算法第四次作业

6.2-5

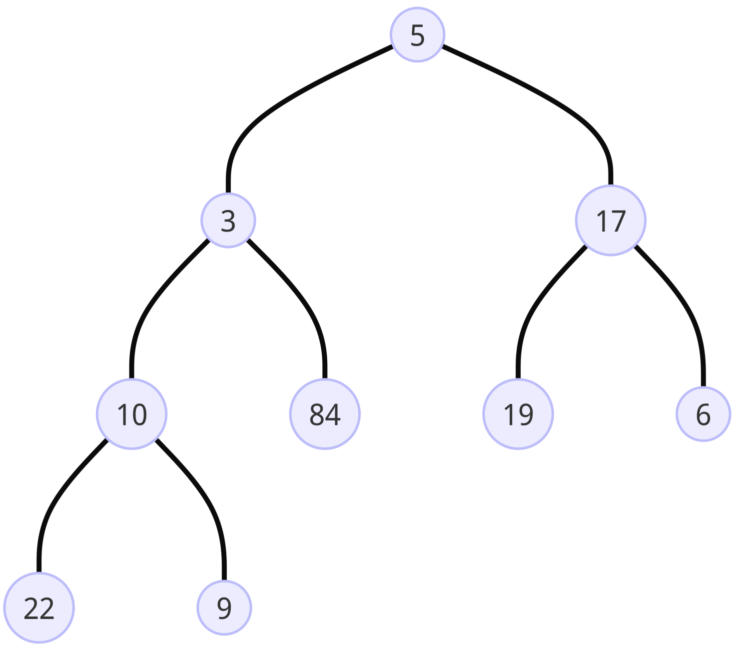
```
def MAX-HEAPIFY(A, i)
    while i < A.heap-size
        l = LEFT(i)
        r = RIGHT(i)
        if l <= A.heap-size and A[l] > A[i]
            largest = l
        else largest = i
        if r <= A.heap-size and A[r] > A[i]
            largest = r
        if largest != i
            exchange A[i] with A[largest]
            i = largest
        else
            return
    return

```

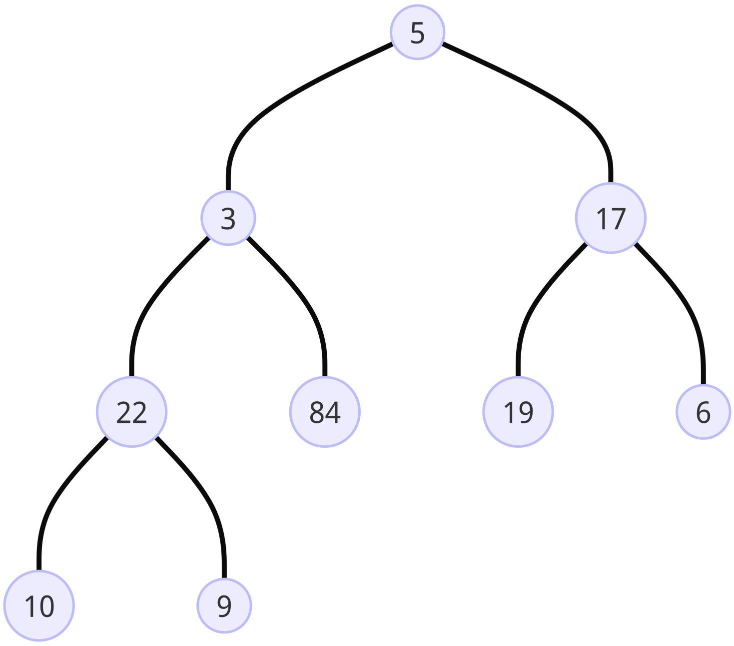
相比较与递归形式的 `MAX-HEAPIFY`，非递归的形式多了整个函数体内的 `while` 循环。循环的结束条件在这里也换成了当 `largest == i` 时，因为此时需要鉴别的元素 `A[i]` 已经满足大根堆的性质，无需继续与子树交换。

