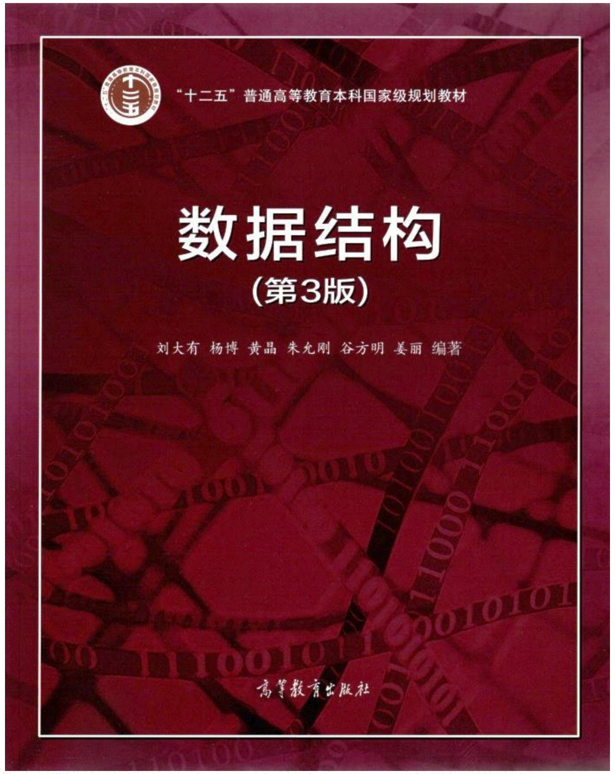


堆排序

- 锦标赛排序
- 堆的概念及基本操作
- 堆排序算法

数据之法 结构之美 算法之道

zhuyungang@jlu.edu.cn

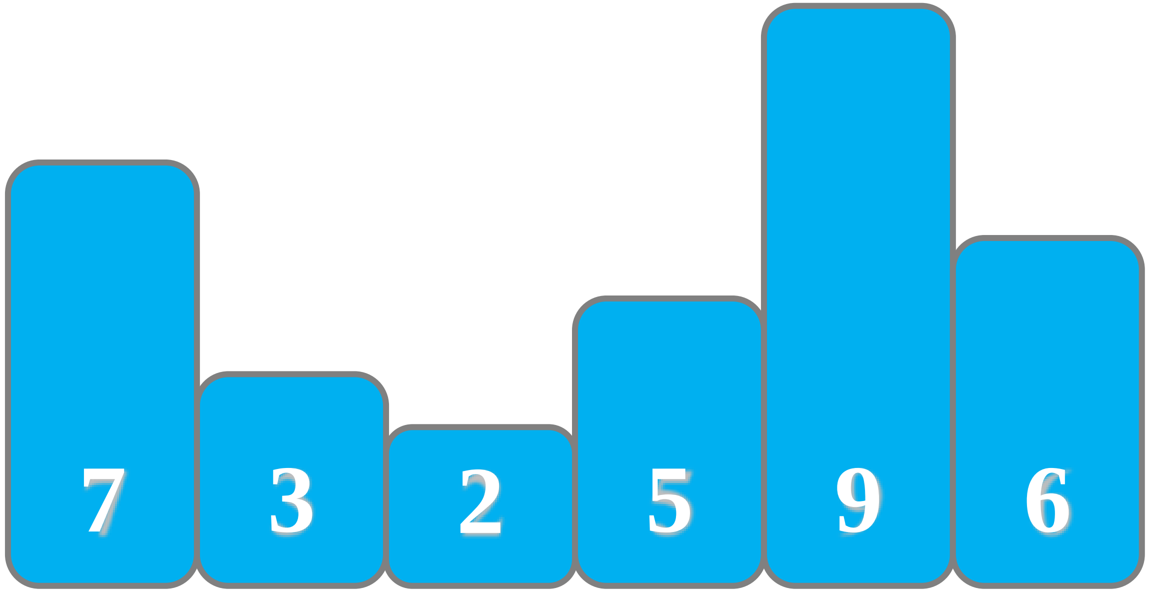
楼天城
清华大学博士
2次获ACM/ICPC全球总决赛金牌亚军
2次获Google全球编程挑战赛冠军
2次获Facebook骇客杯世界编程大赛季军
2次获百度之星程序设计大赛冠军
小马智行创始人兼CTO
身价75亿元人民币（胡润财富榜）

第一名可能会让你望尘莫及，直到最后也没办法追上。在此之前，要有一个正确的心态，不能自暴自弃，更不能妄自菲薄。然后要很实实在在的努力，每次必须做的事要做好，先把距离咬住再说。

人生发展是个积累的过程，基本上99%的成功都是通过日日夜夜的积累完成的。努力是你唯一能依赖的东西。

动机

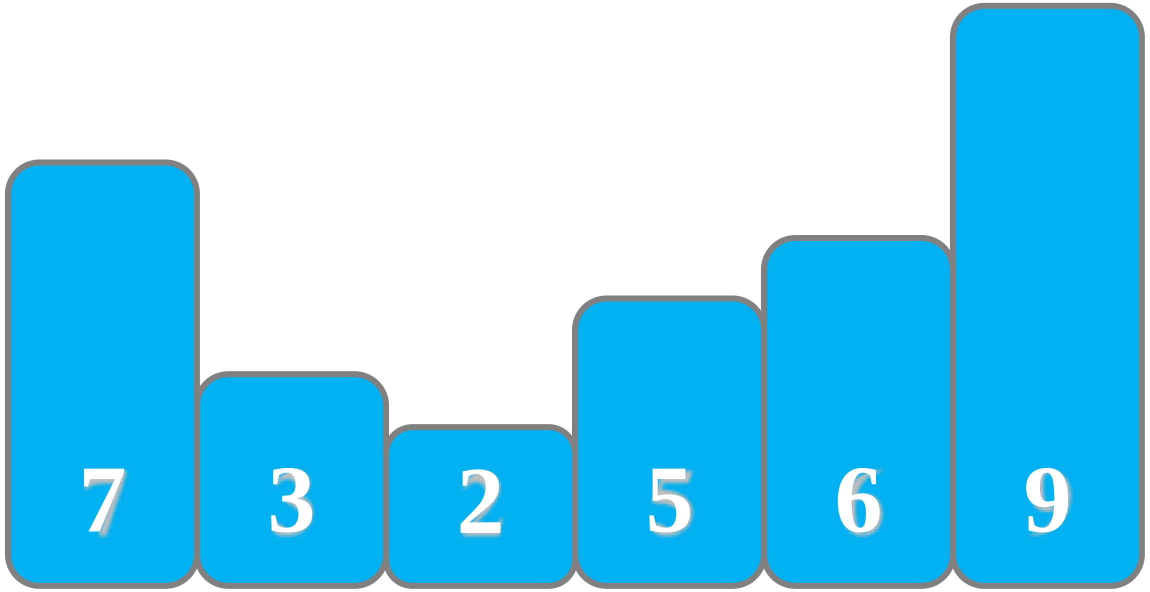
- 直接选择排序低效原因：每次找最大元素需花费O(n)时间，涉及大量重复比较。



动机

A

- 直接选择排序低效原因：每次找最大元素需花费O(n)时间，涉及大量重复比较。
- 能否借助树形结构，减少元素的重复比较次数。



体育比赛
淘汰赛

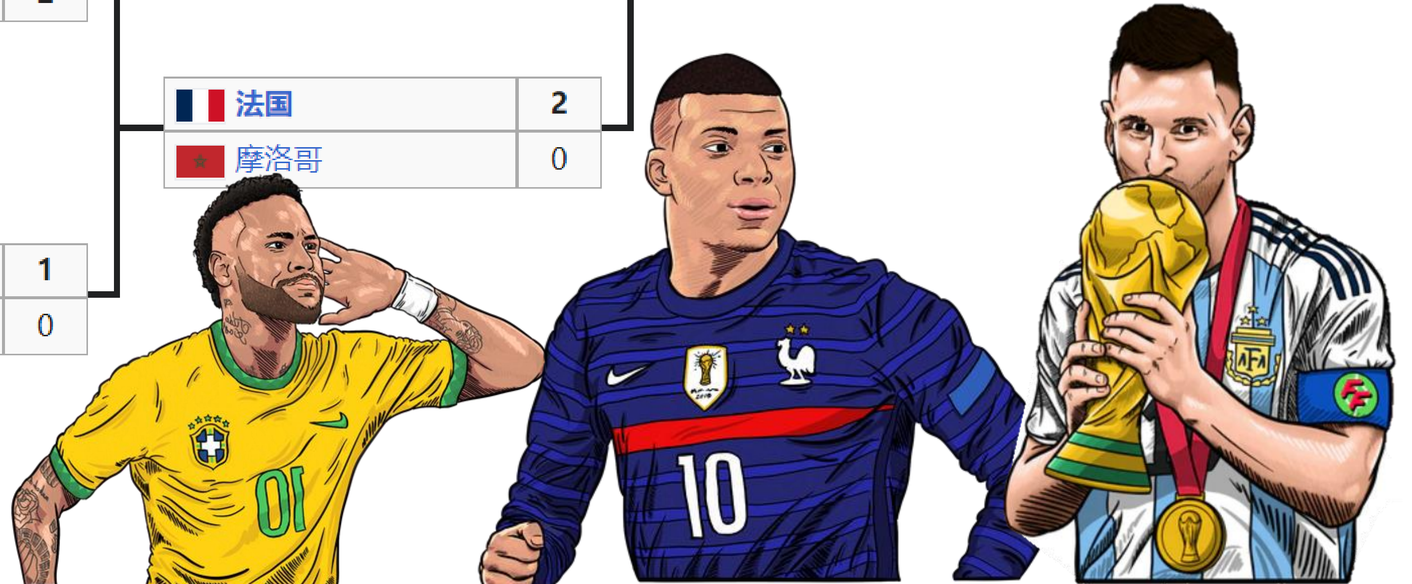
FIFA WORLD CUP Qatar2022

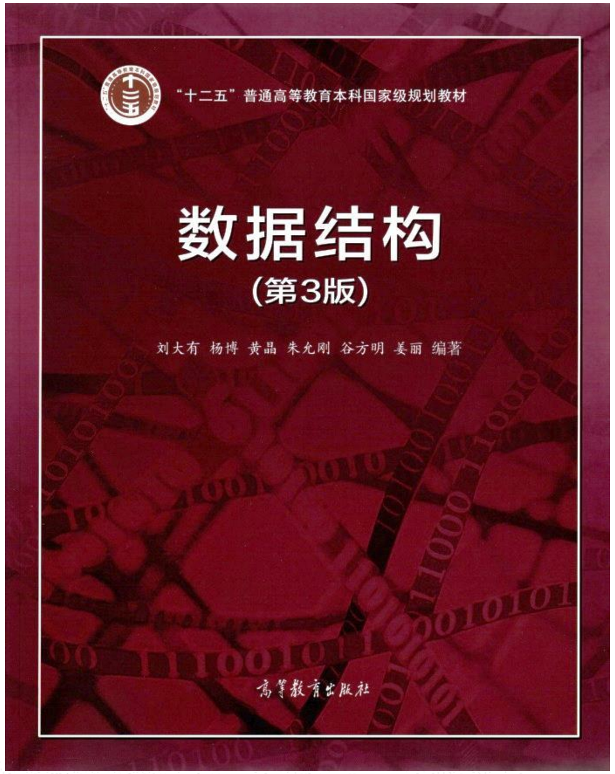
荷兰	3
美国	1
阿根廷	2
澳大利亚	1
日本	1 (1)
克罗地亚（点）	1 (3)
巴西	4
韩国	1
英格兰	3
塞内加尔	0
法国	3
波兰	1
摩洛哥（点）	0 (3)
西班牙	0 (0)
葡萄牙	6
瑞士	1

