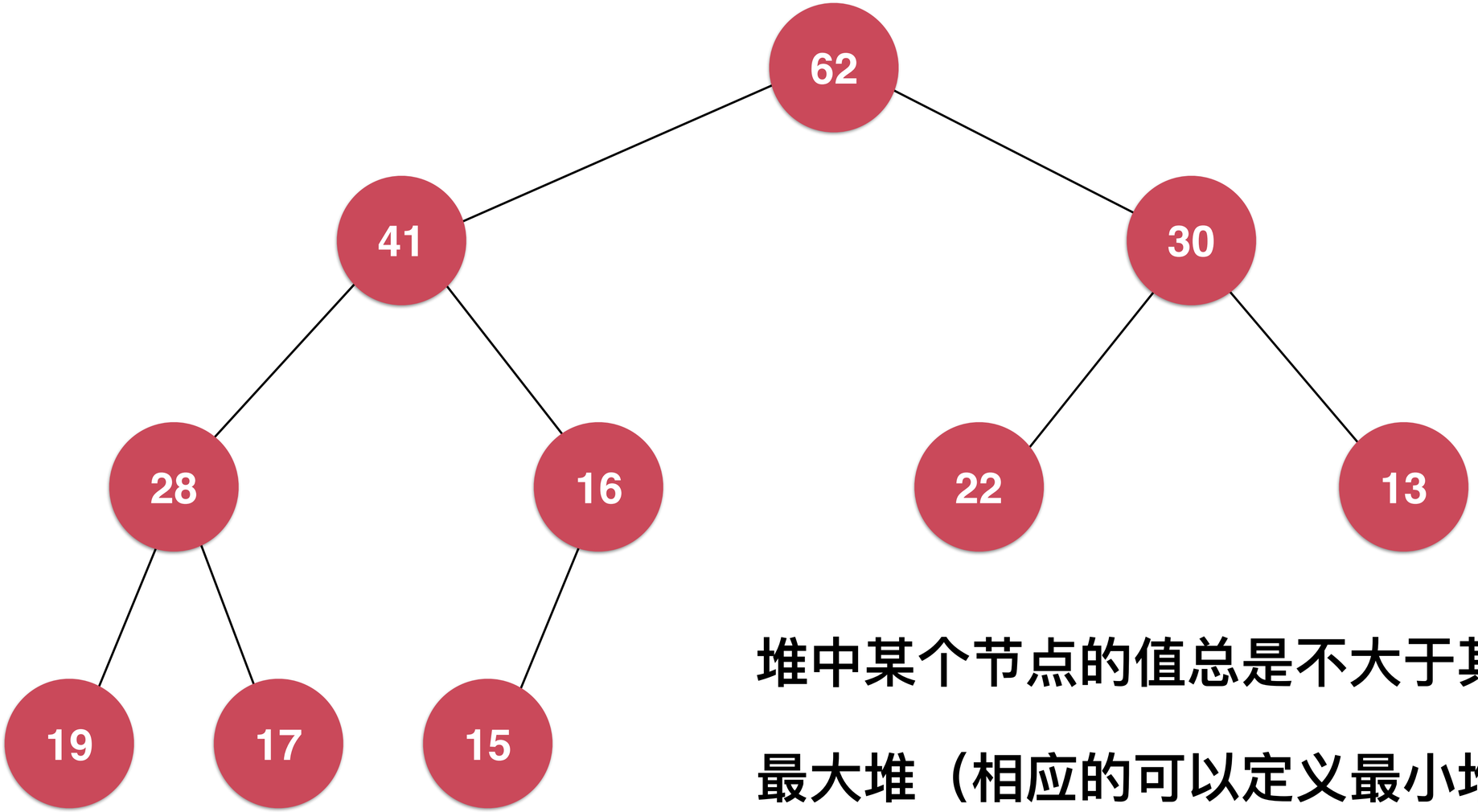


二叉堆是一棵完全二叉树



完全二叉树：把元素顺序排列成树的形状

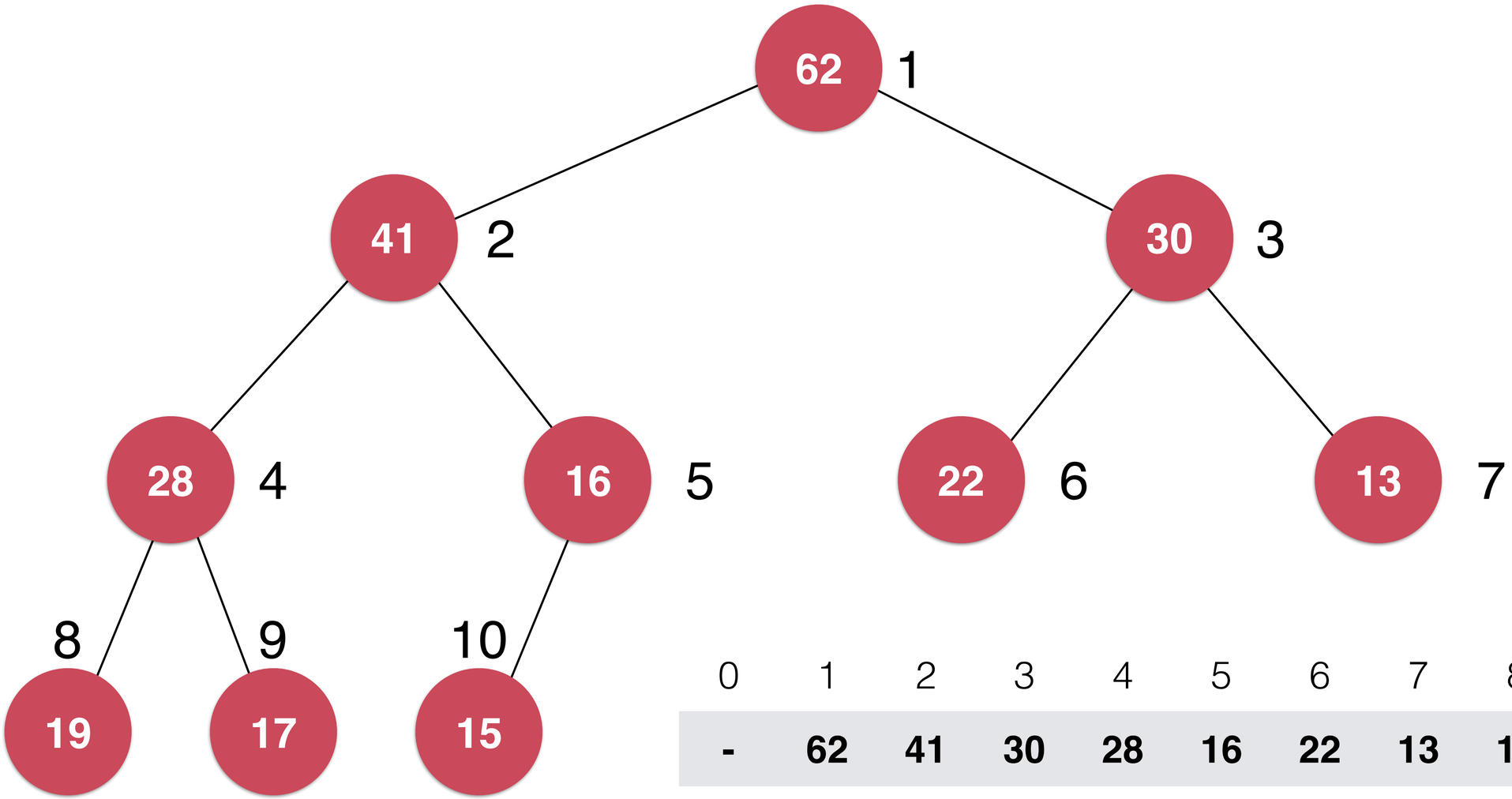
二叉堆的性质



堆中某个节点的值总是不大于其父节点的值

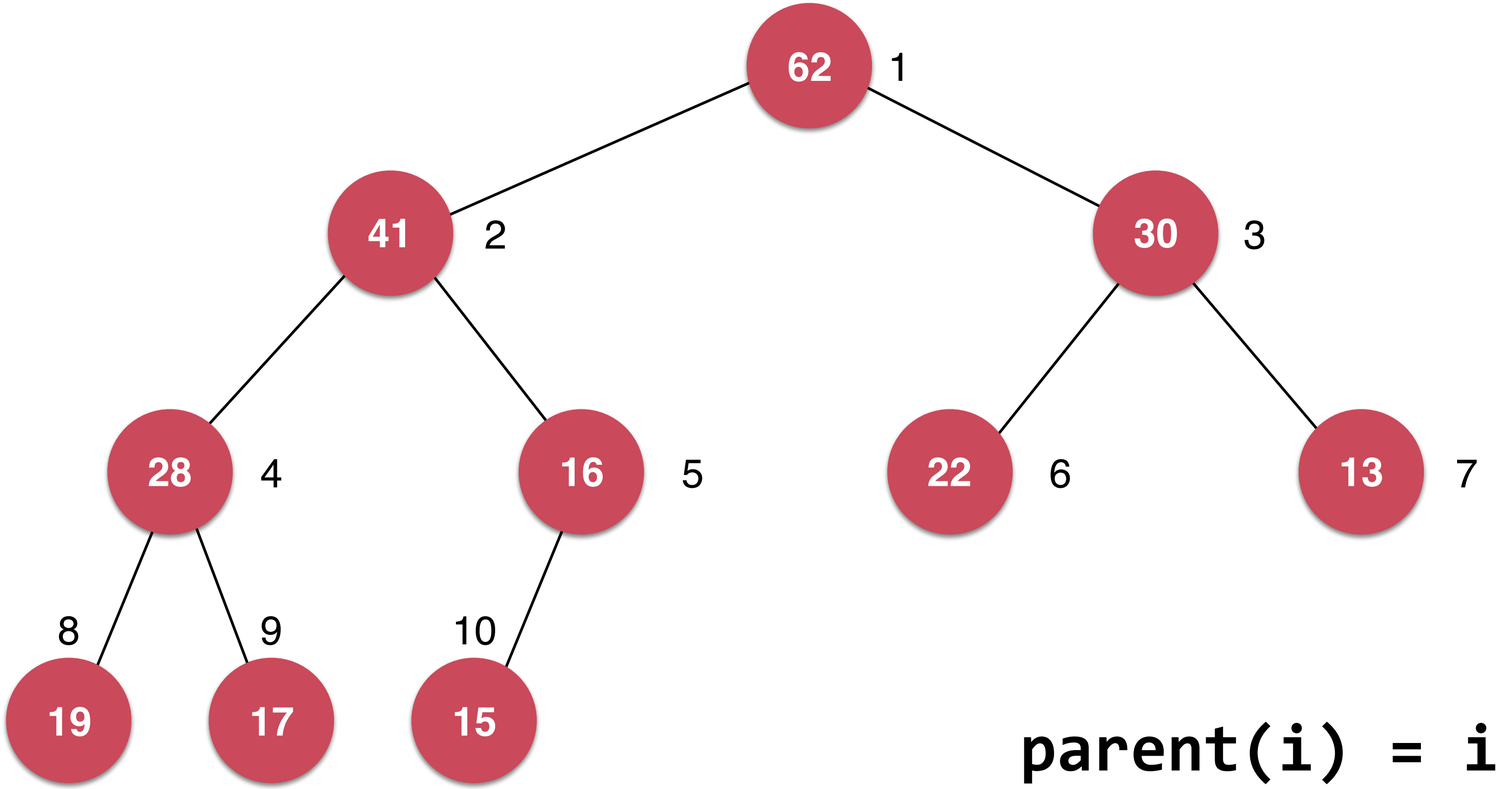
最大堆（相应的可以定义最小堆）

最大堆



0	1	2	3	4	5	6	7	8	9	10
-	62	41	30	28	16	22	13	19	17	15

用数组存储二叉堆

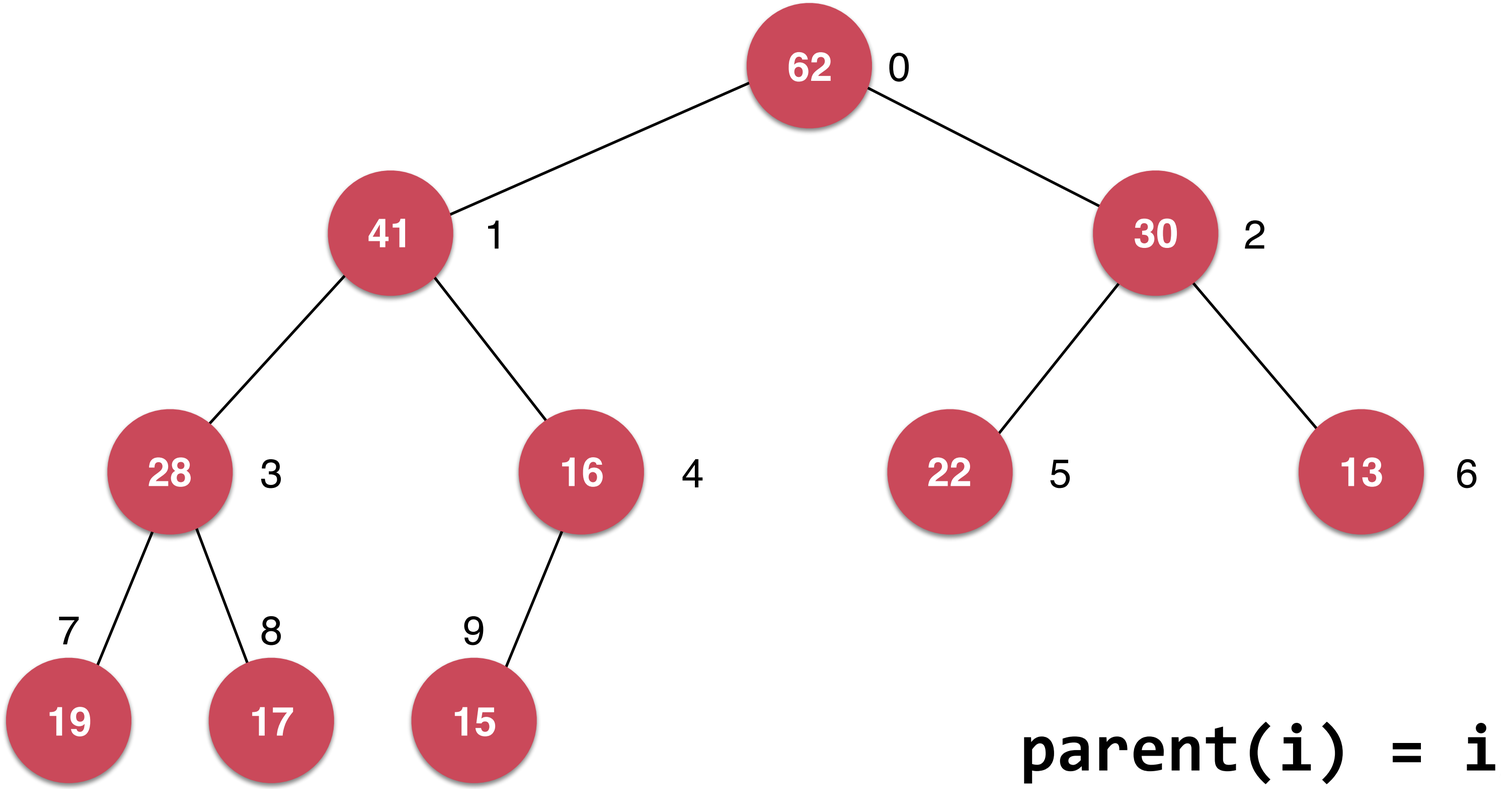


0	1	2	3	4	5	6	7	8	9	10
-	62	41	30	28	16	22	13	19	17	15

```
parent(i) = i/2

left child  (i) = 2*i
right child (i) = 2*i + 1
```

用数组存储二叉堆

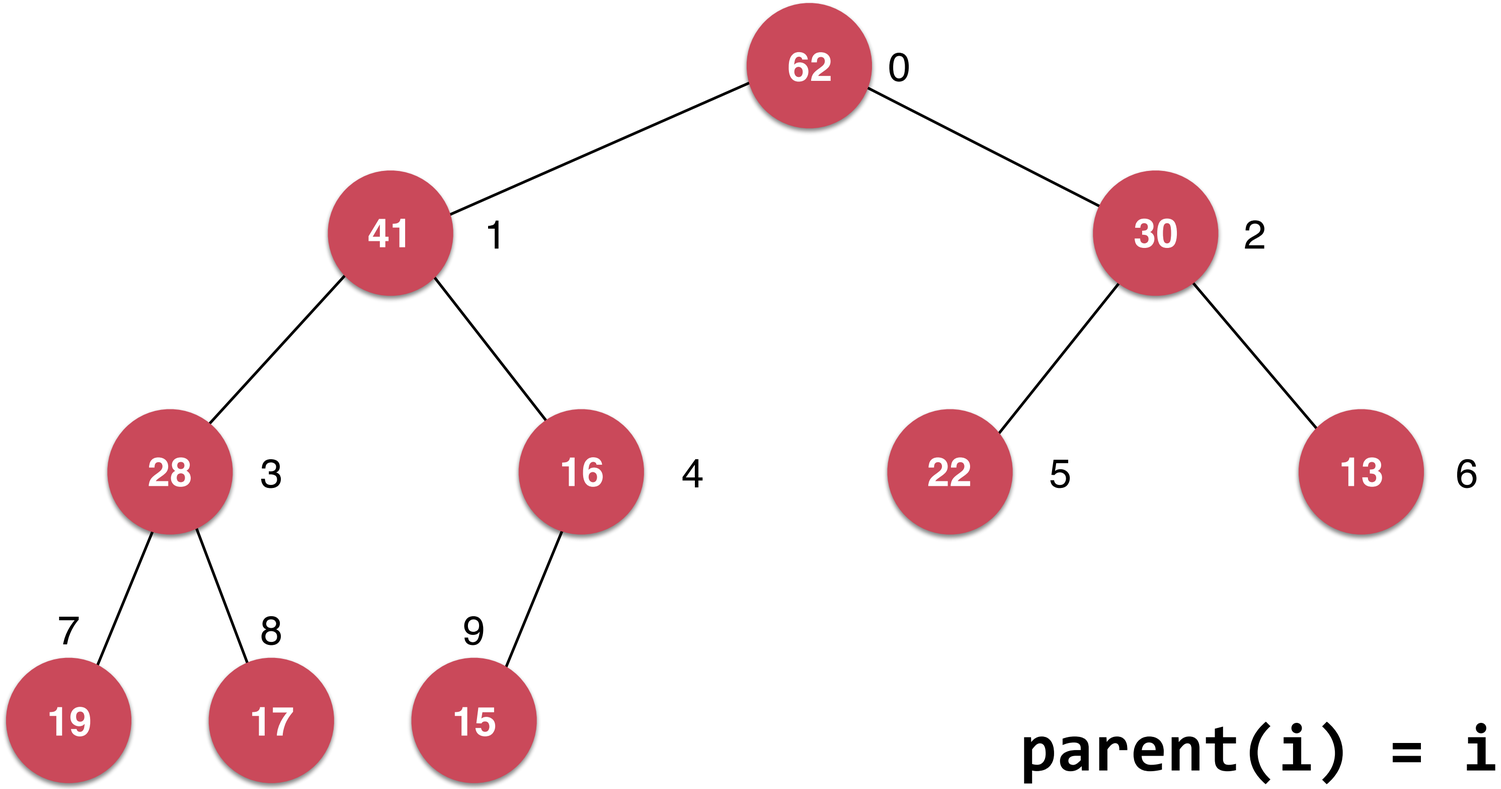


0	1	2	3	4	5	6	7	8	9
62	**41**	**30**	**28**	**16**	**22**	**13**	**19**	**17**	**15**

```
parent(i) = i/2

left child  (i) = 2*i
right child (i) = 2*i + 1
```

