

算法与数据结构体系课程

liuyubobobo

堆和优先队列

优先队列基础

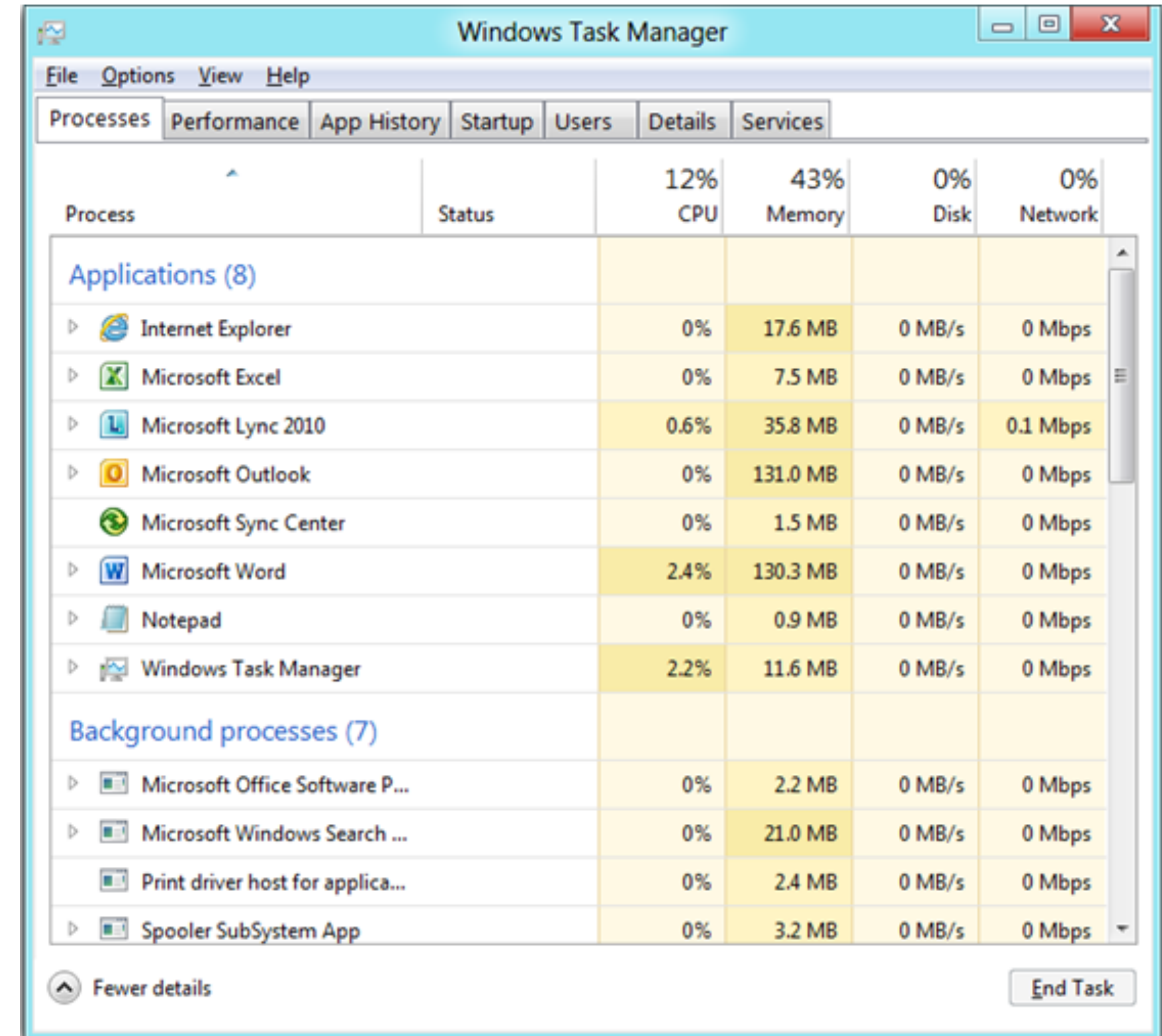
什么是优先队列？

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

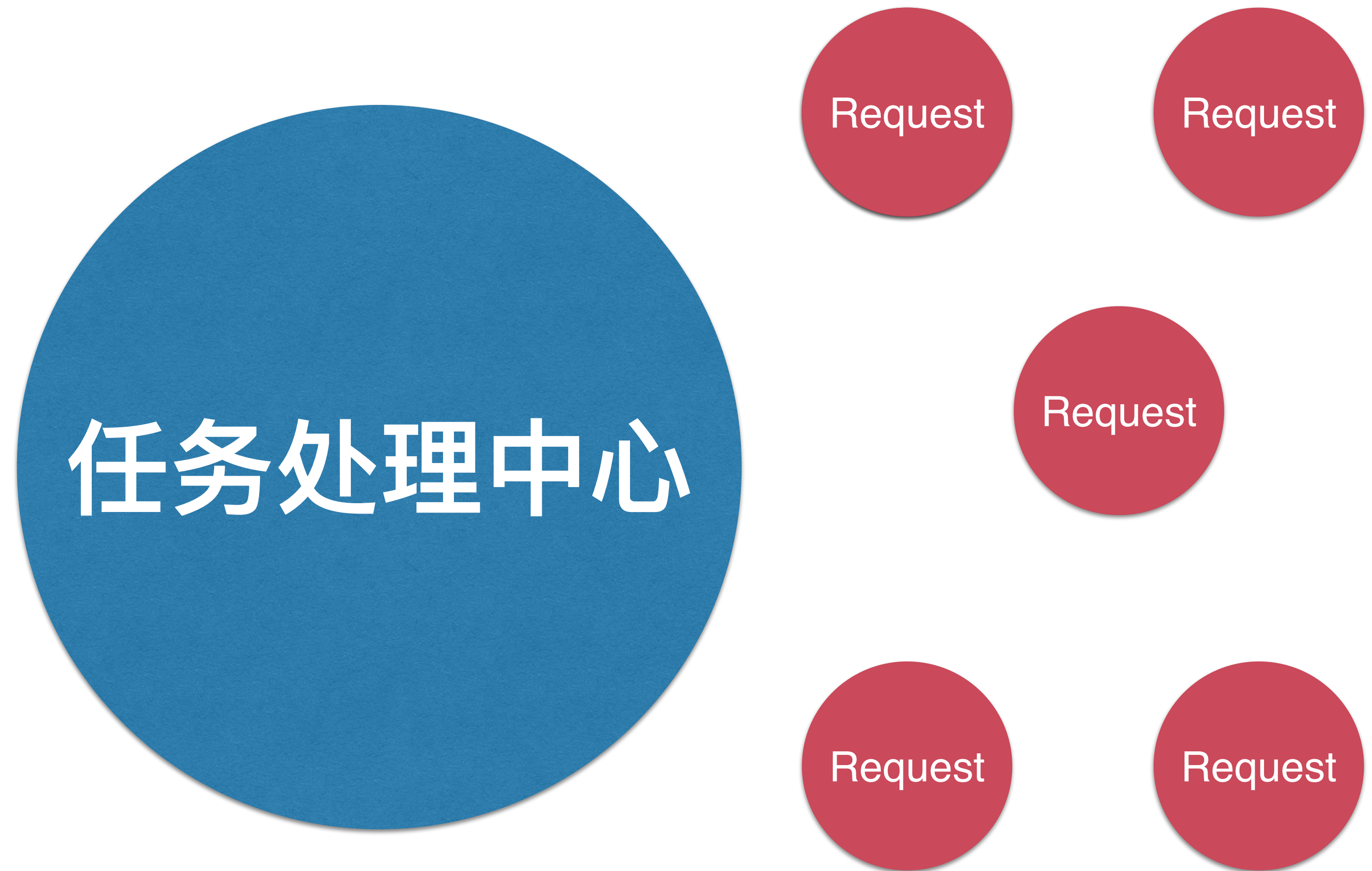


The screenshot shows the Windows Task Manager window with the 'Processes' tab selected. It displays a list of running applications and background processes, along with their CPU, Memory, Disk, and Network usage. The 'Applications (8)' section lists Internet Explorer, Microsoft Excel, Microsoft Lync 2010, Microsoft Outlook, Microsoft Sync Center, Microsoft Word, Notepad, and Windows Task Manager. The 'Background processes (7)' section lists Microsoft Office Software P..., Microsoft Windows Search ..., Print driver host for applica..., and Spooler SubSystem App. The 'End Task' button is visible at the bottom right.

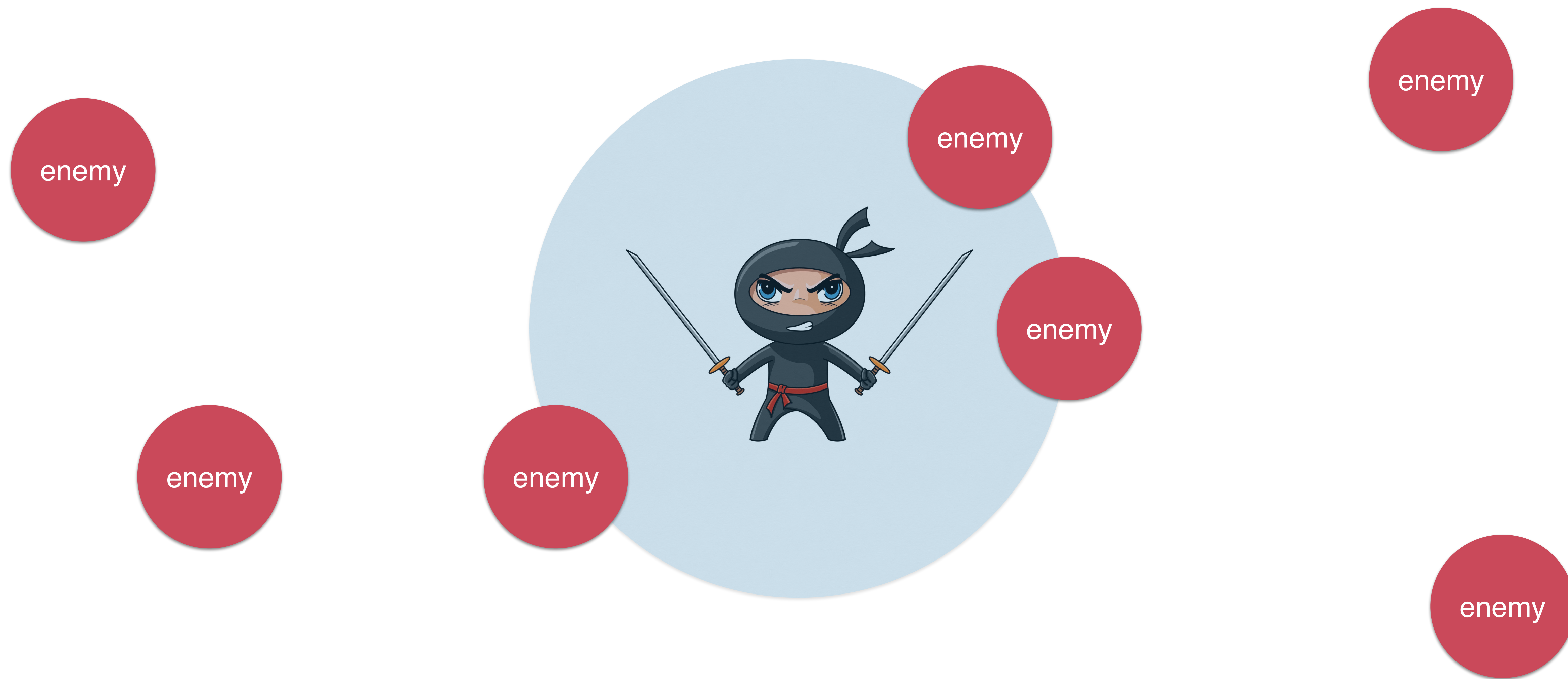
Process	Status	12% CPU	43% Memory	0% Disk	0% Network
Applications (8)					
Internet Explorer		0%	17.6 MB	0 MB/s	0 Mbps
Microsoft Excel		0%	7.5 MB	0 MB/s	0 Mbps
Microsoft Lync 2010		0.6%	35.8 MB	0 MB/s	0.1 Mbps
Microsoft Outlook		0%	131.0 MB	0 MB/s	0 Mbps
Microsoft Sync Center		0%	1.5 MB	0 MB/s	0 Mbps
Microsoft Word		2.4%	130.3 MB	0 MB/s	0 Mbps
Notepad		0%	0.9 MB	0 MB/s	0 Mbps
Windows Task Manager		2.2%	11.6 MB	0 MB/s	0 Mbps
Background processes (7)					
Microsoft Office Software P...		0%	2.2 MB	0 MB/s	0 Mbps
Microsoft Windows Search ...		0%	21.0 MB	0 MB/s	0 Mbps
Print driver host for applica...		0%	2.4 MB	0 MB/s	0 Mbps
Spooler SubSystem App		0%	3.2 MB	0 MB/s	0 Mbps

为什么使用优先队列?


关键词： 动态



为什么使用优先队列?



优先队列

Interface Queue<E>  PriorityQueue<E>
implement

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

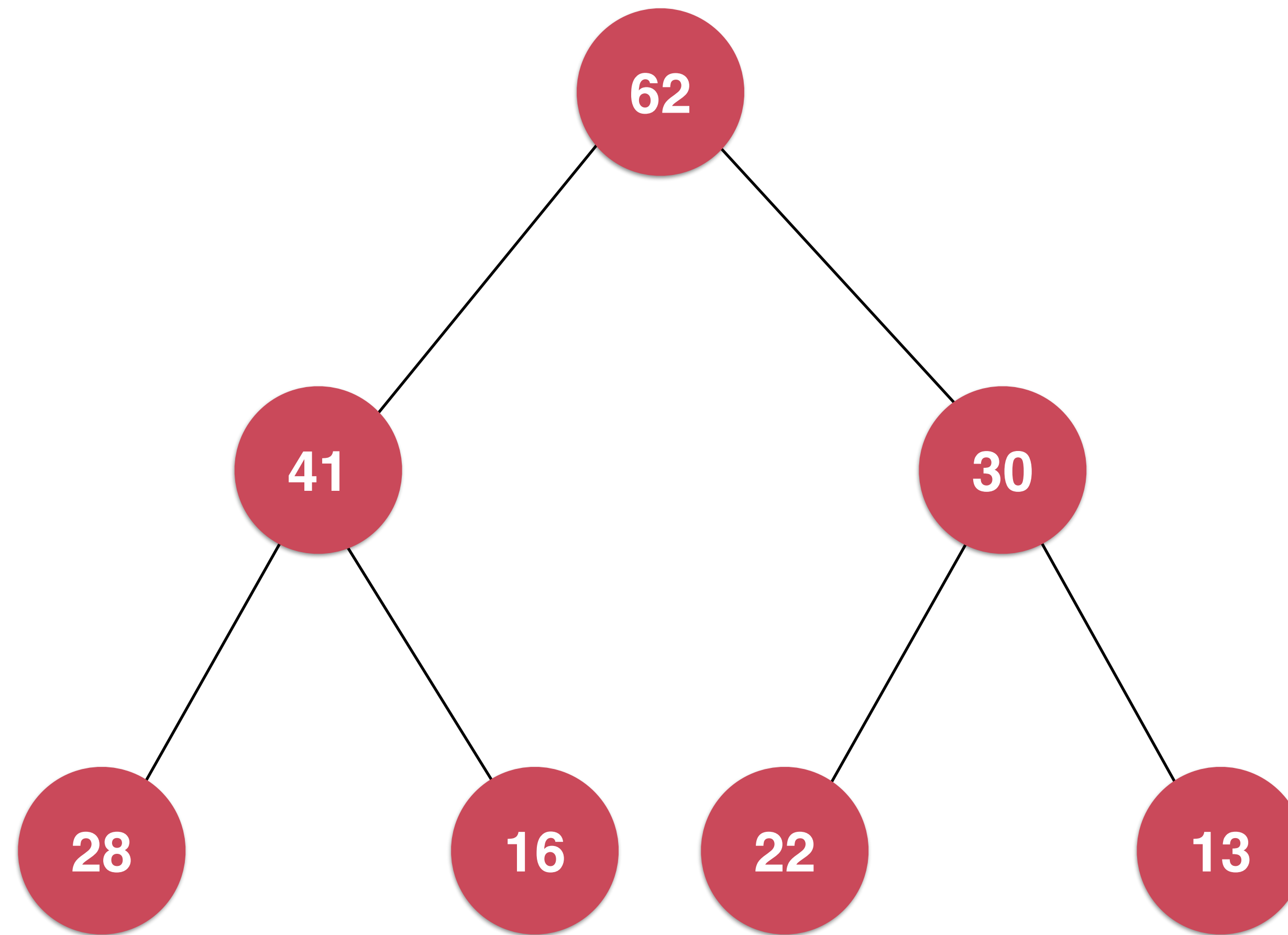
可以使用不同的底层实现

优先队列

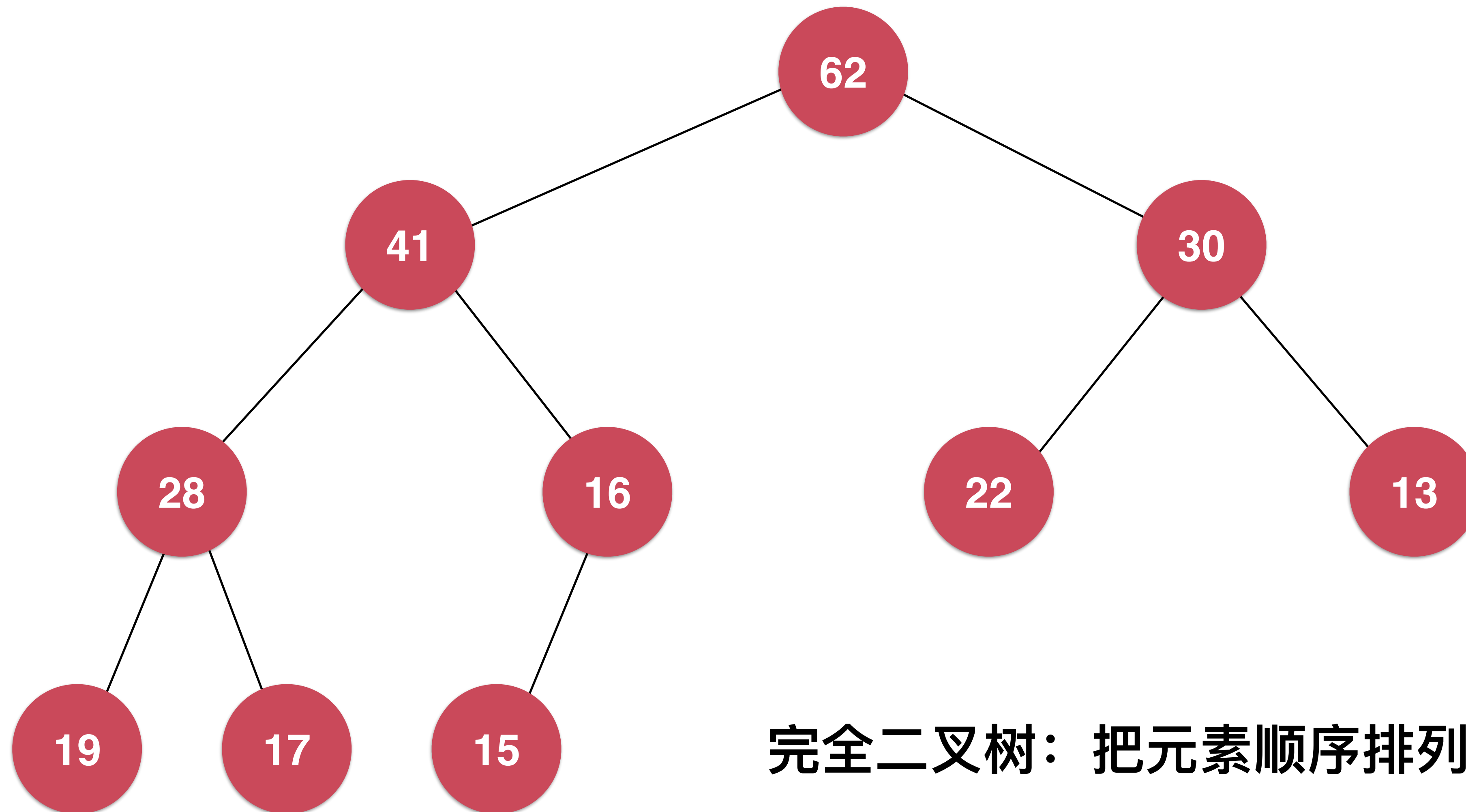
	入队	出队 (拿出最大元素)
普通线性结构	$O(1)$	$O(n)$
顺序线性结构	$O(n)$	$O(1)$
堆	$O(\log n)$	$O(\log n)$