荷兰	2 (3)
阿根廷（点）	2 (4)
克罗地亚（点）	1 (4)
巴西	1 (2)
英格兰	1
法国	2
摩洛哥	1
葡萄牙	0

阿根廷	3
克罗地亚	0
法国	2
摩洛哥	0

阿根廷（点）	3 (4)
法国	3 (2)





堆排序

- 锦标赛排序
- 堆的概念及基本操作
- 堆排序算法

数据之法
结构之美
算法之道

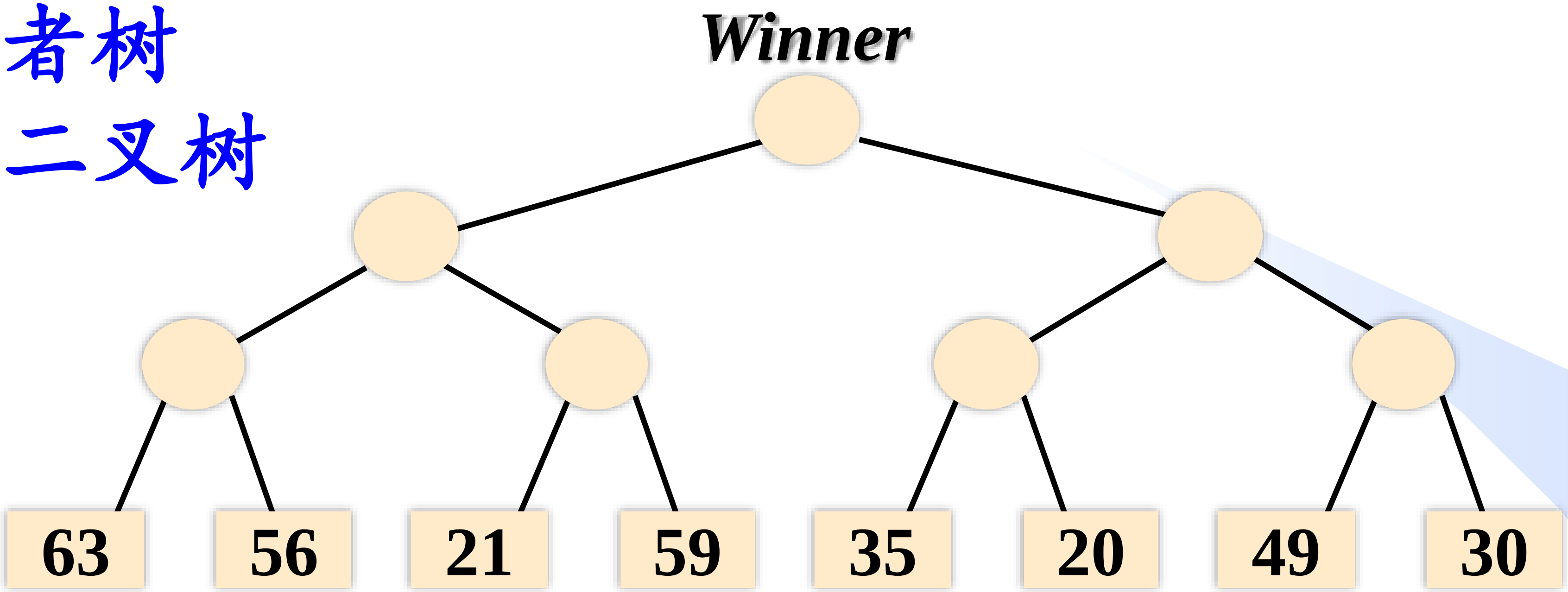
zhuyungang@jlu.edu.cn

锦标赛排序

锦标赛排序

B

胜者树
完全二叉树

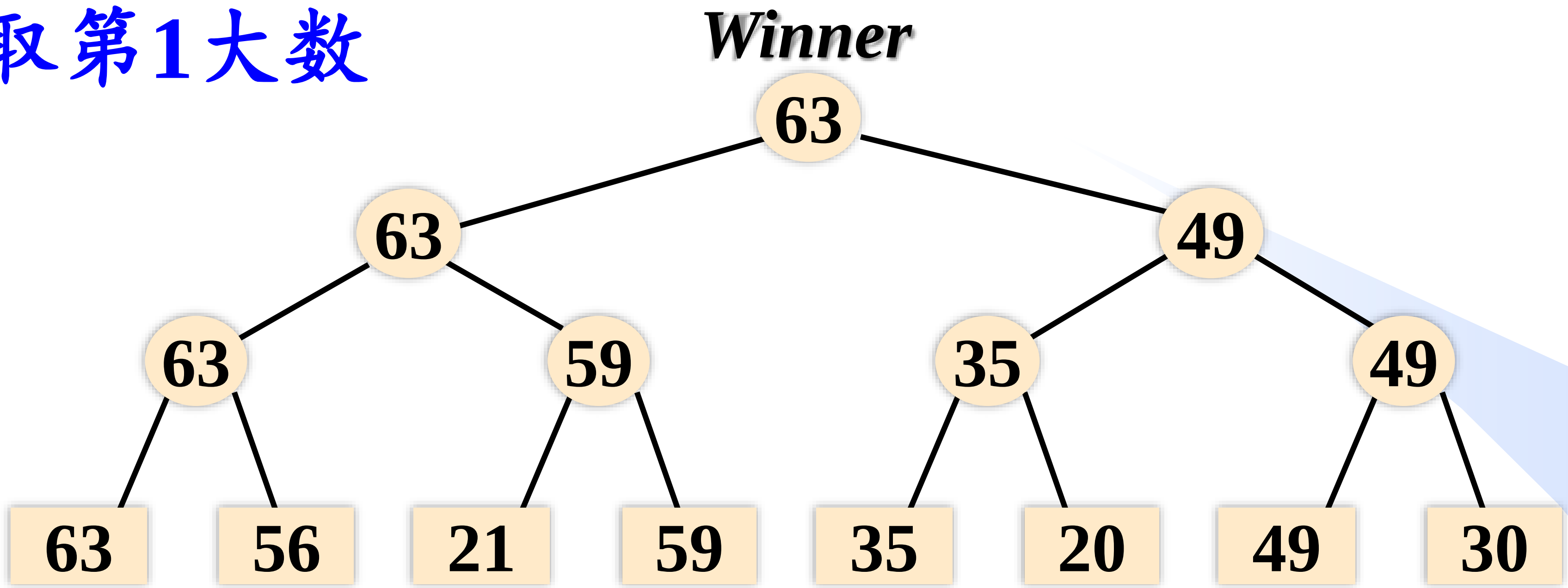


- 叶结点：存放待排序元素的关键词；
- 非叶结点：存放关键词两两比较的结果，即父结点的关键词是子结点的关键词的最大值
- 若有*n*个待排序元素，即胜者树有*n*个叶结点，因胜者树中无度为1的结点，故必有*n*-1个非叶结点。

锦标赛排序

B

取第1大数

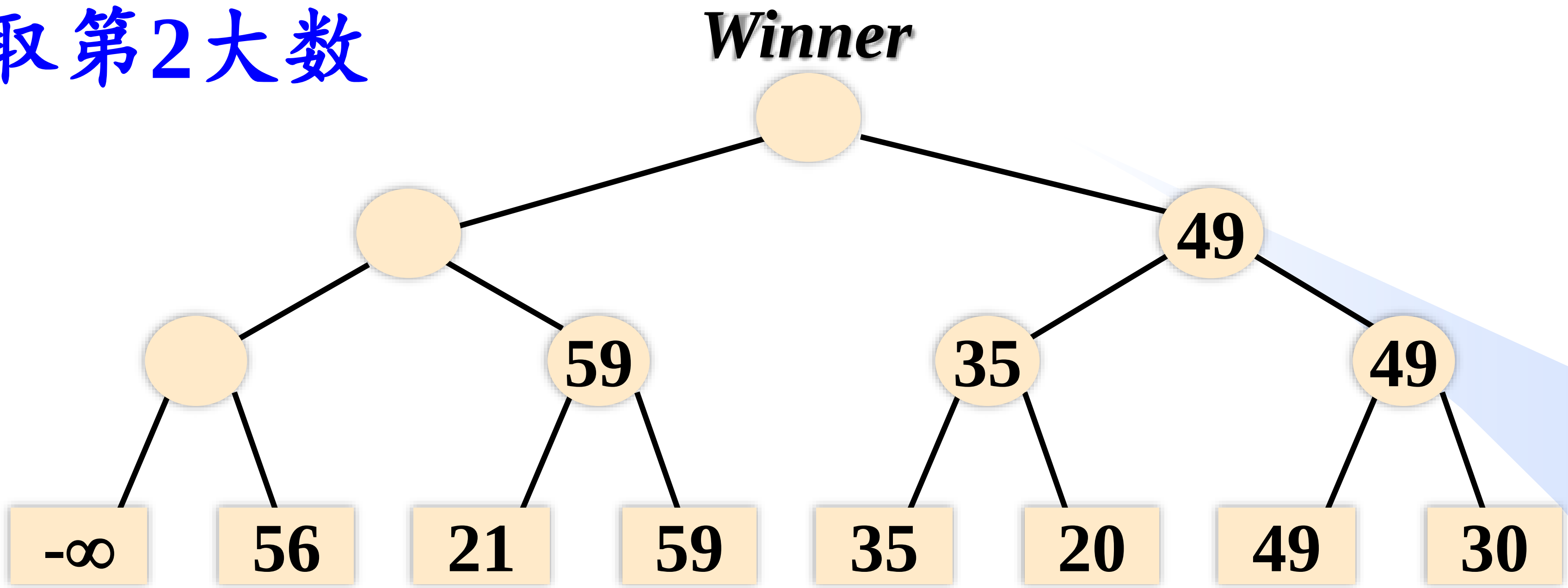


- 最大关键词即为根
- 关键词比较次数O(*n*)