用数组存储二叉堆

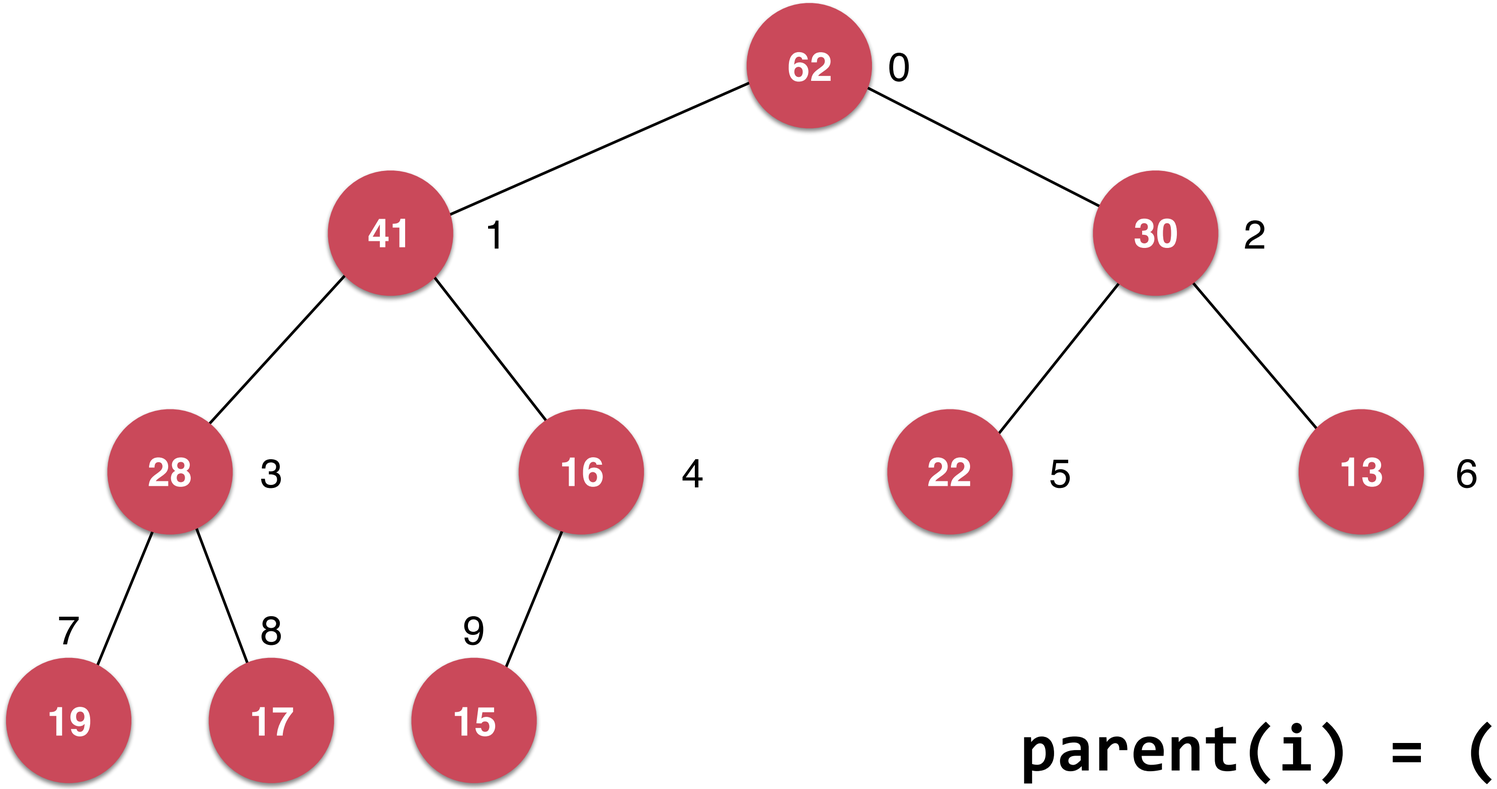


0	1	2	3	4	5	6	7	8	9
62	**41**	**30**	**28**	**16**	**22**	**13**	**19**	**17**	**15**

```
parent(i) = i/2

left child  (i) = 2*i
right child (i) = 2*i + 1
```

用数组存储二叉堆



0	1	2	3	4	5	6	7	8	9
62	**41**	**30**	**28**	**16**	**22**	**13**	**19**	**17**	**15**

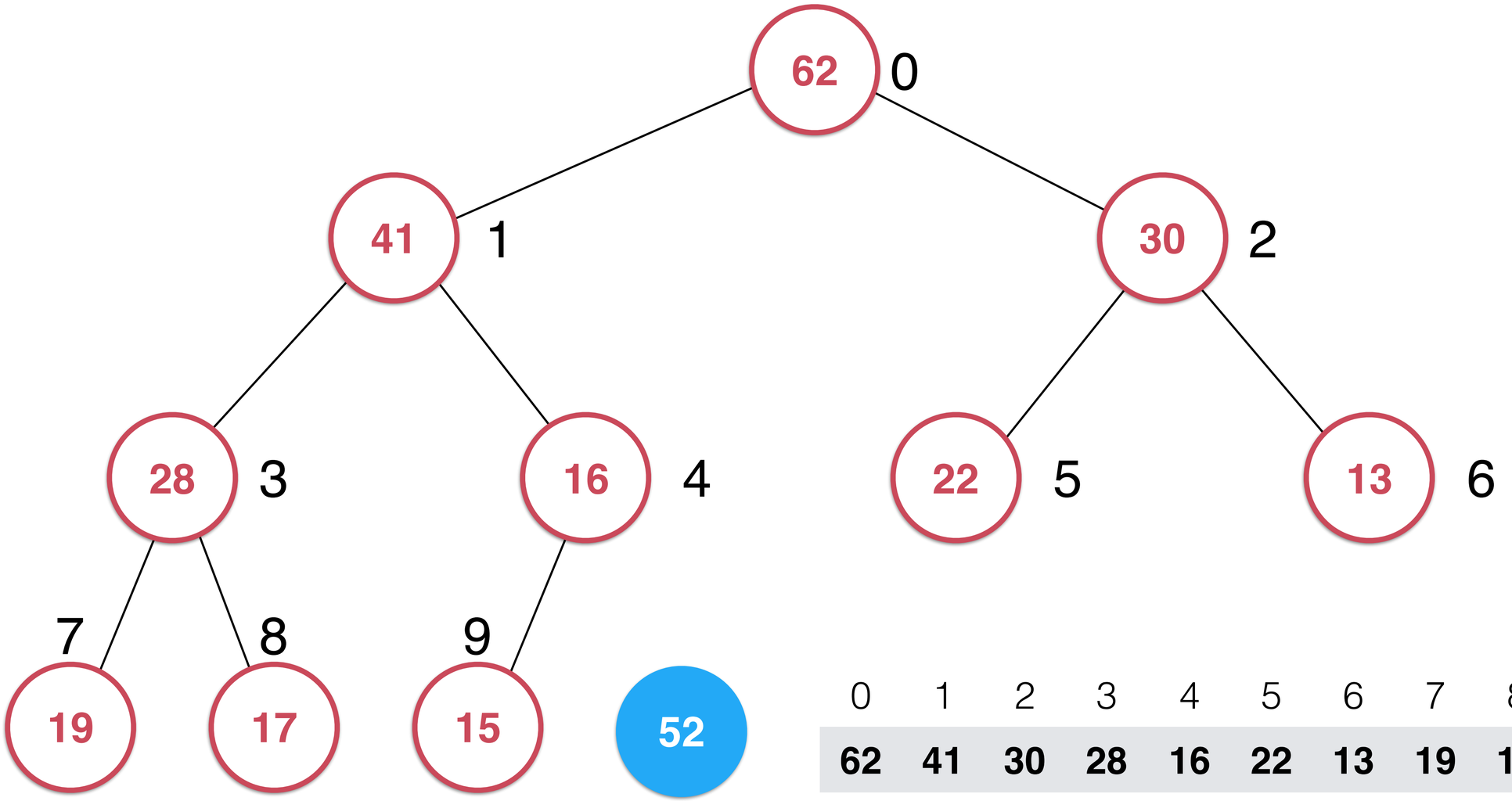
```
parent(i) = (i - 1)/2

left child  (i) = 2*i + 1
right child (i) = 2*i + 2
```

实践：堆的基本框架

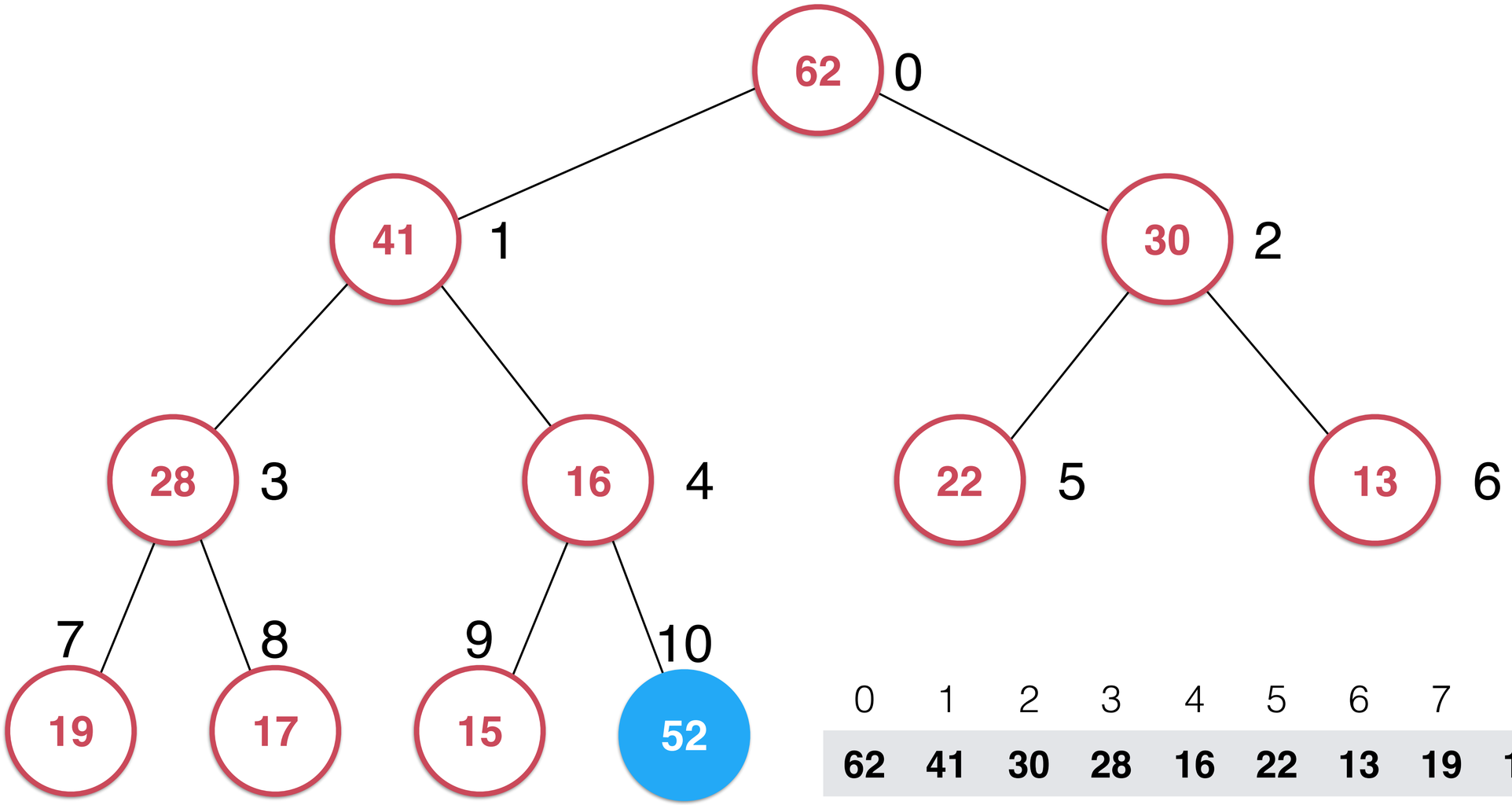
向堆中添加元素和Sift Up

Sift Up



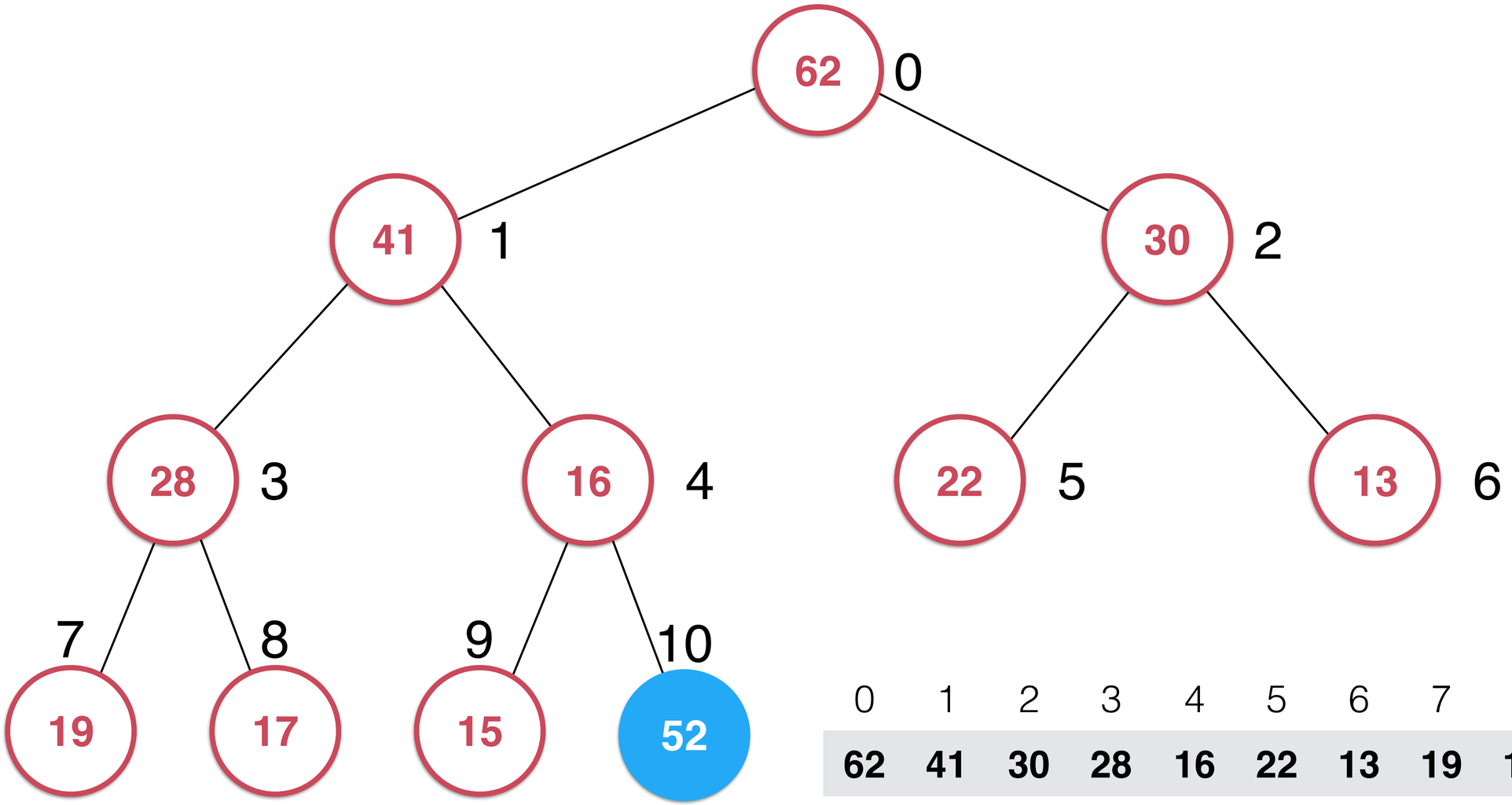
0	1	2	3	4	5	6	7	8	9	10
62	**41**	**30**	**28**	**16**	**22**	**13**	**19**	**17**	**15**	**-**

Sift Up



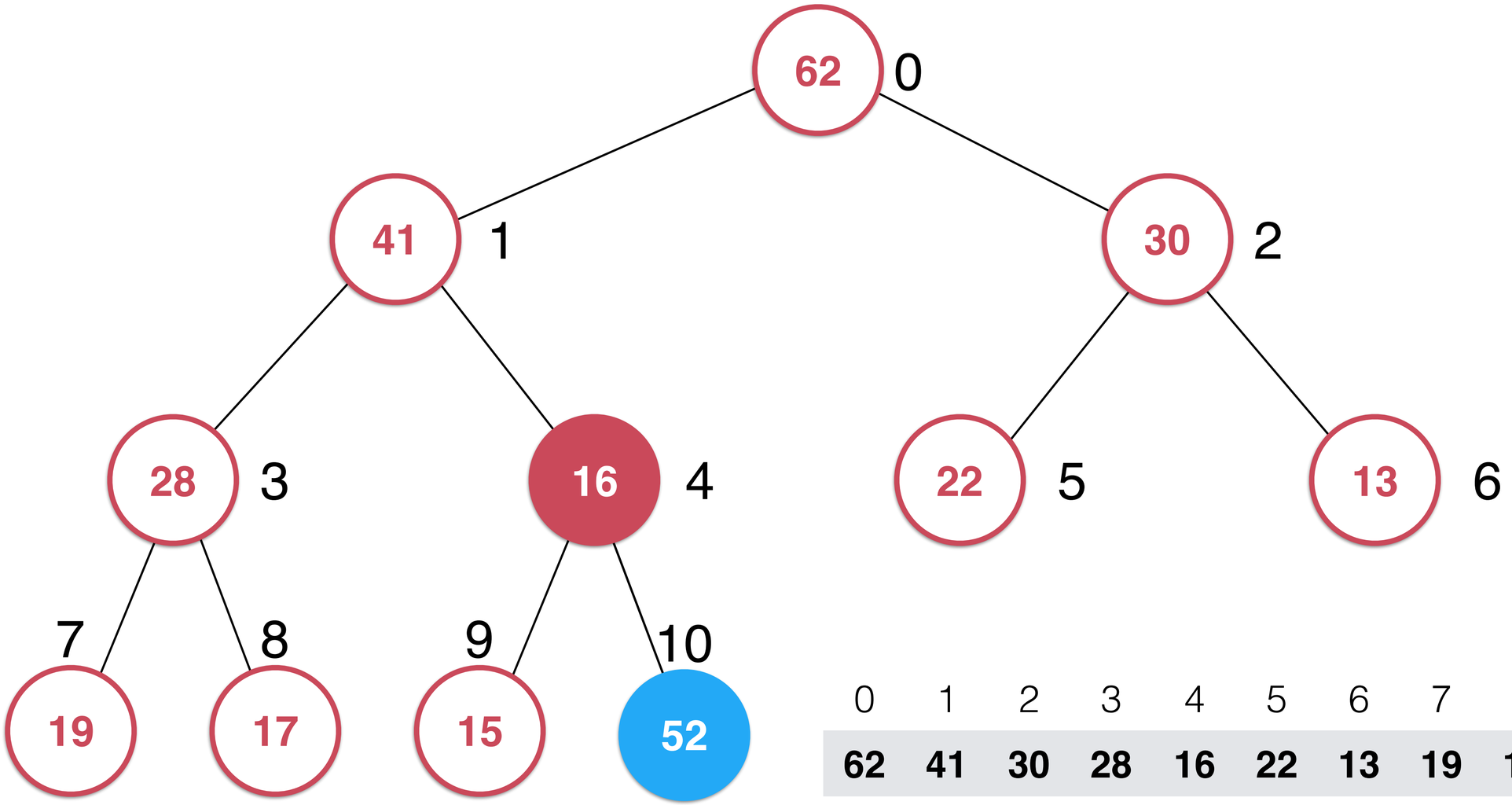
0	1	2	3	4	5	6	7	8	9	10
62	**41**	**30**	**28**	**16**	**22**	**13**	**19**	**17**	**15**	**-**

Sift Up



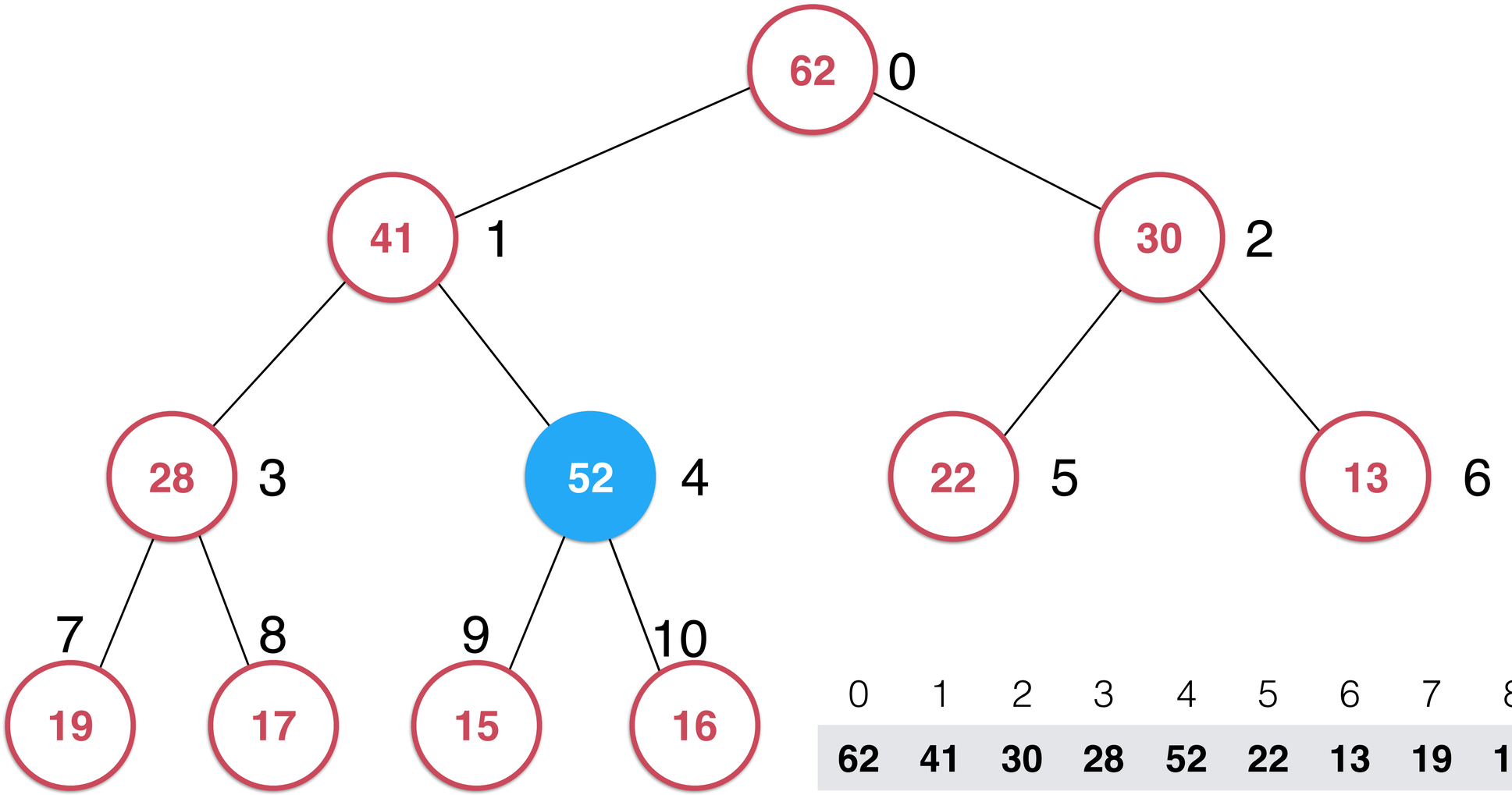
0	1	2	3	4	5	6	7	8	9	10
62	**41**	**30**	**28**	**16**	**22**	**13**	**19**	**17**	**15**	**52**