6.3-1

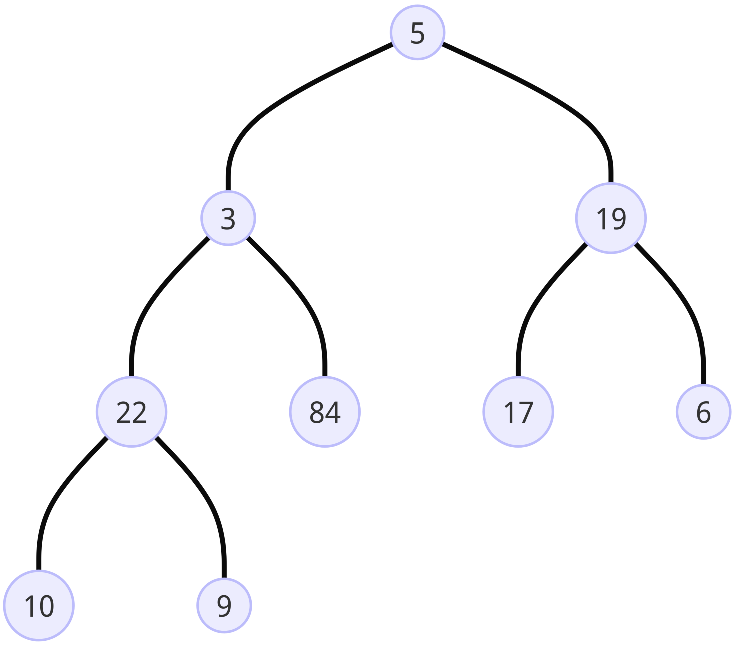
- 初始二叉树如下：



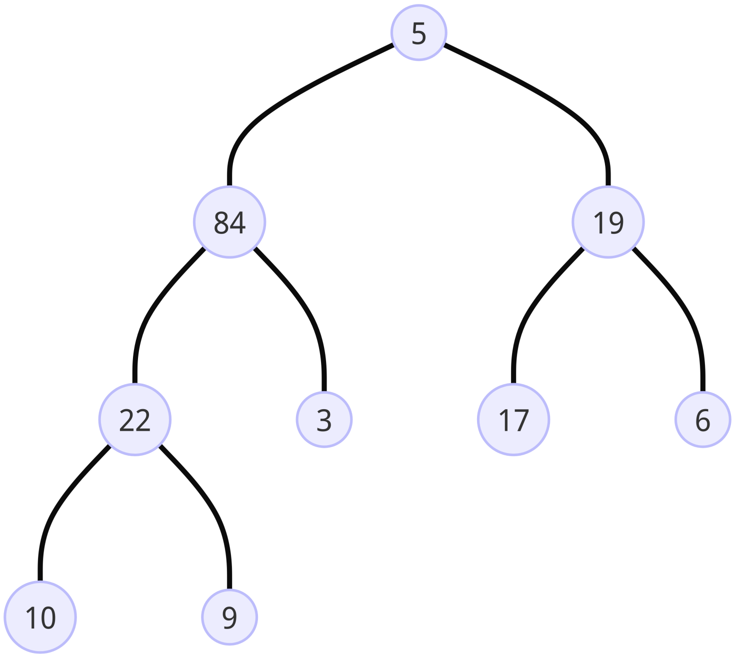
- 对结点 `10` 做操作：



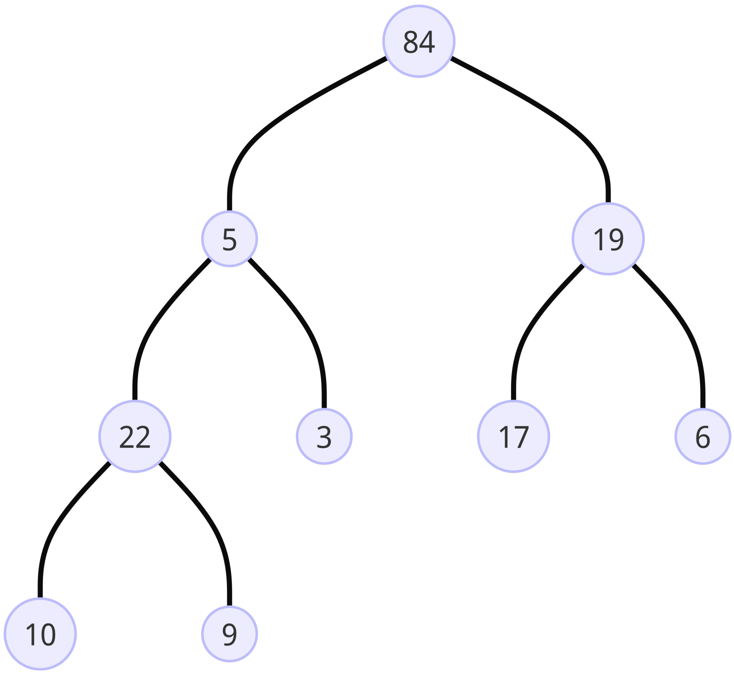
- 对结点 `17` 做操作：

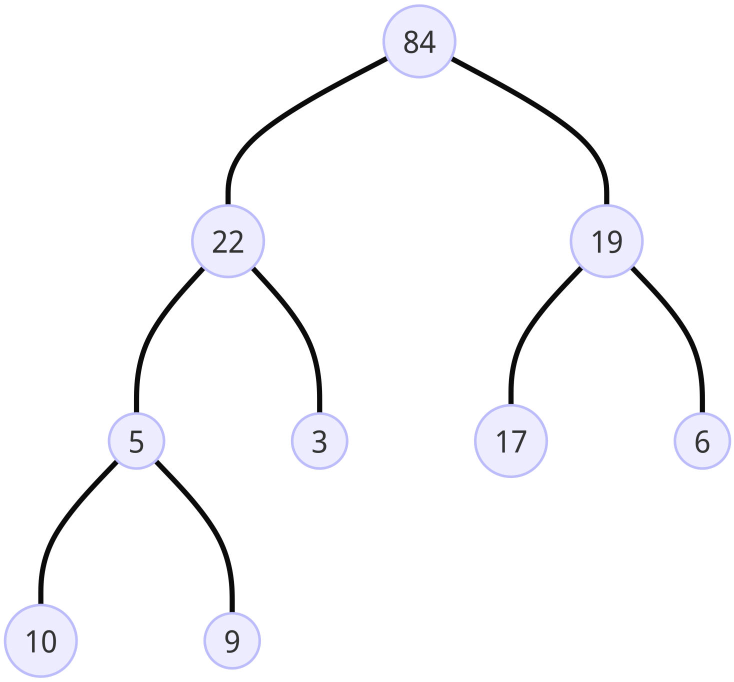


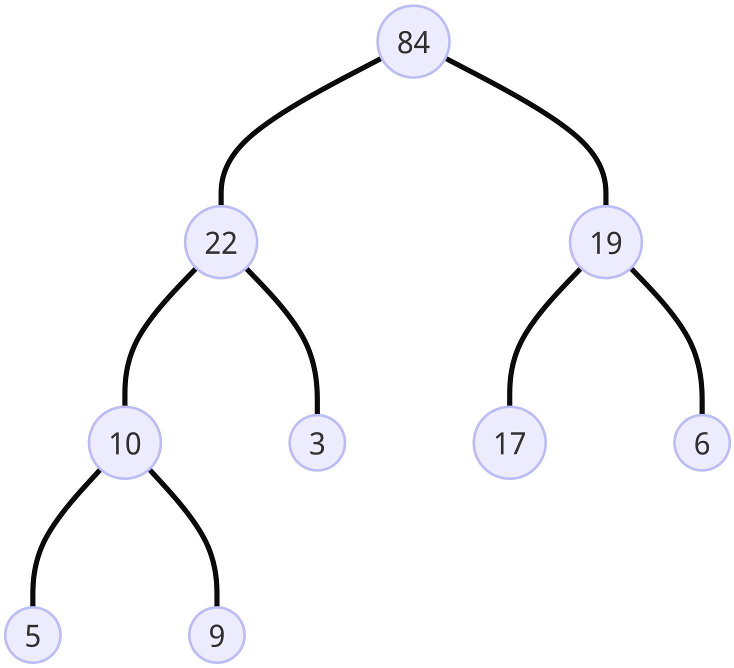
- 对结点 `3` 做操作：



- 对结点 `5` 做操作：







6.3-3

假设高度为 h 的结点的个数为 $N(h)$，下证 $N(h) \le \lceil \frac{n}{2^{h+1}} \rceil$：

首先证 n 个结点的完全二叉树中，非叶子结点有 $\lfloor \frac{n}{2} \rfloor$ 个（即 $N(0) = n - \lfloor \frac{n}{2} \rfloor = \lceil \frac{n}{2} \rceil$）。