堆的基本结构

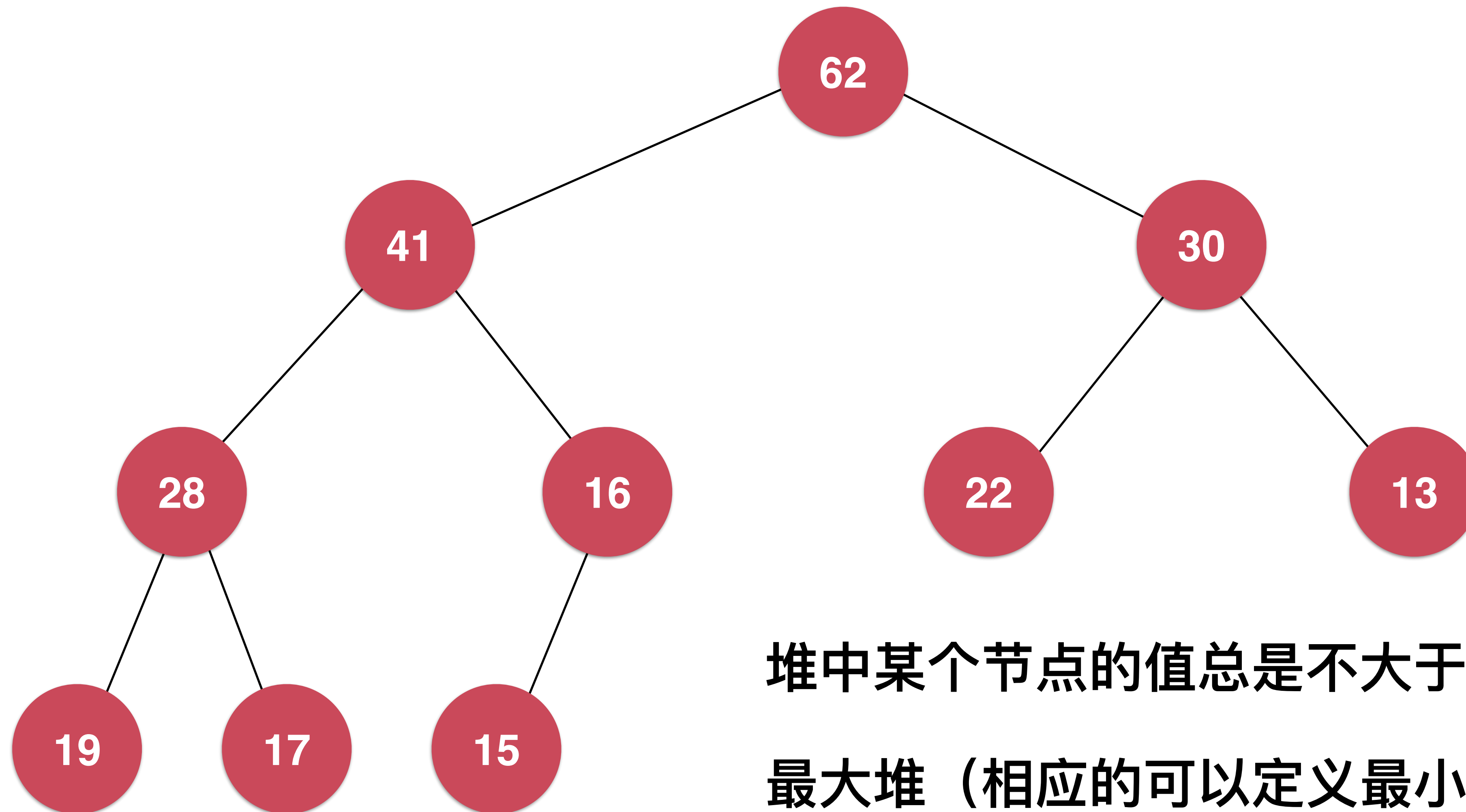
二叉堆 Binary Heap



二叉堆是一棵完全二叉树



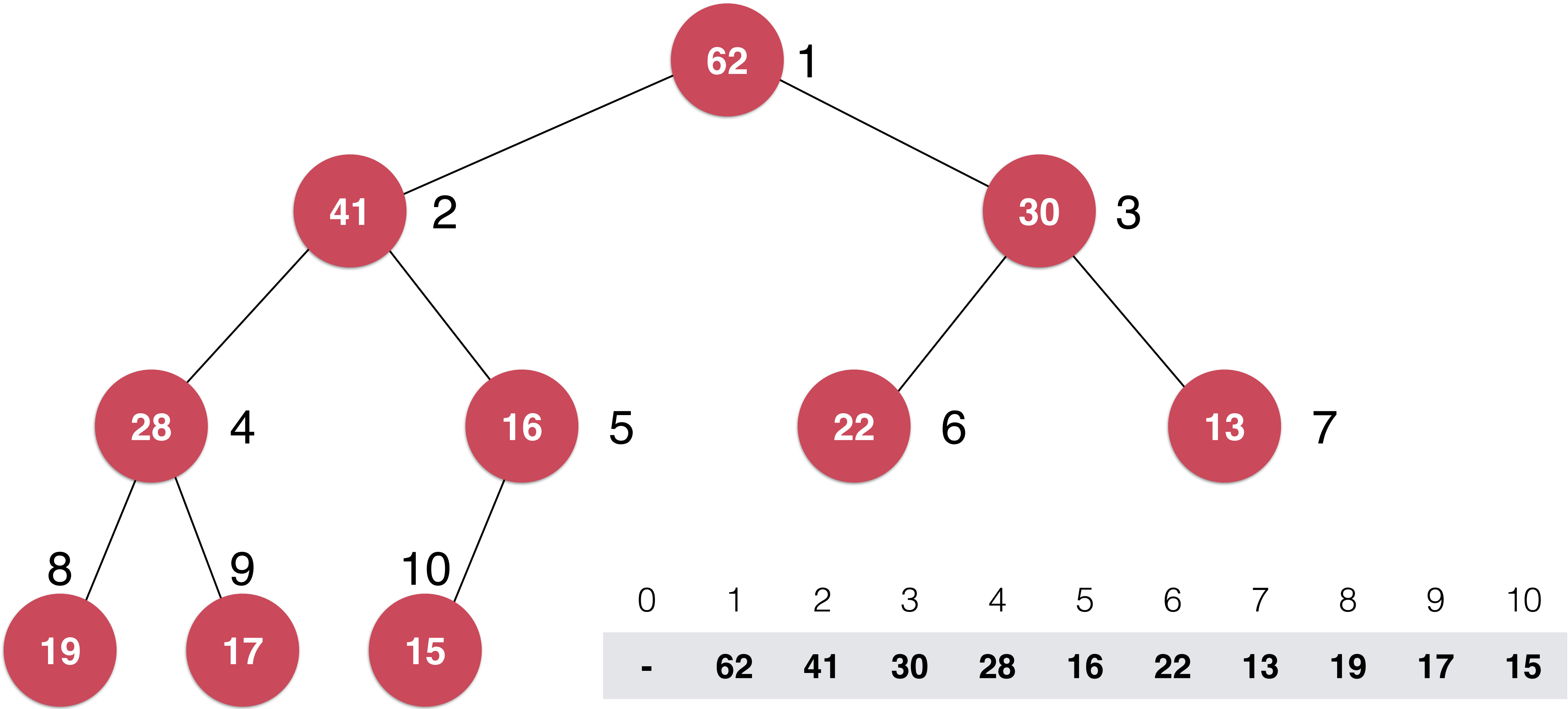
二叉堆的性质



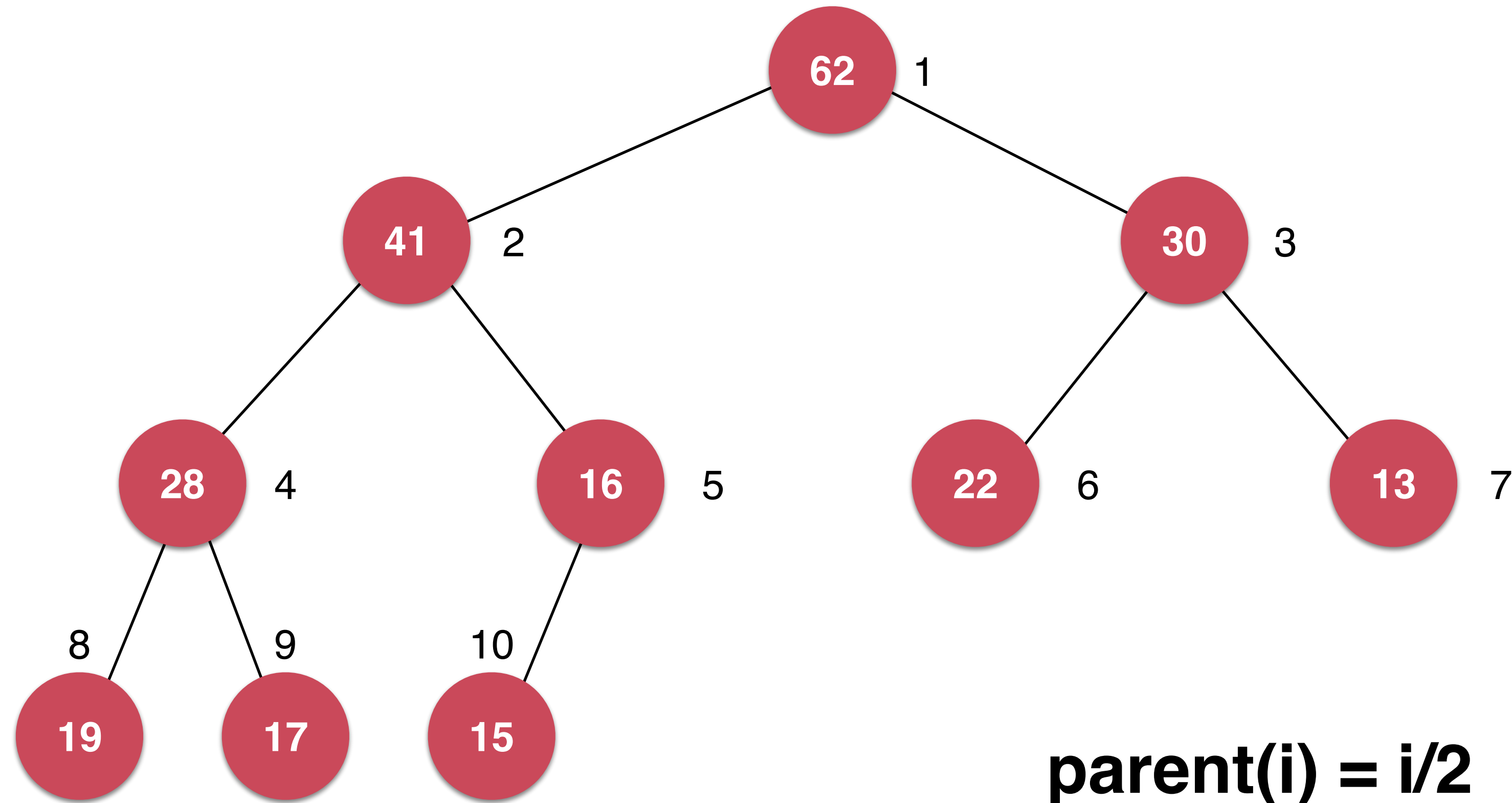
堆中某个节点的值总是不大于其父节点的值

最大堆（相应的可以定义最小堆）

最大堆



用数组存储二叉堆



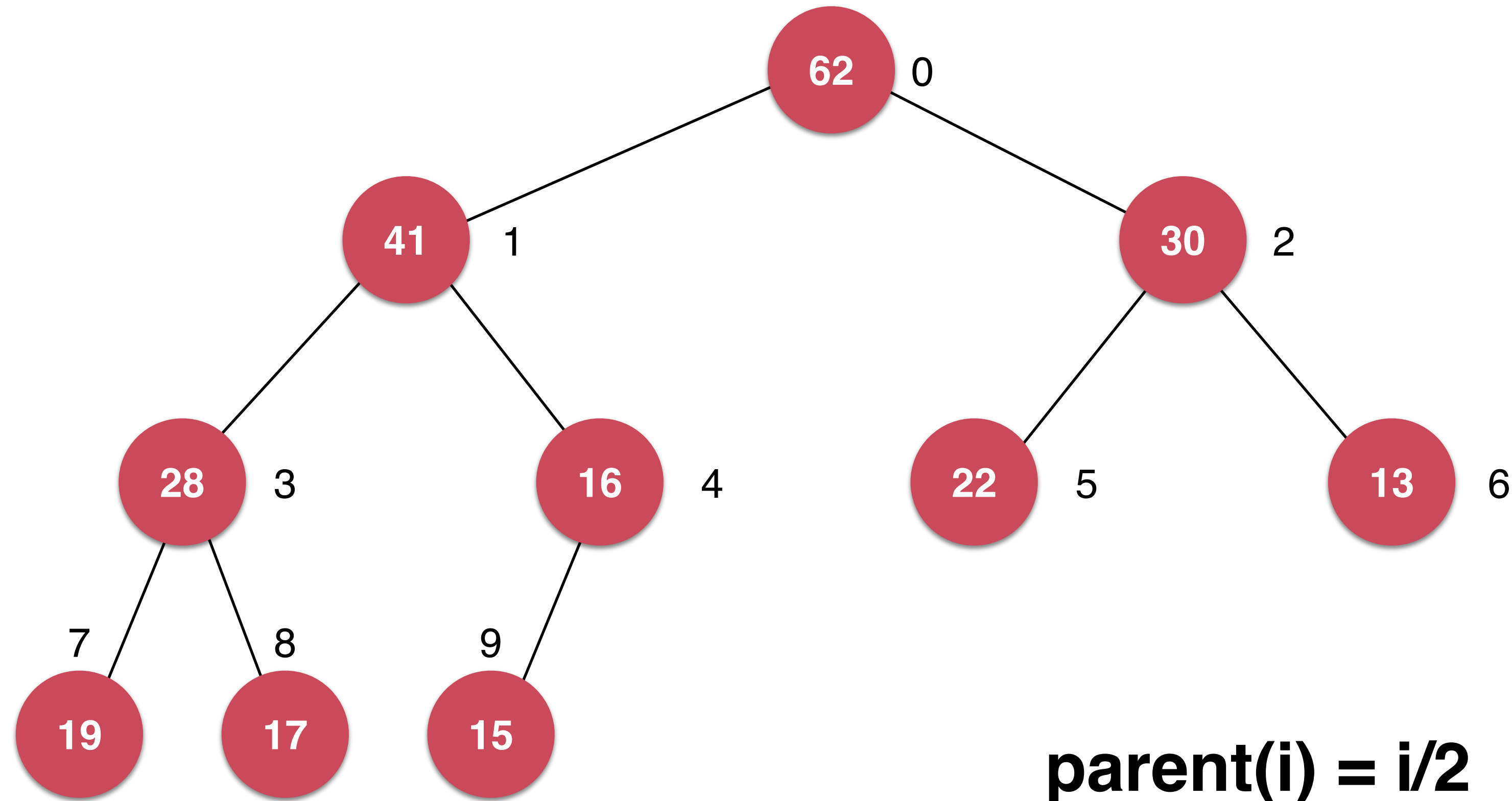
$$\text{parent}(i) = i/2$$

$$\text{left child } (i) = 2*i$$

$$\text{right child } (i) = 2*i + 1$$

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



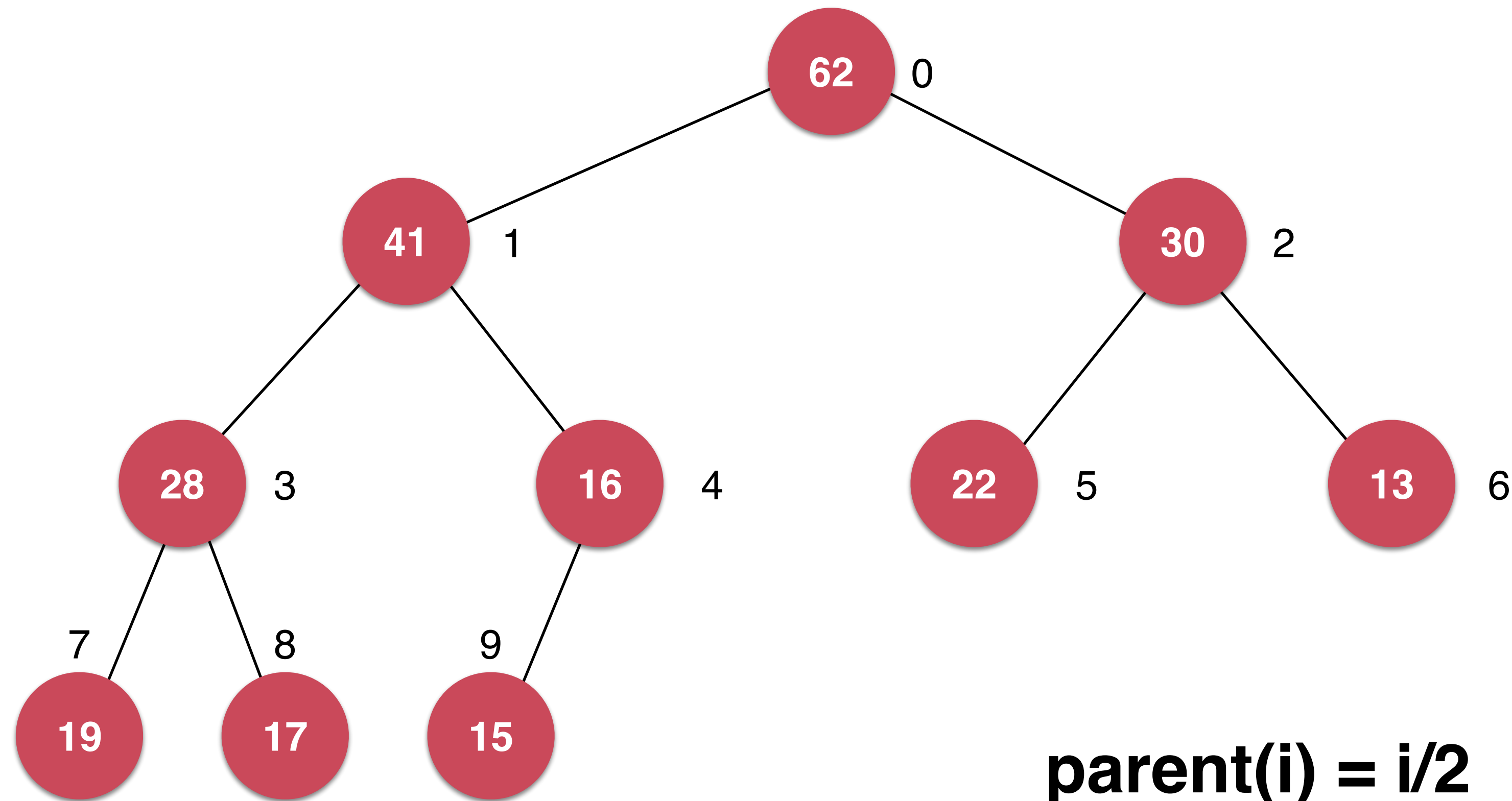
$$\text{parent}(i) = i/2$$

$$\text{left child } (i) = 2*i$$

$$\text{right child } (i) = 2*i + 1$$

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