Sift Up



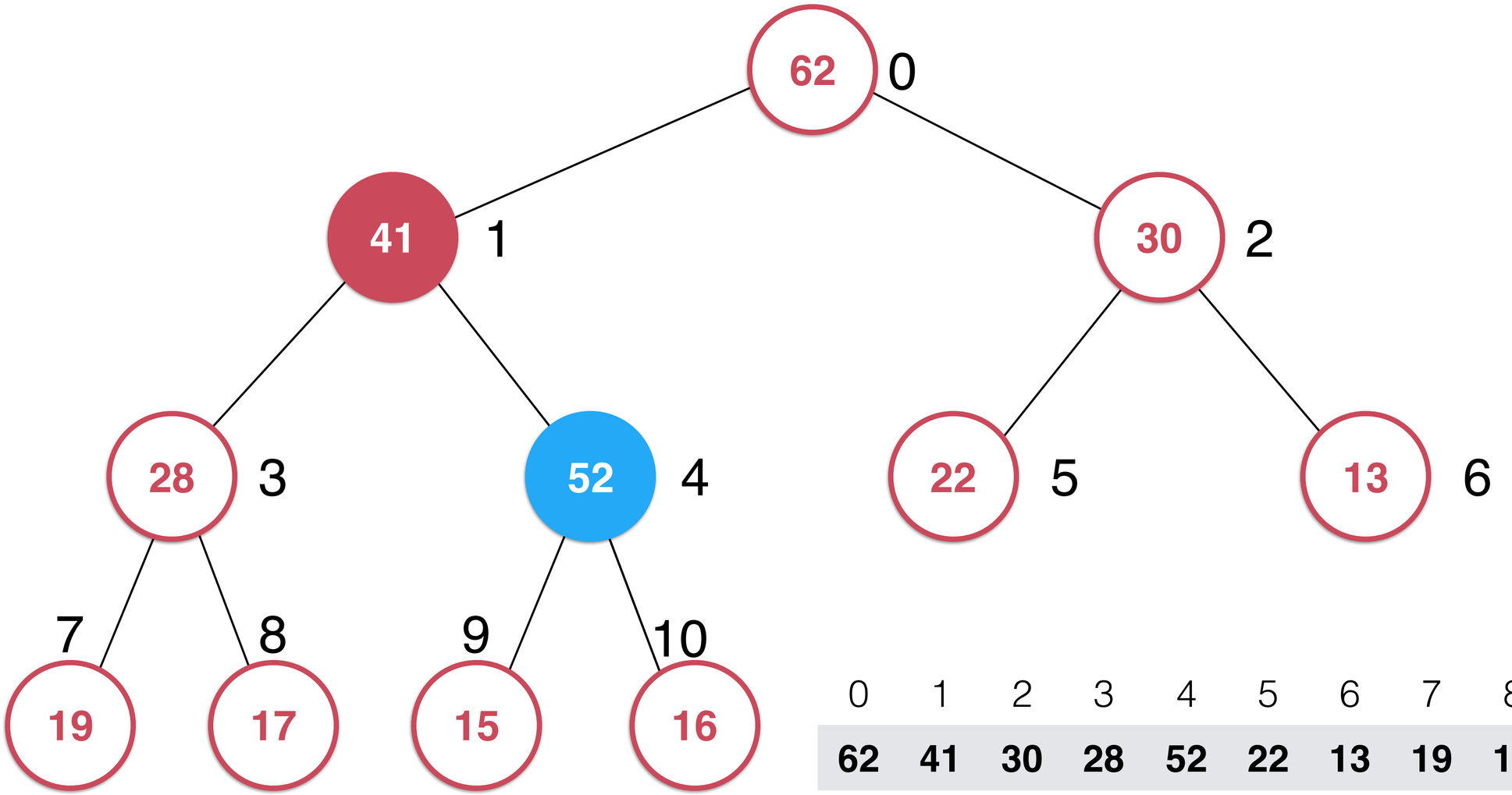
0	1	2	3	4	5	6	7	8	9	10
62	**41**	**30**	**28**	**16**	**22**	**13**	**19**	**17**	**15**	**52**

Sift Up



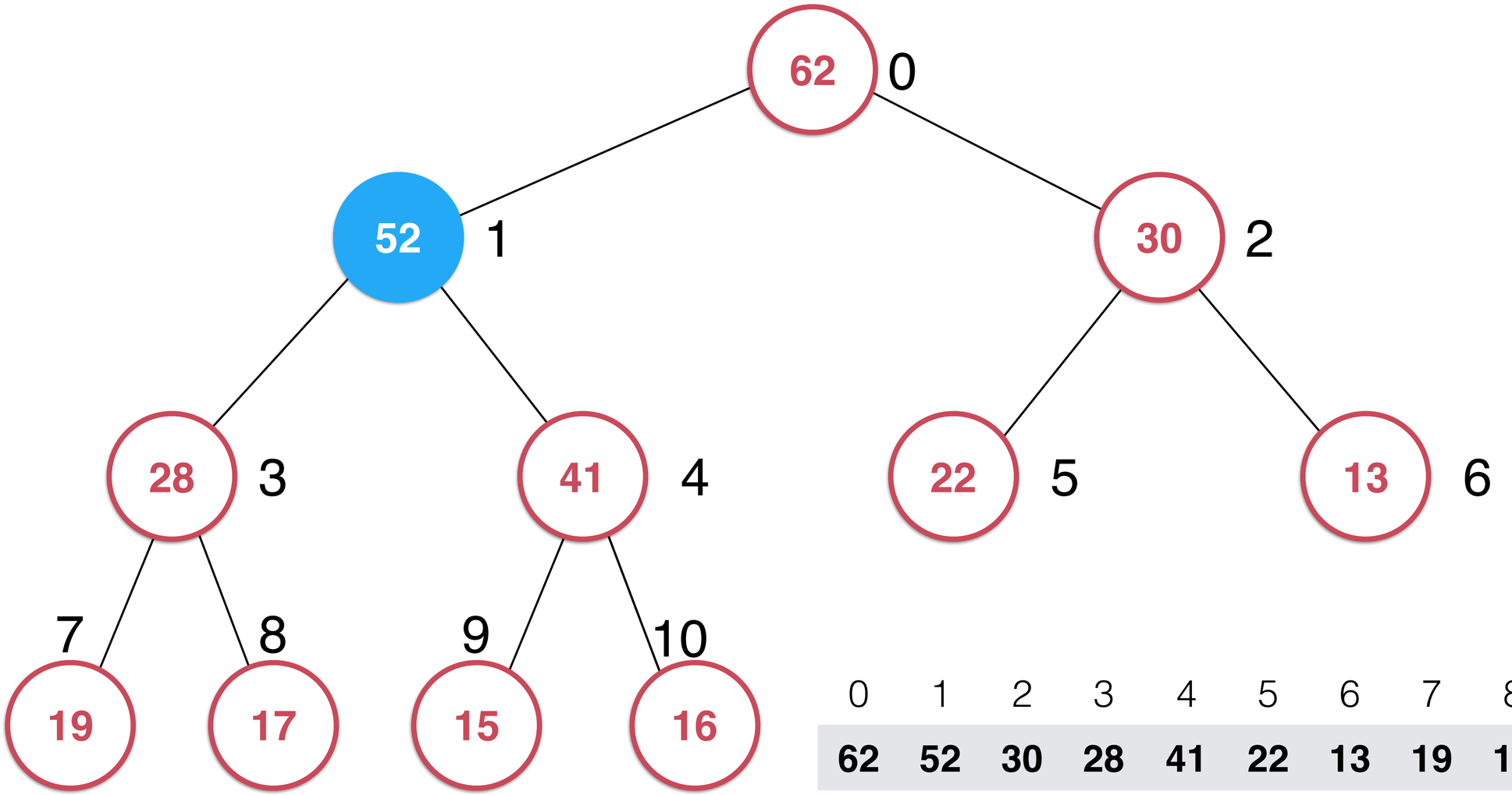
0	1	2	3	4	5	6	7	8	9	10
62	**41**	**30**	**28**	**52**	**22**	**13**	**19**	**17**	**15**	**16**

Sift Up



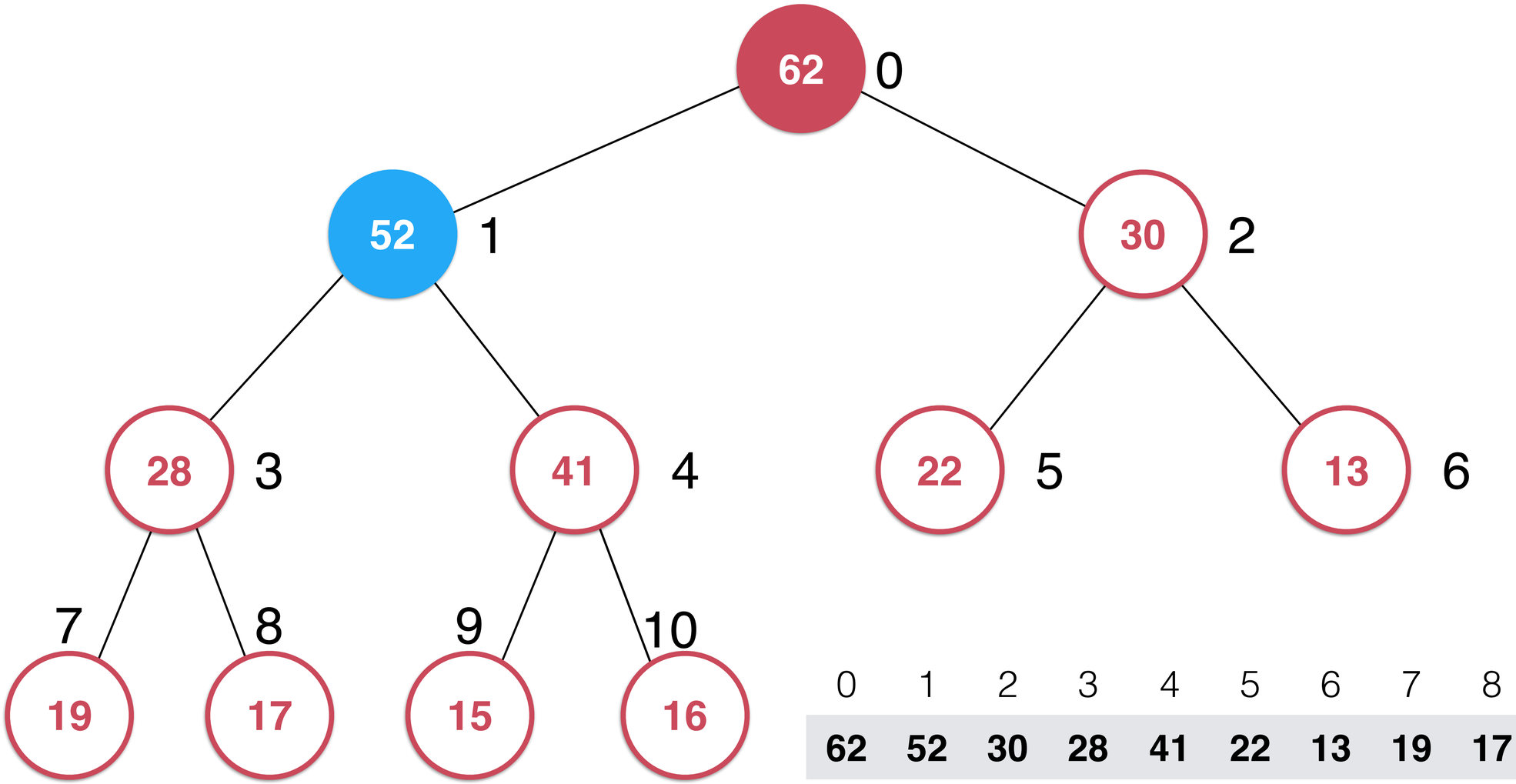
0	1	2	3	4	5	6	7	8	9	10
62	**41**	**30**	**28**	**52**	**22**	**13**	**19**	**17**	**15**	**16**

Sift Up



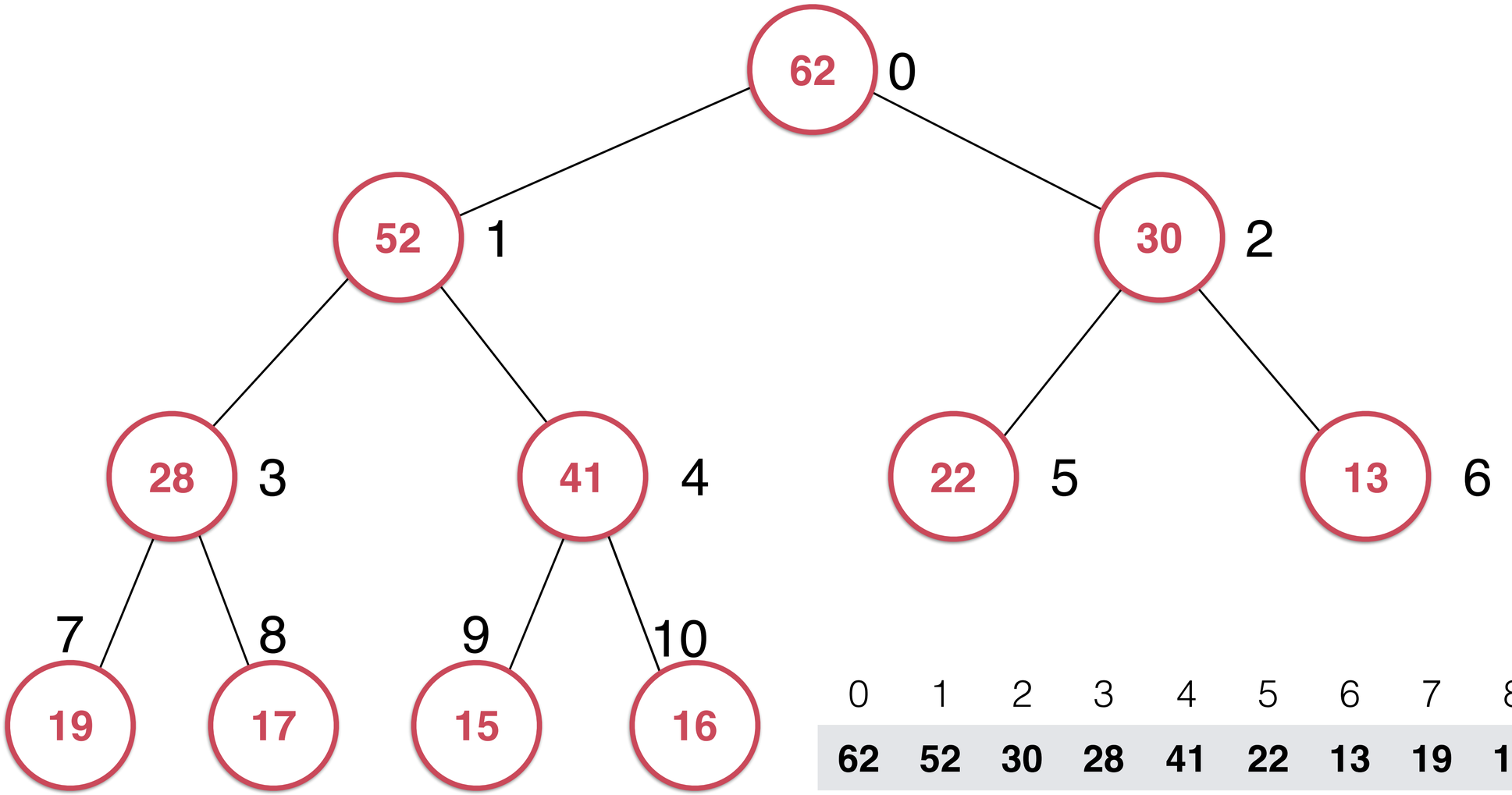
0	1	2	3	4	5	6	7	8	9	10
62	**52**	**30**	**28**	**41**	**22**	**13**	**19**	**17**	**15**	**16**

Sift Up



0	1	2	3	4	5	6	7	8	9	10
62	**52**	**30**	**28**	**41**	**22**	**13**	**19**	**17**	**15**	**16**

Sift Up

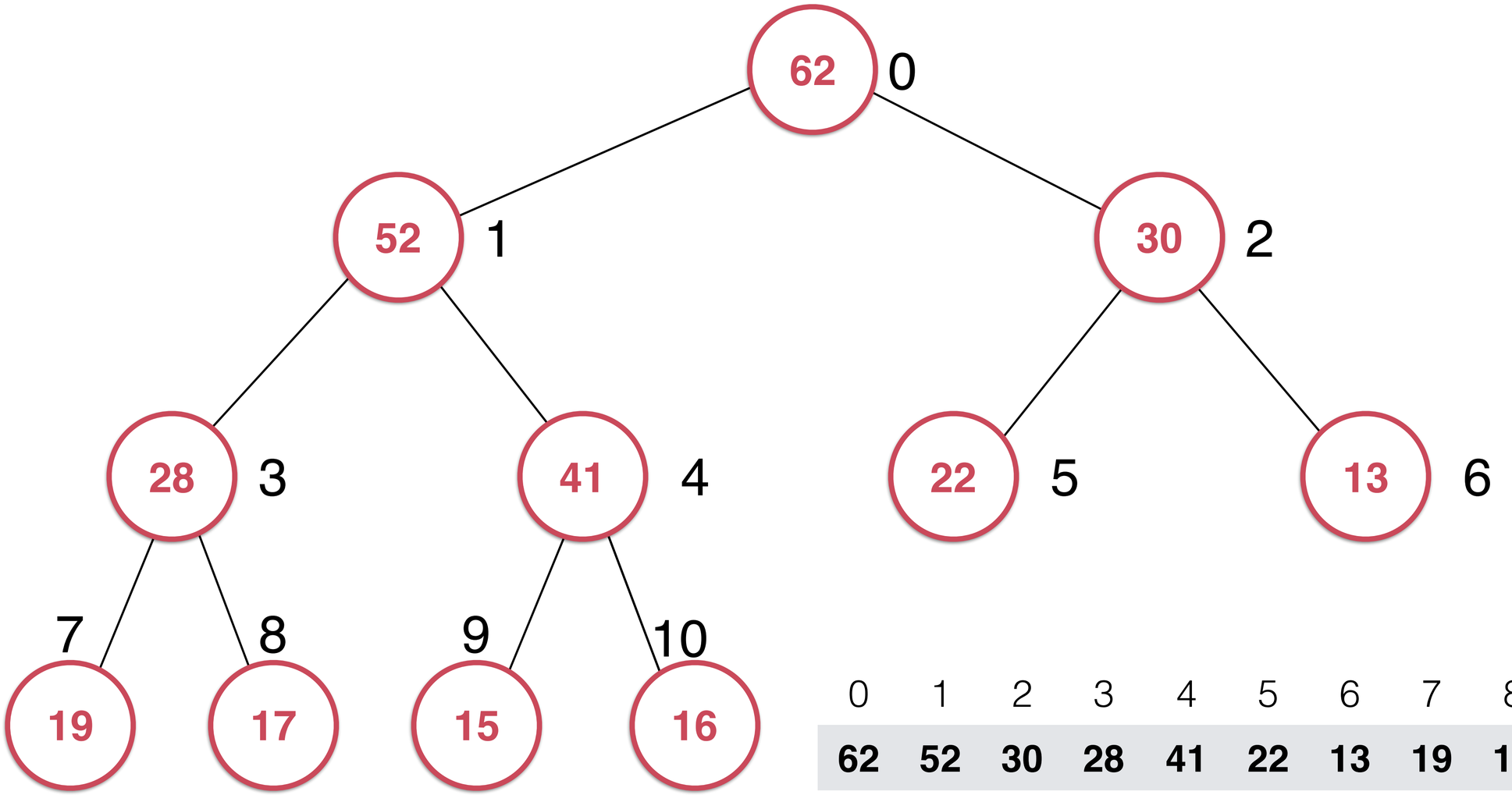


0	1	2	3	4	5	6	7	8	9	10
62	**52**	**30**	**28**	**41**	**22**	**13**	**19**	**17**	**15**	**16**