由图论中的经典结论：度数为 2 的结点个数为度数为 0 的结点个数少一个。在完全二叉树中，度数为 1 的结点数不会超过 1 个。而度数为 0 的结点就是叶子结点，记为 n_0，其余同理，有：

$$n_0 = n_2 + 1 \tag{1}$$

$$n_0 + n_2 \le n_0 + n_1 + n_2 = n \le n_0 + n_2 + 1 \tag{2}$$

由 (1)，(2)式可知：

$$2n_0 - 1 \le n \le 2n_0 \tag{3}$$

$$\frac{n}{2} \le n_0 \le \frac{n+1}{2} \tag{4}$$

由 (4) 有:

$$N(0) = n_0 = \lceil \frac{n}{2} \rceil$$

当然，非叶子结点有：$\lfloor \frac{n}{2} \rfloor$ 个。

做下文操作：删除所有叶子结点，此时所有新的叶子结点就是原先高度为 1 的结点，总结点个数为 $\lfloor \frac{n}{2} \rfloor$
。

根据上述结论，原先高度为 1 的结点的个数是：

$$N(1) = \lceil \frac{\lfloor \frac{n}{2} \rfloor}{2} \rceil \leq \lceil \frac{\frac{n}{2}}{2} \rceil = \lceil \frac{n}{2^2} \rceil$$

继续做上述操作，设 $g(n) = \lfloor \frac{g(n-1)}{2} \rfloor$，其中 $g(0) = n$，那么在重复 n 次操作之后，剩余的总结点个数就是 $g(n)$，同时：$N(n) = \lceil g(n) \rceil$，那么对$g(h)$ 和 $N(h)$：

$$g(h) \leq \frac{g(h-1)}{2} \leq \frac{g(h-2)}{2^2} \leq \cdots \leq \frac{n}{2^h}$$

$$N(h) = g(h) - \lfloor \frac{g(h)}{2} \rfloor = \lceil \frac{g(h)}{2} \rceil \leq \lceil \frac{n}{2^{h+1}} \rceil$$

得证。

6.4-4

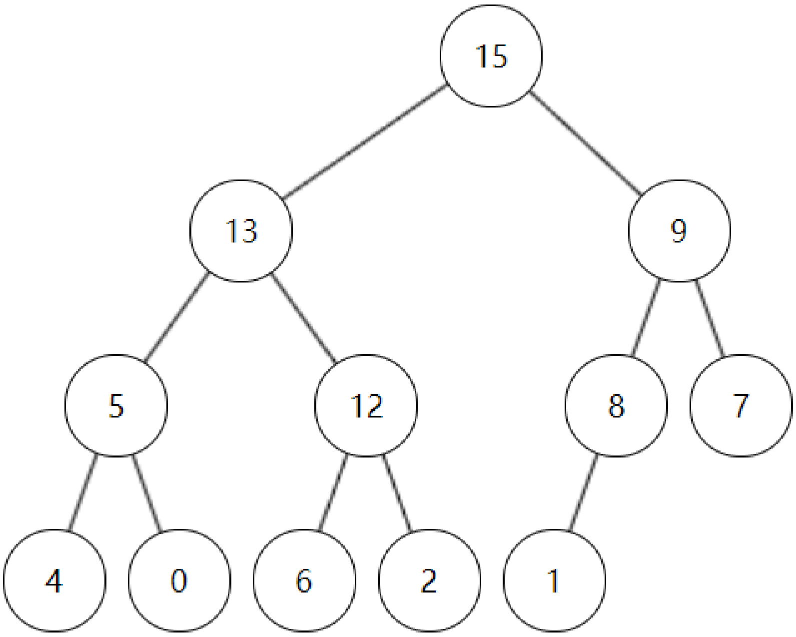
若输入数组是一个严格从小到大的数组。那么在建堆时调用 `BUILD-MAX-HEAP` 函数时复杂度为 $O(1)$，堆排序使用 `MAX-HEAPIFY` 时，$A[1]$ 和 $A[i]$ 交换时，均需要操作树高 h 次。而在树高为 h 时结点数为 $\Theta(2^h)$，此时运行时间为：

$$\sum_{h=0}^{\lfloor log(n) \rfloor} \Theta(2^h)h = \Theta(\sum_{h=1}^{\lfloor log(n) \rfloor} h \cdot 2^h) = \Theta(2 + (\lfloor log(n) \rfloor - 1)2^{\lfloor log(n) \rfloor}) = \Theta(nlog(n))$$