锦标赛排序

B

取第2大数

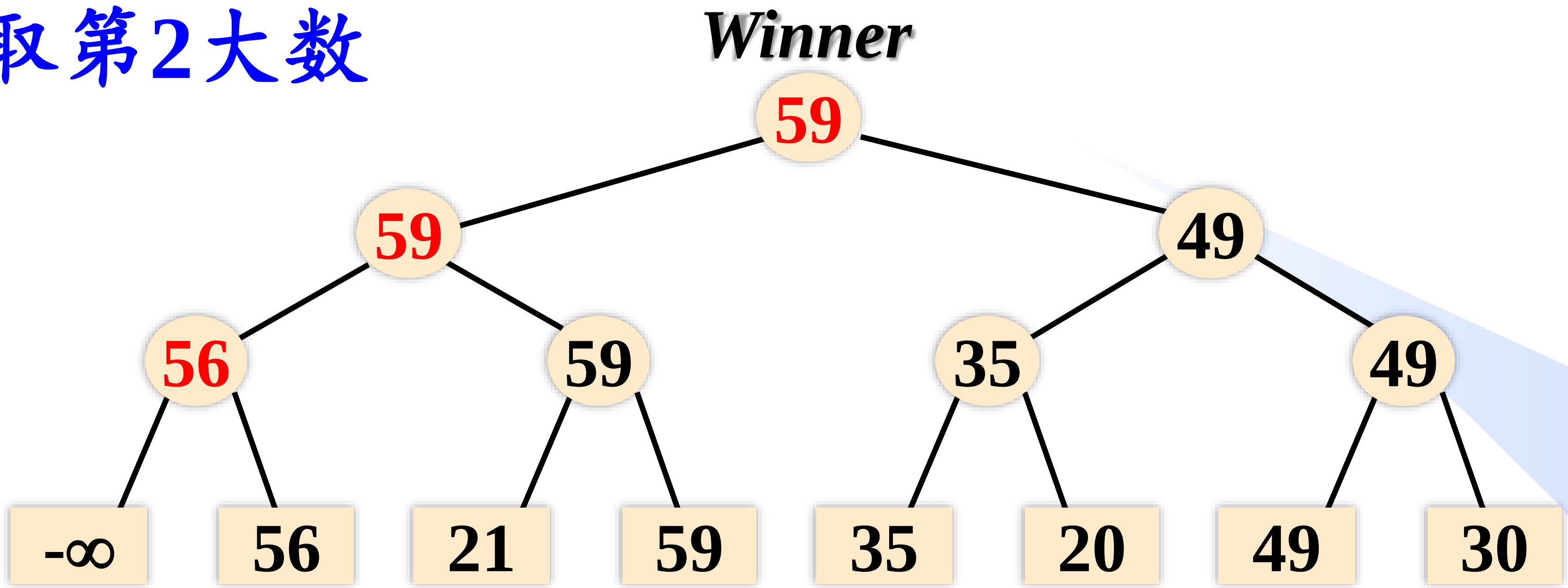


胜者树保存了前面比较的结果，选取第2大元素时直接利用前面比较的结果，从而大幅减少关键词比较次数。

锦标赛排序

B

取第2大数

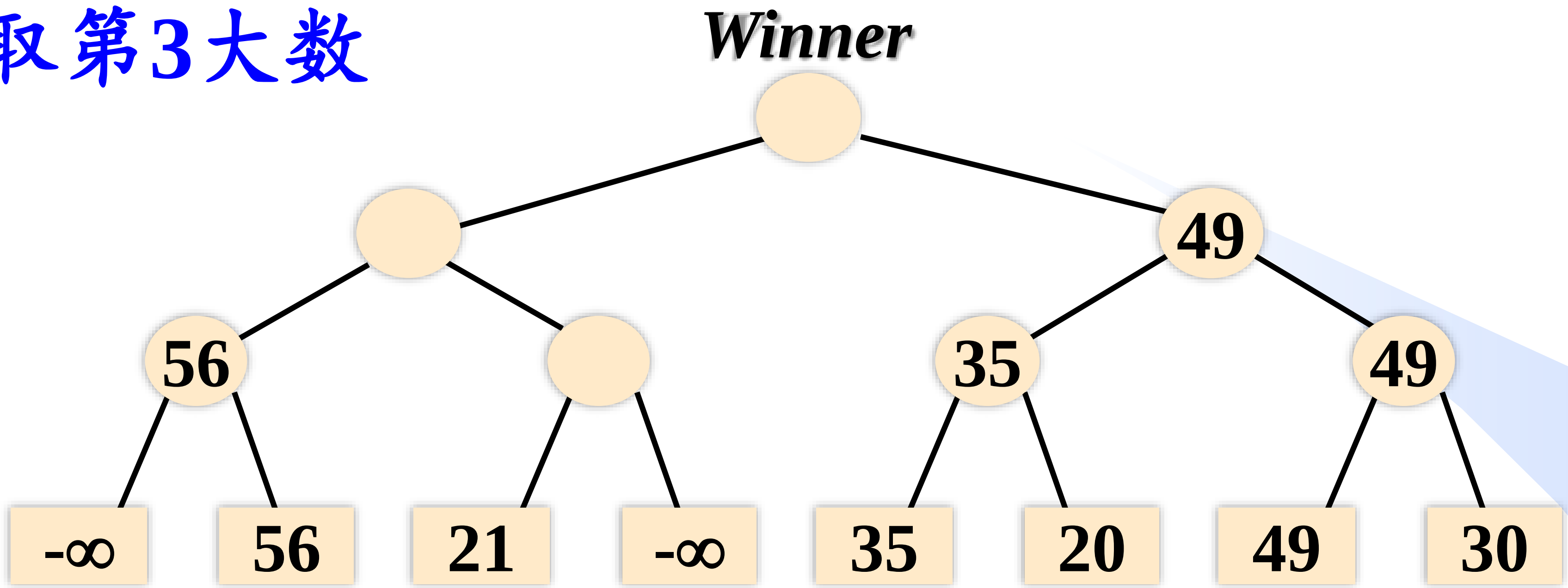


- 将剩余记录调整为新的胜者树，得到第2大元素
- 只涉及从刚被置为-∞的叶结点到根结点的路径，关键词比较次数O(log*n*)

锦标赛排序

B

取第3大数

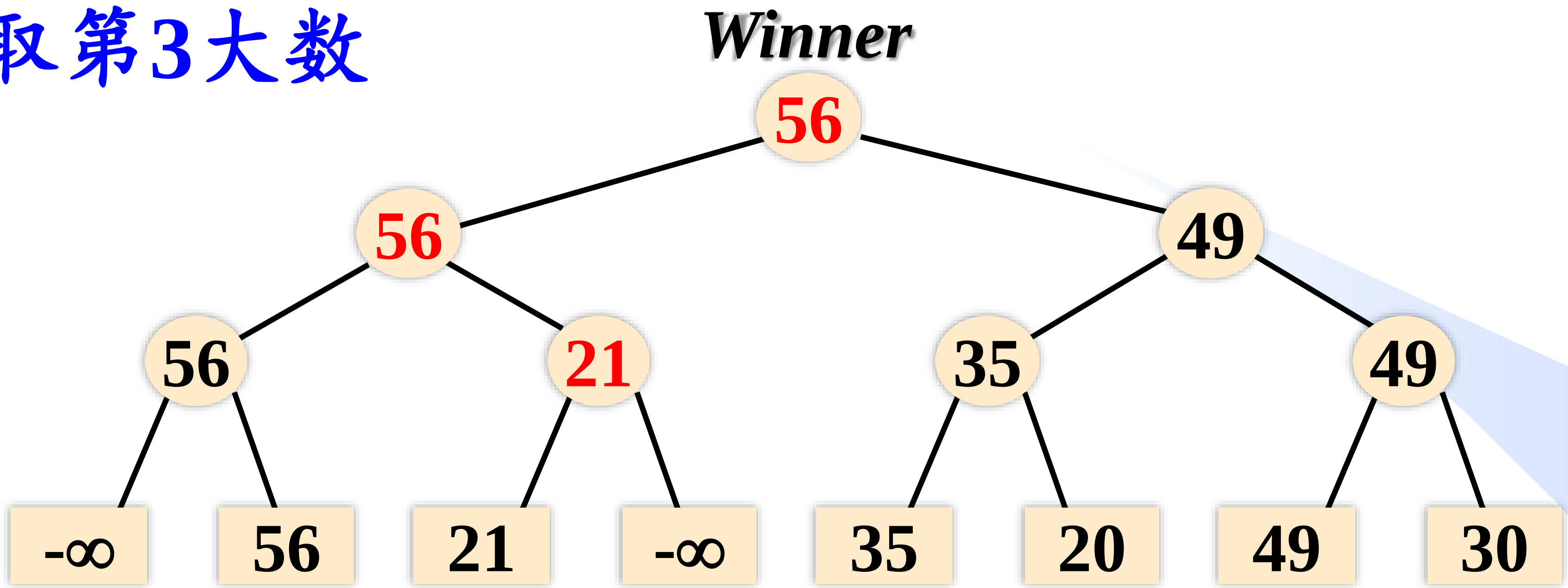


胜者树保存了前面比较的结果，选取第3大元素时直接利用前面比较的结果，从而大幅减少关键词比较次数。

锦标赛排序

B

取第3大数

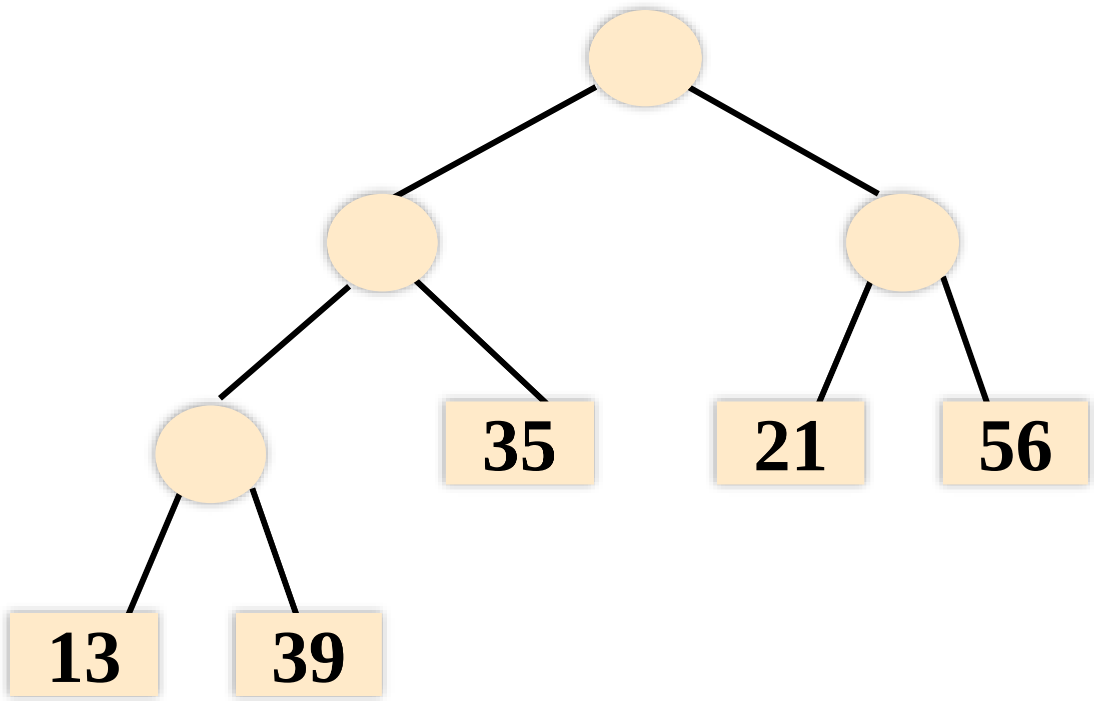


- 将剩余记录调整为新的胜者树，得到第3大元素
- 关键词比较次数O(log*n*)
- 总时间复杂度O(*n*) + O(log*n*)+…+ O(log*n*) = O(*n*log*n*)
- 对于*n*个待排序记录，锦标赛算法至少需要2*n*-1个结点来存放胜者树，故这是一个拿空间换时间的算法。

如何构建胜者树

B

给定5个元素35, 21, 56, 13, 39.