用数组存储二叉堆



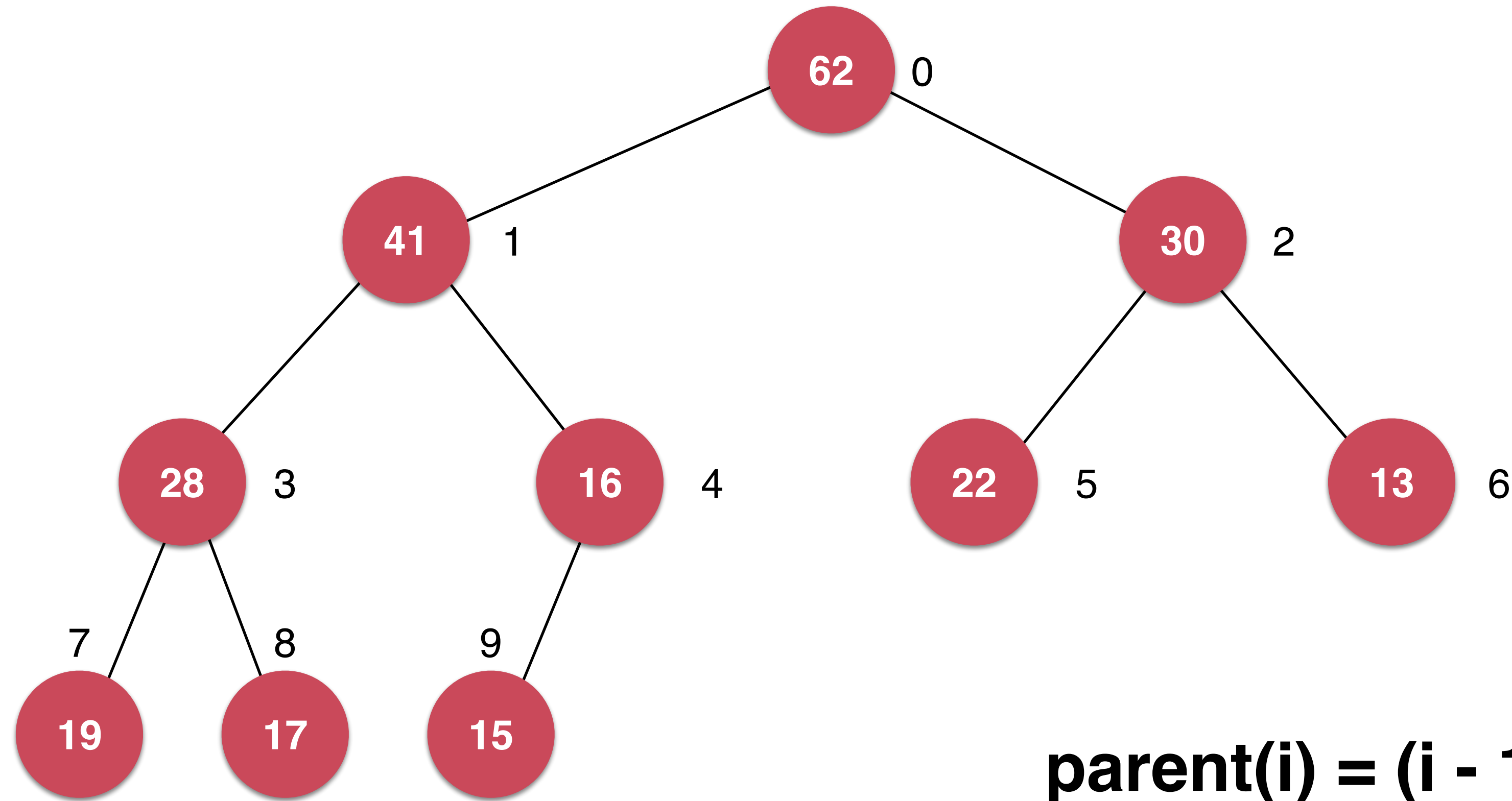
$$\text{parent}(i) = i/2$$

$$\text{left child } (i) = 2*i$$

$$\text{right child } (i) = 2*i + 1$$

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

用数组存储二叉堆



$$\text{parent}(i) = (i - 1)/2$$

$$\text{left child } (i) = 2*i + 1$$

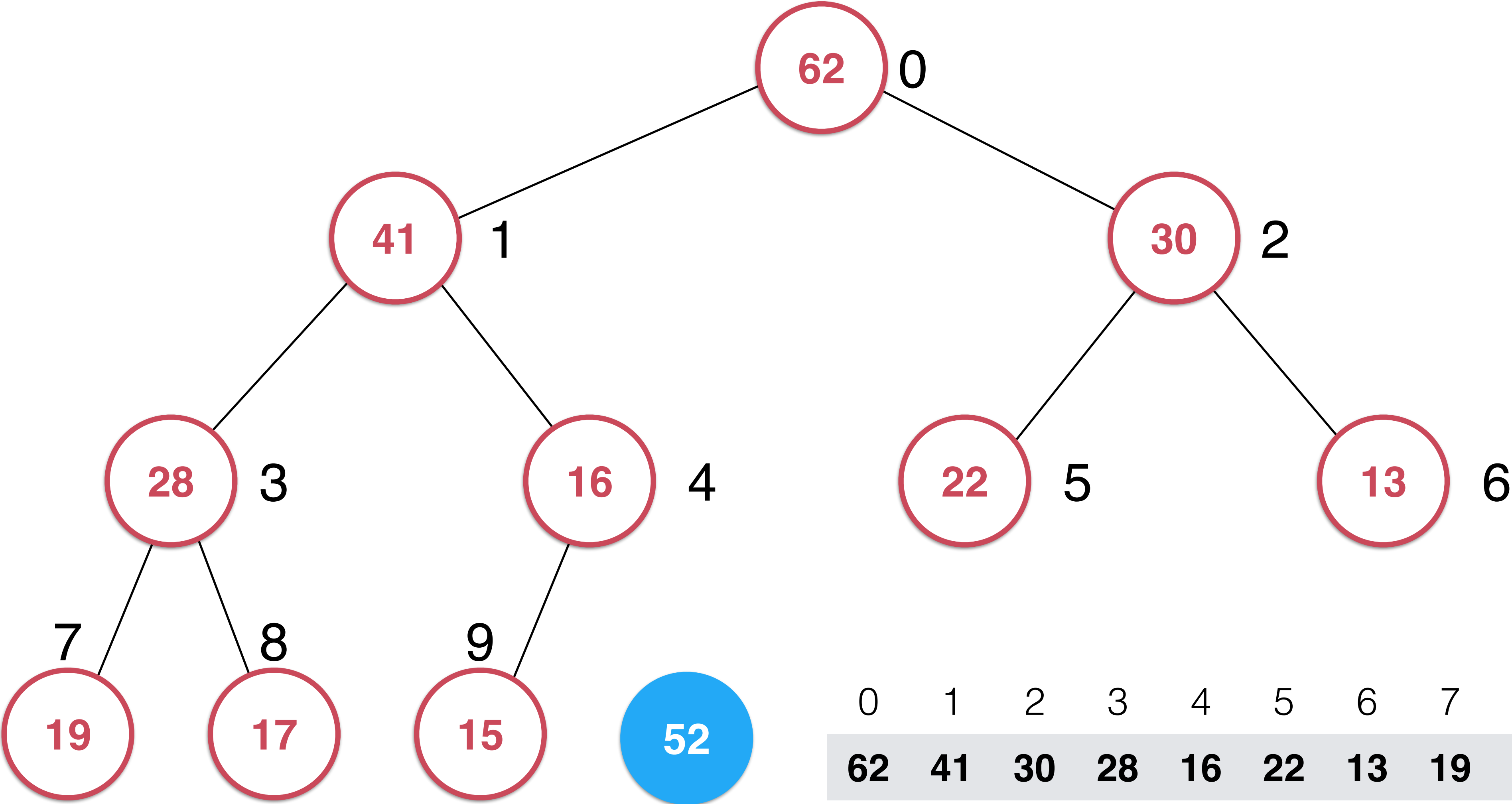
$$\text{right child } (i) = 2*i + 2$$

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

实践：堆的基本框架

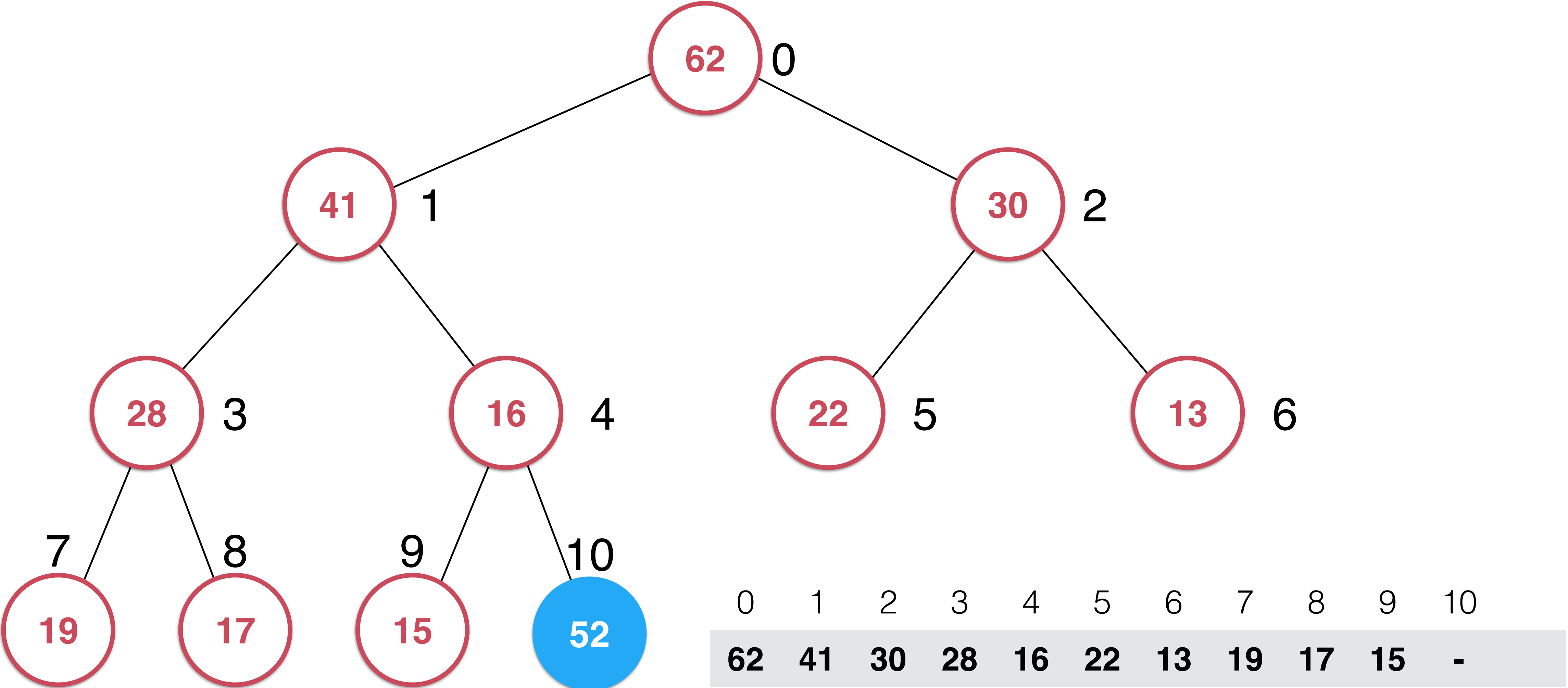
向堆中添加元素和Sift Up

Sift Up

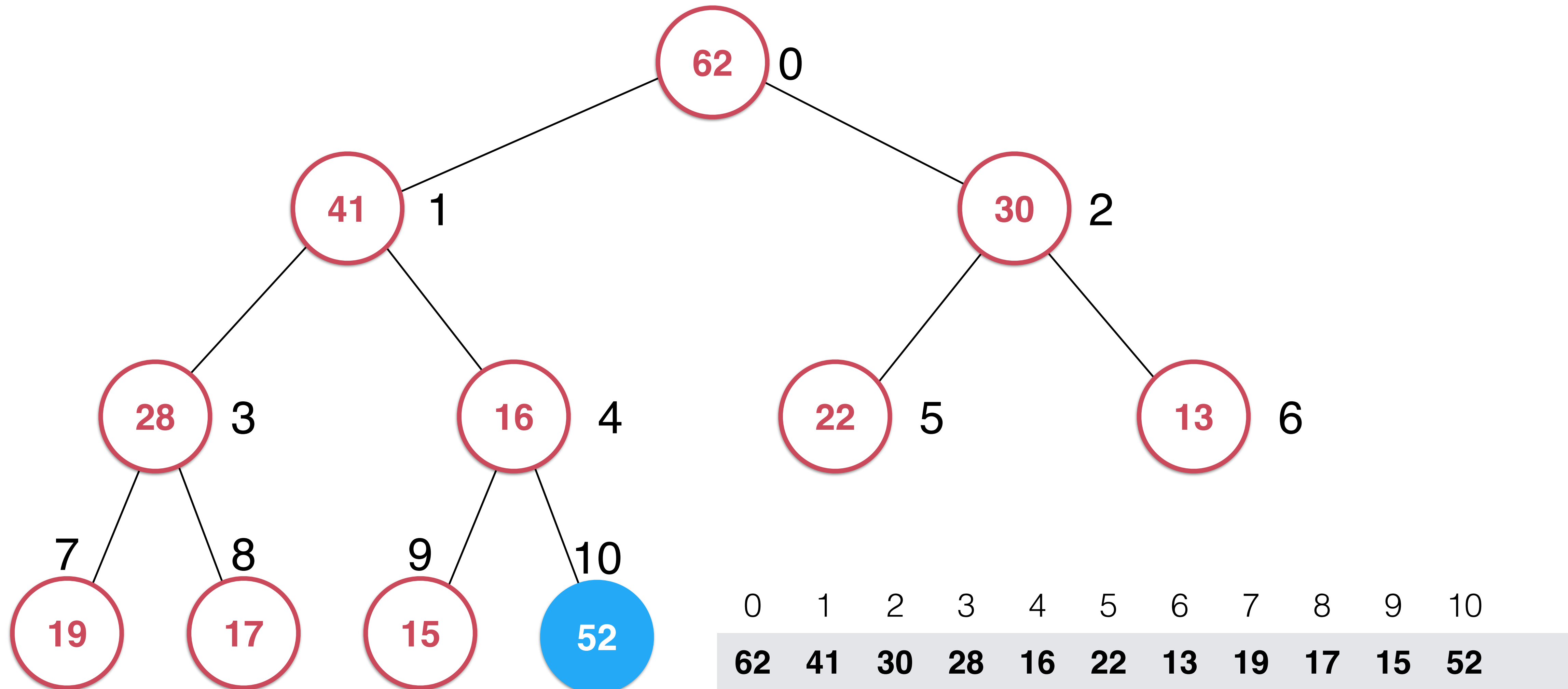


0	1	2	3	4	5	6	7	8	9	10
62	41	30	28	16	22	13	19	17	15	-