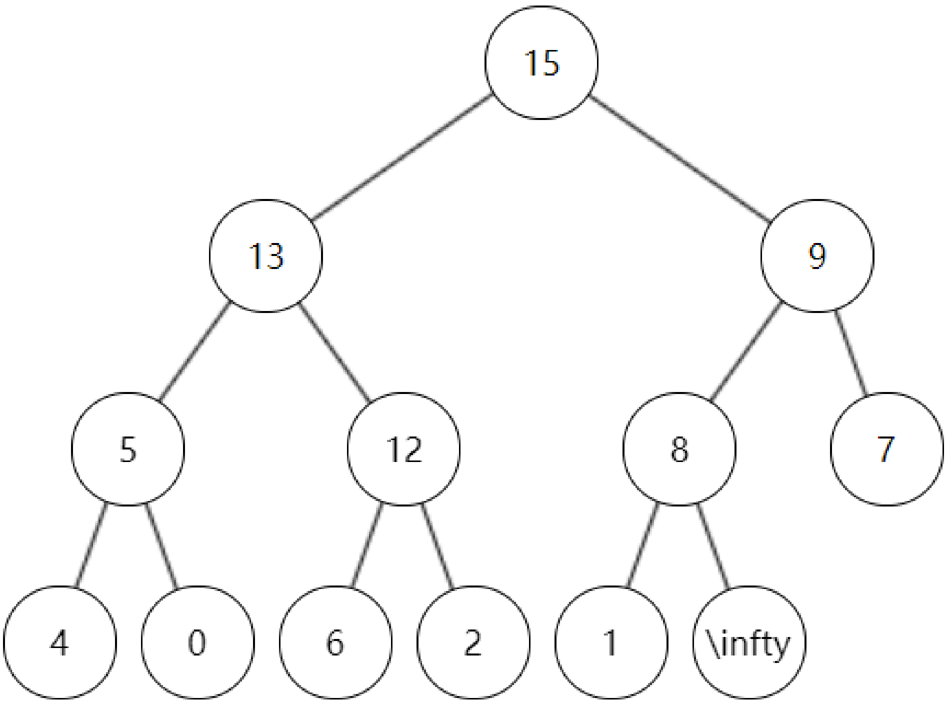
当然有最坏情况下 `HEAPSORT` 的时间复杂度是 $\Omega(nlog(n))$