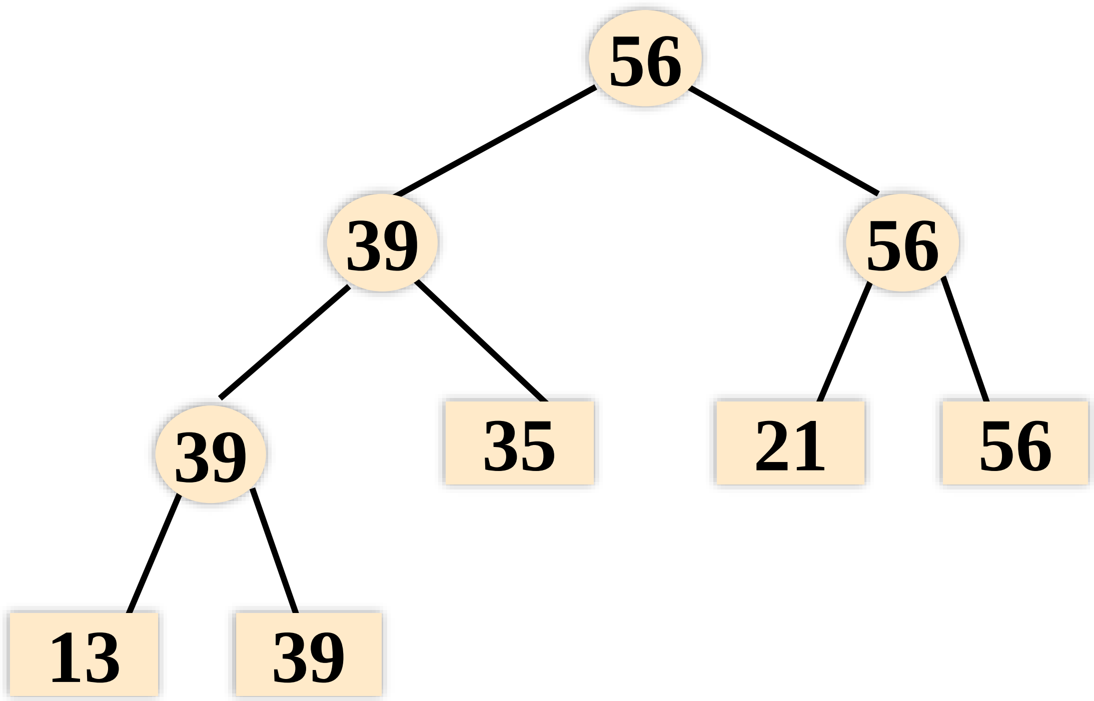
1	2	3	4	5	6	7	8	9
				35	21	56	13	39

- ✓ 因胜者树是一棵完全二叉树，可用一长度为2n-1的数组存胜者树；
- ✓ 首先将待排序的n个元素填入数组的后n个位置；
- ✓ 然后从右至左两两比较，将比较结果自右向左依次填入数组的前n-1个位置，从而实现胜者树的构建并求出最大元素。

如何构建胜者树

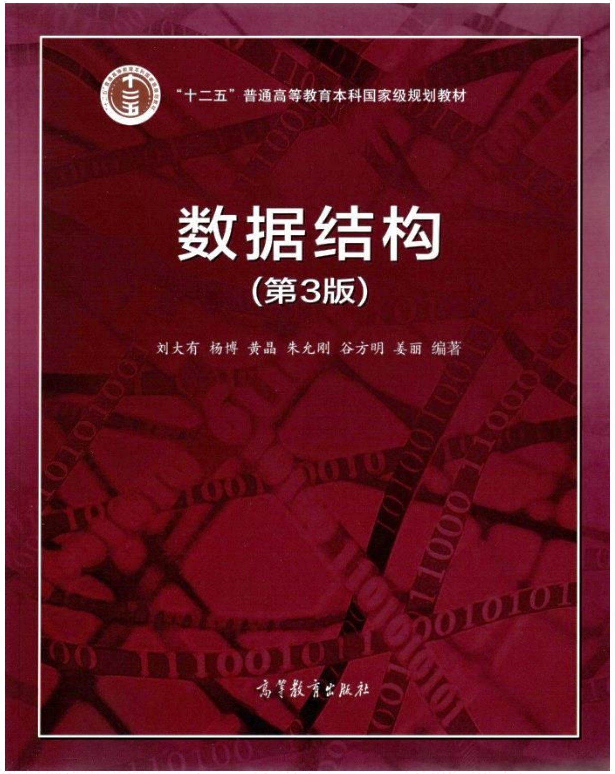
B

给定5个元素35, 21, 56, 13, 39.



56	39	56	39	35	21	56	13	39
1	2	3	4	5	6	7	8	9

- ✓ 因胜者树是一棵完全二叉树，可用一长度为2*n*-1的数组存胜者树；
- ✓ 首先将待排序的*n*个元素填入数组的后*n*个位置；
- ✓ 然后从右至左两两比较，将比较结果自右向左依次填入数组的前*n*-1个位置，从而实现胜者树的构建并求出最大元素。



堆排序

- 锦标赛排序
- **堆的概念及基本操作**
- 堆排序算法

数据之法
结构之美
算法之道

Last updated on 2024.11

zhuyungang@jlu.edu.cn

动机

堆排序

➢ **由J.W.J. Williams、Robert Floyd提出。**

ALGORITHM 232
HEAPSORT
J. W. J. WILLIAMS (Recd 1 Oct. 1963 and, revised, 15 Feb. 1964)
Elliott Bros. (London) Ltd., Borehamwood, Herts, England

comment The following procedures are related to *TREESORT* [R. W. Floyd, Alg. 113, *Comm. ACM 5* (Aug. 1962), 434, and A. F. Kaupe, Jr., Alg. 143 and 144, *Comm. ACM 5* (Dec. 1962), 604] but avoid the use of pointers and so preserve storage space. All the procedures operate on single word items, stored as elements 1 to n of the array A. The elements are normally so arranged that $A[i] \leqq A[j]$ for $2 \leqq j \leqq n$, $i = j \div 2$. Such an arrange-

ALGORITHM 245
TREESORT 3 [M1]
ROBERT W. FLOYD (Recd. 22 June 1964 and 17 Aug. 1964)
Computer Associates, Inc., Wakefield, Mass.

procedure *TREESORT* 3 (M, n);
  value n; **array** M; **integer** n;
comment *TREESORT* 3 is a major revision of *TREESORT* [R. W. Floyd, Alg. 113, *Comm. ACM 5* (Aug. 1962), 434] suggested by *HEAPSORT* [J. W. J. Williams, Alg. 232, *Comm. ACM 7* (June 1964), 347] from which it differs in being an in-place sort. It is shorter and probably faster, requiring fewer comparisons and only one division. It sorts the array $M[1:n]$, requiring no more than $2 \times (2 \uparrow p-2) \times (p-1)$, or approximately $2 \times n \times (\log_2(n)-1)$ comparisons and half as many exchanges in the worst case to sort $n = 2 \uparrow p - 1$ items. The algorithm is

[1] **J. W. J. Williams**. Heapsort, *Communications of the ACM*, 7(6):378-348, 1964.
[2] **Robert W. Floyd**. Treesort 3, *Communications of the ACM*, 7(12):701, 1964.

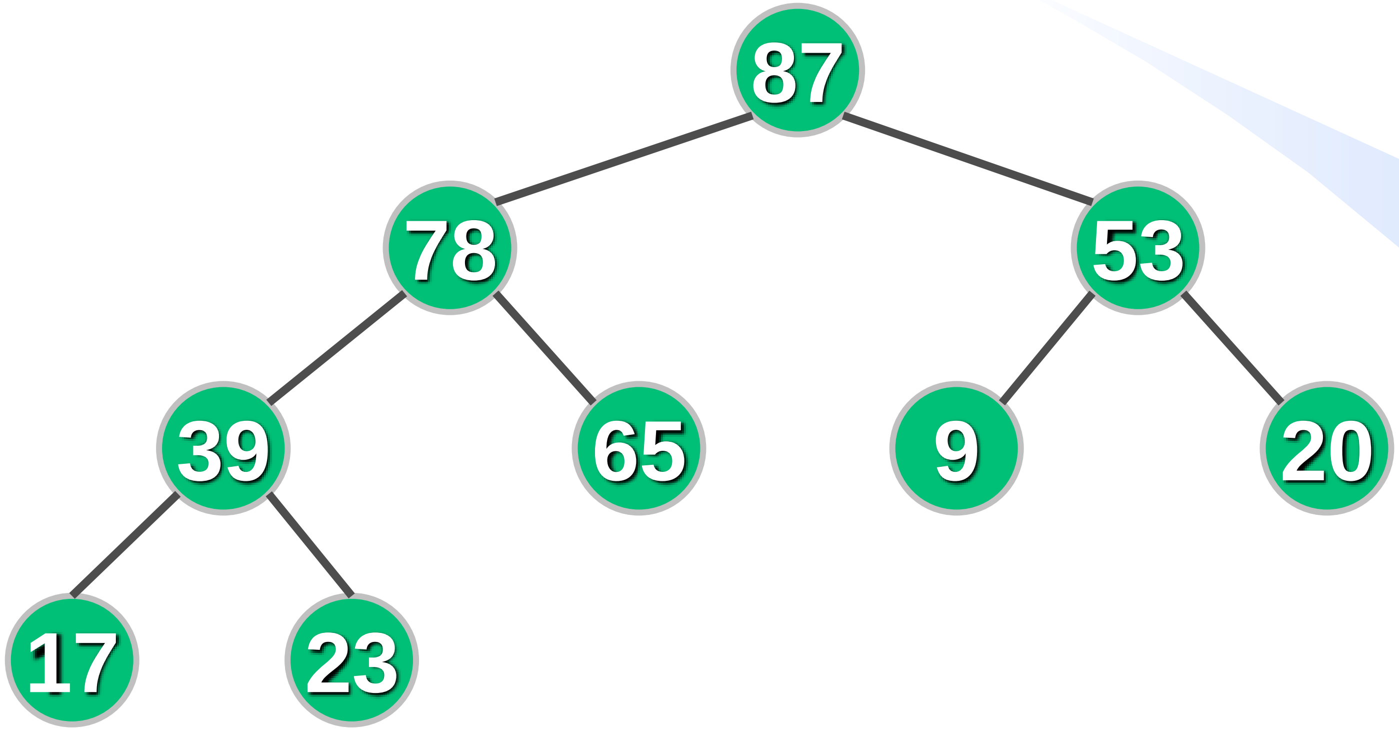
Coca-Cola
Estrella Damm
movistar
LA VANGUARDIA
SEREM
3
3
REPSOL
BASF
WE WANT TO VOTE!
www.jnc.cat
TARRAGONA

二叉堆（Binary Heap），简称堆（Heap）

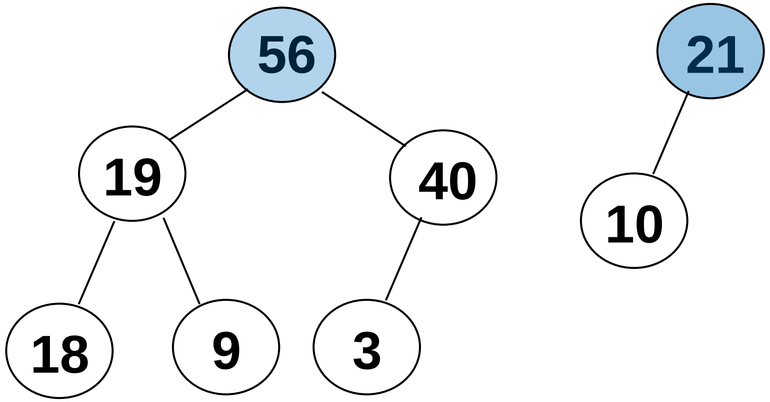
A

- 最大堆（大根堆）：一棵完全二叉树，其中任意结点的关键词大于等于它的两个孩子的关键词。
  - ✓ 结构性：完全二叉树。
  - ✓ 堆序性：任意结点的关键词大于等于其孩子的关键词。
- 最小堆（小根堆）：一棵完全二叉树，其中任意结点的关键词小于等于它的两个孩子的关键词。
- 堆的优势：最大堆中根结点的关键词最大，最小堆中根结点的关键词最小。