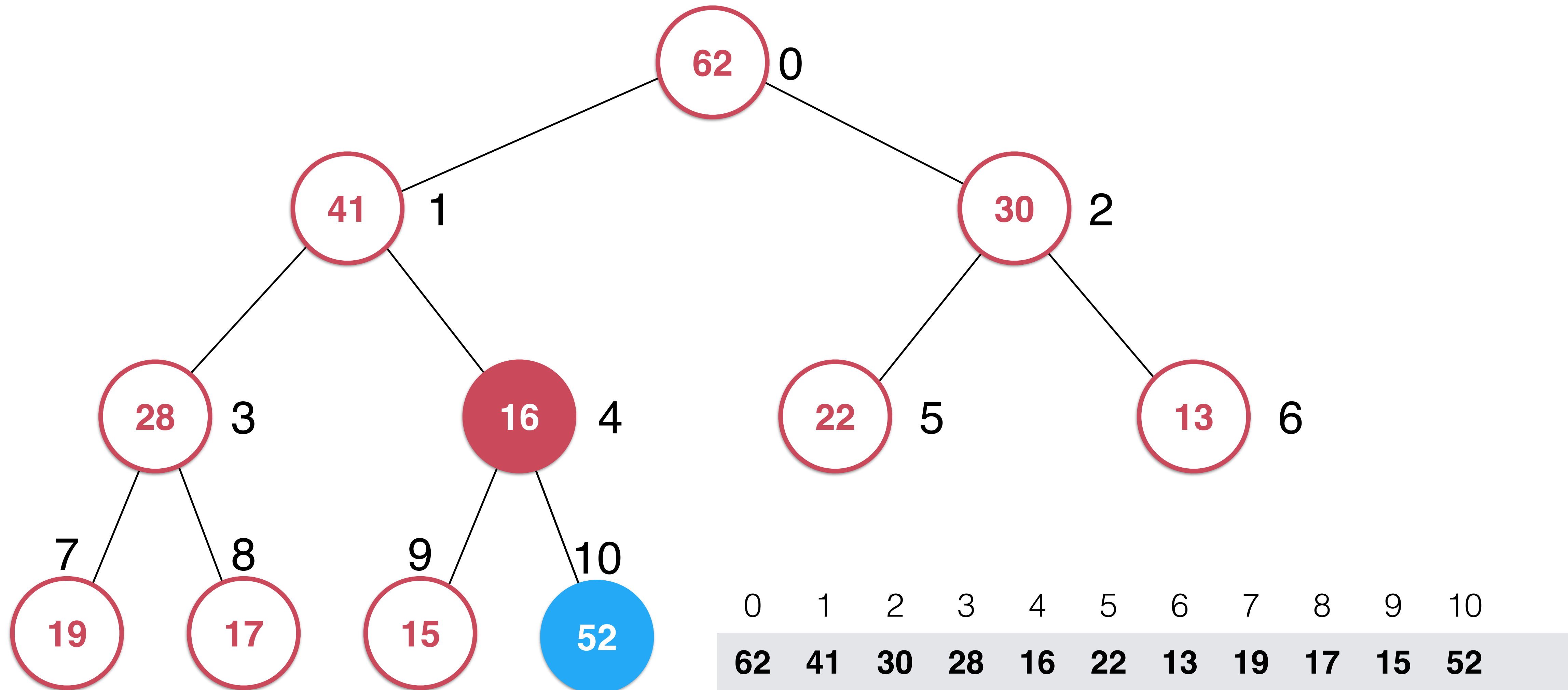
Sift Up



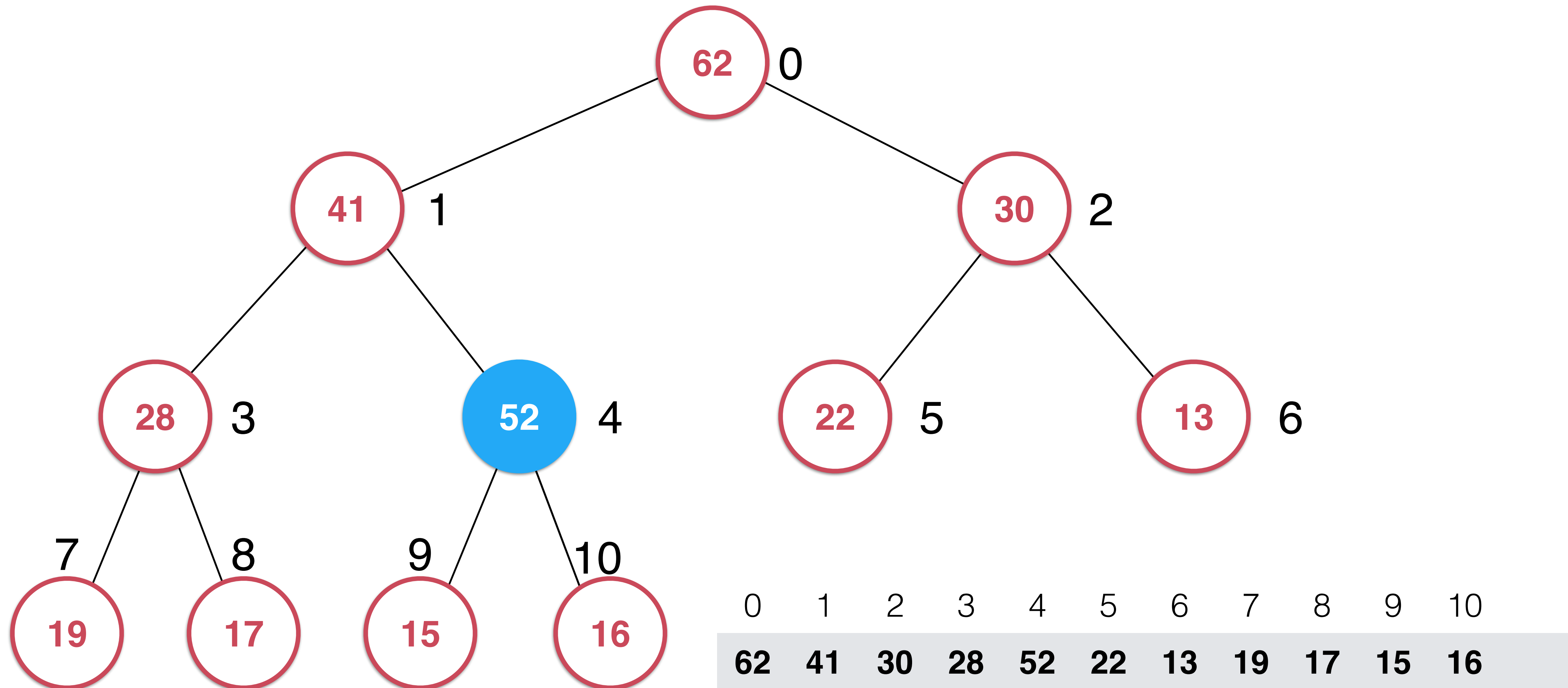
Sift Up



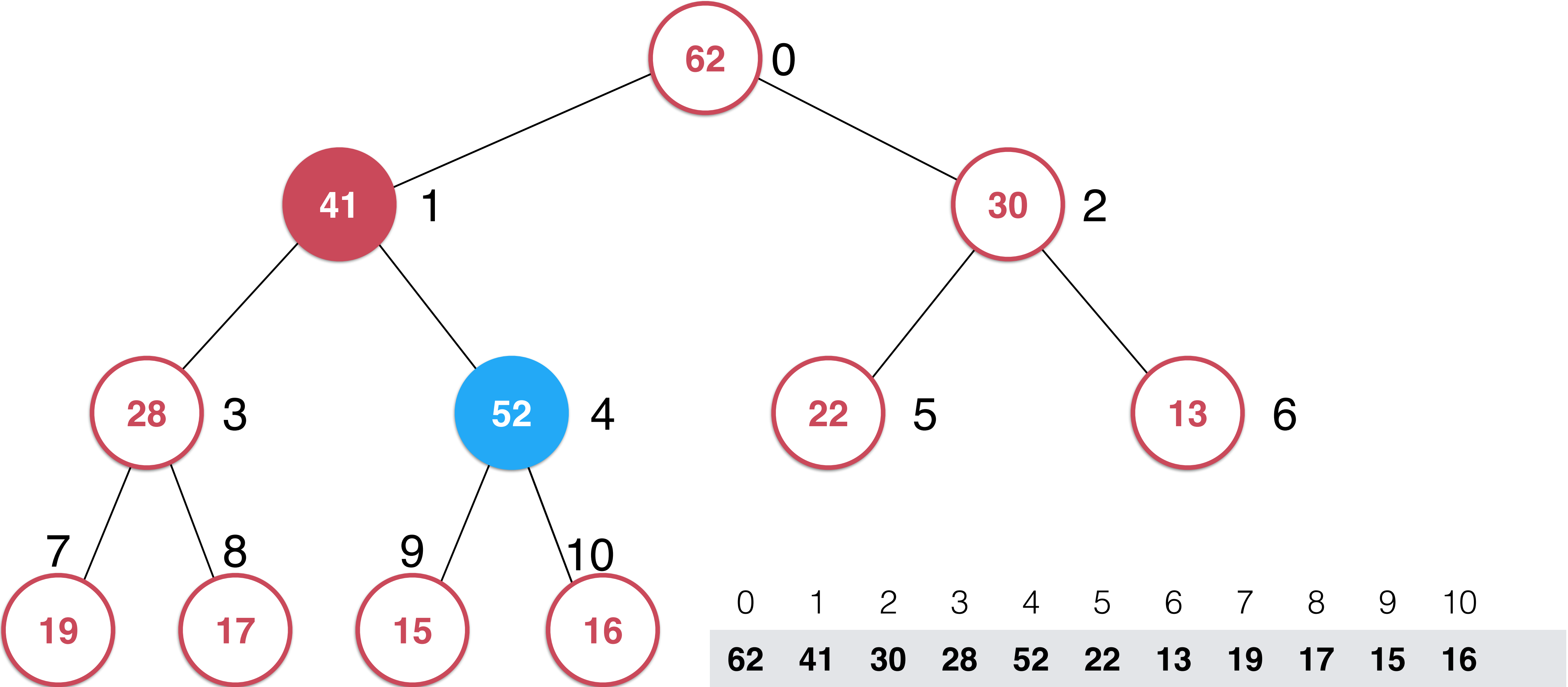
Sift Up



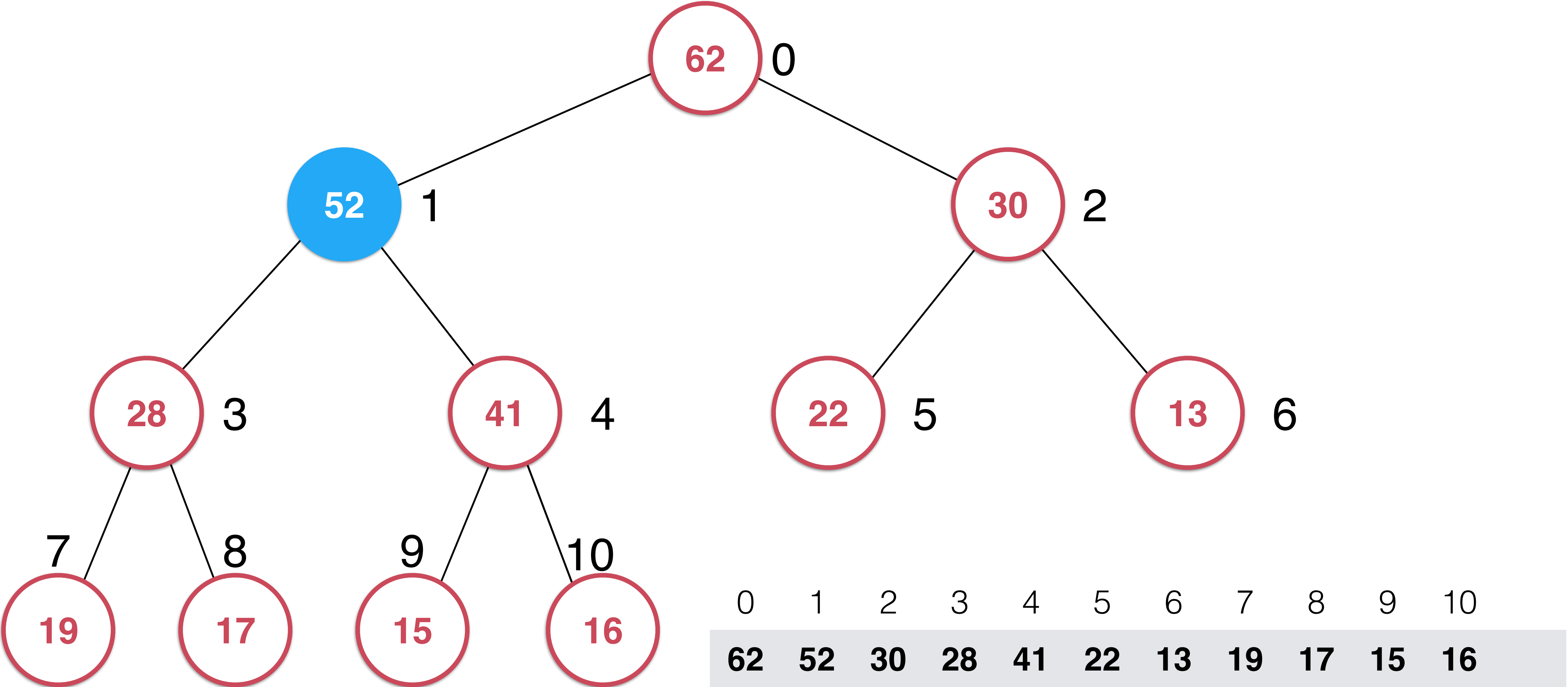
Sift Up



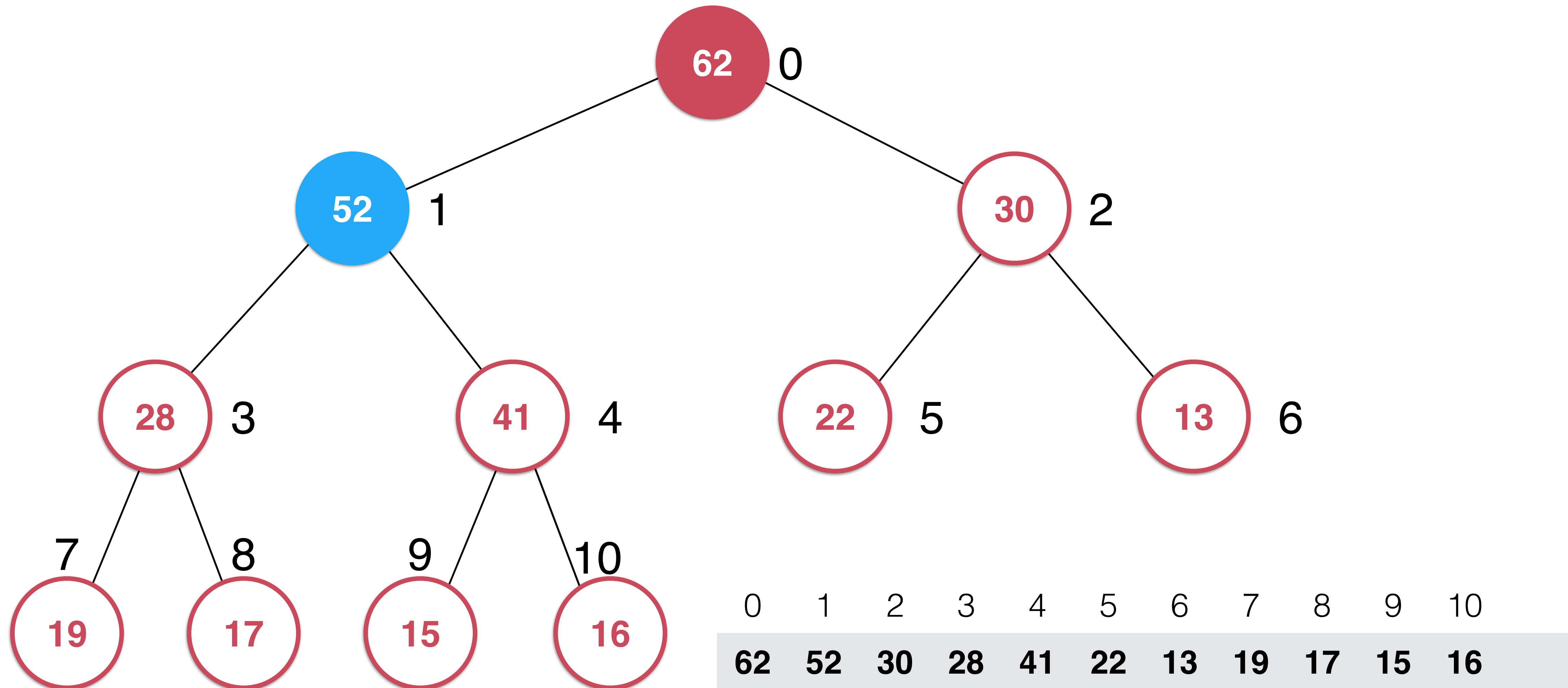
Sift Up



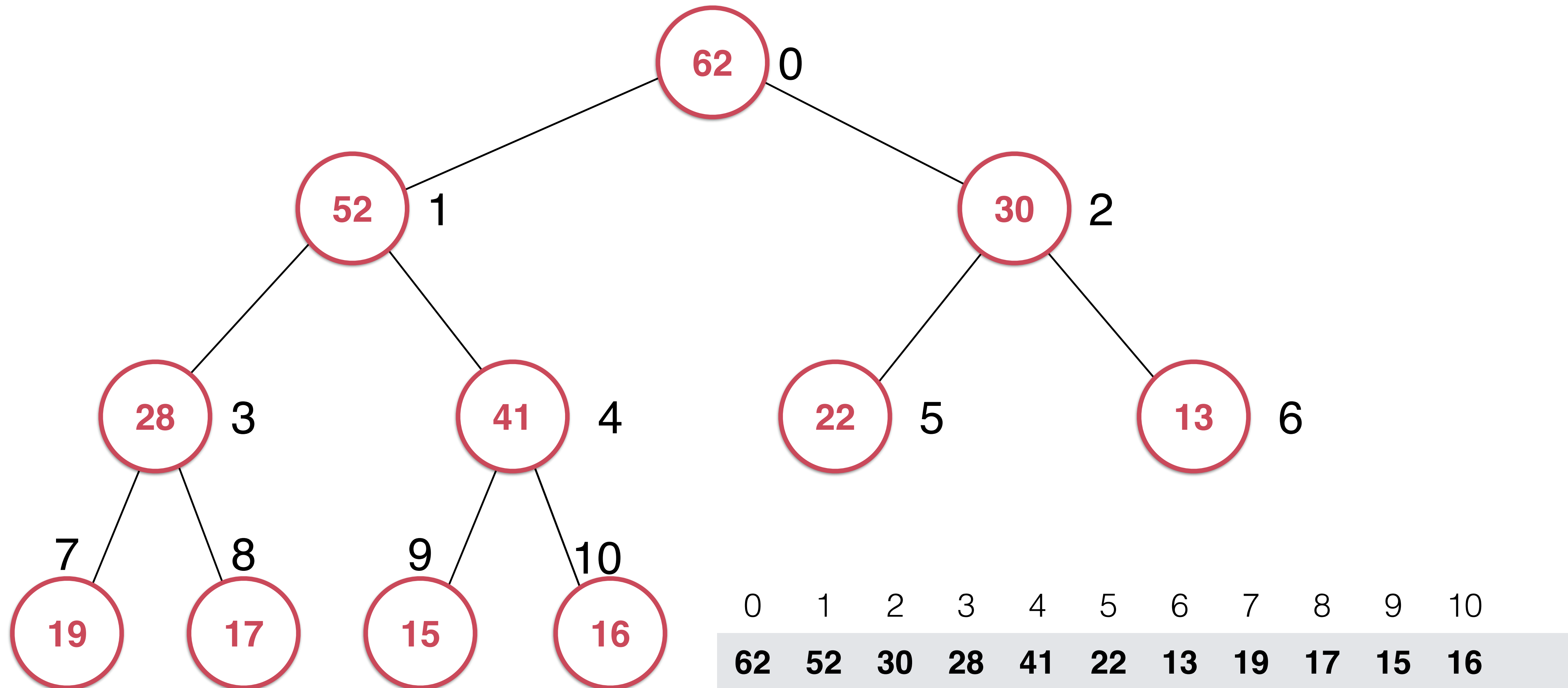
Sift Up



Sift Up



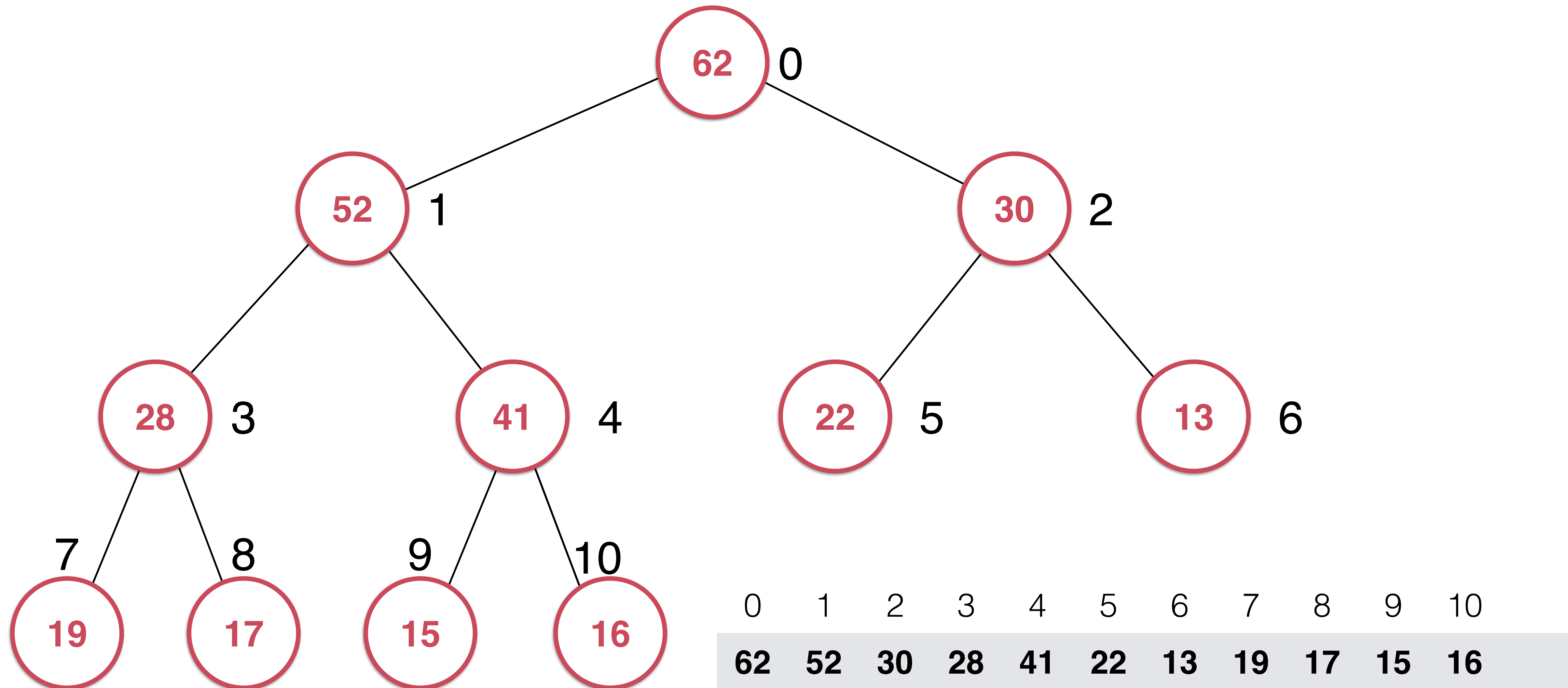
Sift Up



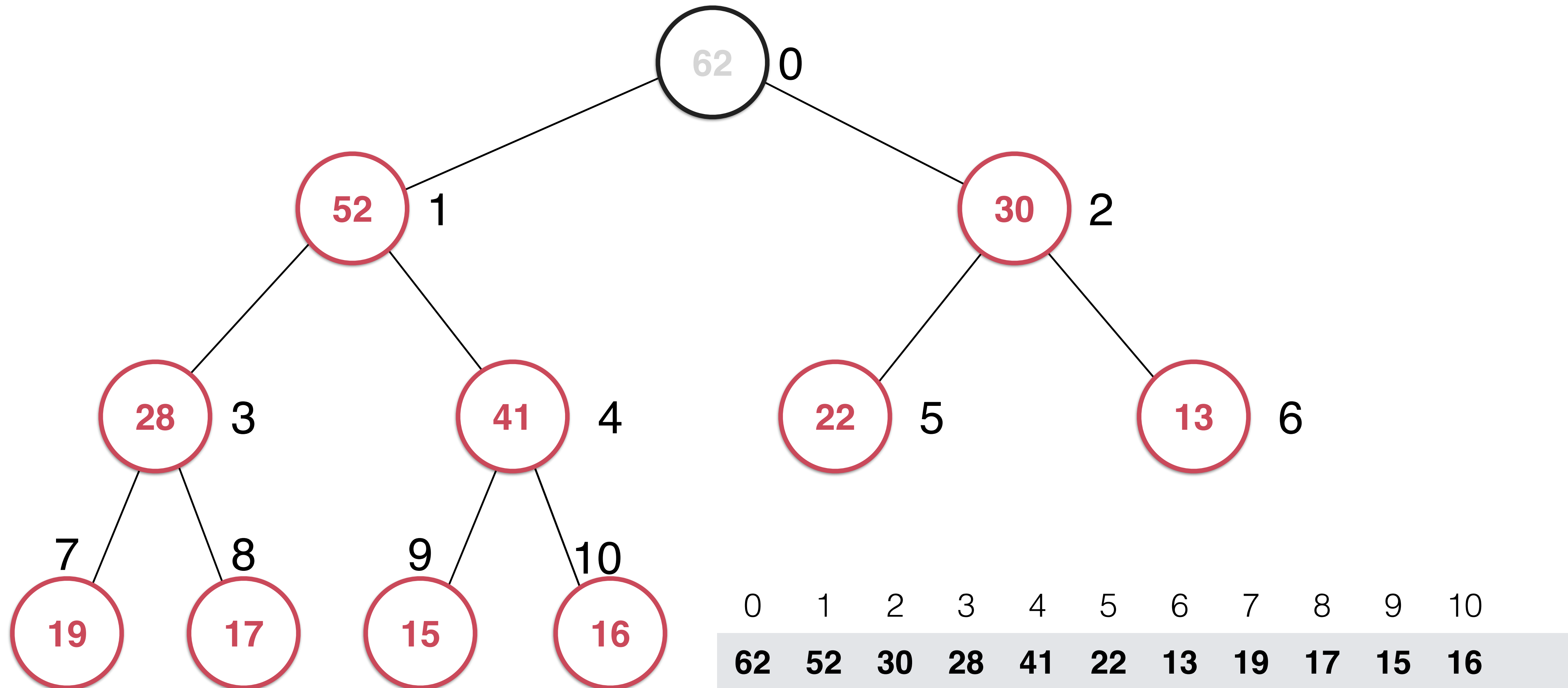
实践： Sift Up 和 add

取出堆中的最大元素和Sift Down

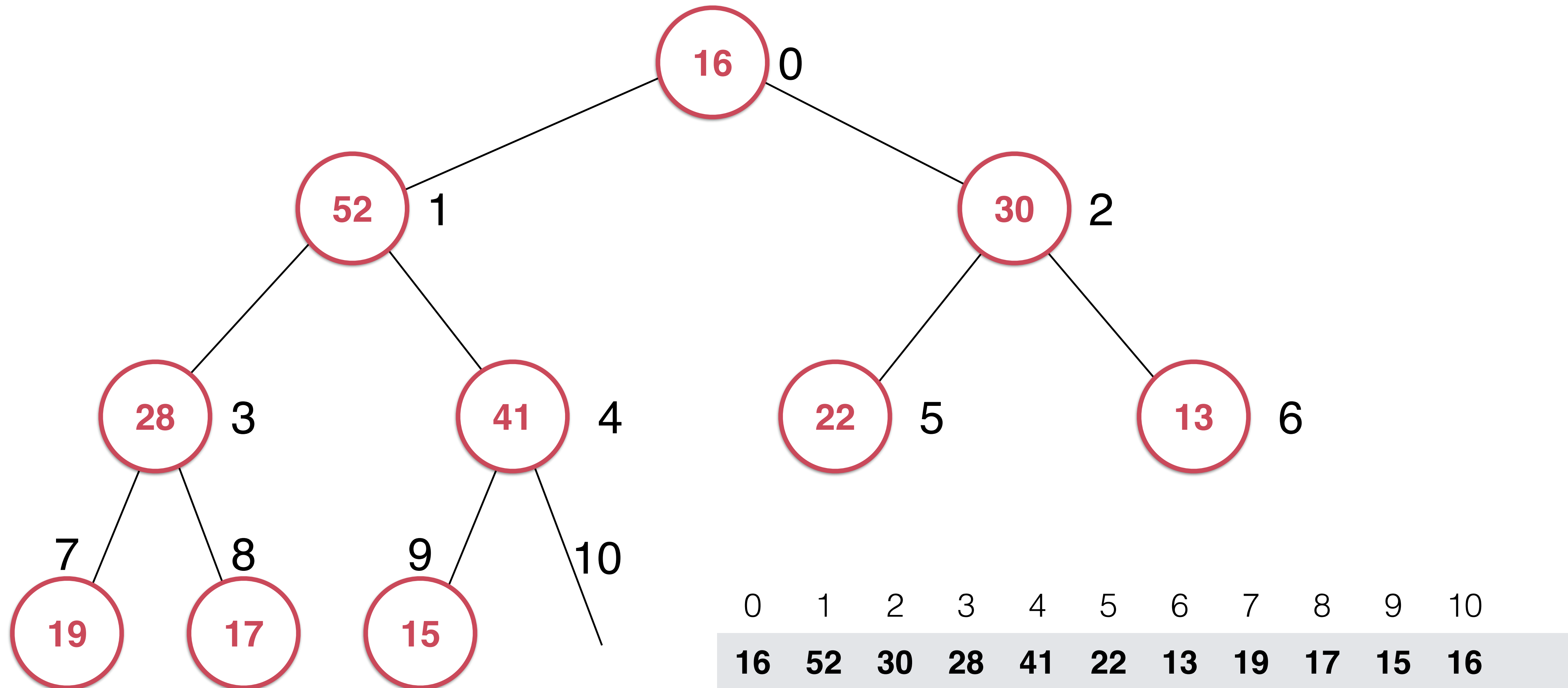
Sift Down



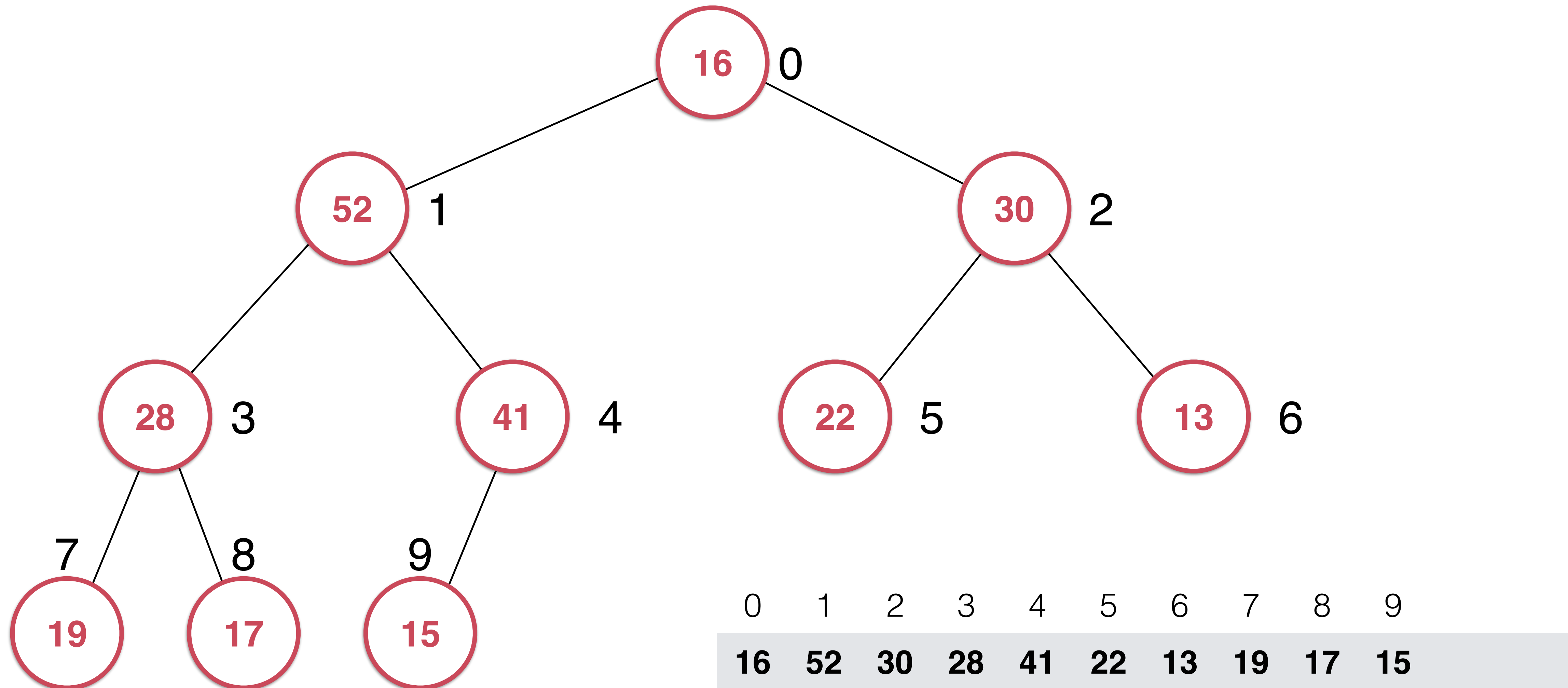
Sift Down



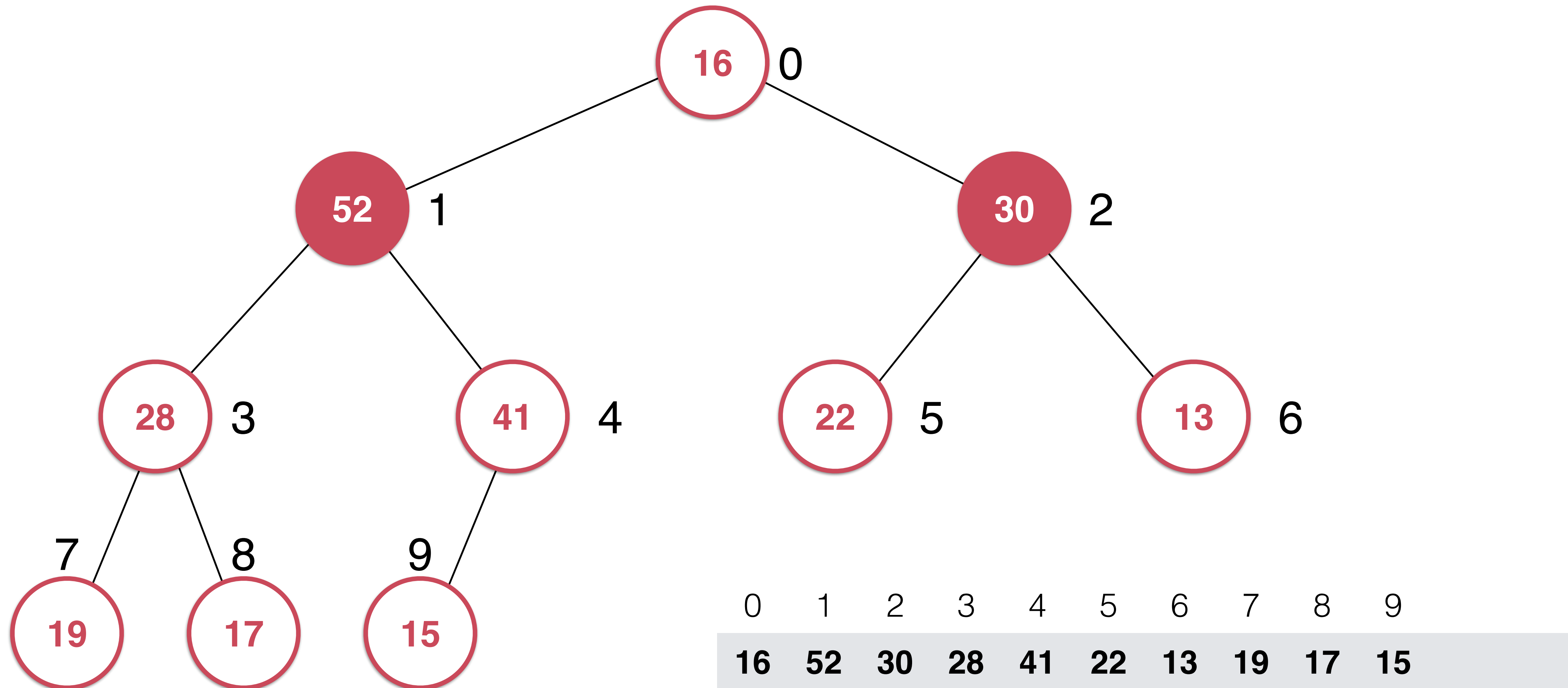