实践： Sift Up 和 add

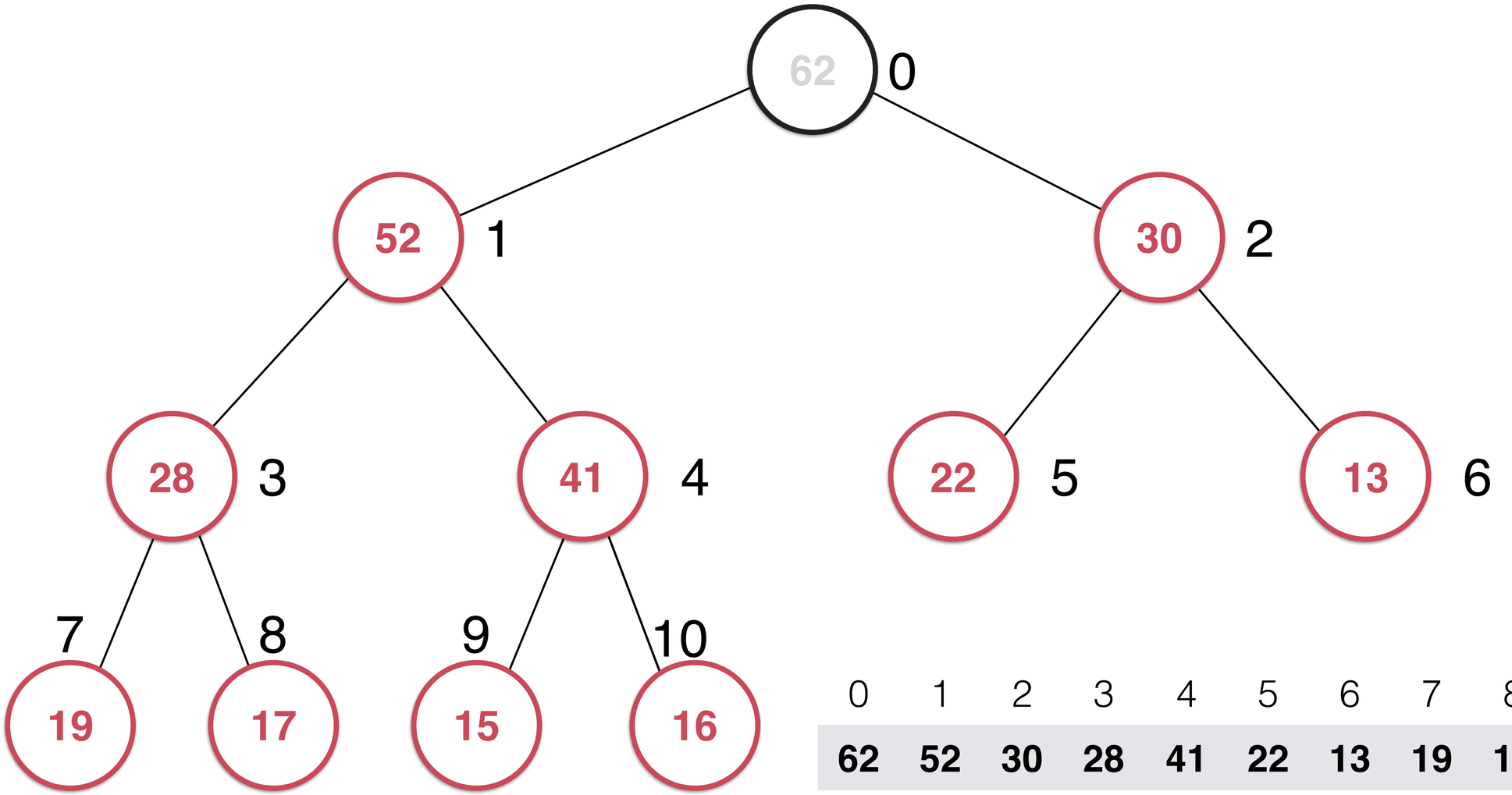
取出堆中的最大元素和Sift Down

Sift Down



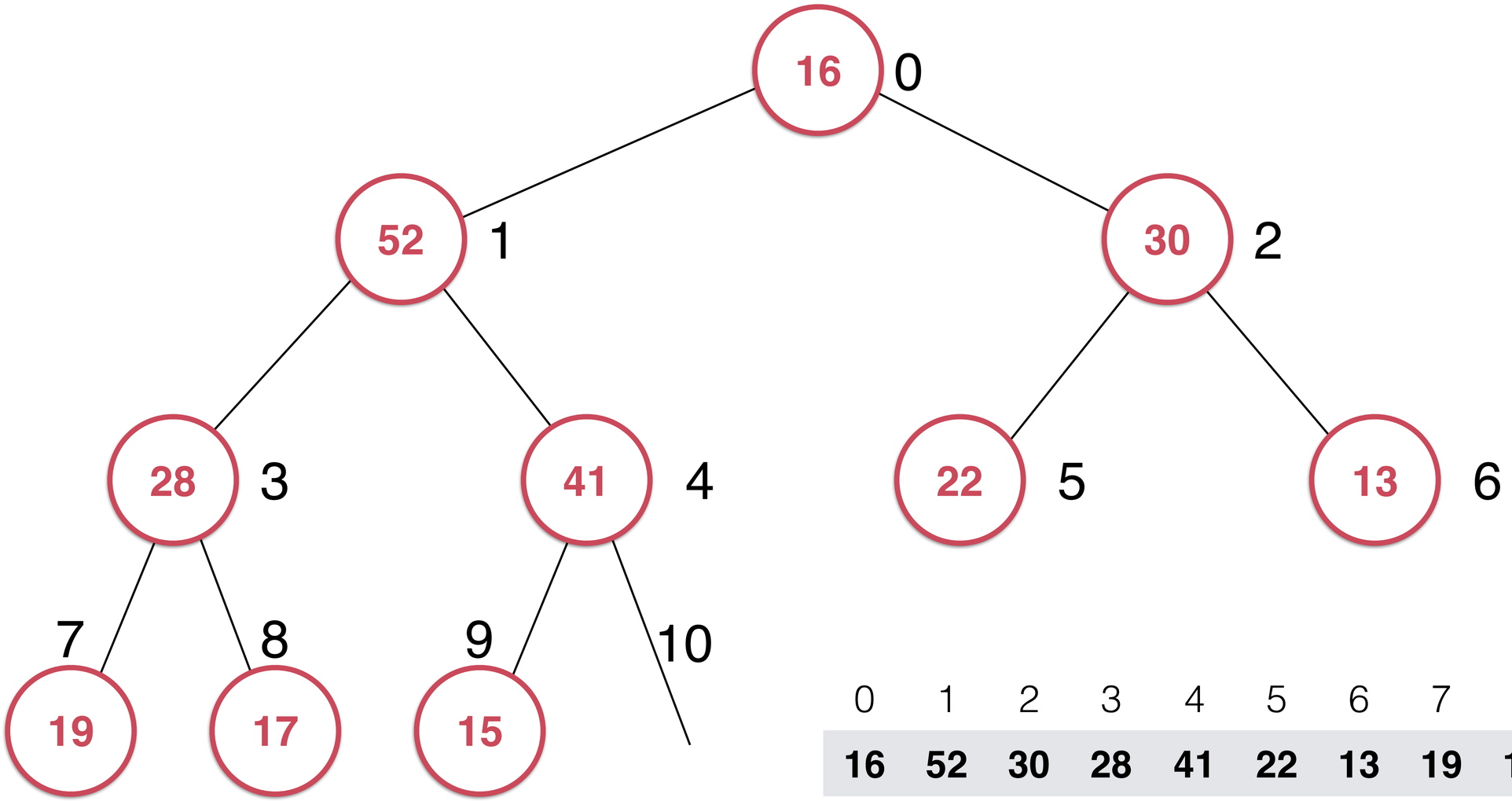
0	1	2	3	4	5	6	7	8	9	10
62	**52**	**30**	**28**	**41**	**22**	**13**	**19**	**17**	**15**	**16**

Sift Down



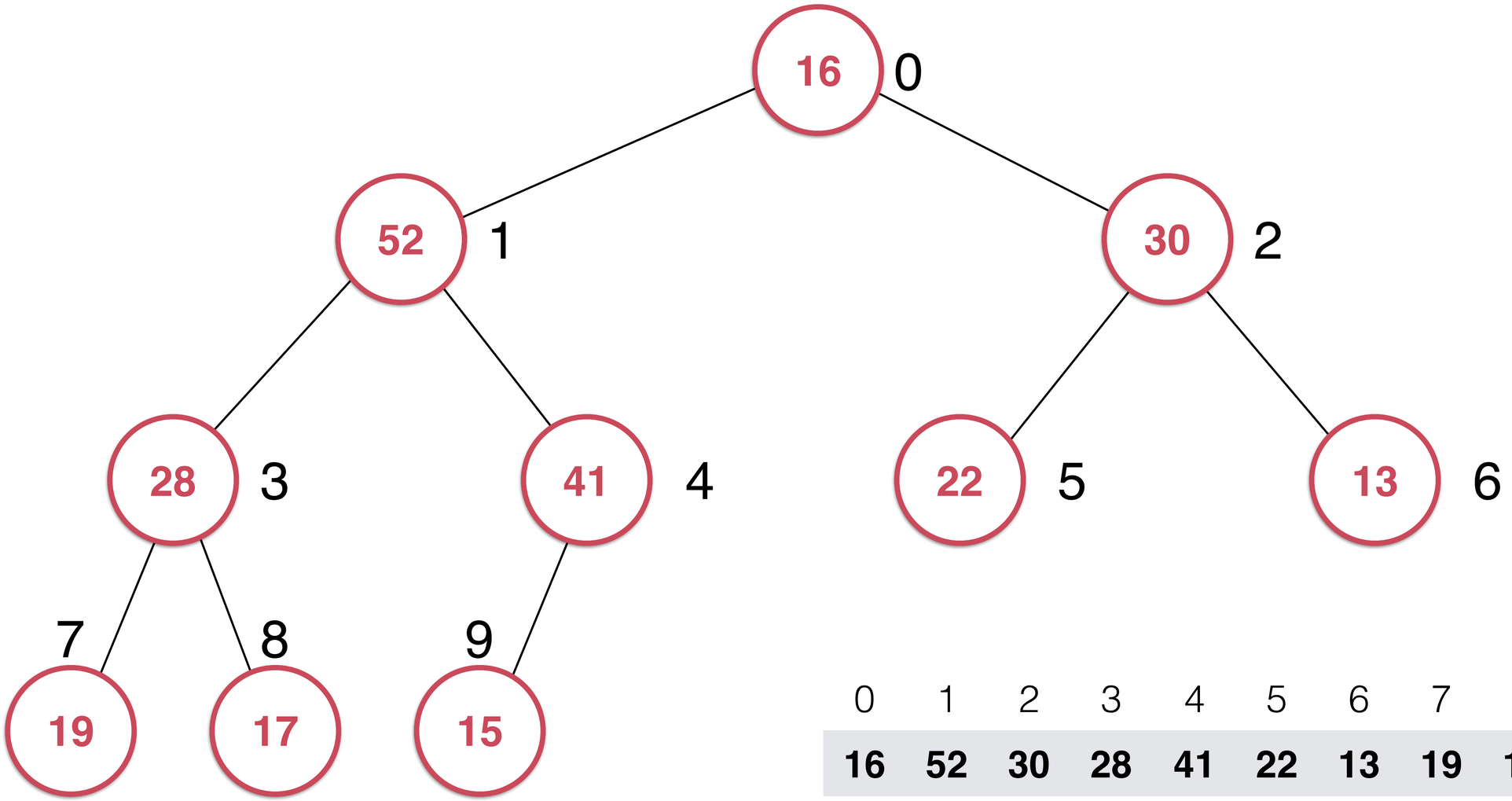
0	1	2	3	4	5	6	7	8	9	10
62	**52**	**30**	**28**	**41**	**22**	**13**	**19**	**17**	**15**	**16**

Sift Down



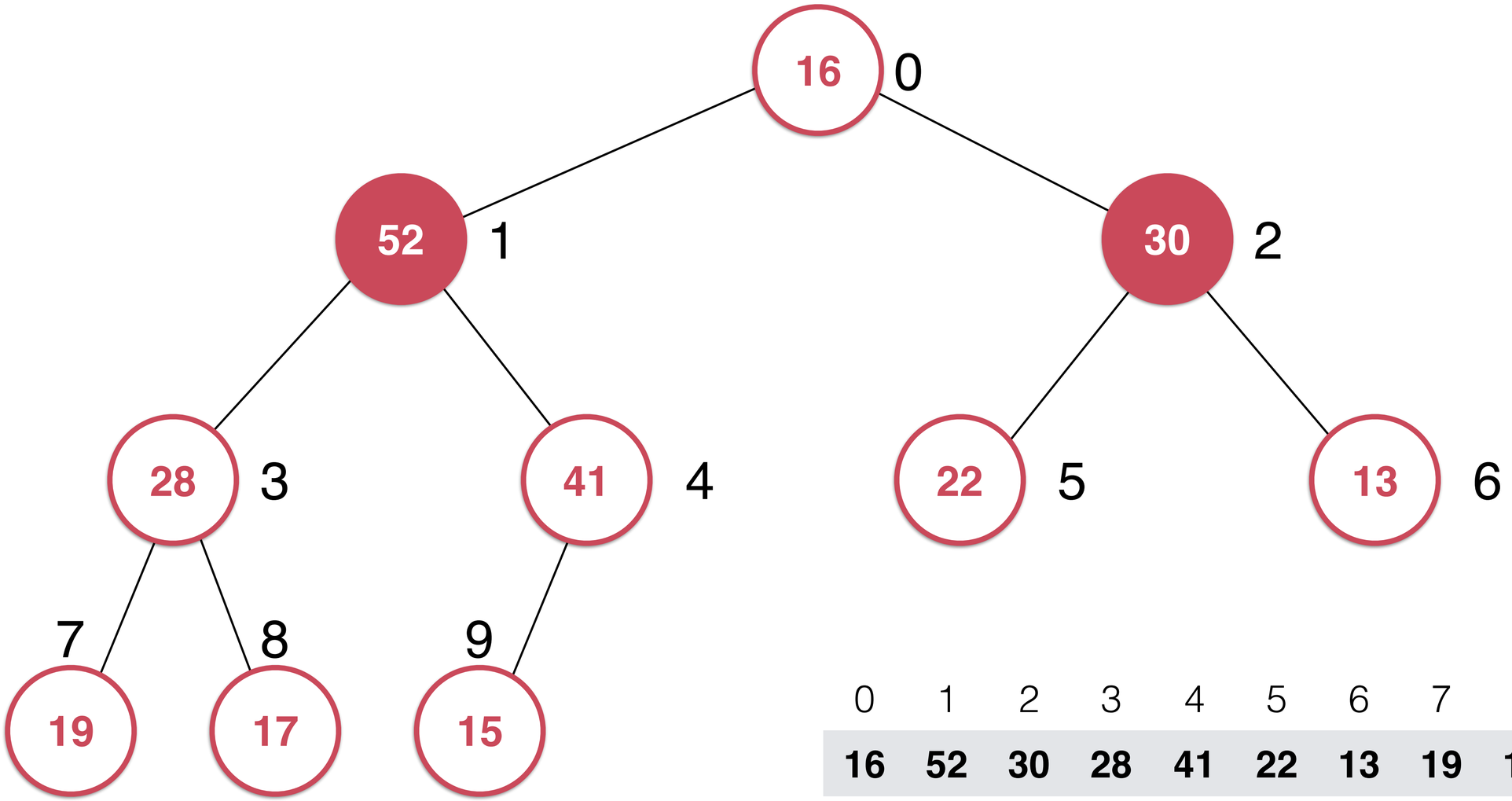
0	1	2	3	4	5	6	7	8	9	10
16	**52**	**30**	**28**	**41**	**22**	**13**	**19**	**17**	**15**	**16**

Sift Down



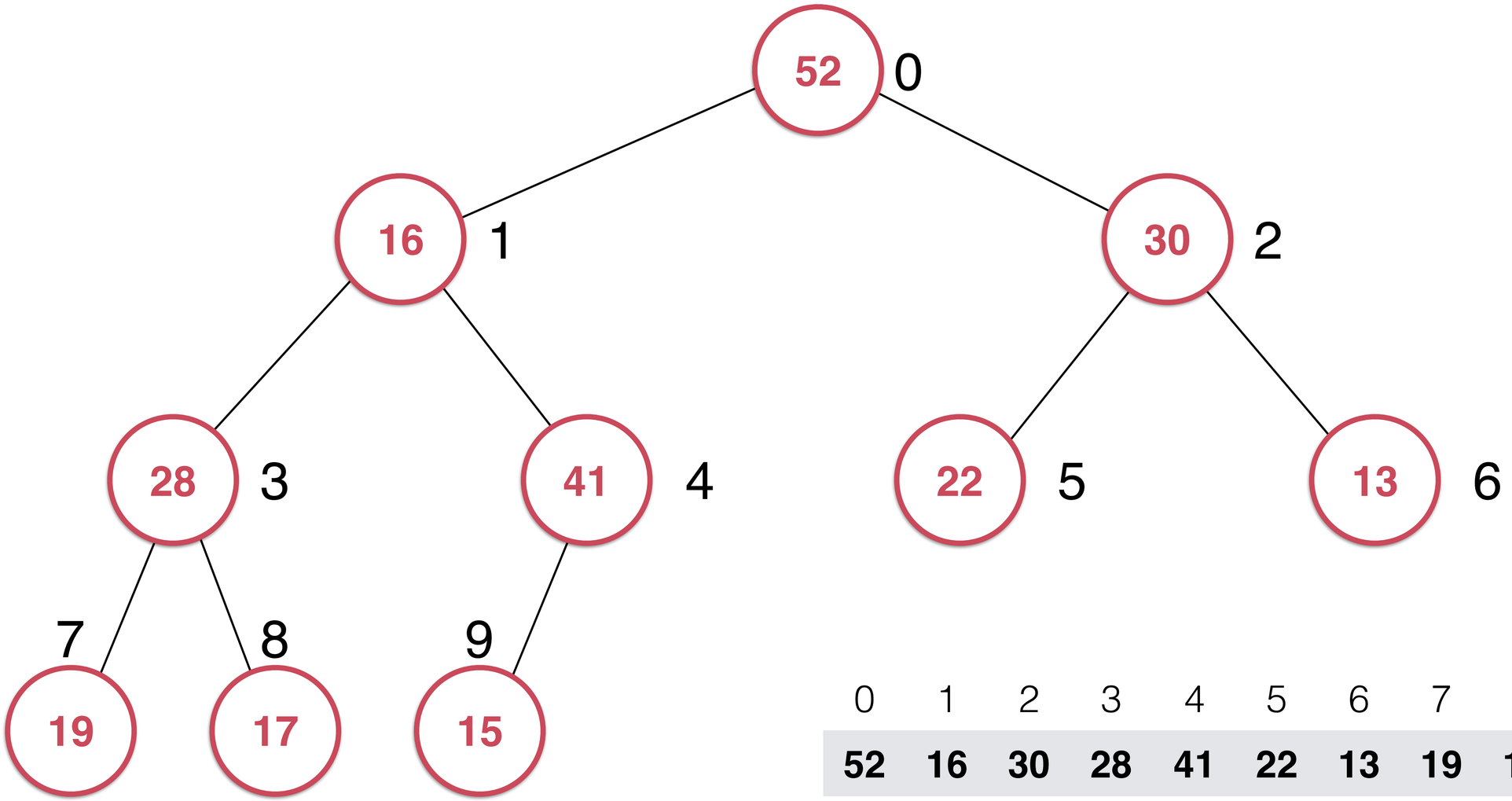
0	1	2	3	4	5	6	7	8	9
16	**52**	**30**	**28**	**41**	**22**	**13**	**19**	**17**	**15**

Sift Down



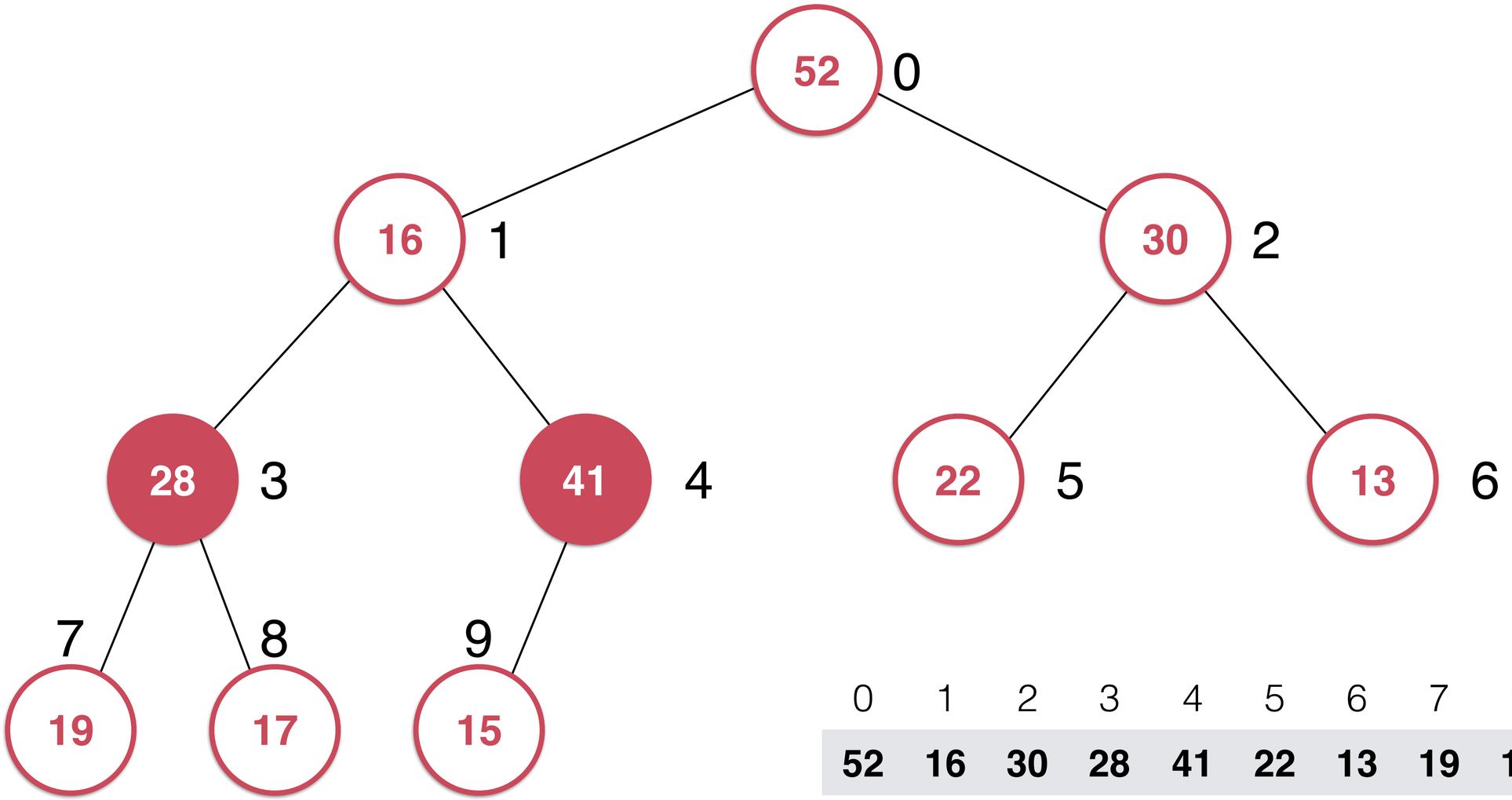
0	1	2	3	4	5	6	7	8	9
16	**52**	**30**	**28**	**41**	**22**	**13**	**19**	**17**	**15**