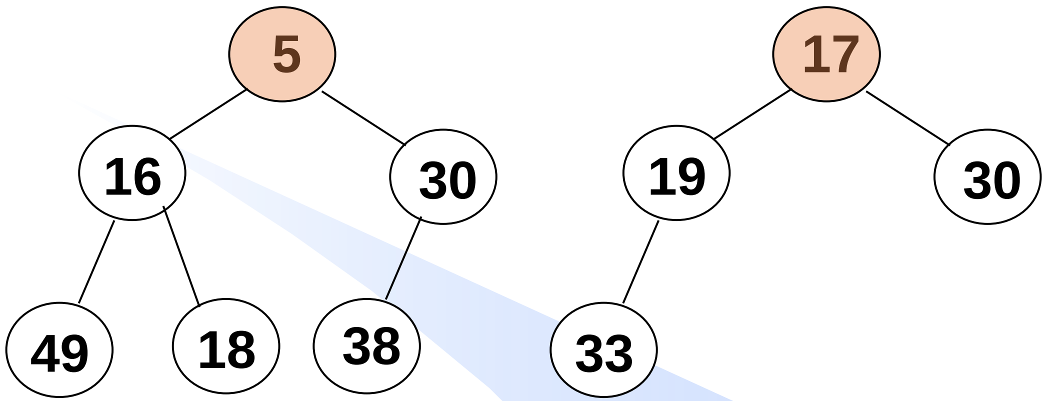
最大堆

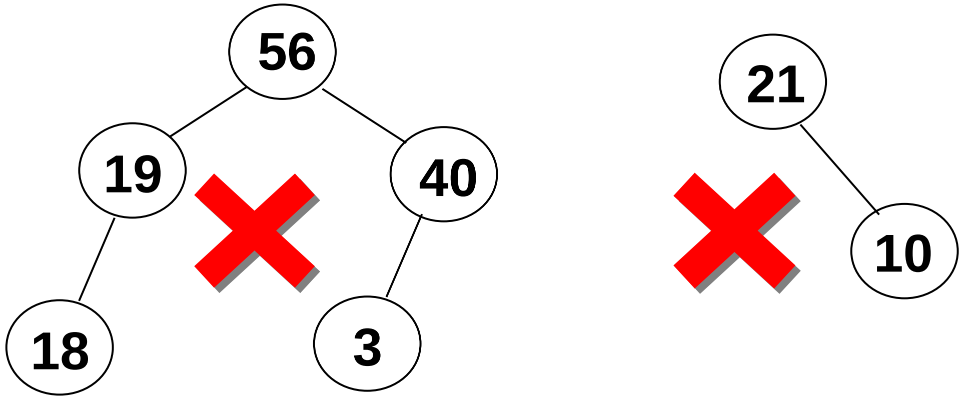


最大堆

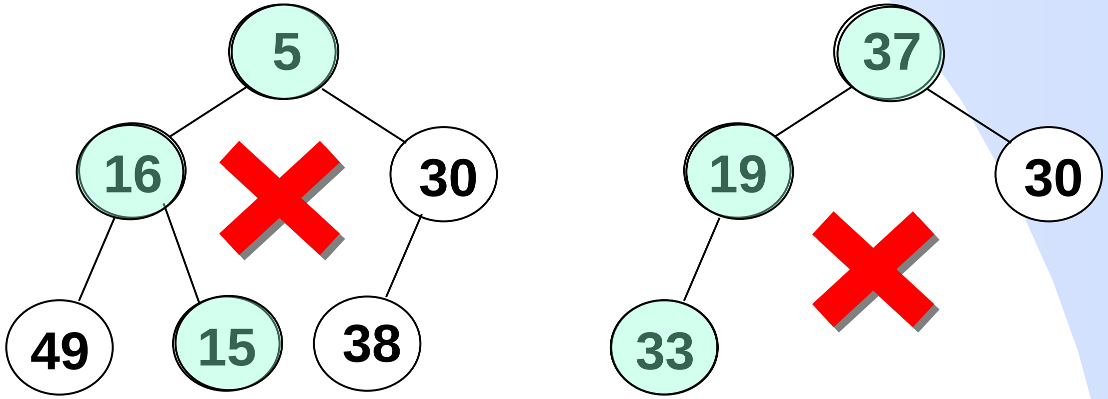


最小堆





不满足结构性



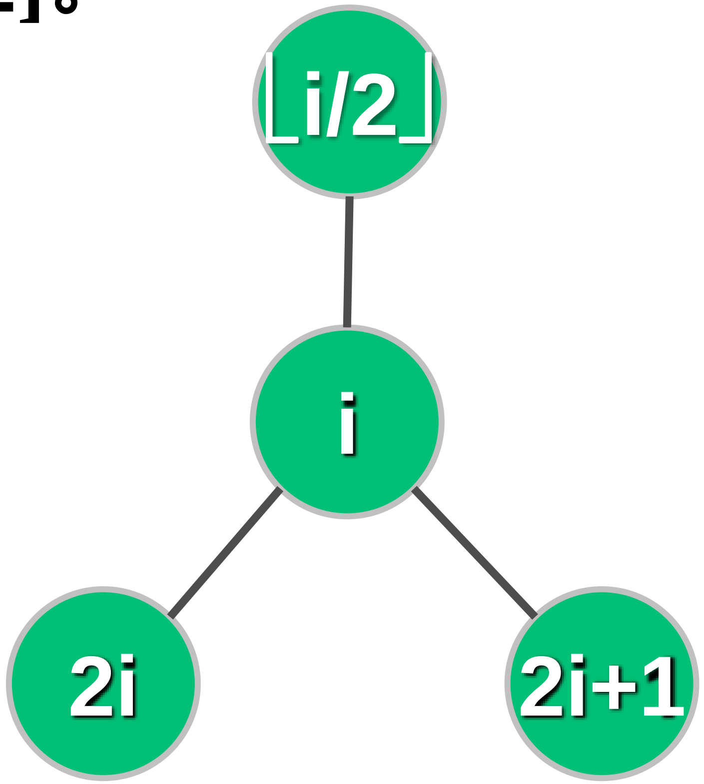
不满足堆序性

不是堆

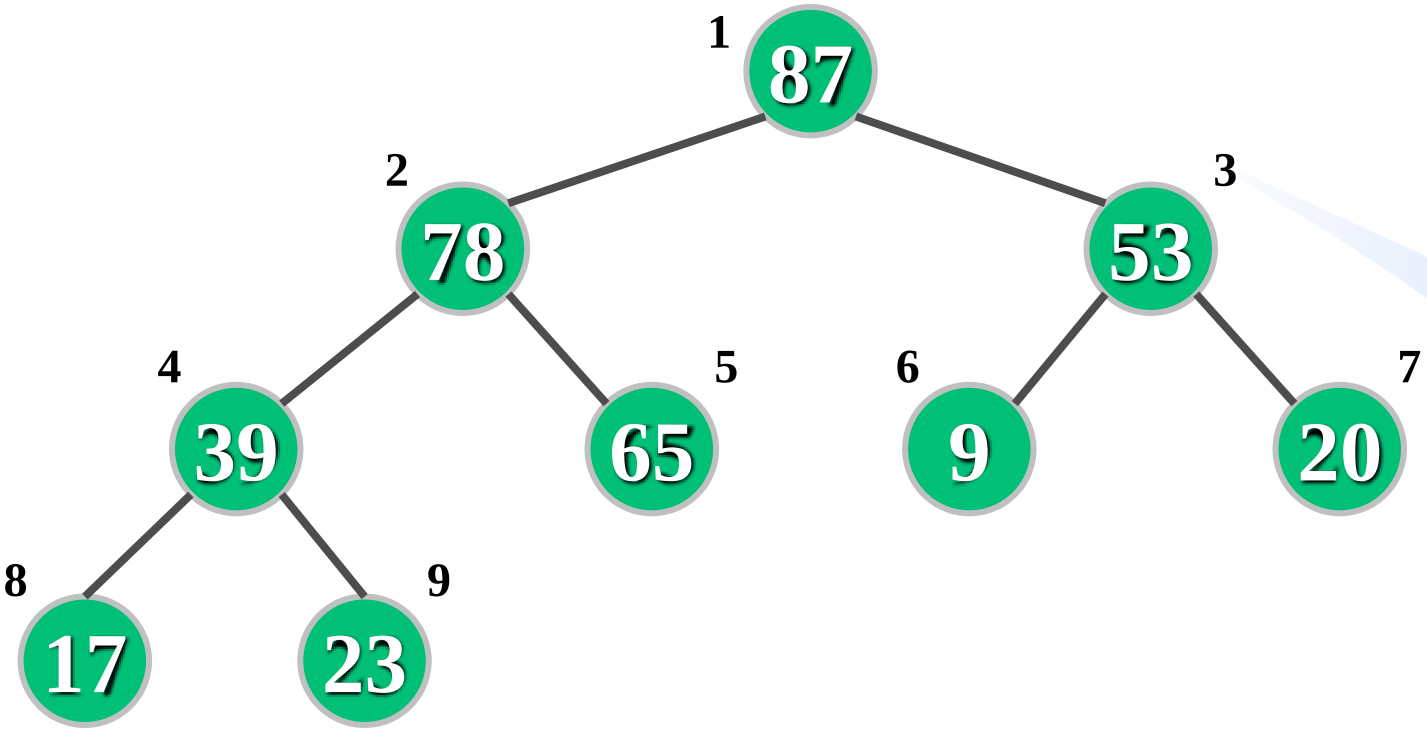
回顾：　完全二叉树的顺序存储

A

$R[1]$存储二叉树的根结点。结点$R[i]$的左孩子（若有的话）存放在$R[2i]$处，而$R[i]$的右孩子（若有的话）存放在$R[2i+1]$处。$R[i]$的父结点是$R[i/2]$。



堆的顺序存储



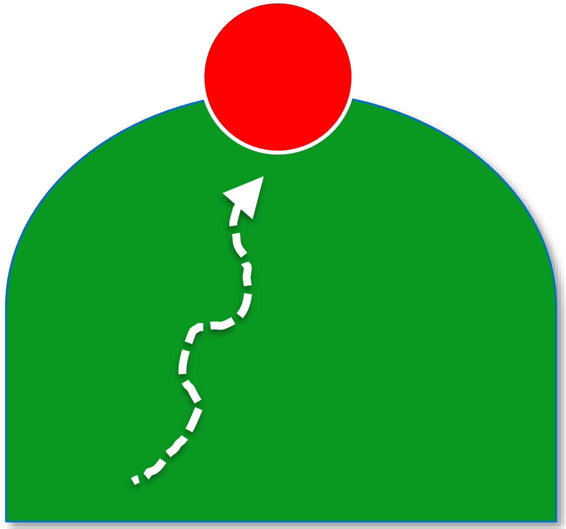
R		87	78	53	39	65	9	20	17	23
i	0	1	2	3	4	5	6	7	8	9