Sift Down



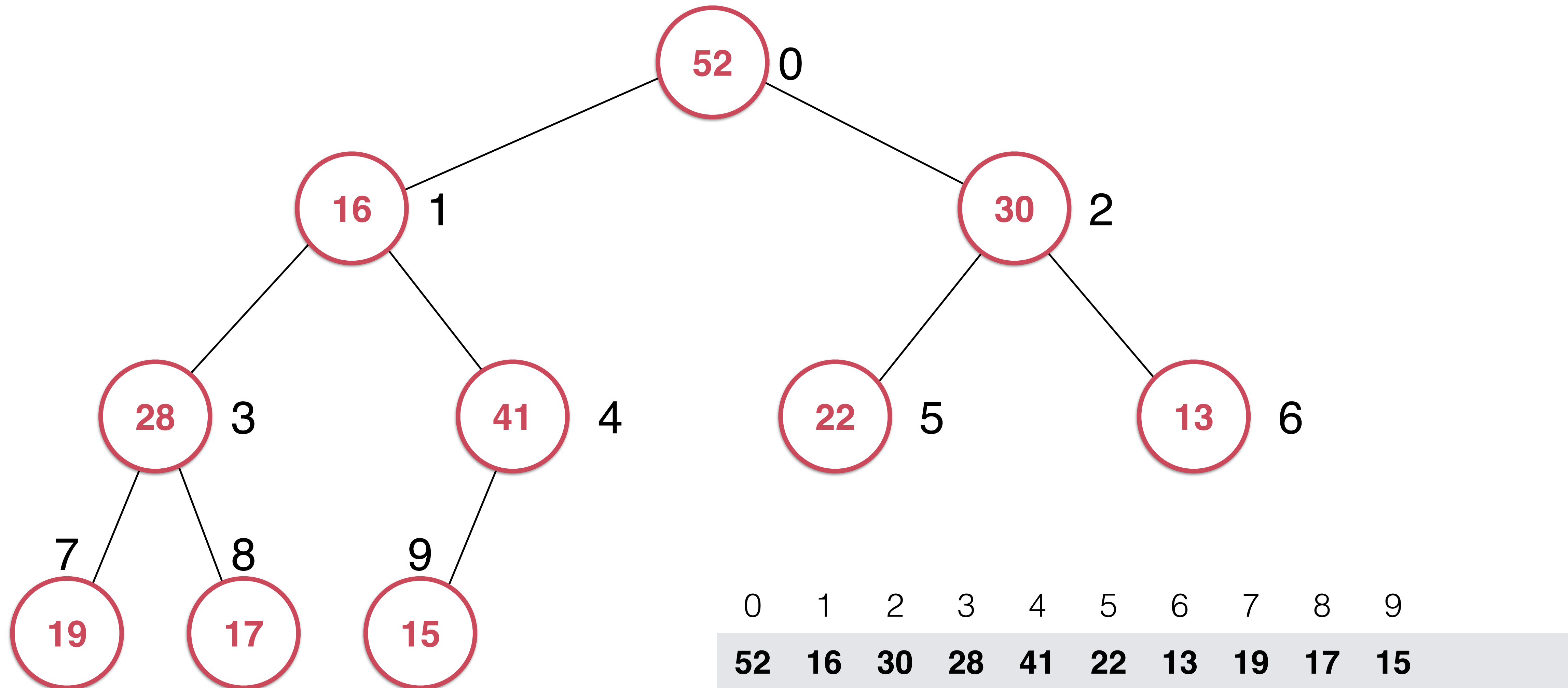
Sift Down



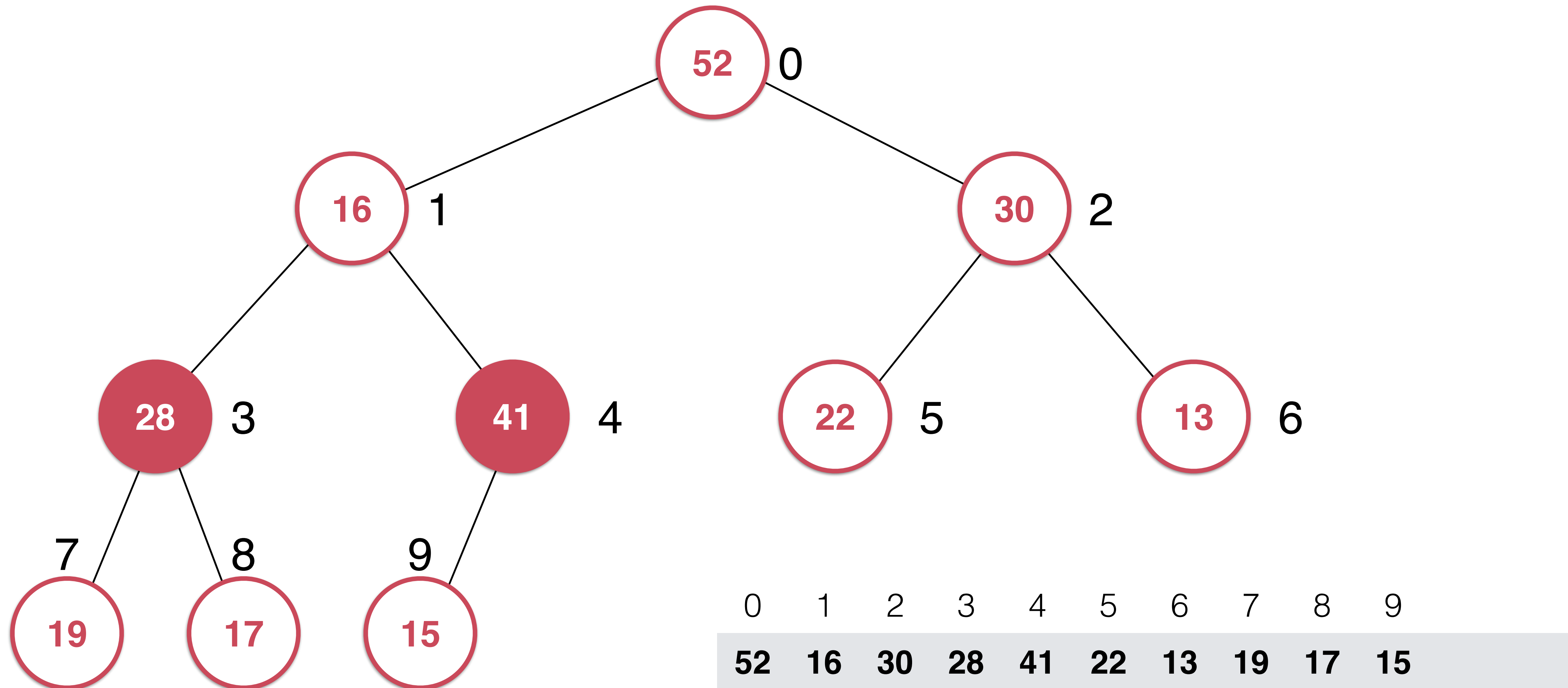
Sift Down



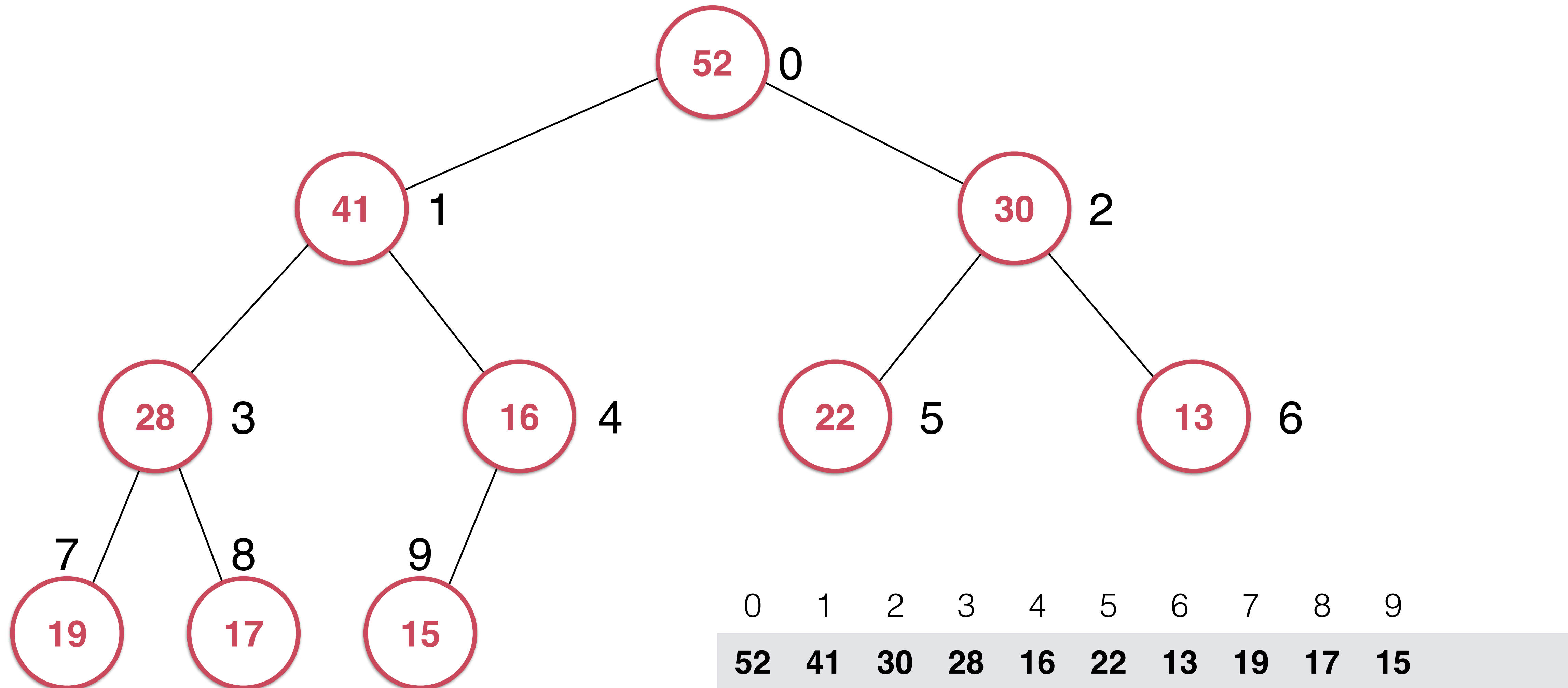
Sift Down



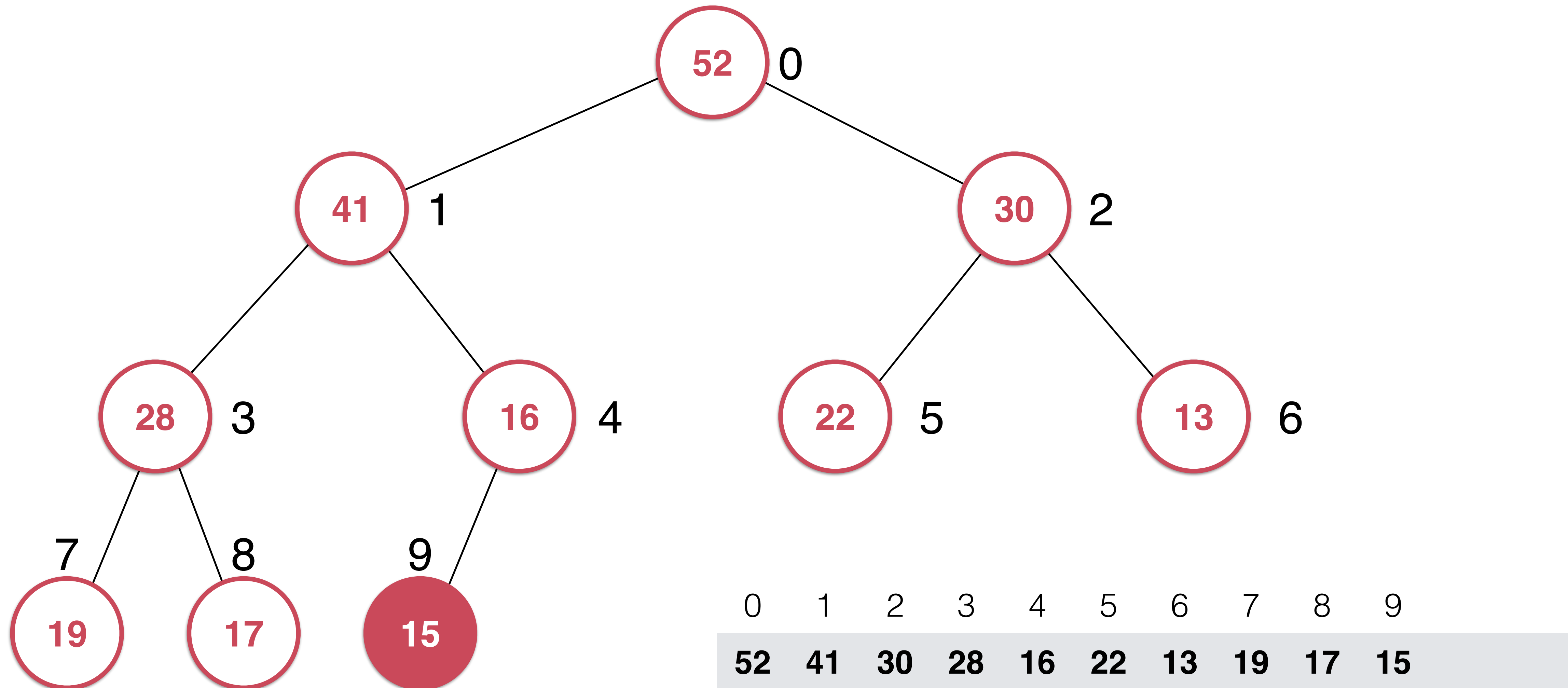
Sift Down



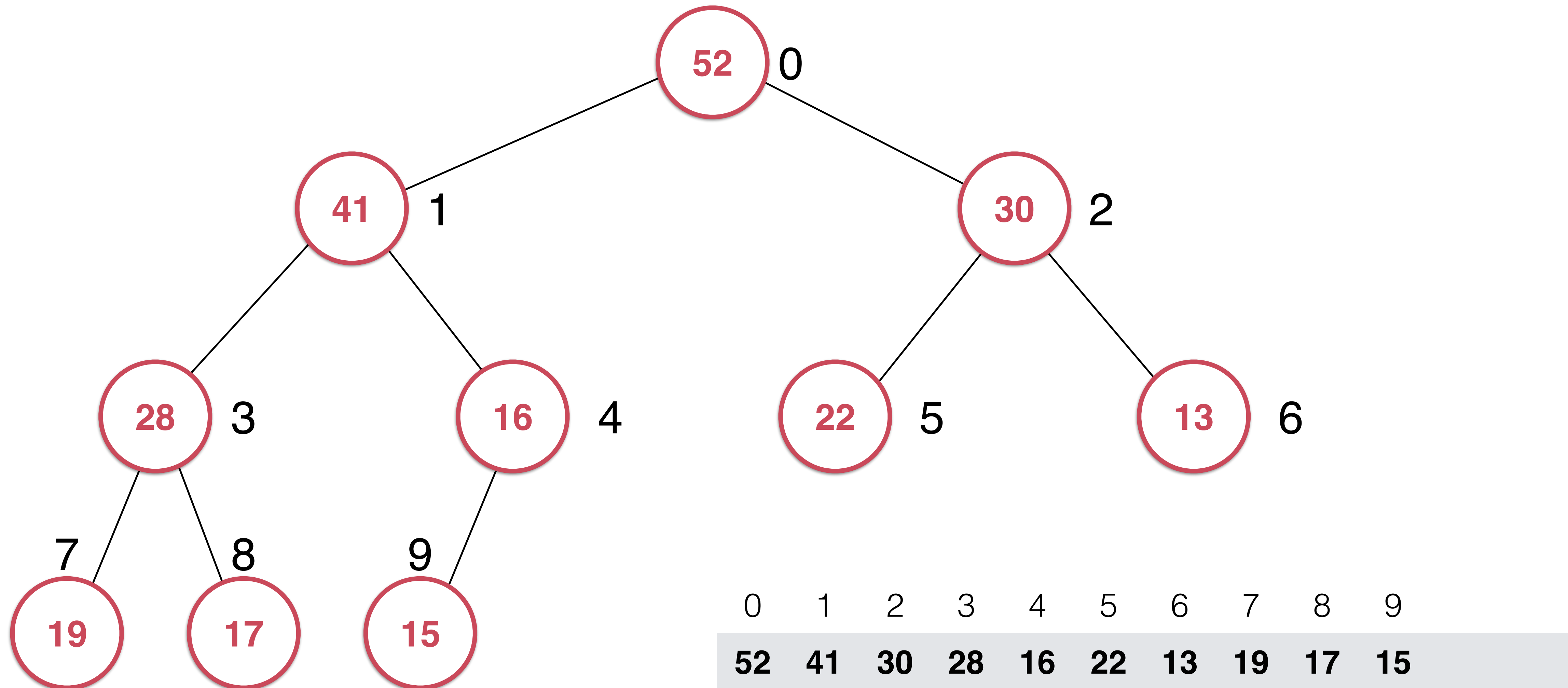
Sift Down



Sift Down



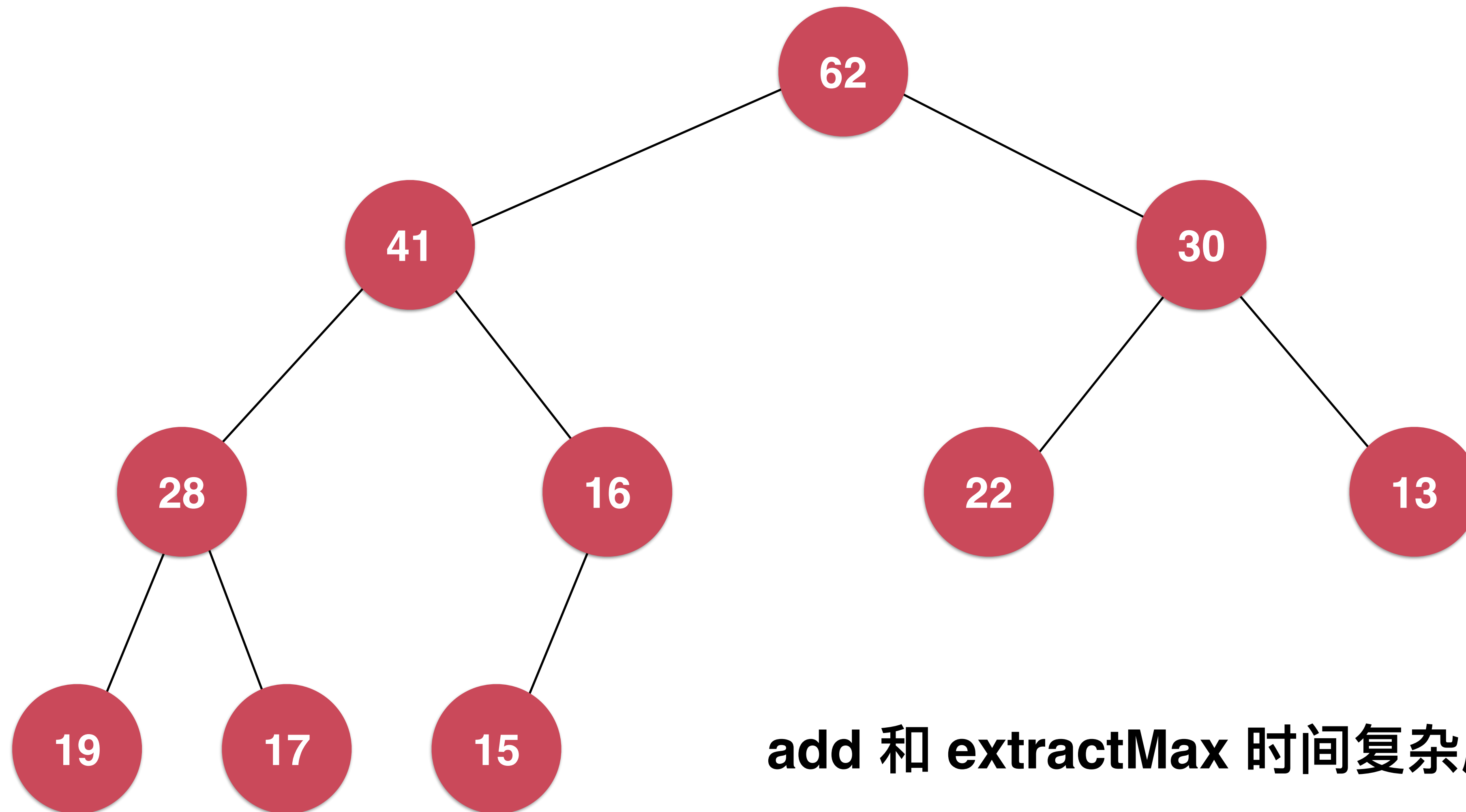
Sift Down



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 $O(\log n)$

最简单的堆排序

实践：最简单的堆排序

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 $O(\log n)$ 的操作

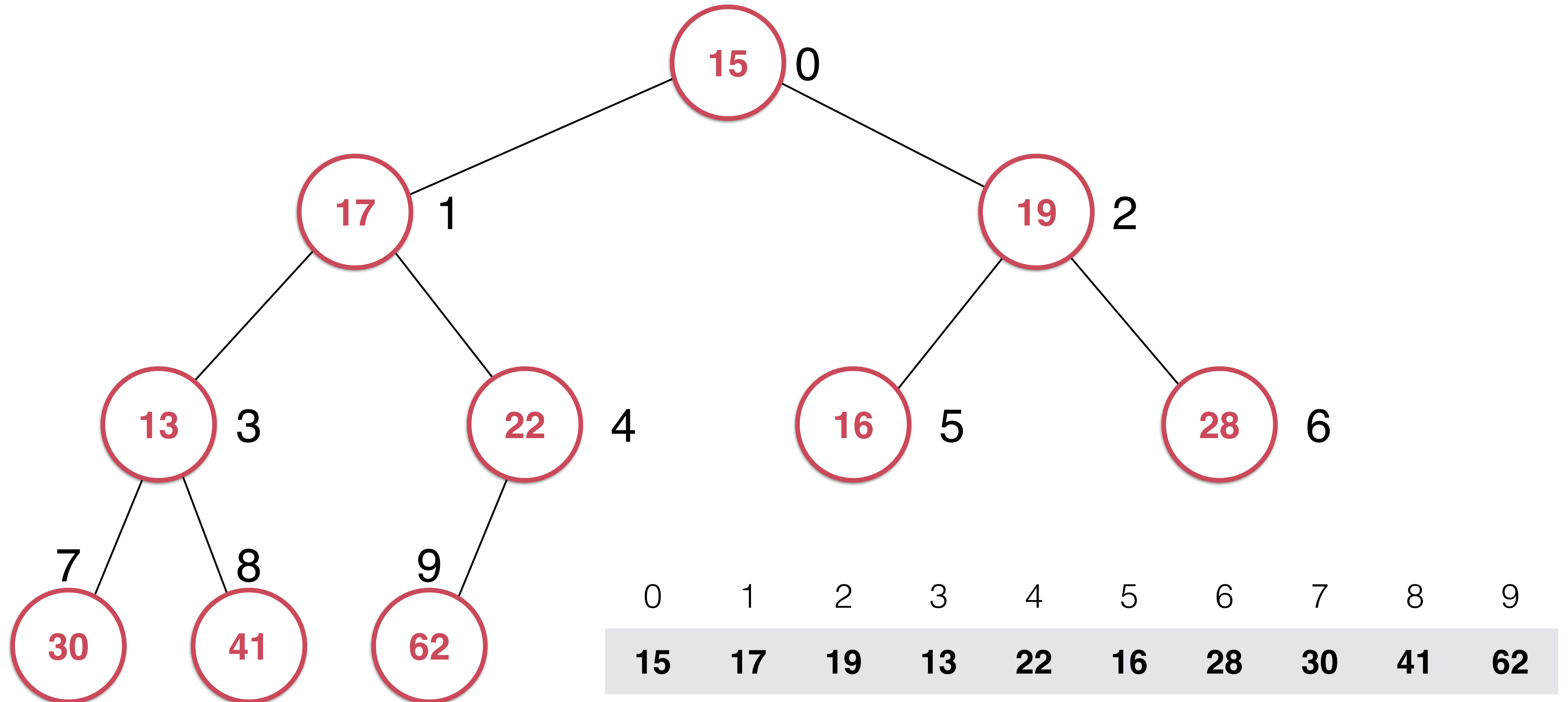
实现: 可以直接将堆顶元素替换以后Sift Down, 一次 $O(\log n)$ 的操作