Sift Down



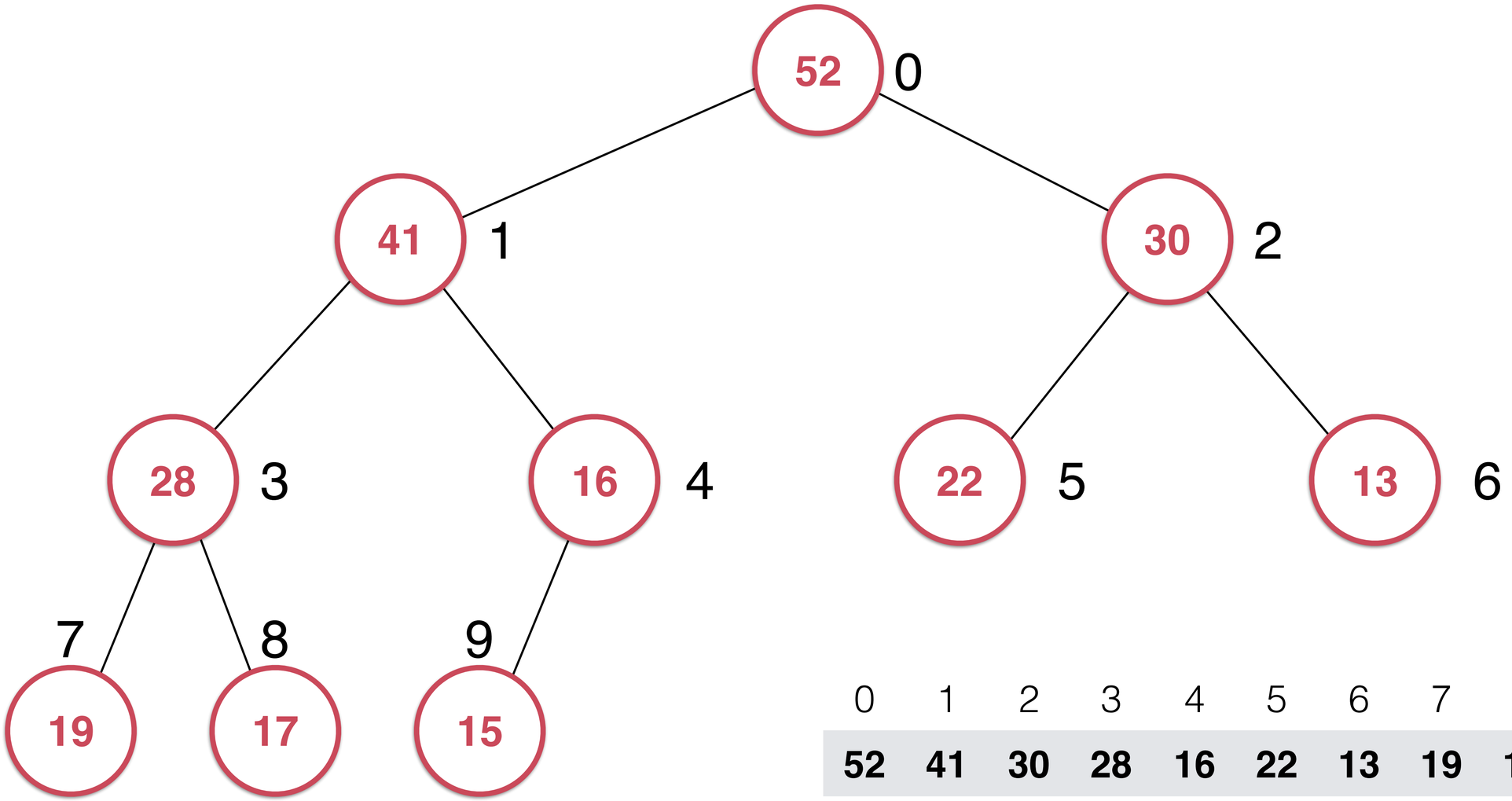
0	1	2	3	4	5	6	7	8	9
52	**16**	**30**	**28**	**41**	**22**	**13**	**19**	**17**	**15**

Sift Down



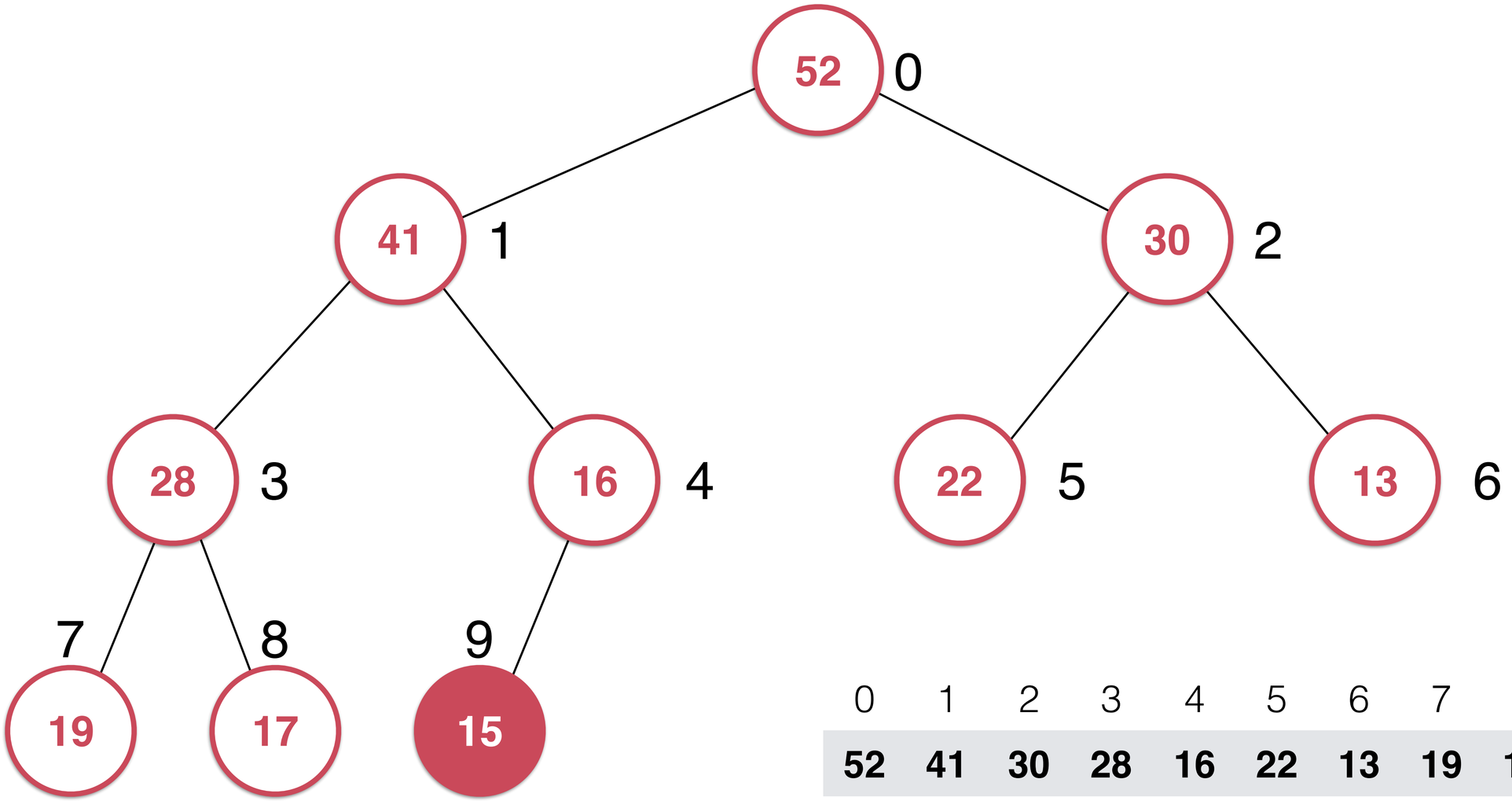
0	1	2	3	4	5	6	7	8	9
52	**16**	**30**	**28**	**41**	**22**	**13**	**19**	**17**	**15**

Sift Down



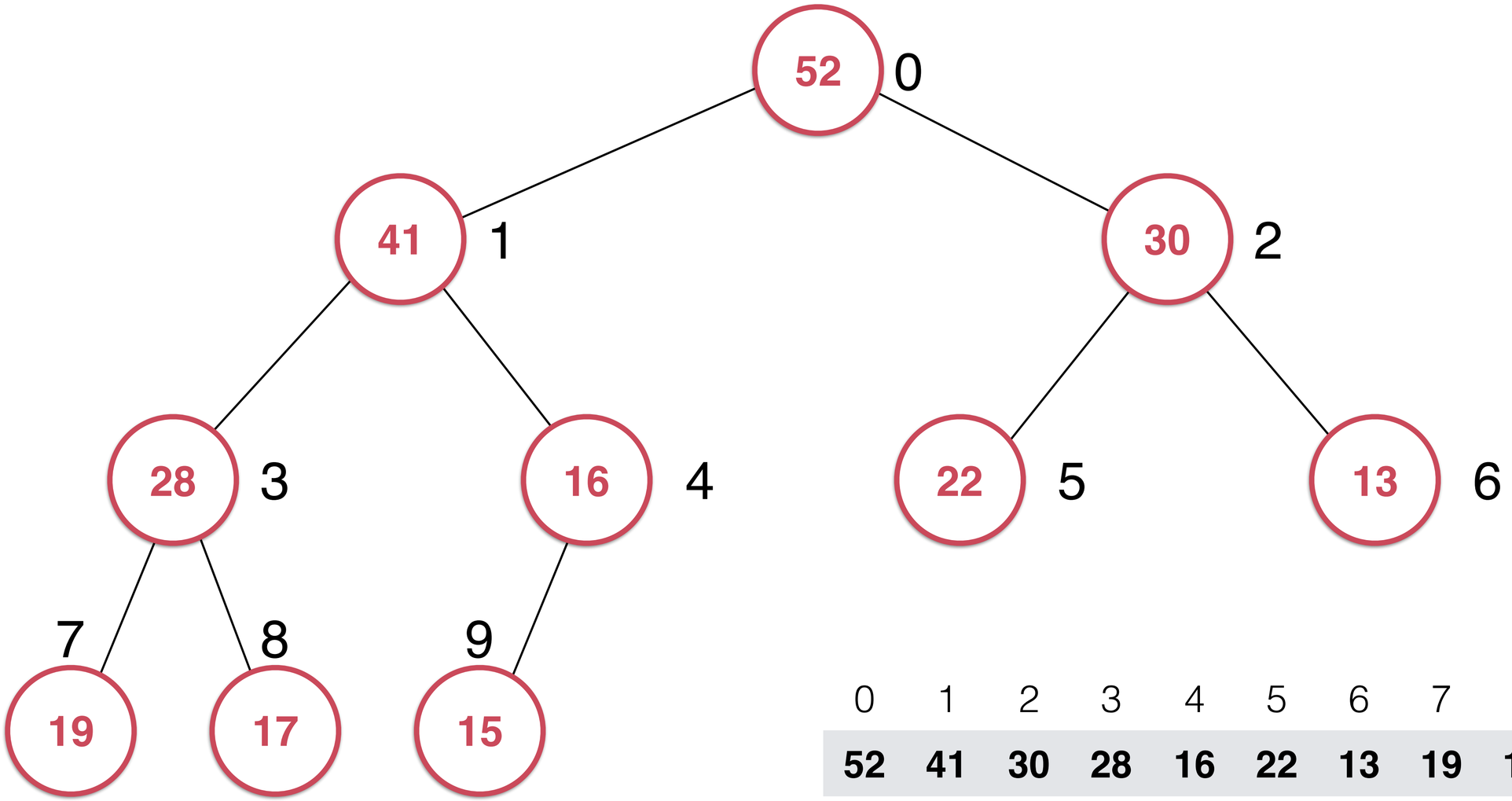
0	1	2	3	4	5	6	7	8	9
52	**41**	**30**	**28**	**16**	**22**	**13**	**19**	**17**	**15**

Sift Down



0	1	2	3	4	5	6	7	8	9
52	**41**	**30**	**28**	**16**	**22**	**13**	**19**	**17**	**15**

Sift Down

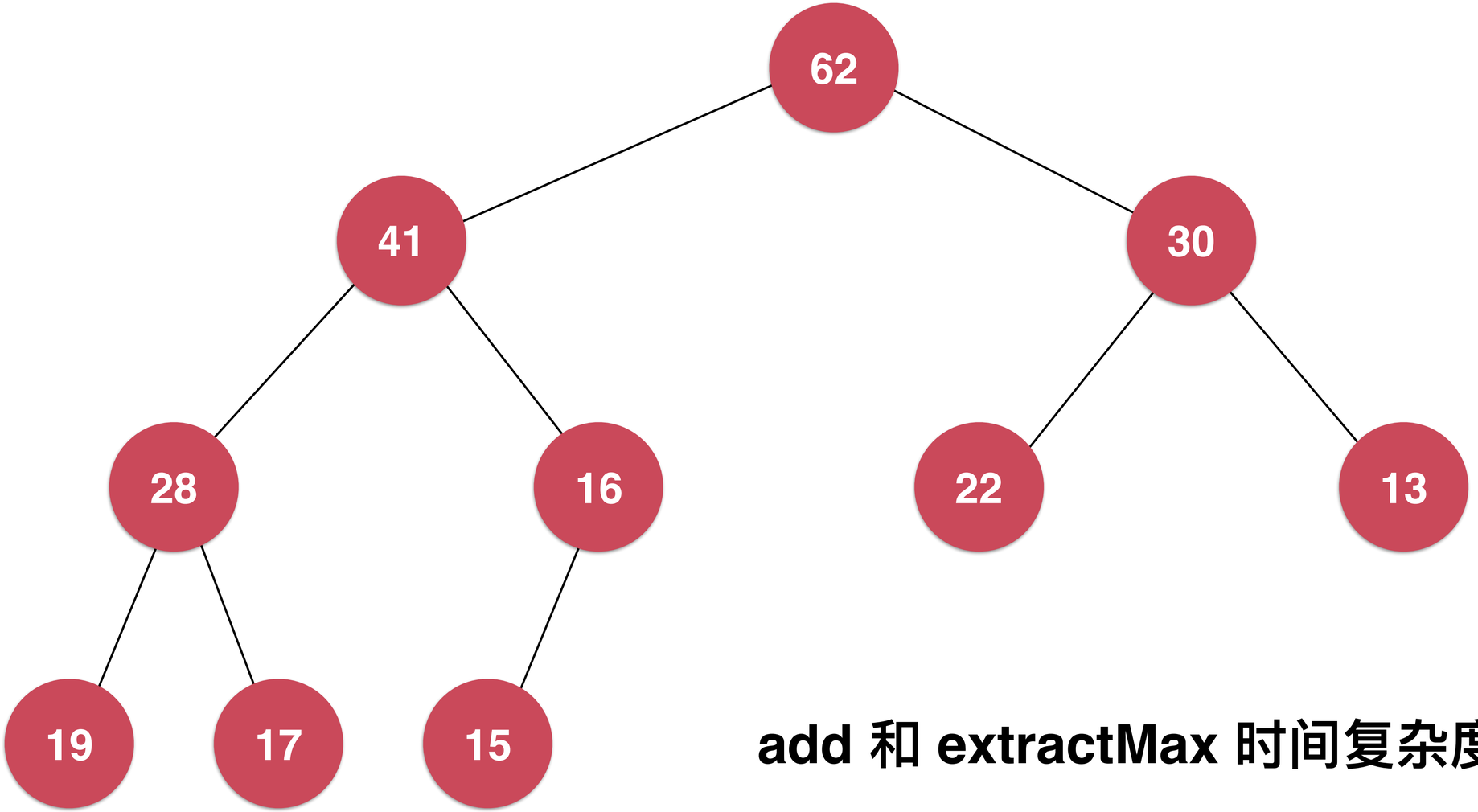


0	1	2	3	4	5	6	7	8	9
52	**41**	**30**	**28**	**16**	**22**	**13**	**19**	**17**	**15**

实践：Sift Down 和 extractMax

实践：测试堆

堆的时间复杂度分析



add 和 extractMax 时间复杂度都是 O(logn)