- ✓*R*[1]存根结点；
- ✓结点*R*[*i*]的左孩子（若有的话）存放在*R*[2*i*]处；
- ✓*R*[*i*]的右孩子（若有的话）存放在*R*[2*i*+1]处；
- ✓*R*[*i*]的父结点为*R*[*i*/2]。

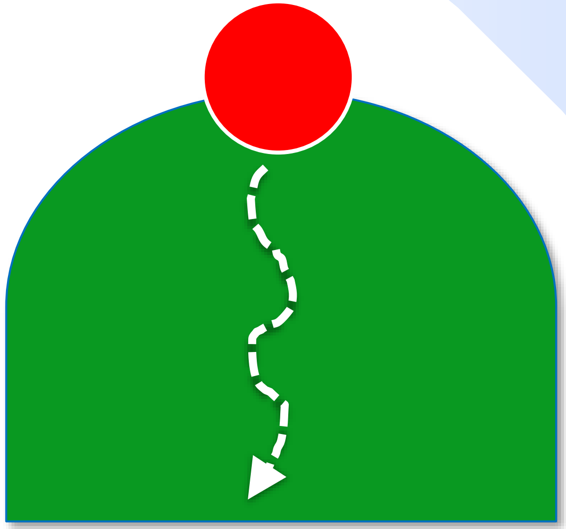
堆的两个基本操作

A

上浮操作

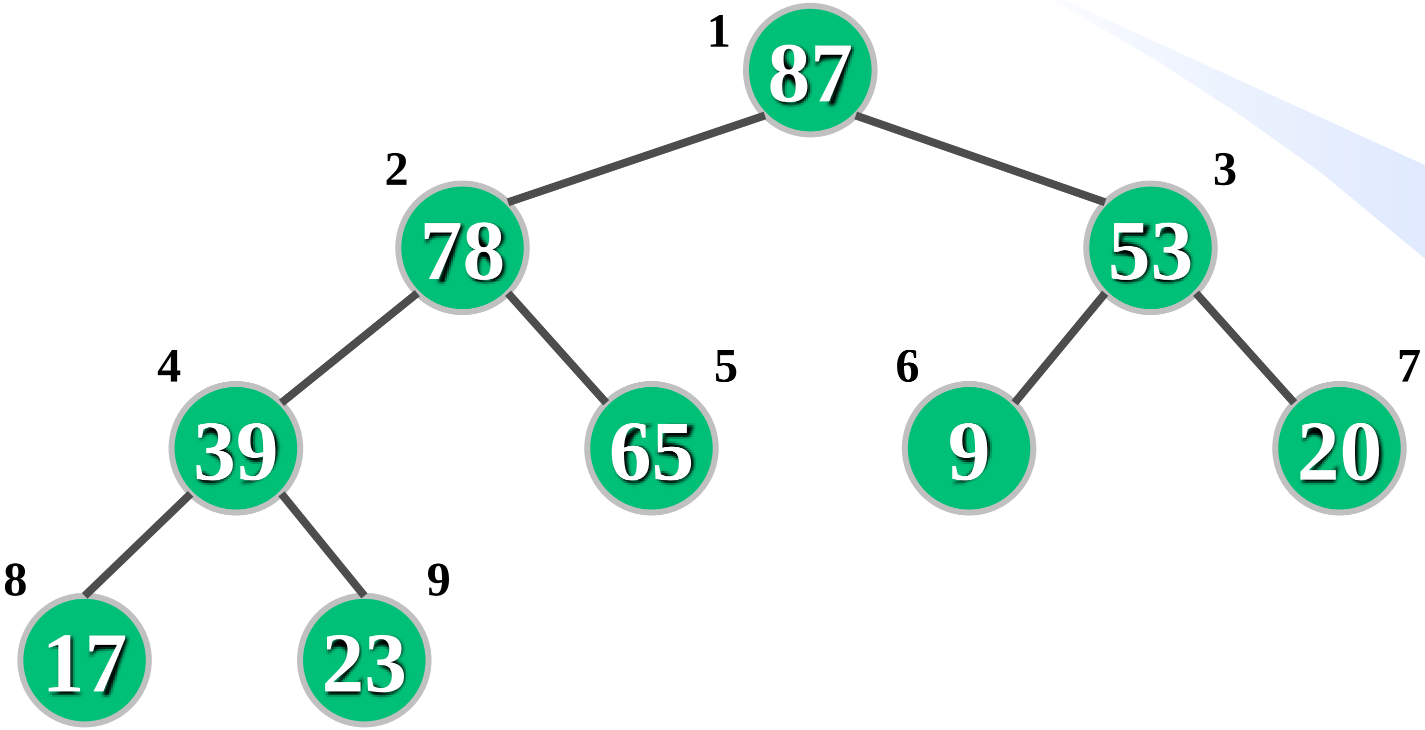


下沉操作



堆的上浮操作

当大根堆的元素值R[*i*]变大时，该结点**可能会**上浮；



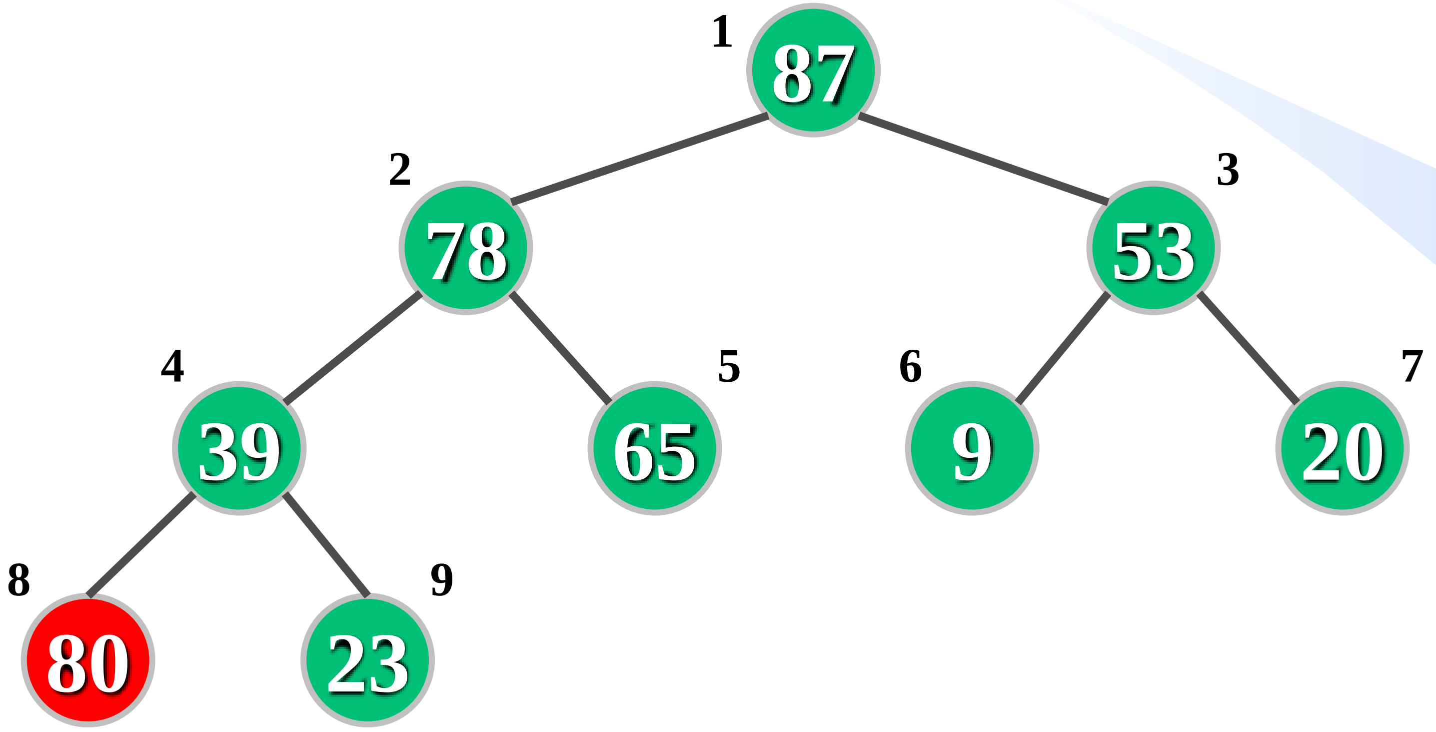
将R[8]修改为80

87	78	53	39	65	9	20	17	23
1	2	3	4	5	6	7	8	9

堆的上浮操作

A

当大根堆的元素值R[*i*]变大时，该结点可能会上浮；

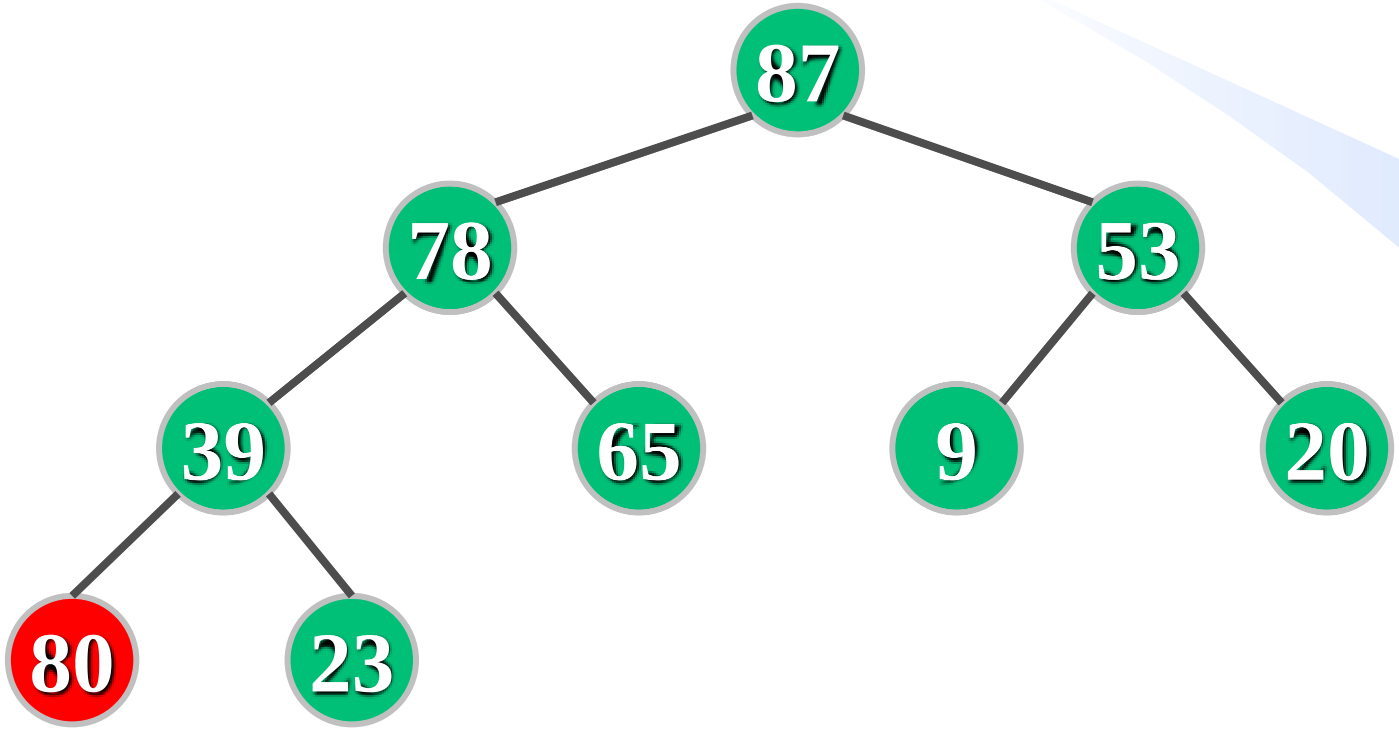


将R[8]修改为80

87	78	53	39	65	9	20	80	23
1	2	3	4	5	6	7	8	9

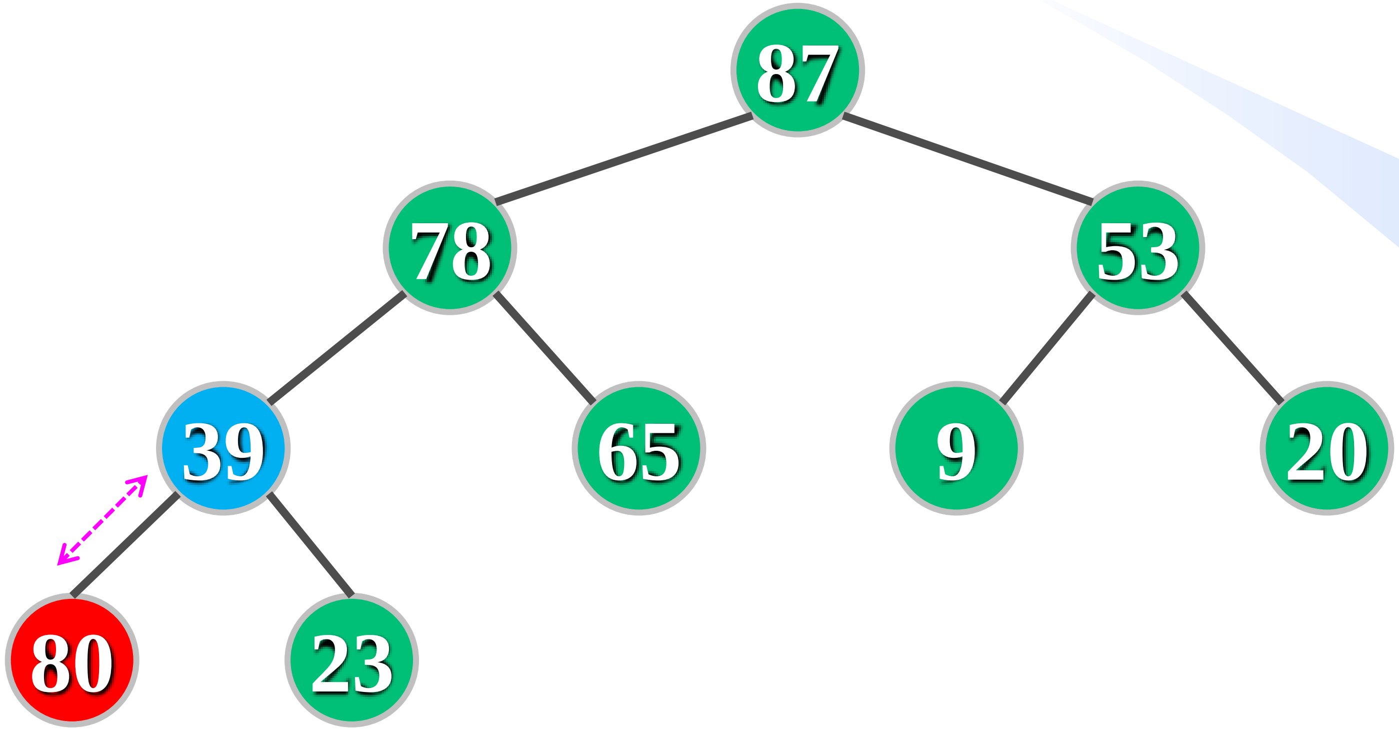
堆的上浮操作