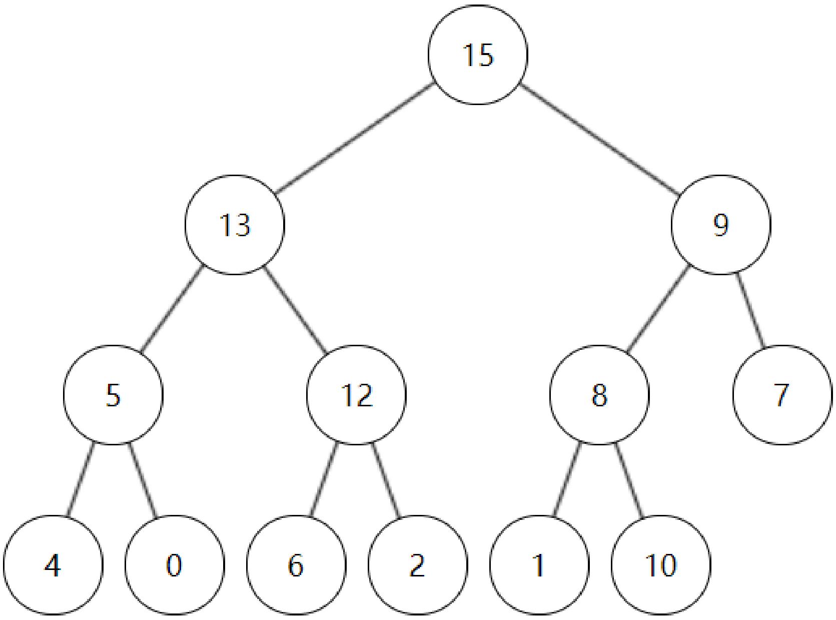
6.5-2

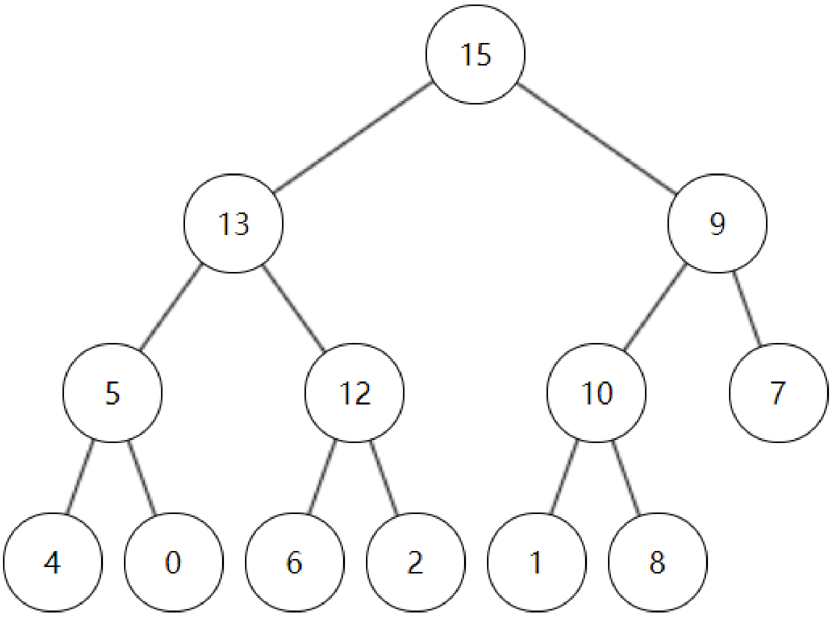
初始状态:

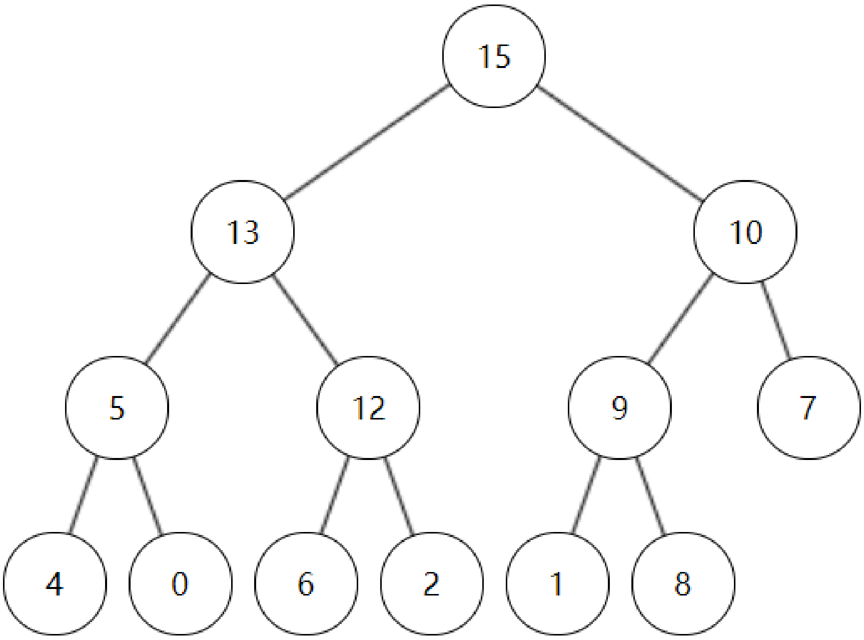


操作:




15
13
9
5
12
8
7
4
0
6
2
1
10


15
13
9
5
12
10
7
4
0
6
2
1
8



6.5-8

先将需要删除的值调用 `HEAP-INCREASE-KEY` 函数变为 $+\infty$，该值大于所有堆上的值。

这时该值会在堆顶，调用 `EXTRACT-MAX` 即可。

伪代码为：

```
HEAP-INCREASE-KEY(A,i,+\infty)
EXTRACT-MAX(A)
```

其中， `HEAP-INCREASE-KEY` 函数的时间复杂度是 $O(log(n))$， `EXTRACT-MAX` 函数的时间复杂度是 $O(log(n))$，所以删除的时间复杂度也是 $O(log(n))$。