Heapify 和 replace

replace

replace：取出最大元素后，放入一个新元素

实现：可以先extractMax，再add，两次O(logn)的操作

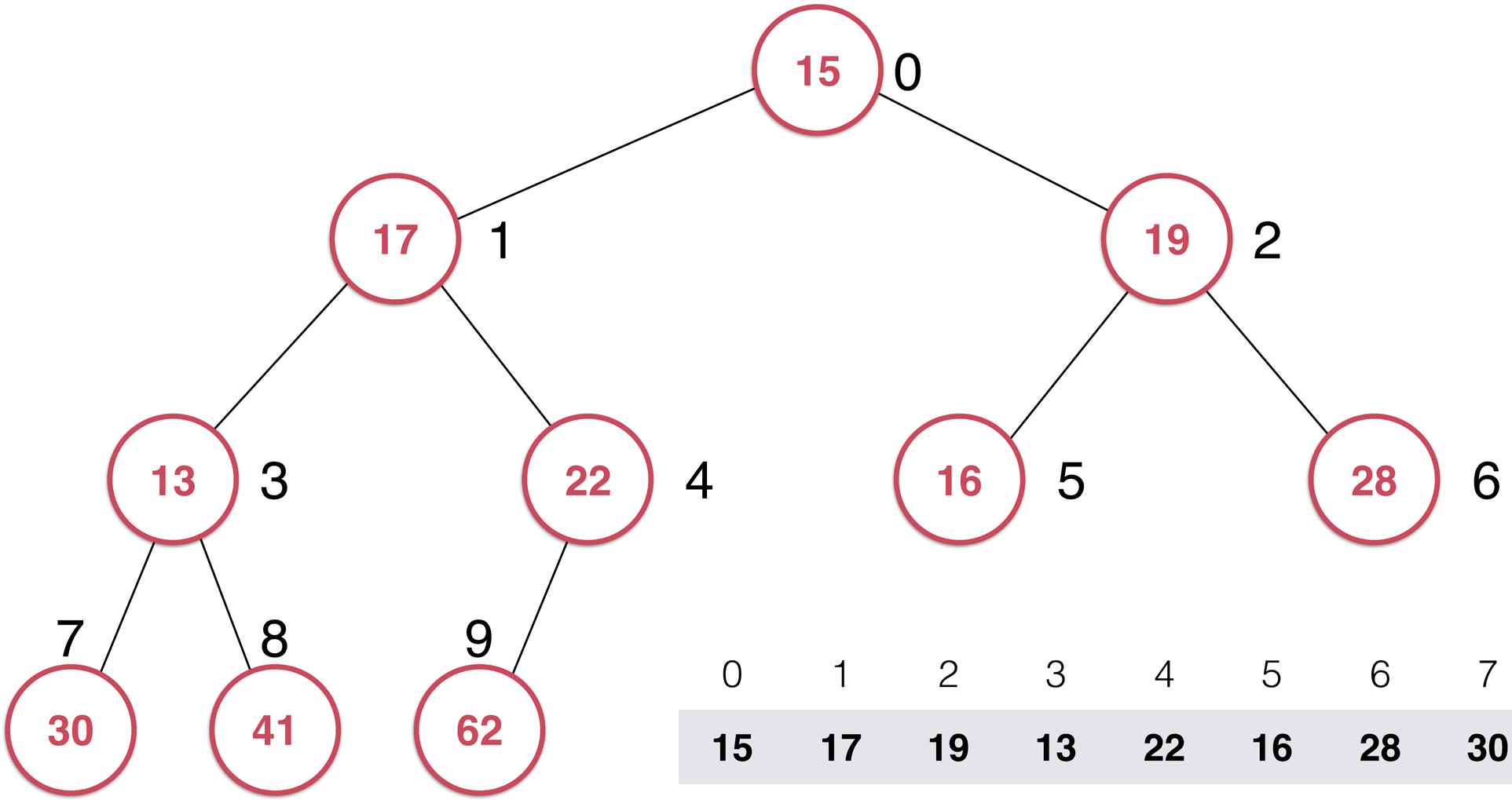
实现：可以直接将堆顶元素替换以后Sift Down，一次O(logn)的操作

实践：replace

heapify

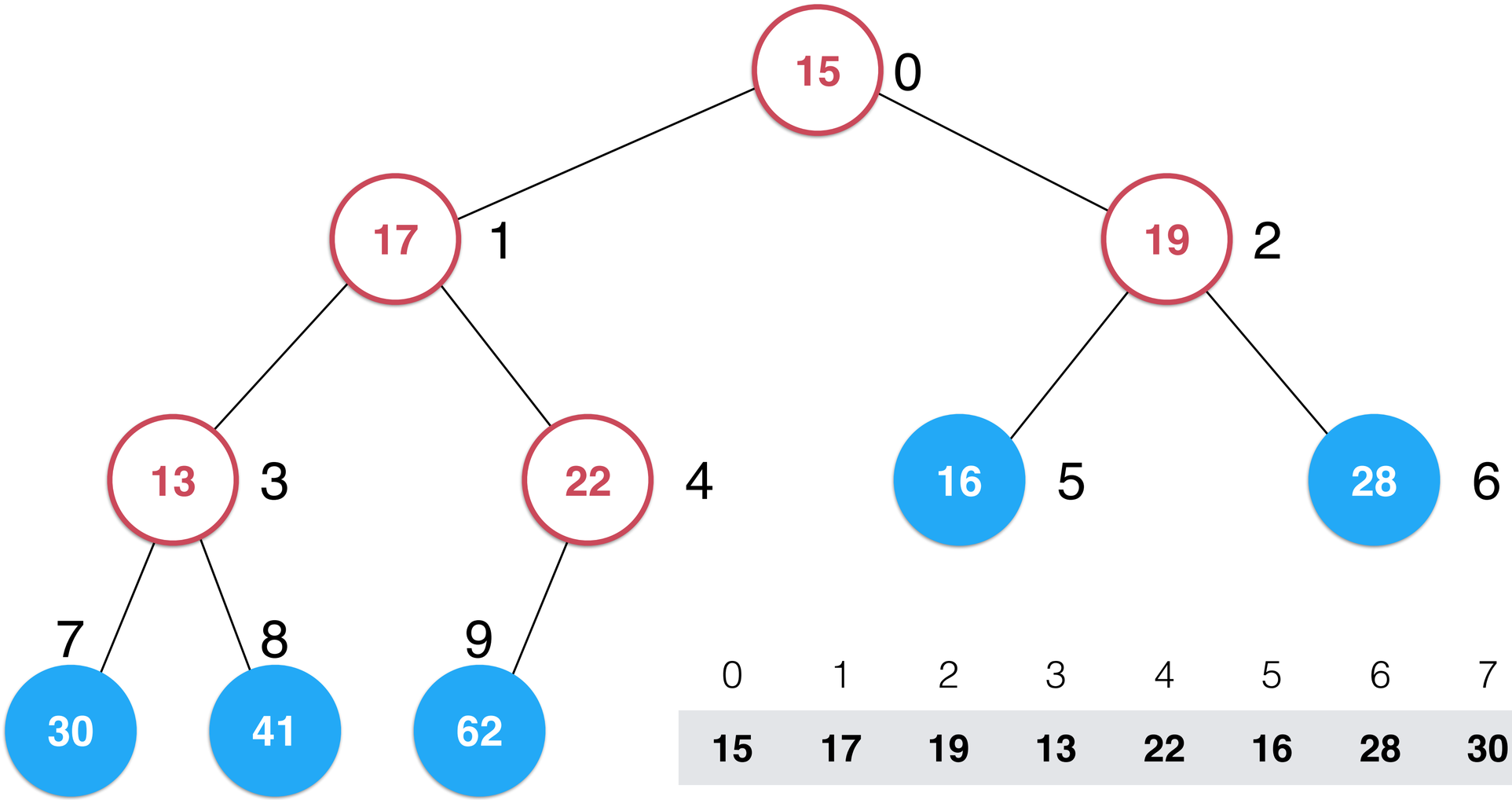
heapify：将任意数组整理成堆的形状

Heapify



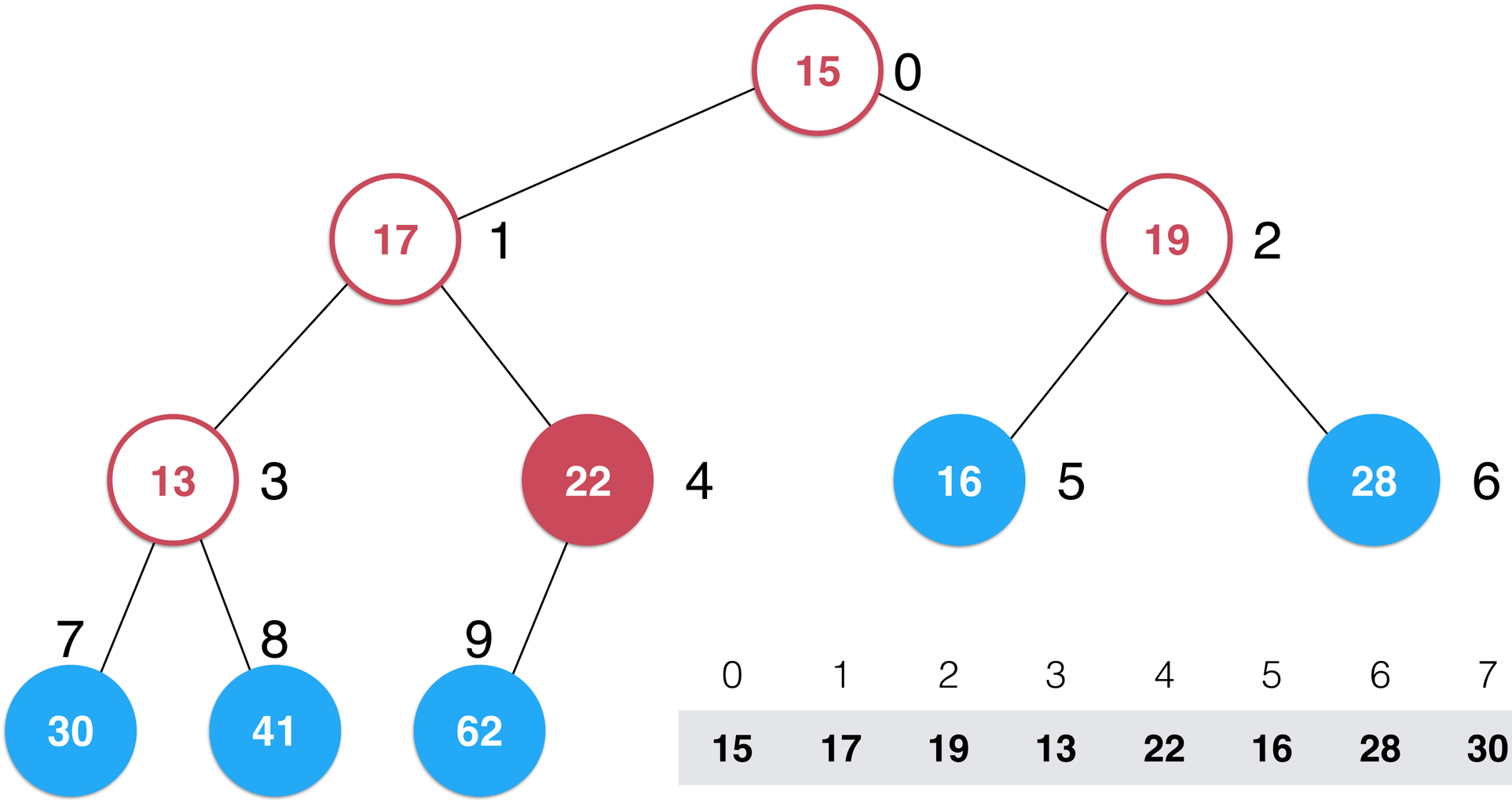
0	1	2	3	4	5	6	7	8	9
15	**17**	**19**	**13**	**22**	**16**	**28**	**30**	**41**	**62**

Heapify



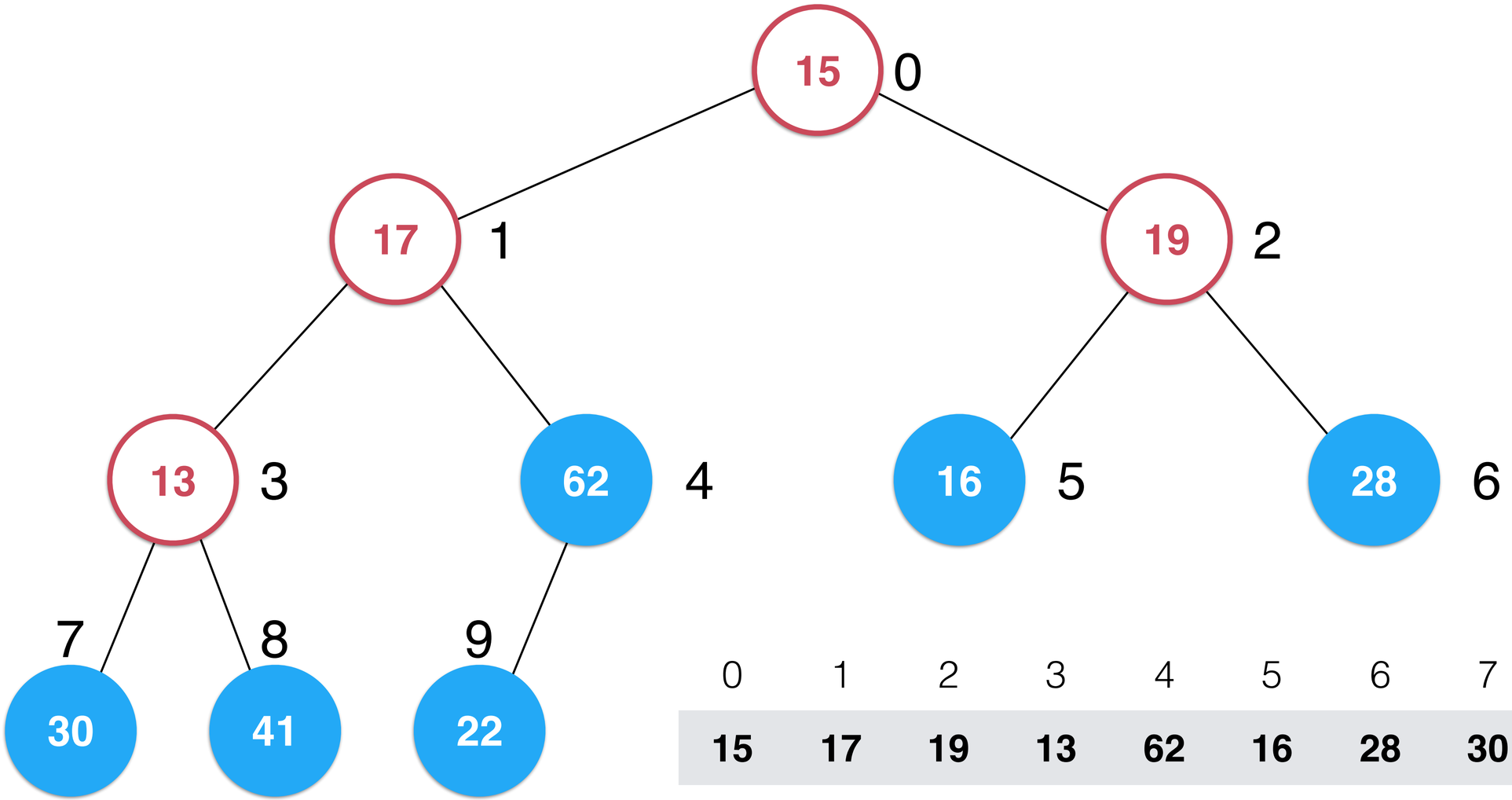
0	1	2	3	4	5	6	7	8	9
15	**17**	**19**	**13**	**22**	**16**	**28**	**30**	**41**	**62**

Heapify



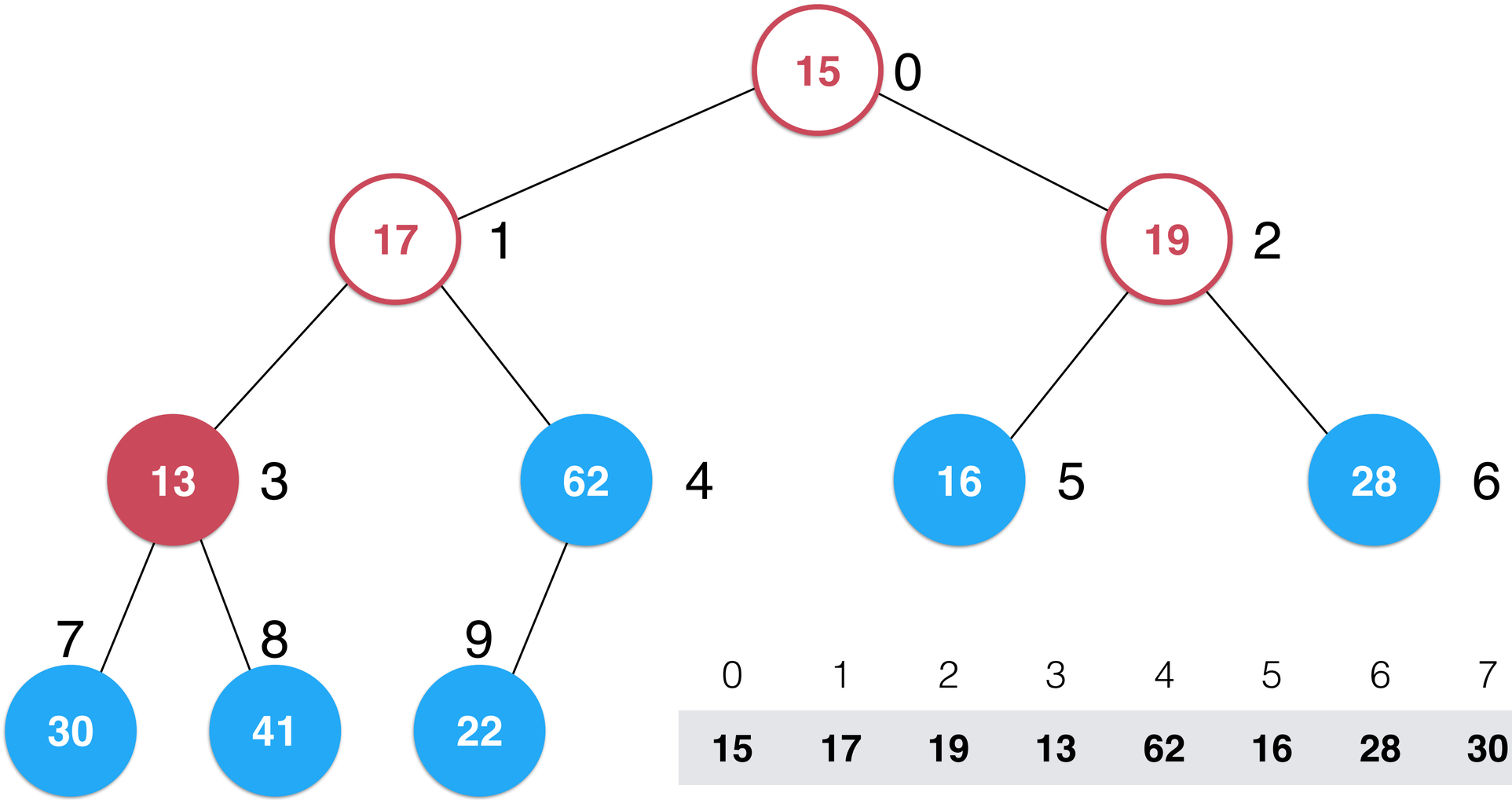
0	1	2	3	4	5	6	7	8	9
15	**17**	**19**	**13**	**22**	**16**	**28**	**30**	**41**	**62**

Heapify



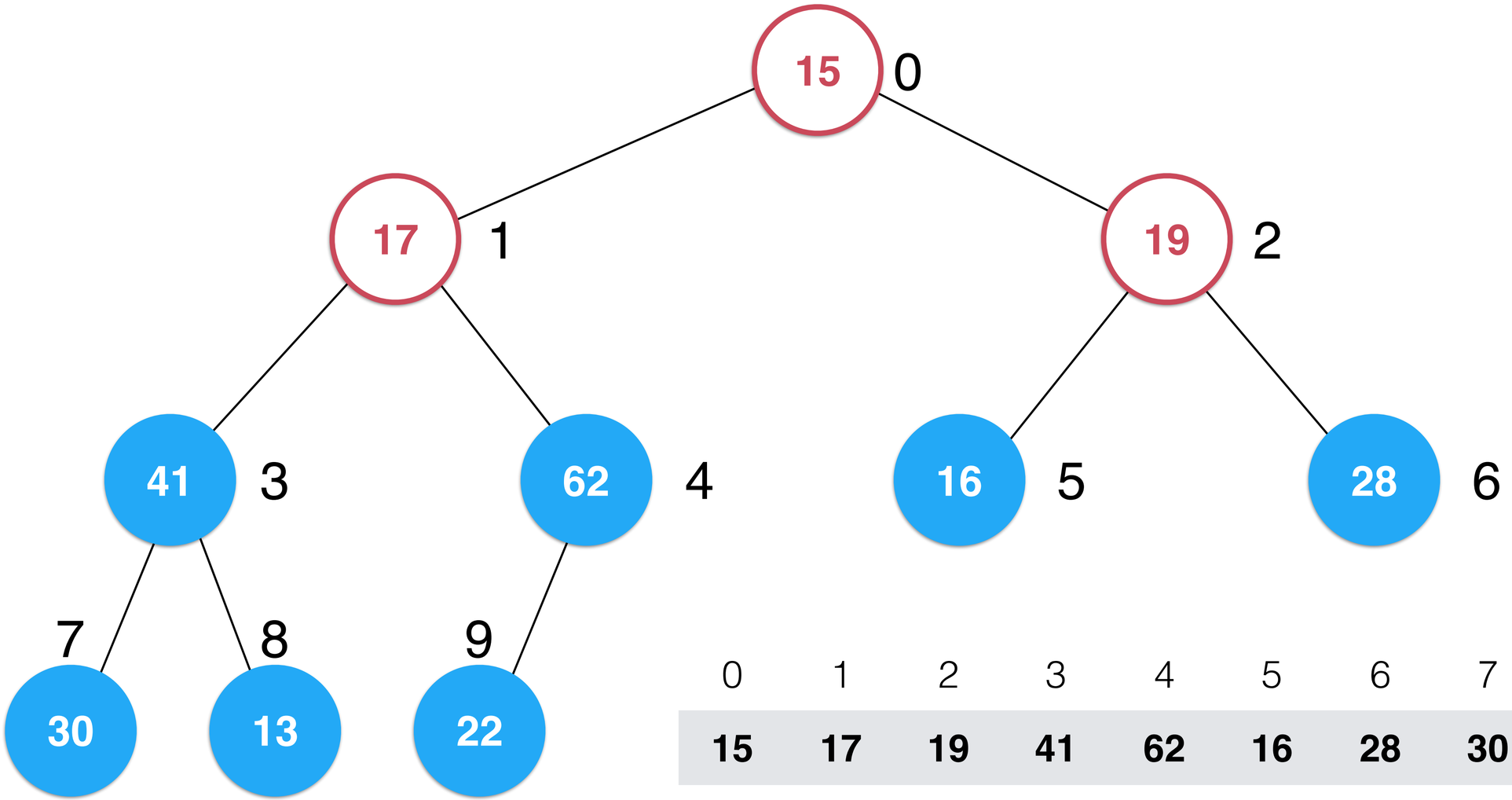
0	1	2	3	4	5	6	7	8	9
15	**17**	**19**	**13**	**62**	**16**	**28**	**30**	**41**	**22**

Heapify



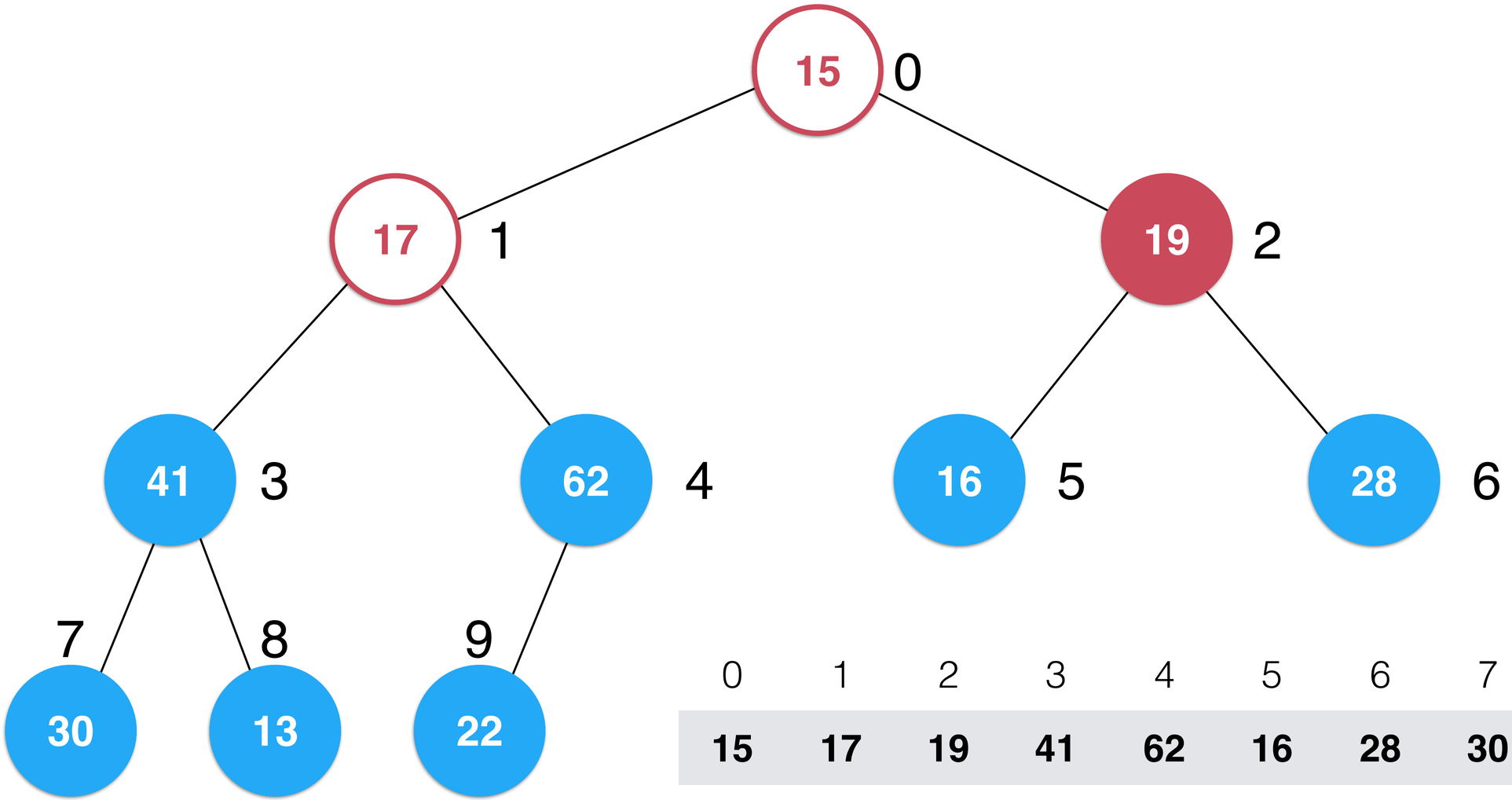
0	1	2	3	4	5	6	7	8	9
15	**17**	**19**	**13**	**62**	**16**	**28**	**30**	**41**	**22**