当大根堆的元素值R[*i*]变大时，该结点可能会上浮；



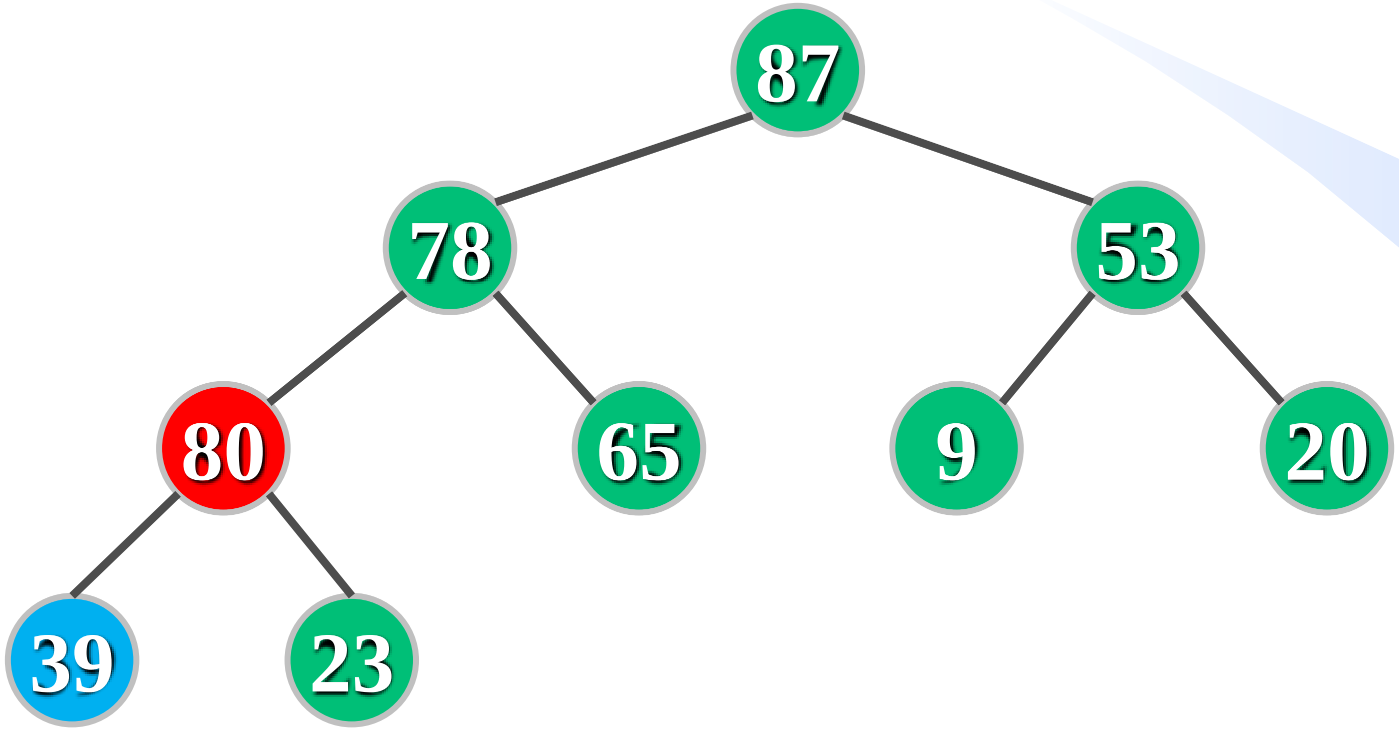
堆的上浮操作

当大根堆的元素值R[*i*]变大时，该结点可能会上浮；



堆的上浮操作

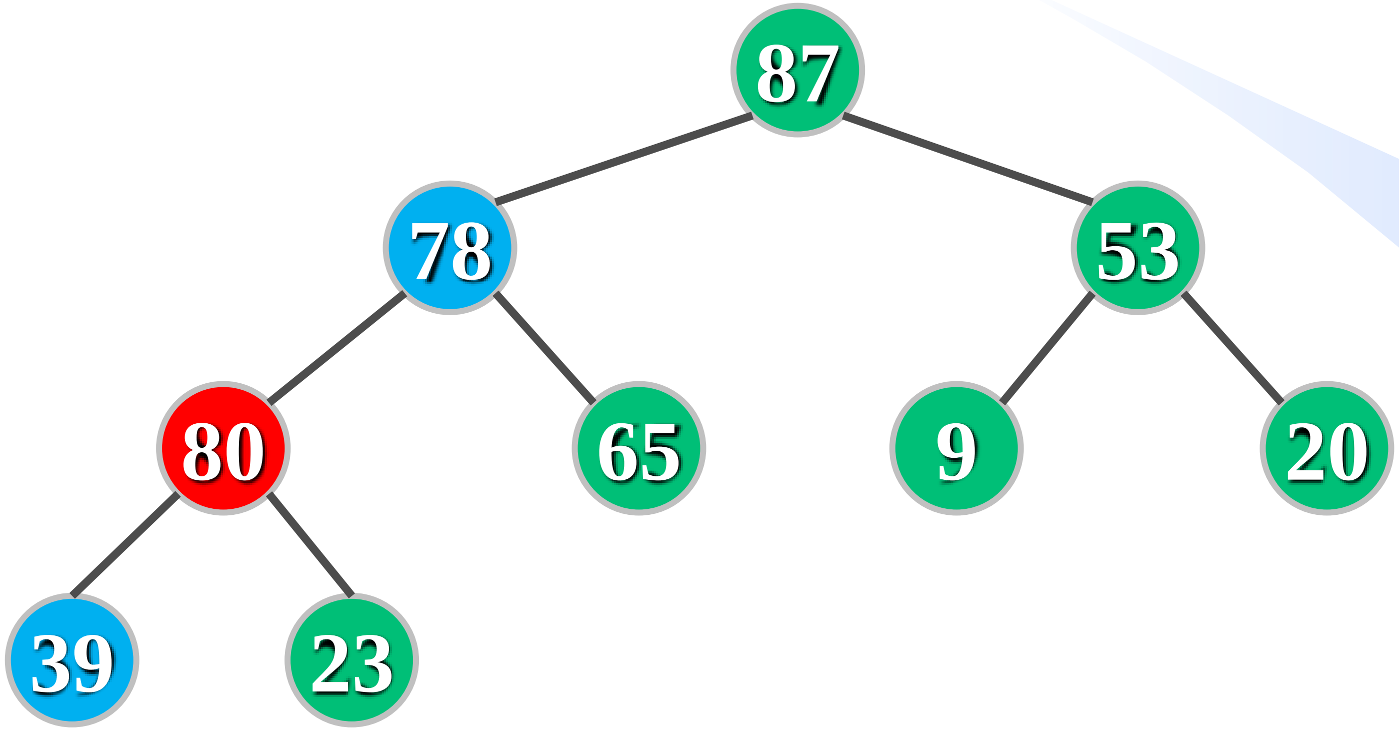
当大根堆的元素值R[*i*]变大时，该结点可能会上浮；



堆的上浮操作

A

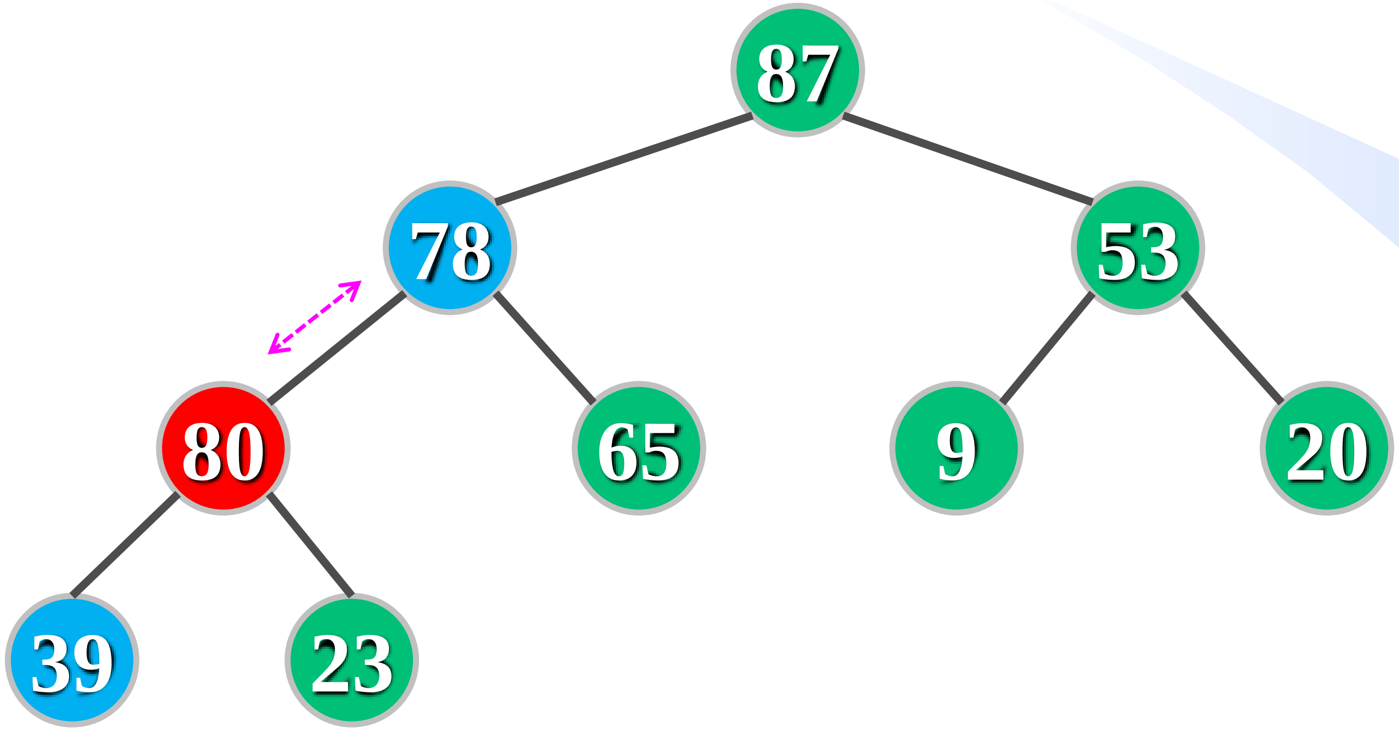
当大根堆的元素值R[*i*]变大时，该结点**可能会**上浮；



堆的上浮操作

A

当大根堆的元素值R[*i*]变大时，该结点**可能会**上浮；

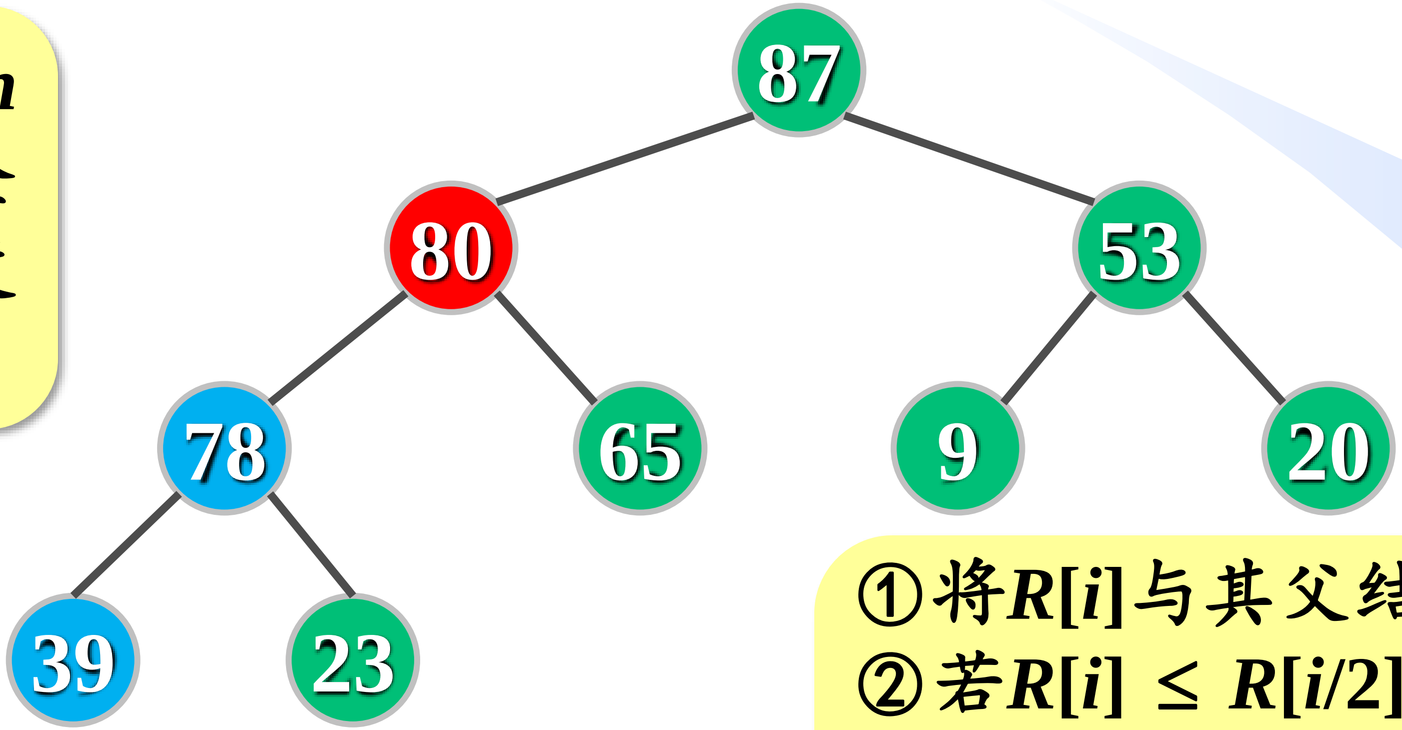


A

堆的上浮操作

当大根堆的元素值R[*i*]变大时，该结点**可能会**上浮；

回顾： 具有n个结点的完全二叉树的高度是$\lfloor \log_2 n \rfloor$.



时间复杂度取决于树的高度$O(\log n)$

①将$R[i]$与其父结点$R[i/2]$比较；
②若$R[i] \le R[i/2]$, 则已满足堆序性, 算法结束；
③否则交换$R[i]$和$R[i/2]$, 令i上行指向其父亲$i \leftarrow i/2$, 执行①。