实践： replace

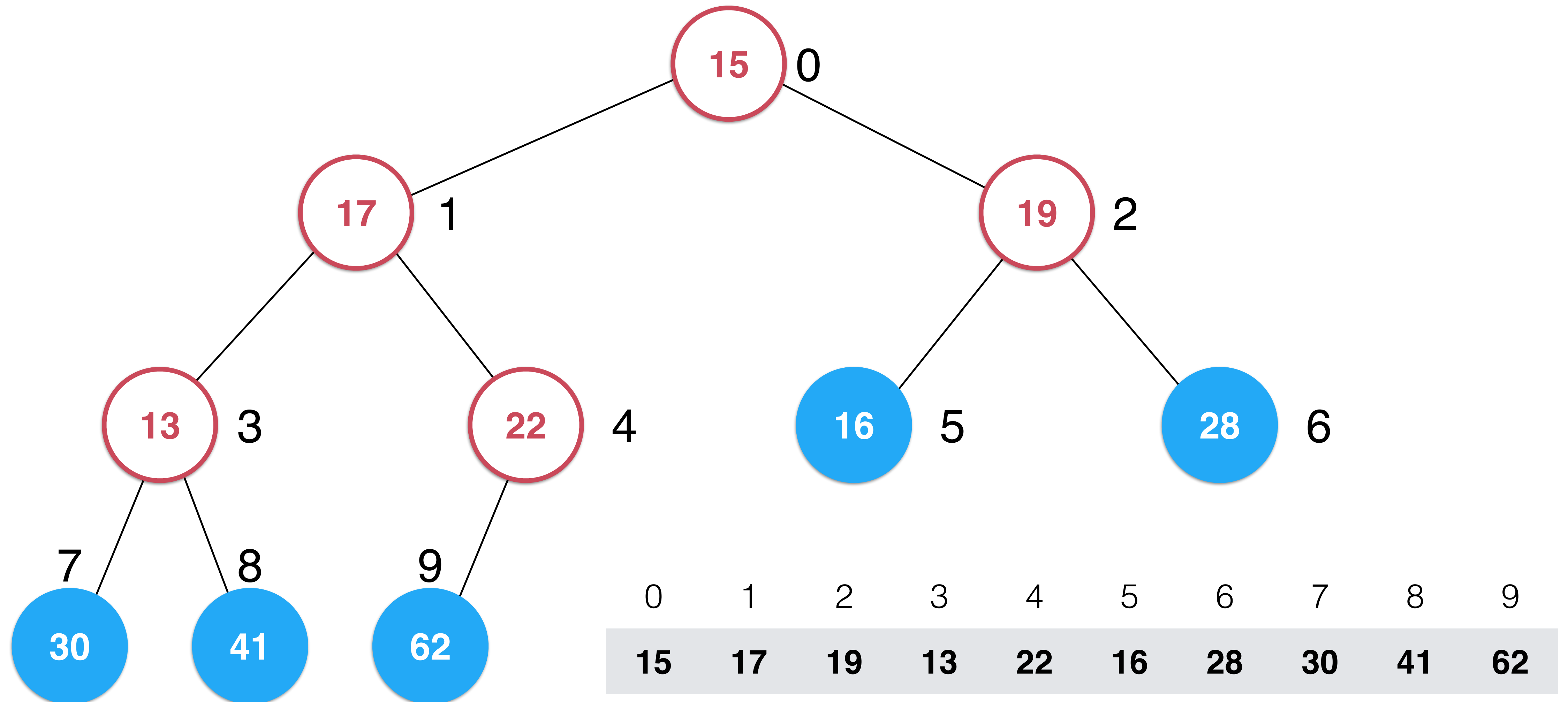
heapify

heapify：将任意数组整理成堆的形状

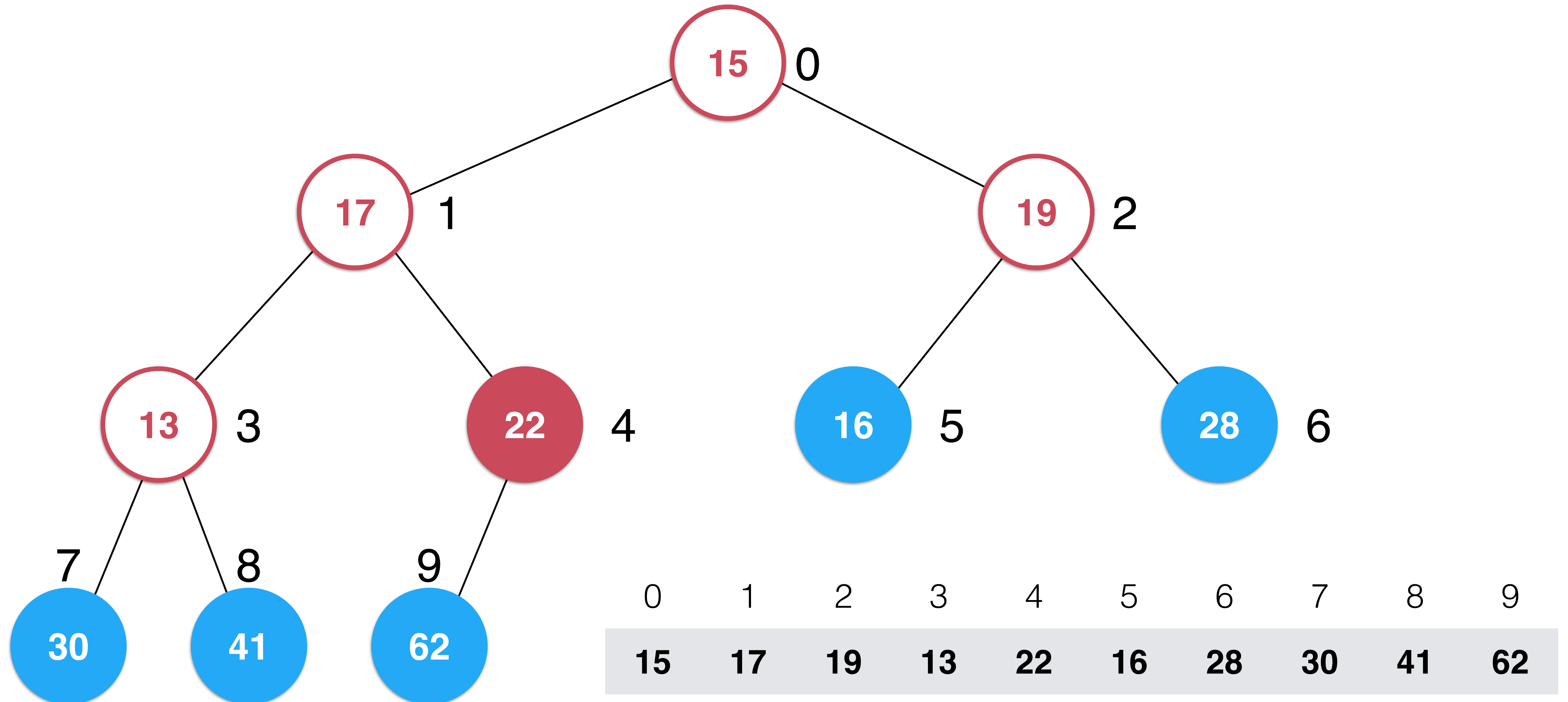
Heapify



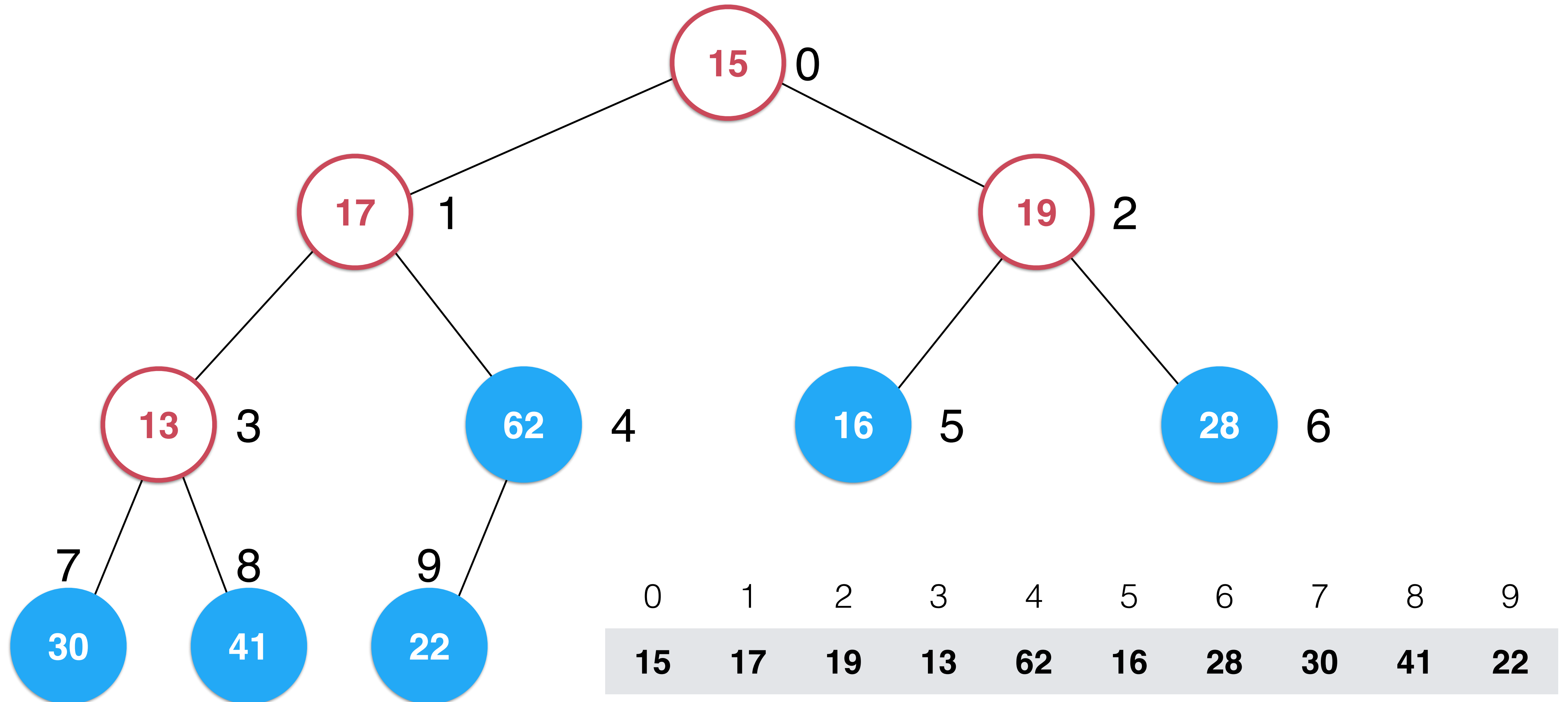
Heapify



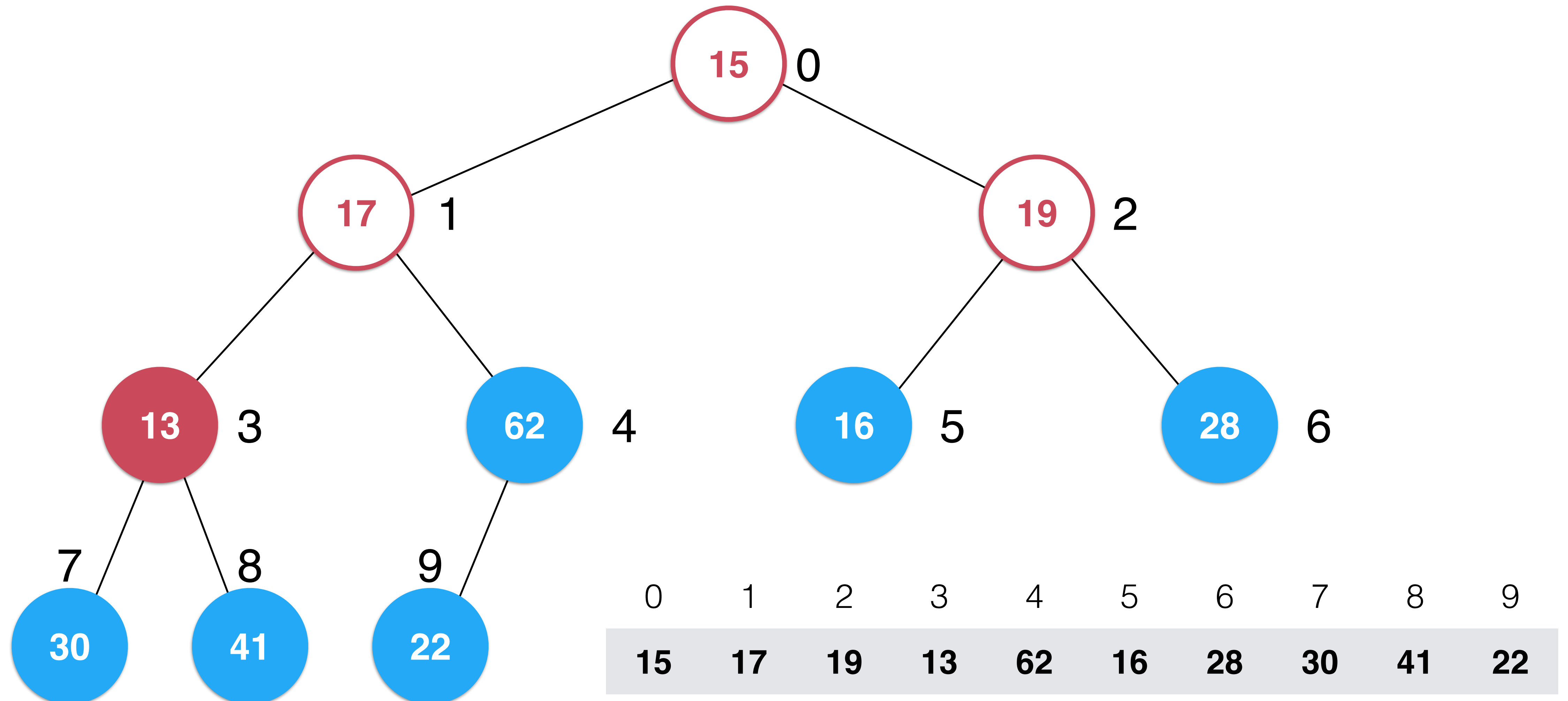
Heapify



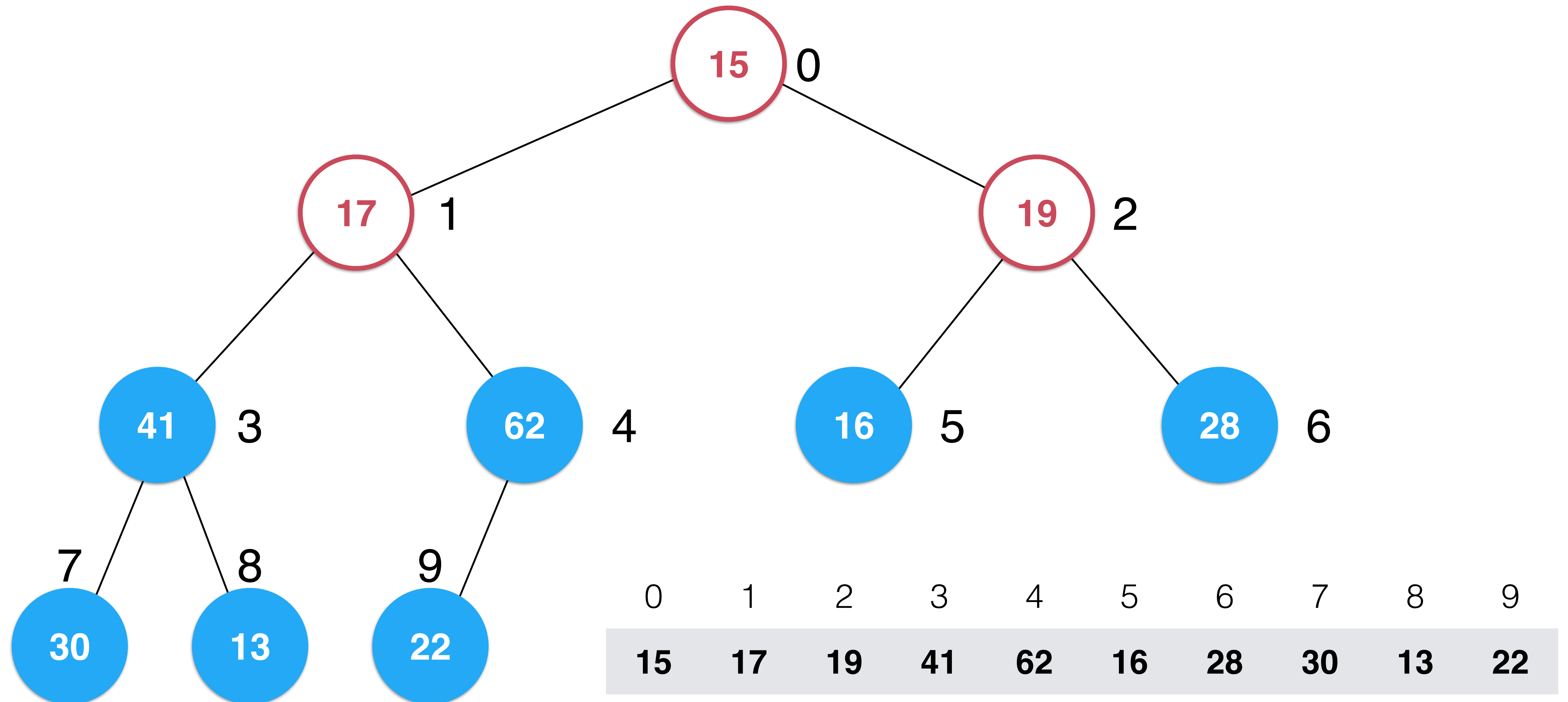
Heapify



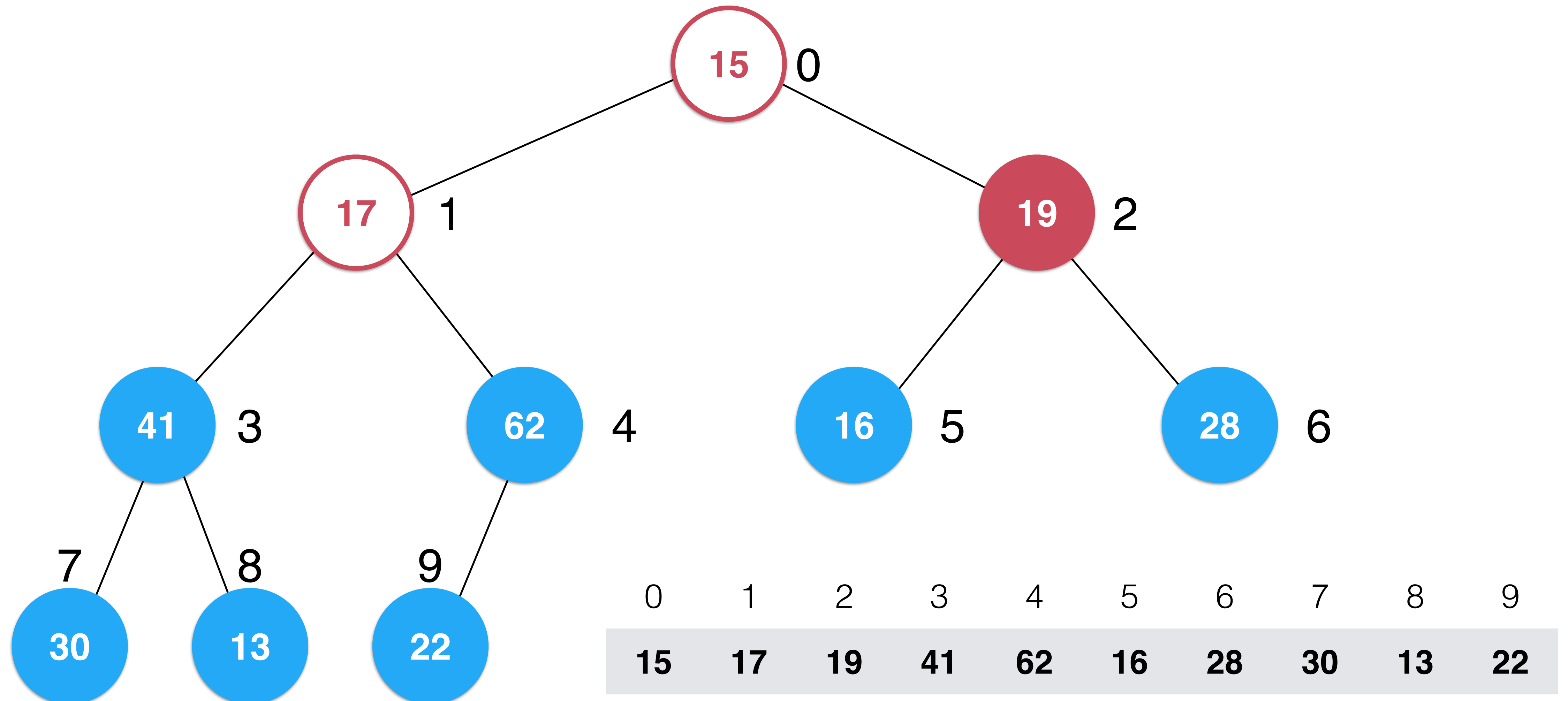
Heapify



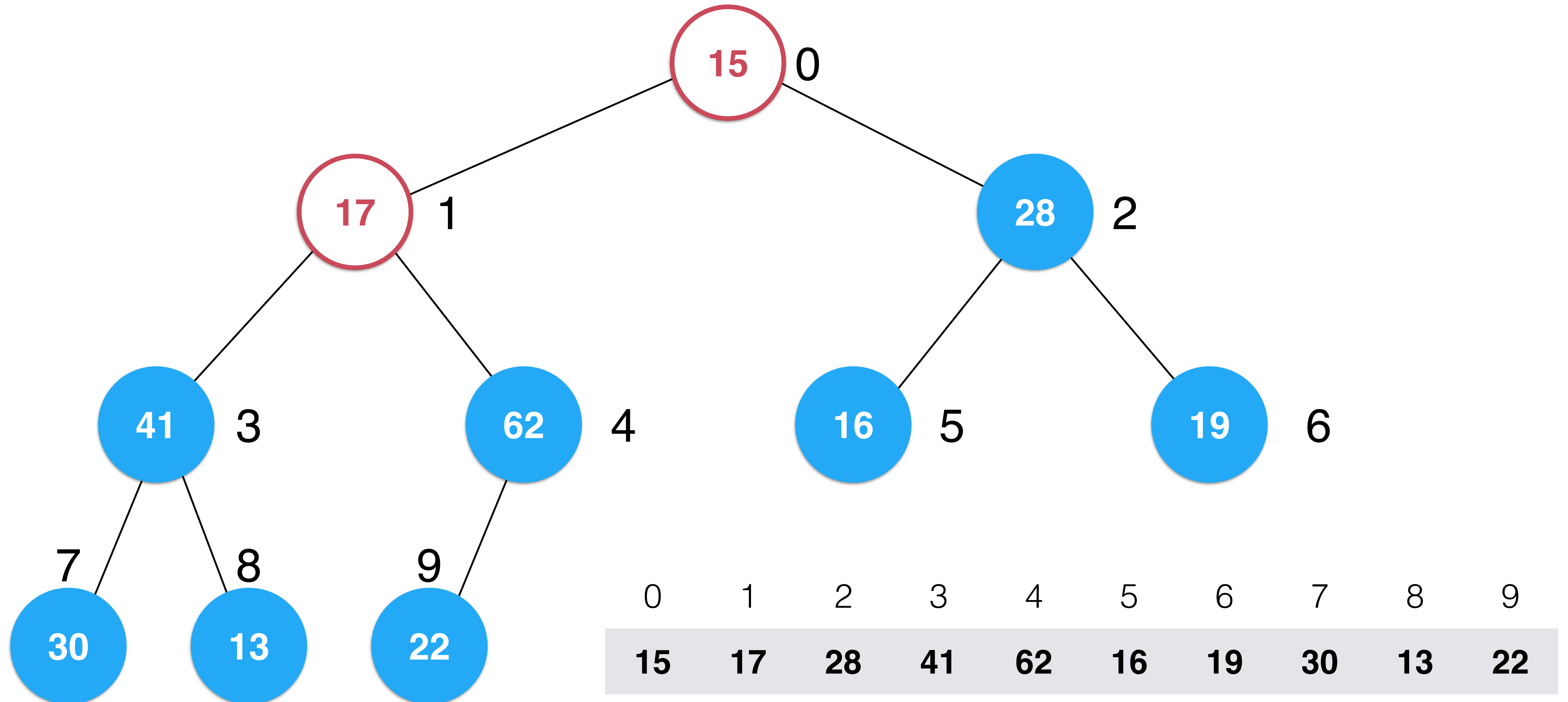
Heapify



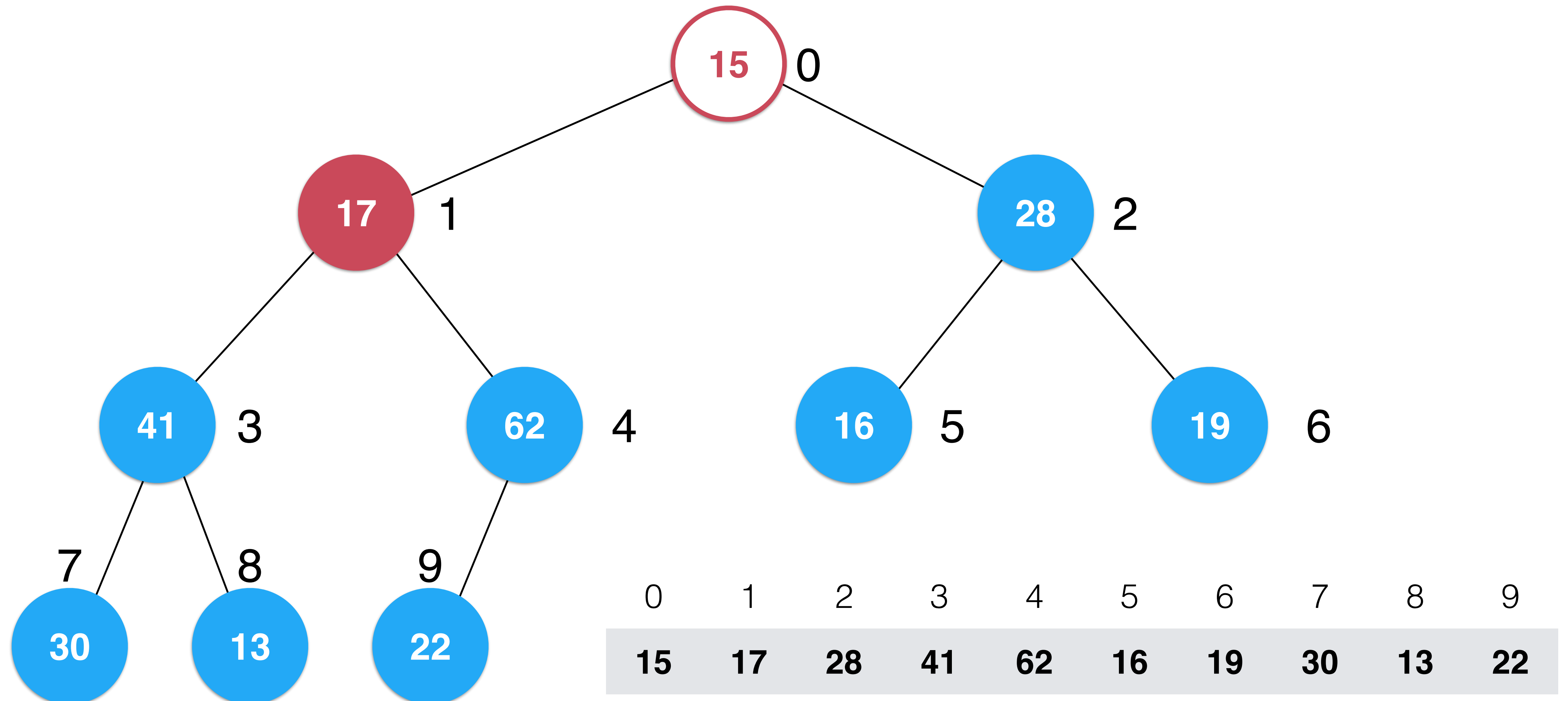
Heapify



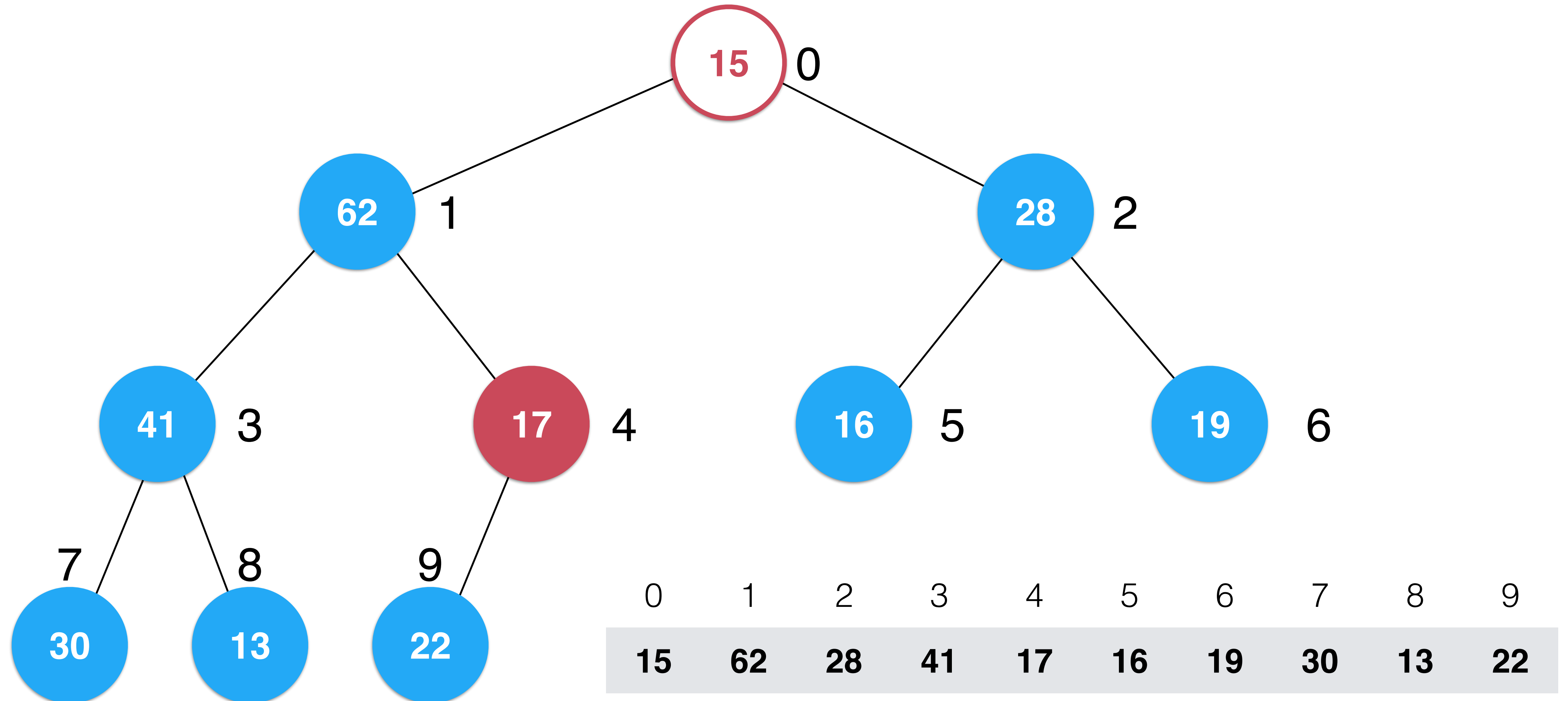
Heapify



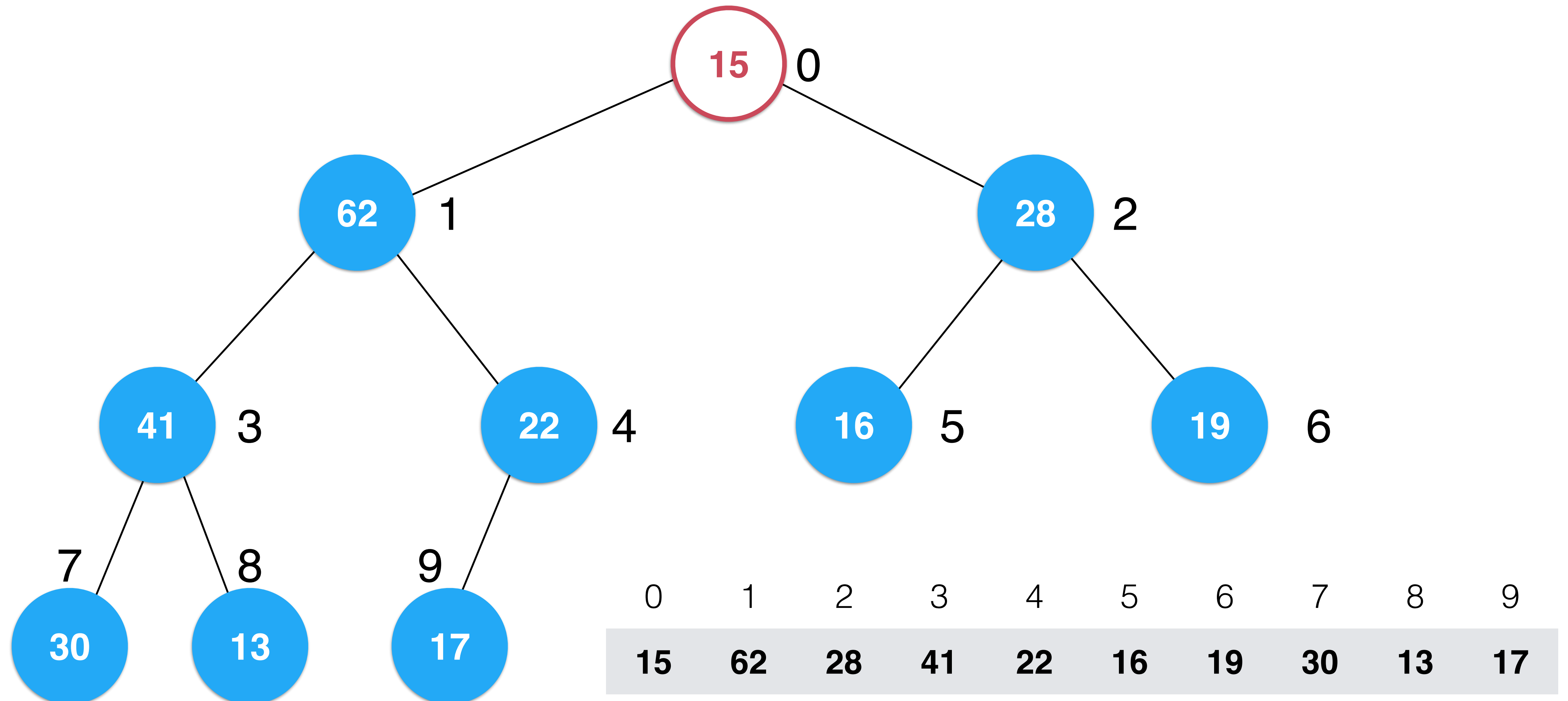
Heapify



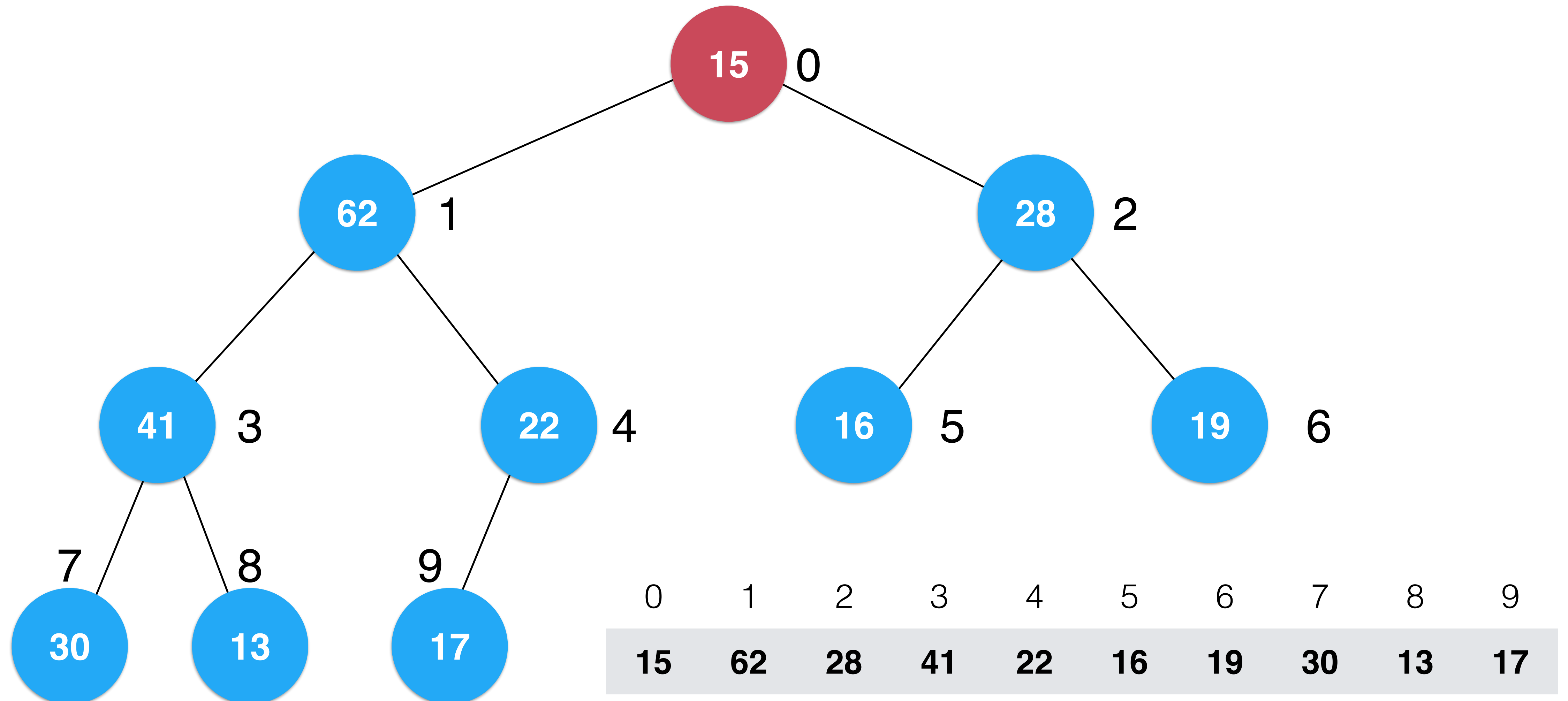
Heapify



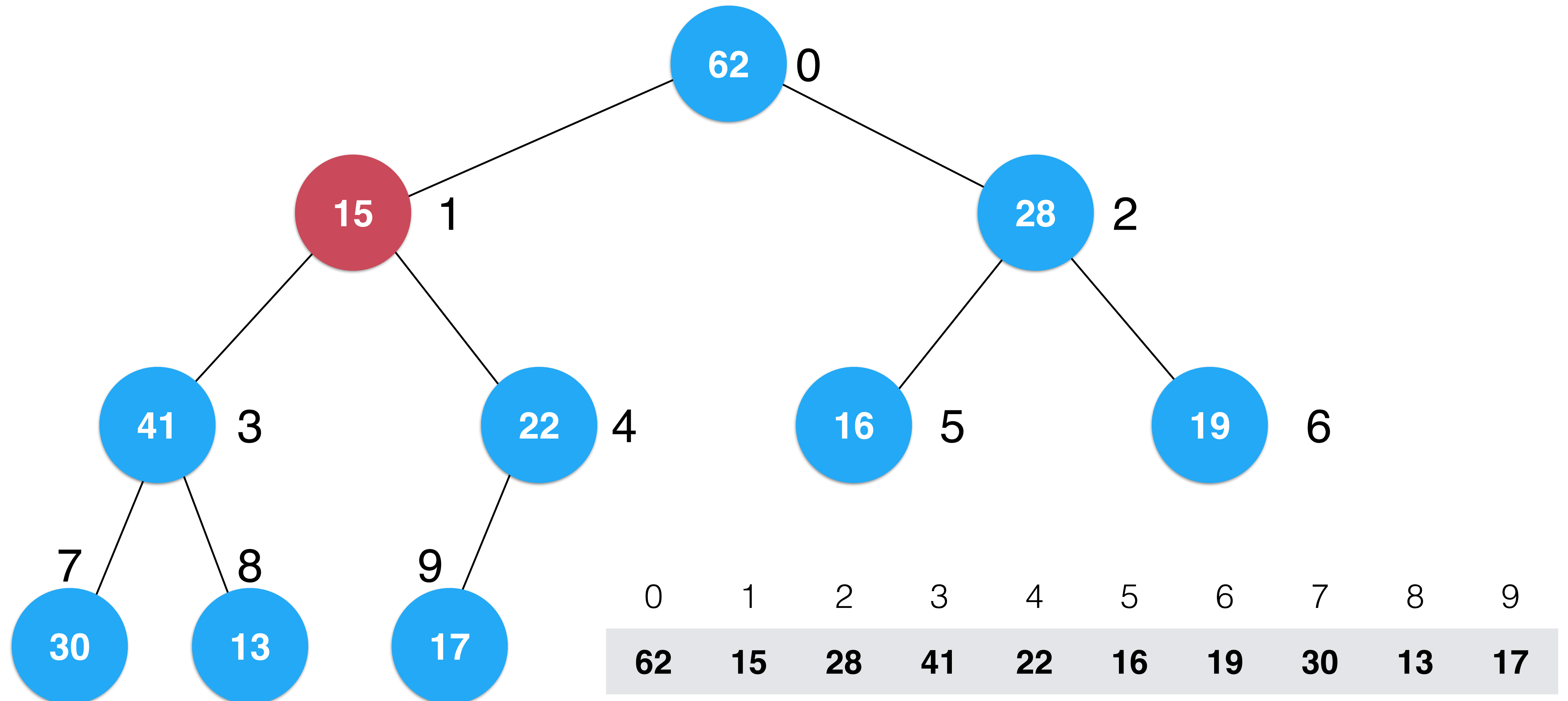
Heapify



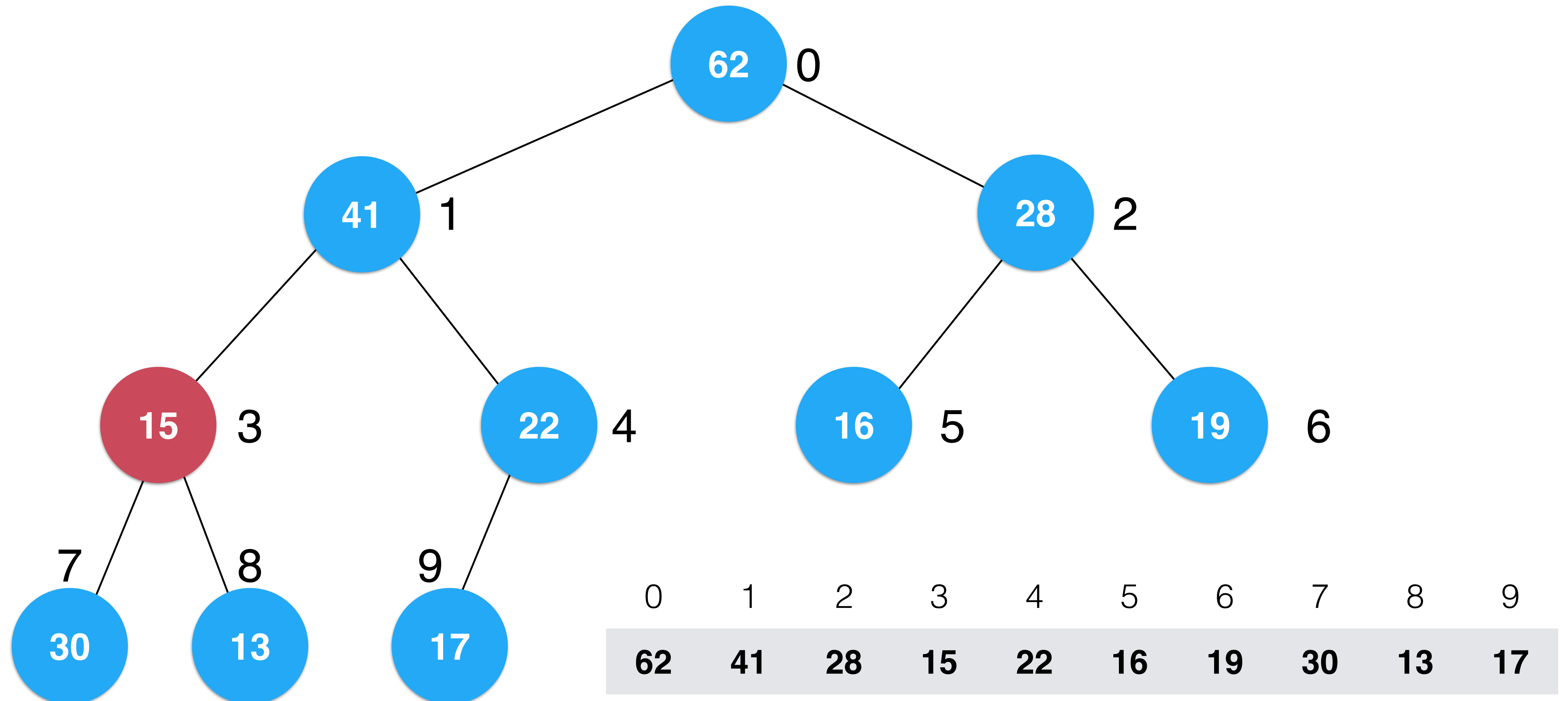
Heapify