Heapify



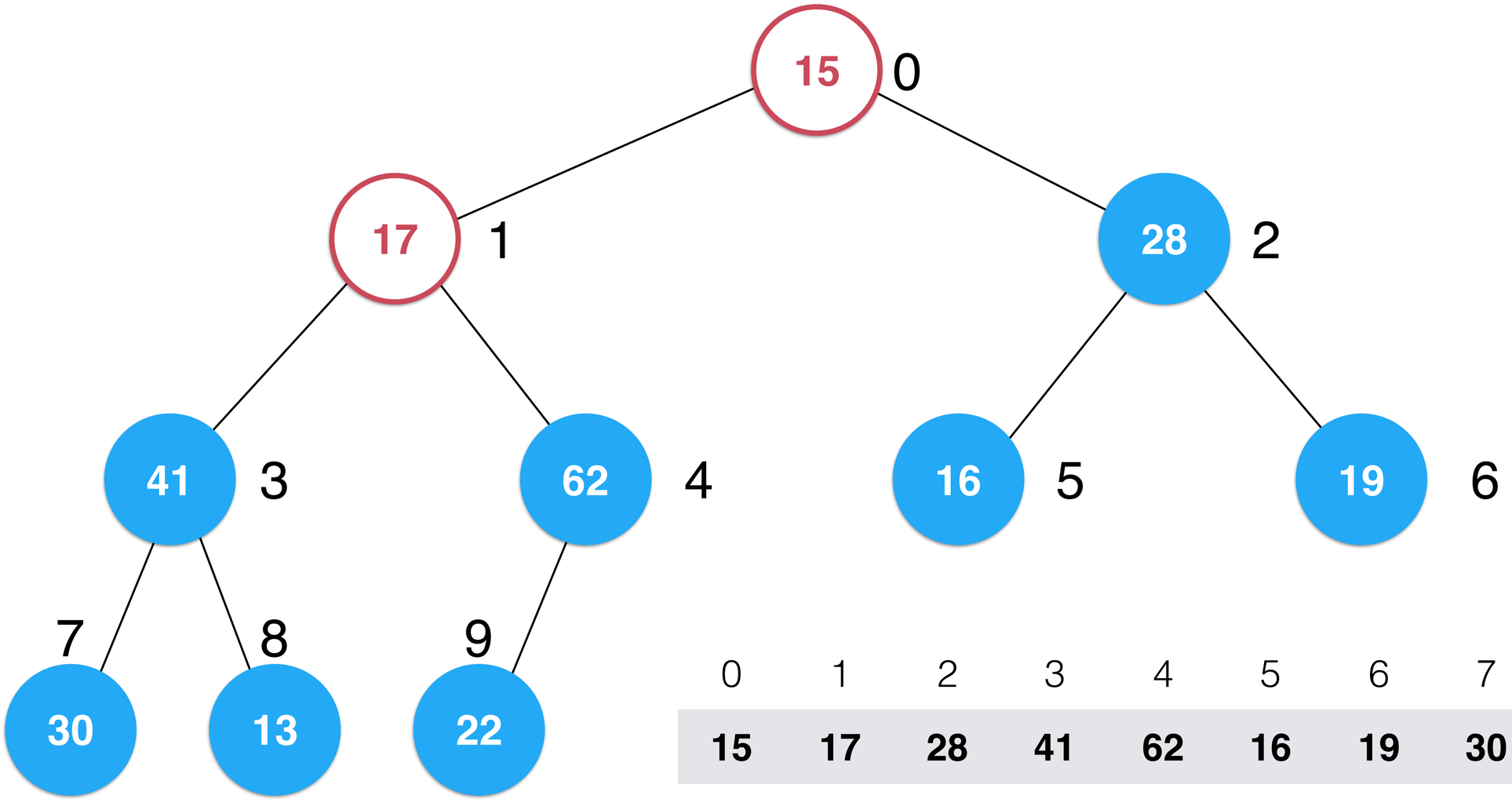
0	1	2	3	4	5	6	7	8	9
15	**17**	**19**	**41**	**62**	**16**	**28**	**30**	**13**	**22**

Heapify



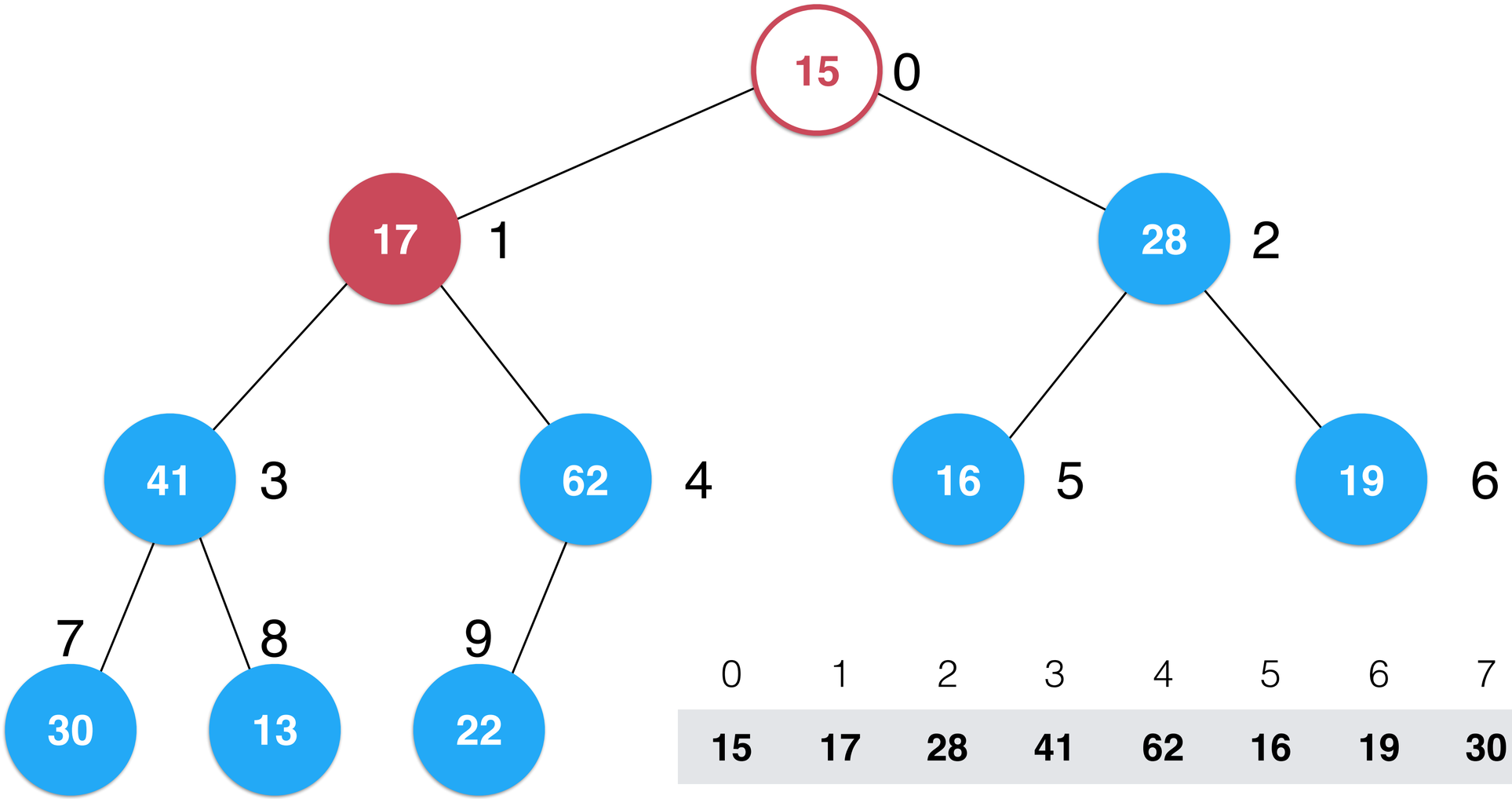
0	1	2	3	4	5	6	7	8	9
15	**17**	**19**	**41**	**62**	**16**	**28**	**30**	**13**	**22**

Heapify



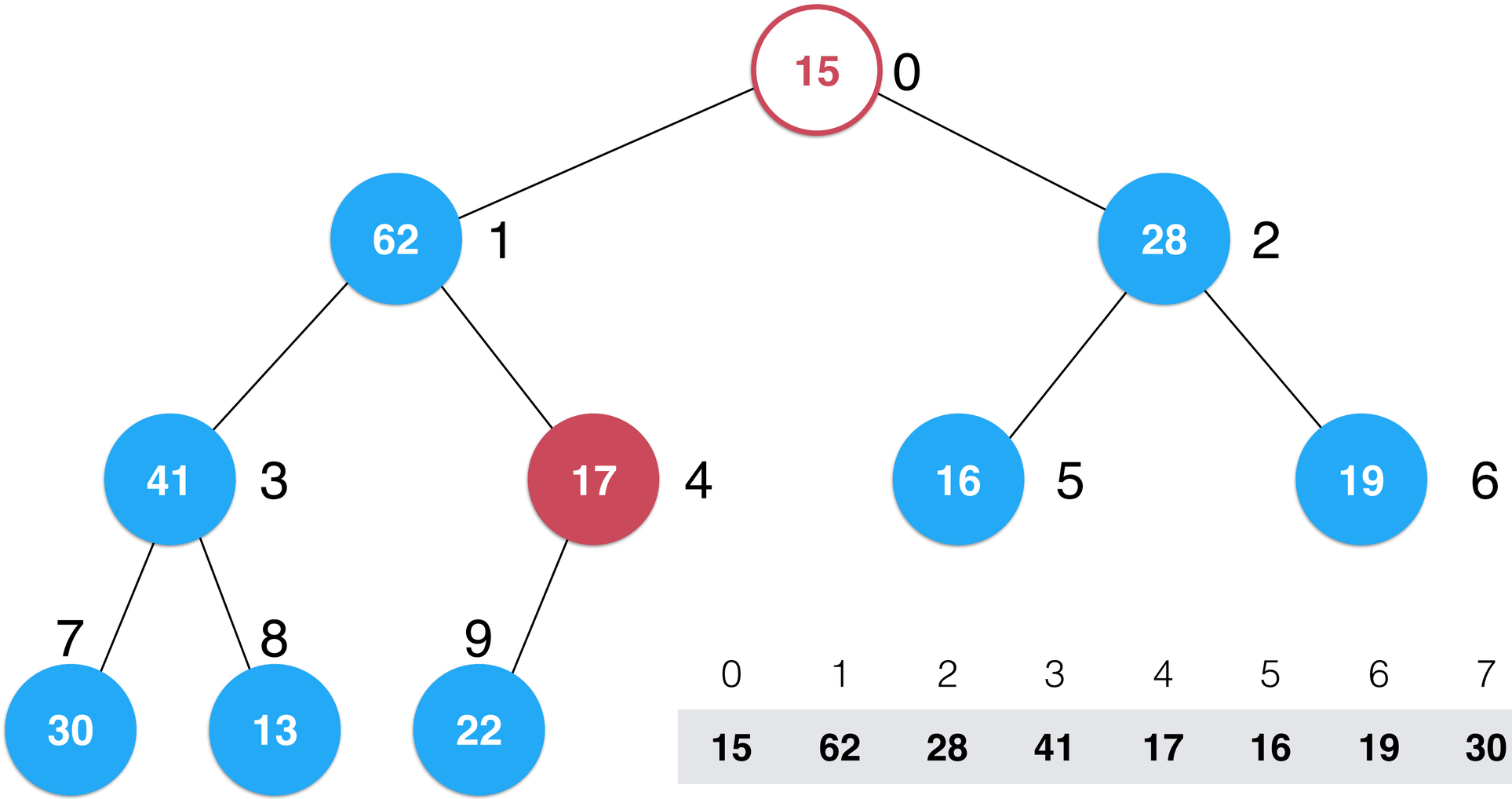
0	1	2	3	4	5	6	7	8	9
15	**17**	**28**	**41**	**62**	**16**	**19**	**30**	**13**	**22**

Heapify



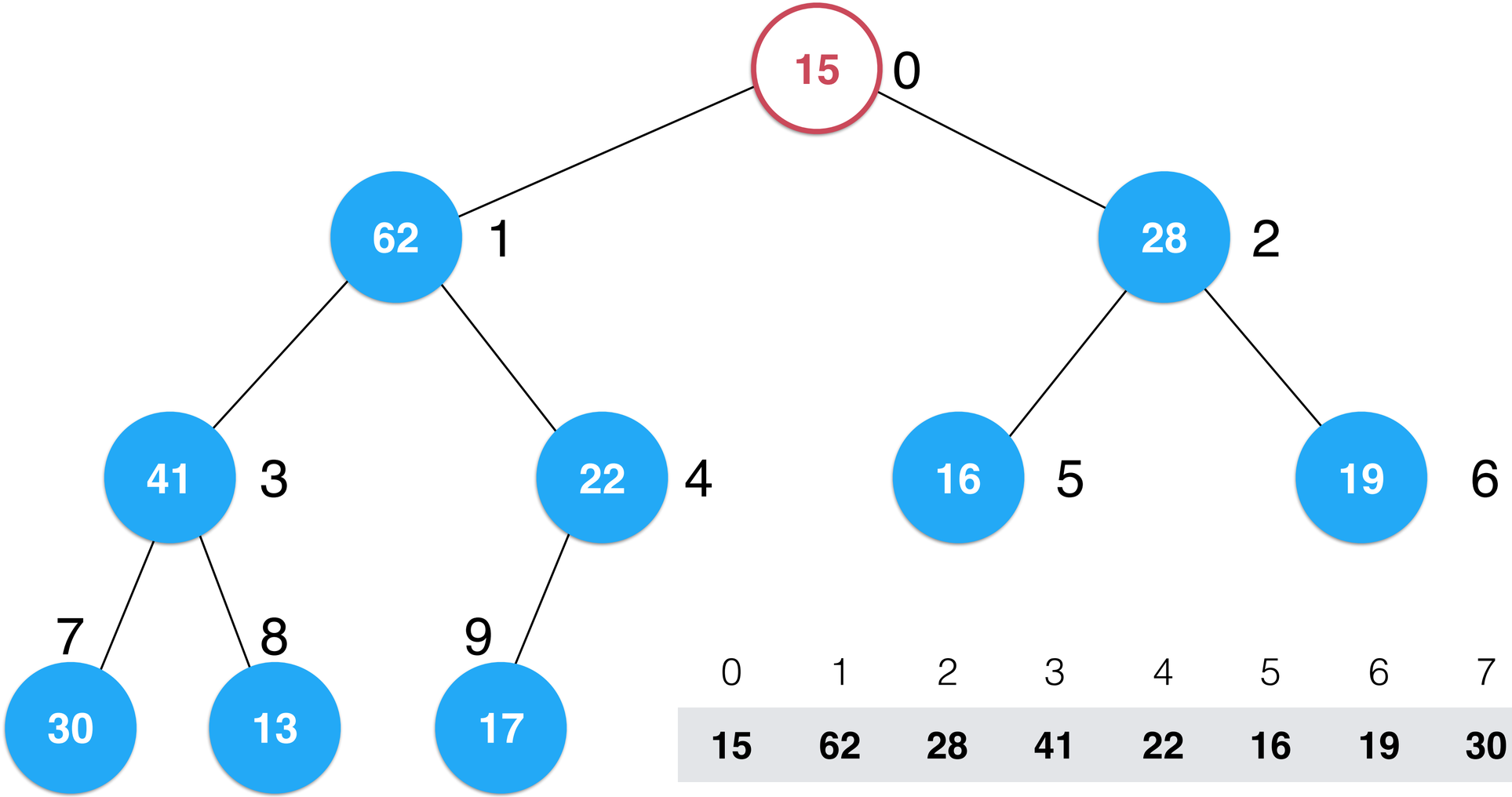
0	1	2	3	4	5	6	7	8	9
15	**17**	**28**	**41**	**62**	**16**	**19**	**30**	**13**	**22**

Heapify



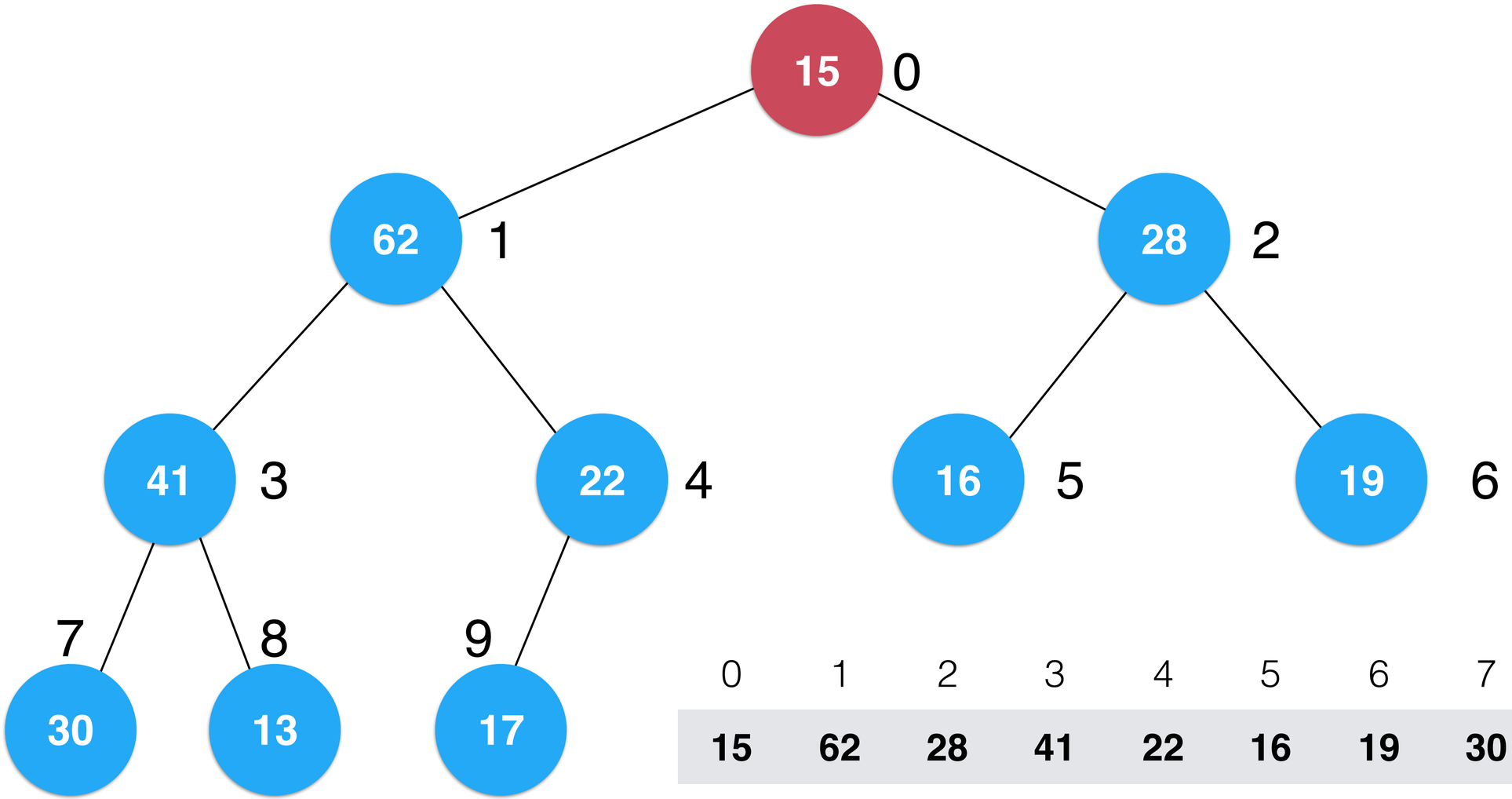
0	1	2	3	4	5	6	7	8	9
15	**62**	**28**	**41**	**17**	**16**	**19**	**30**	**13**	**22**

Heapify



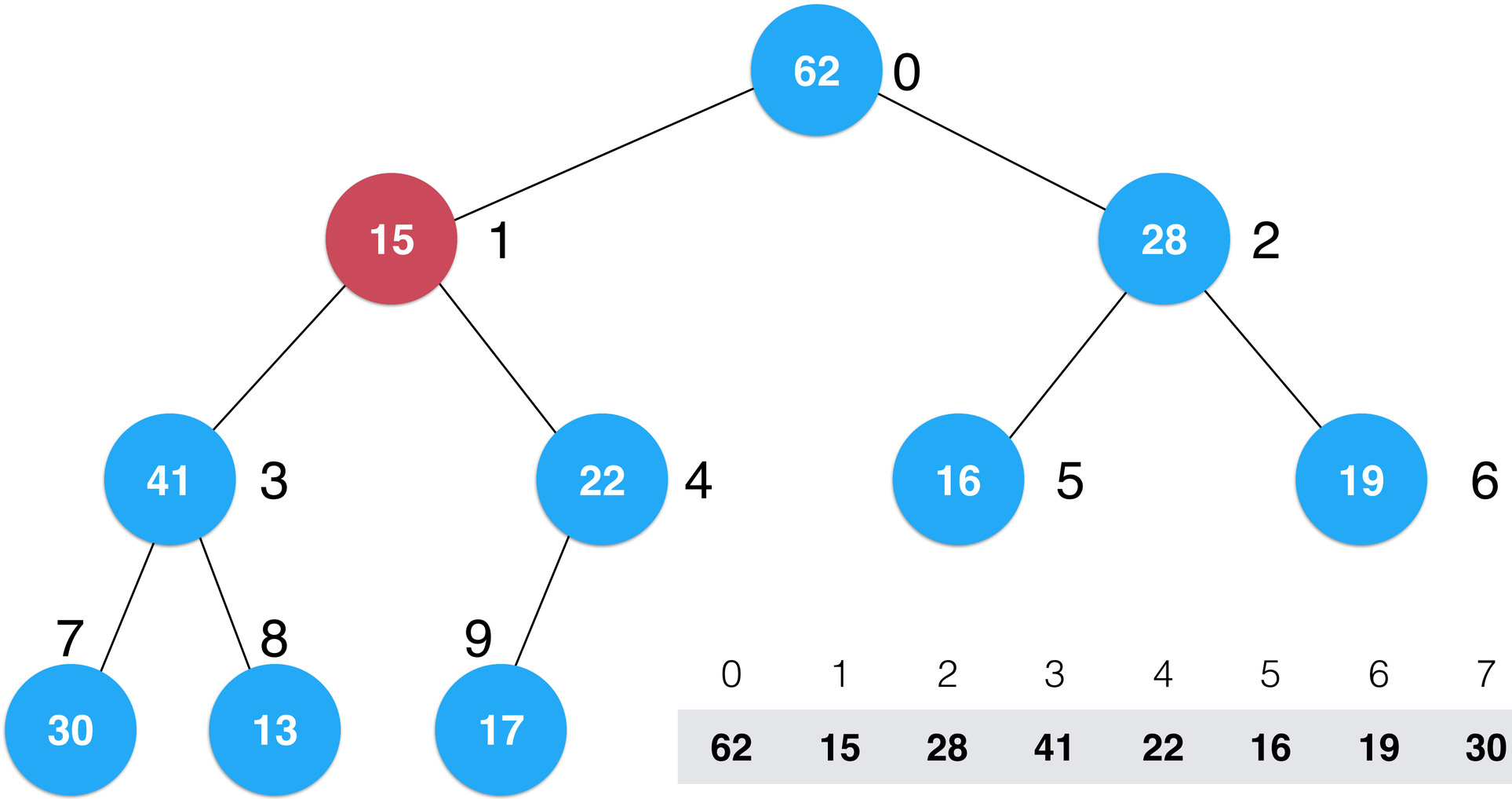
0	1	2	3	4	5	6	7	8	9
15	**62**	**28**	**41**	**22**	**16**	**19**	**30**	**13**	**17**

Heapify



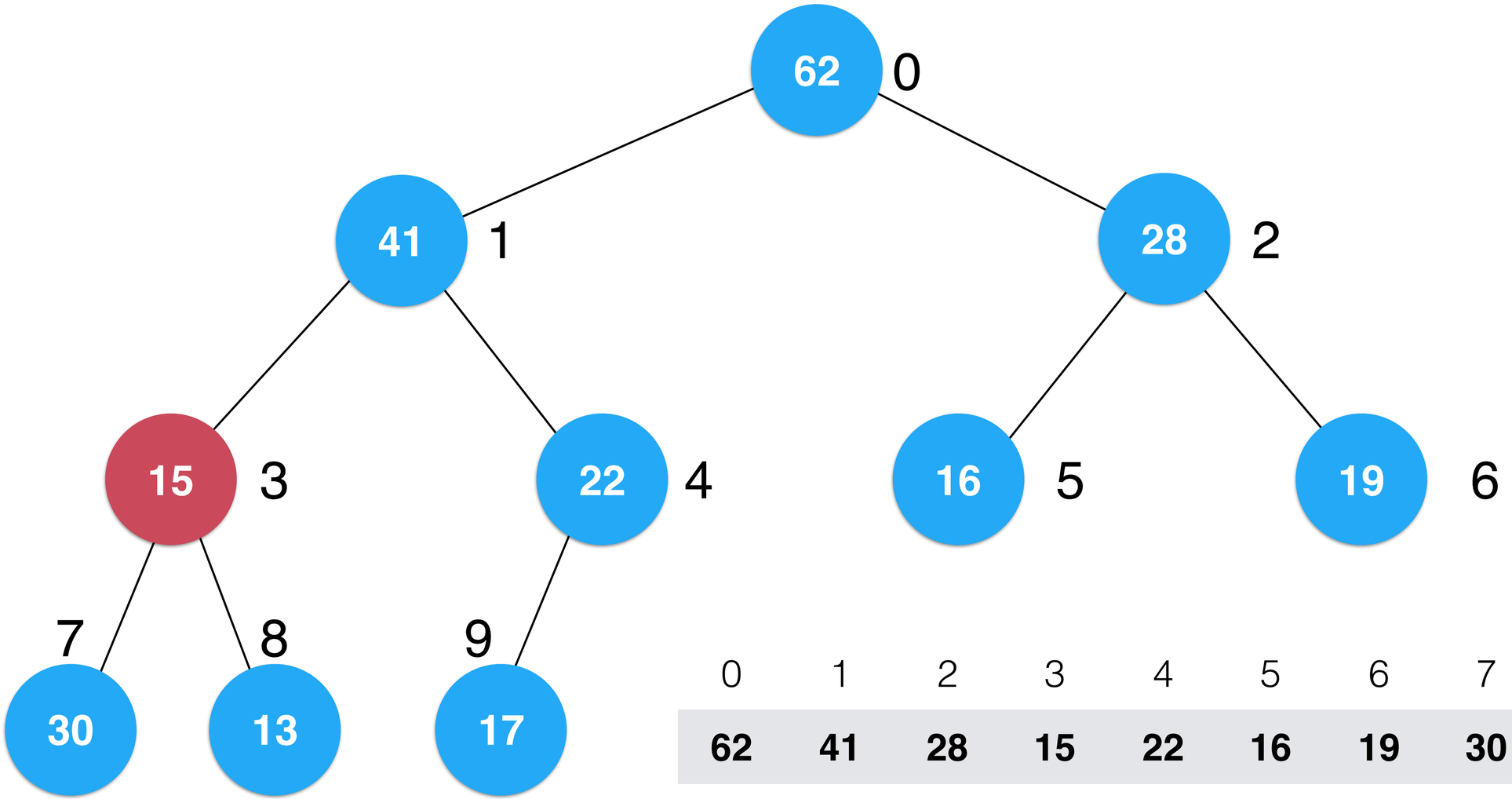
0	1	2	3	4	5	6	7	8	9
15	**62**	**28**	**41**	**22**	**16**	**19**	**30**	**13**	**17**

Heapify



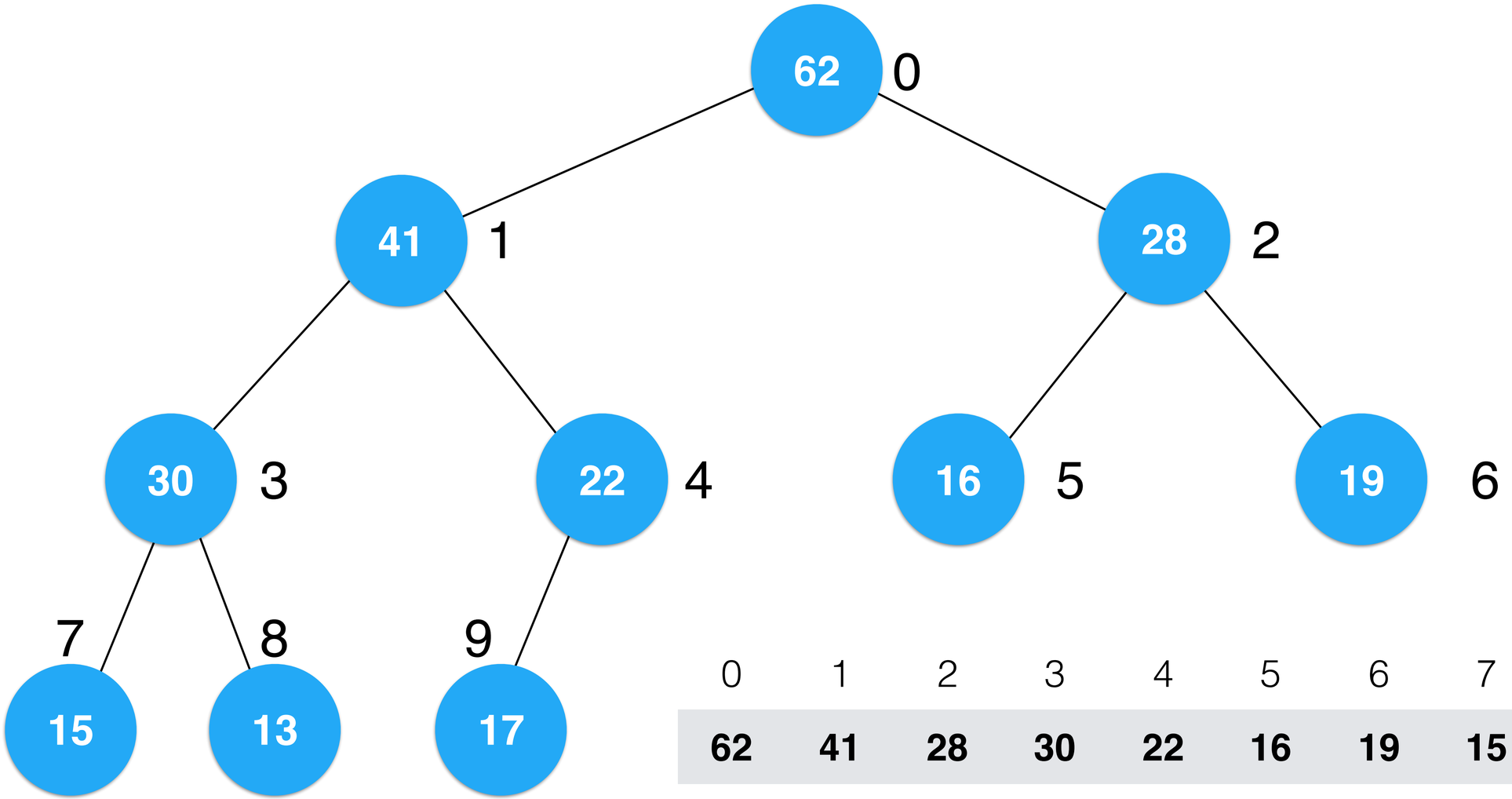
0	1	2	3	4	5	6	7	8	9
62	**15**	**28**	**41**	**22**	**16**	**19**	**30**	**13**	**17**

Heapify



0	1	2	3	4	5	6	7	8	9
62	**41**	**28**	**15**	**22**	**16**	**19**	**30**	**13**	**17**

Heapify



0	1	2	3	4	5	6	7	8	9
62	**41**	**28**	**30**	**22**	**16**	**19**	**15**	**13**	**17**

操作：Heapify

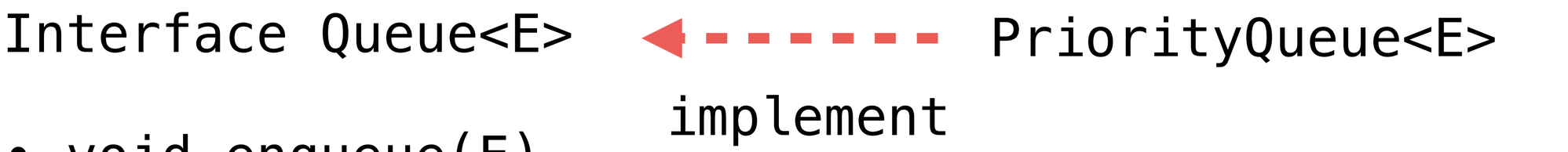
Heapify 的算法复杂度

将n个元素逐个插入到一个空堆中，算法复杂度是O(nlogn)

heapify的过程，算法复杂度为O(n)

更多Heap相关的操作和Priority Queue

优先队列



- void enqueue(E)
- E dequeue()
- E getFront()
- int getSize()
- boolean isEmpty()

可以使用不同的底层实现

实践：基于堆的优先队列实现

Leetcode 上优先队列相关的问题

优先队列的经典问题

在1,000,000个元素中选出前100名?

在N个元素中选出前M个元素

排序? NlogN

使用优先队列? NlogM

优先队列的经典问题

在1,000,000个元素中选出前100名?

在N个元素中选出前M个元素

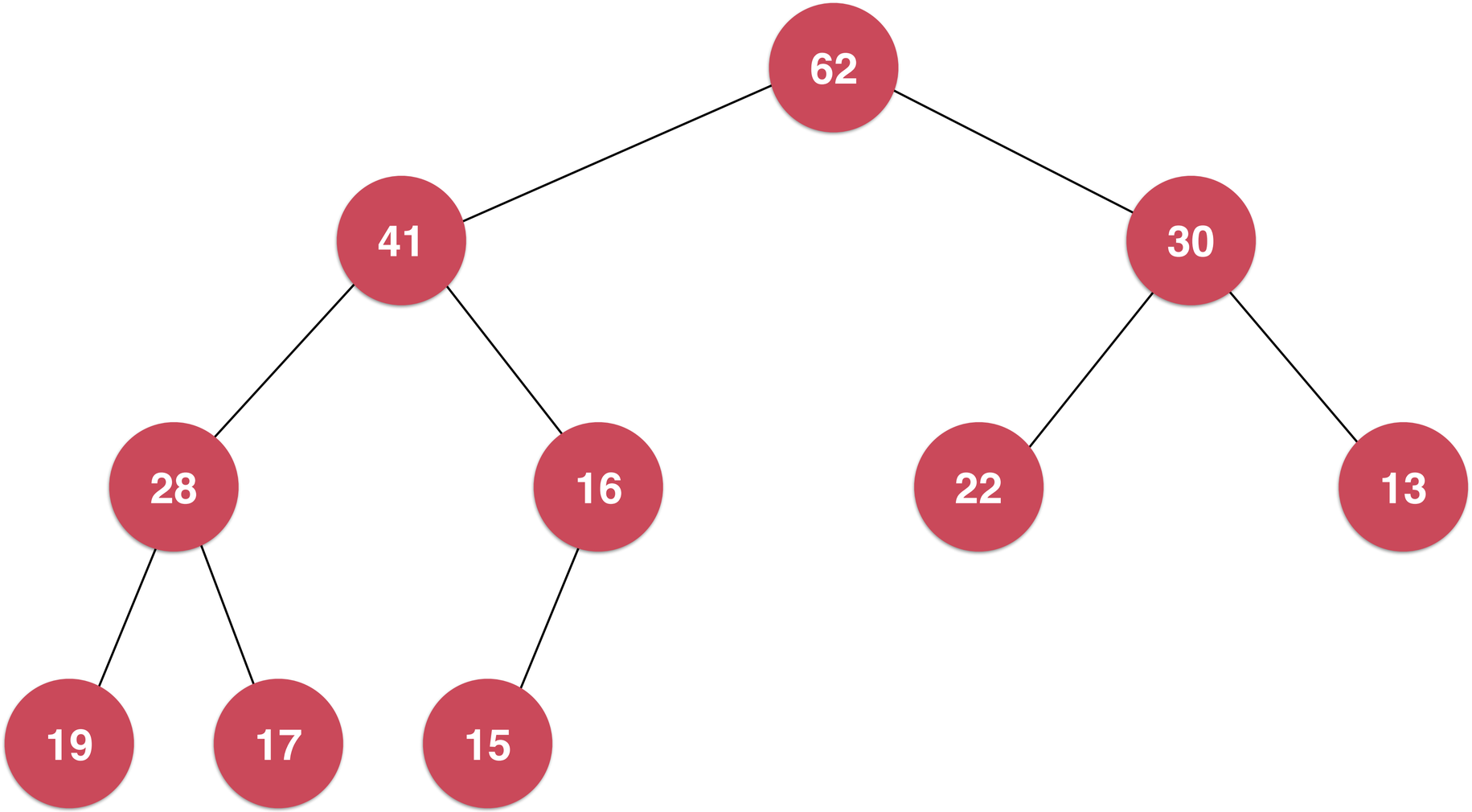
使用优先队列，维护当前看到的前M个元素

需要使用最小堆

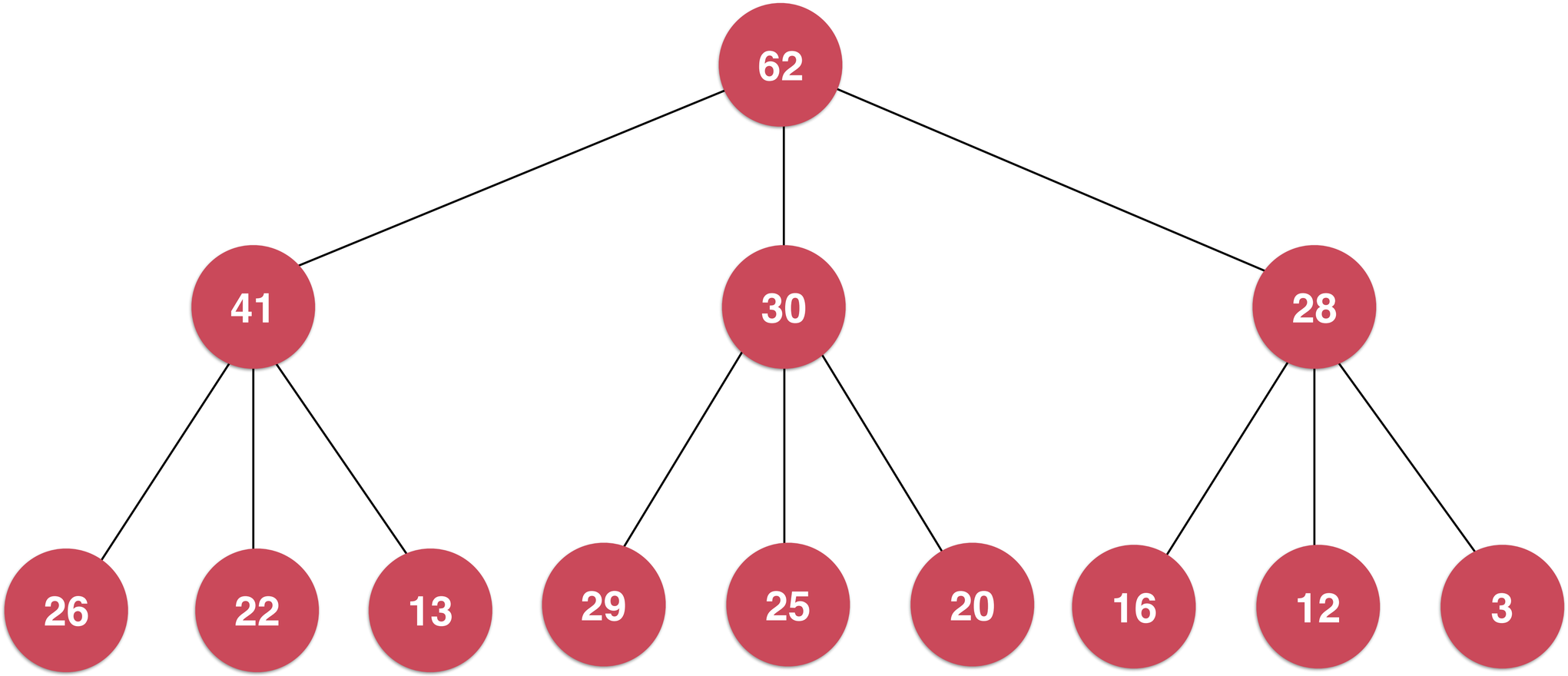
Java标准库中的优先队列

更多和堆，队列相关的话题

二叉堆 Binary Heap



d 叉堆 d-ary heap



索引堆

二项堆

斐波那契堆

广义队列

广义队列

Interface Queue<E>

- void enqueue(E)
- E dequeue()
- E getFront()
- int getSize()
- boolean isEmpty()

广义队列

普通队列，优先队列

栈，也可以理解成是一个队列

随机队列

其他

欢迎大家关注我的个人公众号：是不是很酷

玩儿转数据结构

liuyubobobo