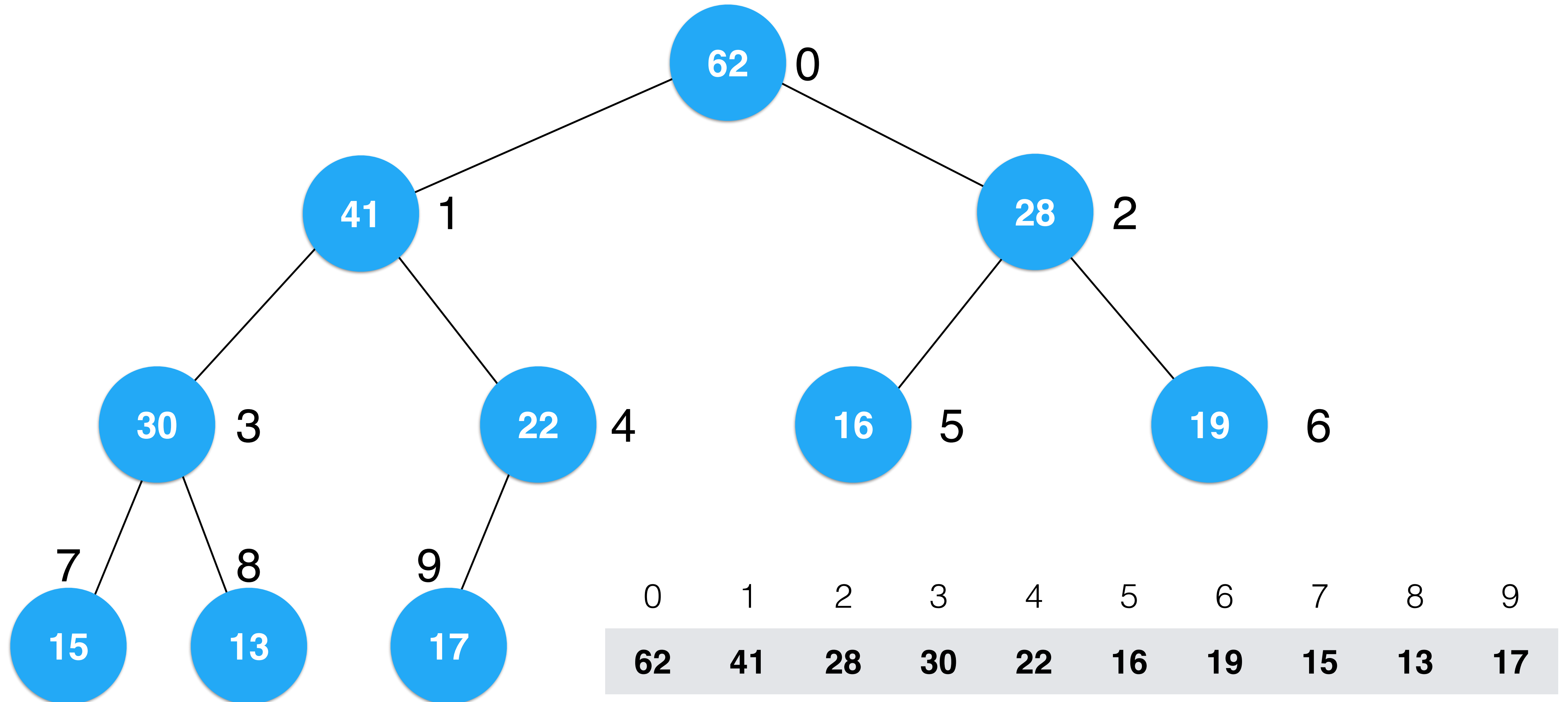
Heapify



Heapify



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 $O(n\log n)$

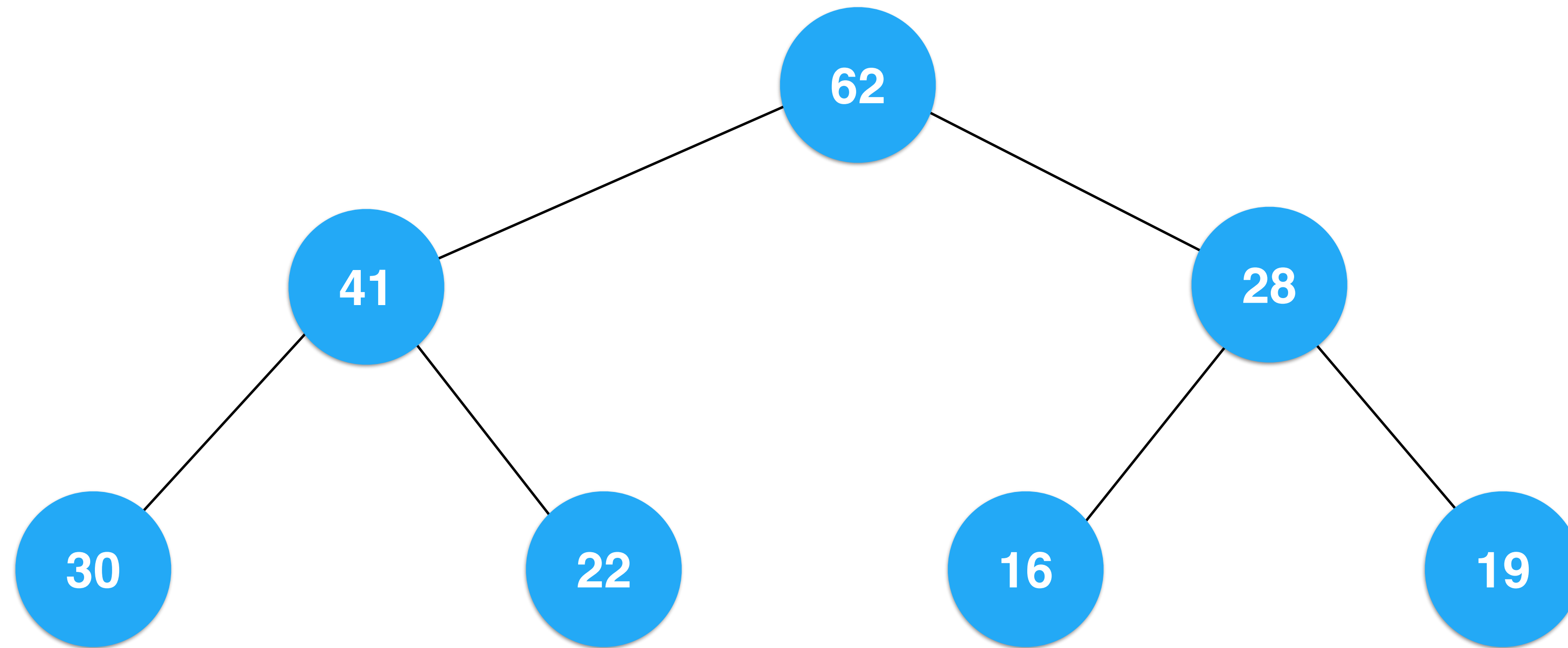
heapify的过程，算法复杂度为 $O(n)$

Heapify 的算法复杂度

heapify的过程，算法复杂度为 $O(n)$

最后一层最多有多少个节点？

Heapify 的算法复杂度



$$1 + 2 + 4 + 8 + \dots + 2^{h-1} = \frac{1(1 - 2^h)}{1 - 2} = 2^h - 1$$

非叶子节点: $2^{h-1} - 1$ 叶子节点: 2^{h-1} 大约是 $n/2$ 个

Heapify 的算法复杂度

heapify的过程，算法复杂度为 $O(n)$

最后一层最多有多少个节点? $n/2$ $n/2 * 0$

倒数第2层最多有多少个节点? $n/4$ $n/4 * 1$

下面有 h 层节点 $\frac{n}{2^{h+1}}$ $\frac{n}{2^{h+1}} * h$

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O \left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h} \right)$$

Heapify 的算法复杂度

heapify的过程，算法复杂度为 $O(n)$

$$\begin{aligned}\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) &= O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right) \\ &= O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right) \\ &= O(n) .\end{aligned}$$

$$\begin{aligned}\sum_{k=0}^{\infty} kx^k &= \frac{x}{(1-x)^2} \\ \text{for } |x| < 1.\end{aligned}$$

$$\begin{aligned}\sum_{h=0}^{\infty} \frac{h}{2^h} &= \frac{1/2}{(1-1/2)^2} \\ &= 2 .\end{aligned}$$

Heapify 的算法复杂度

heapify的过程，算法复杂度为 $O(n)$

实现 Heapify

实践：实现 Heapify

优化的堆排序

原地堆排序

max



v

Max Heap

原地堆排序

max



原地堆排序

max



原地堆排序

max



原地堆排序

max



原地堆排序

max



原地堆排序

max



原地堆排序



原地堆排序



Max Heap

作业：实现最小堆

作业解析：实现最小堆

其他

欢迎大家关注我的个人公众号：是不是很酷



算法与数据结构体系课程

liuyubobobo