堆的上浮操作

A

```
void ShiftUp(int R[], int n, int i){
    //堆元素R[i]上浮, 数组R[ ]存储堆, n为堆包含的元素个数
    while(i>1 && R[i]>R[i/2]){   //R[i]比父亲大且i不是根
        swap(R[i], R[i/2]);       //交换R[i]和父亲
        i/=2;                     //结点i继续上浮
    }
}
```

时间复杂度O(log*n*)

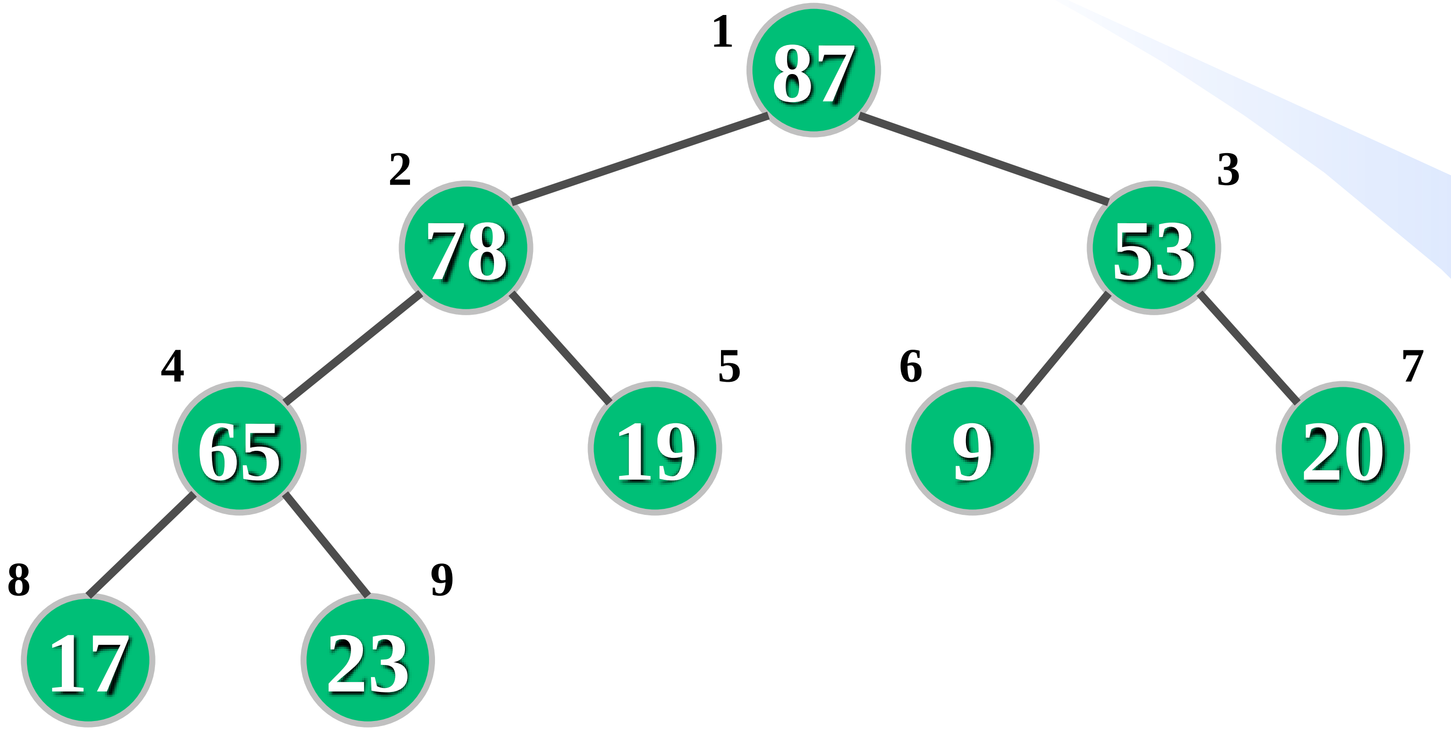
也称为Swim或PercolateUp

```
void swap(int &a, int &b){
   int temp = a;
   a = b;
   b = temp;
}
```

①将*R*[*i*]与其父结点*R*[*i*/2]比较；
②若*R*[*i*] ≤ *R*[*i*/2], 则已满足堆序性, 算法结束；
③否则交换*R*[*i*]和*R*[*i*/2], 令*i*指向其父亲*i*←*i*/2, 执行①。

堆的下沉操作

当大根堆的元素值R[*i*]变小时，该结点**可能会**下沉；



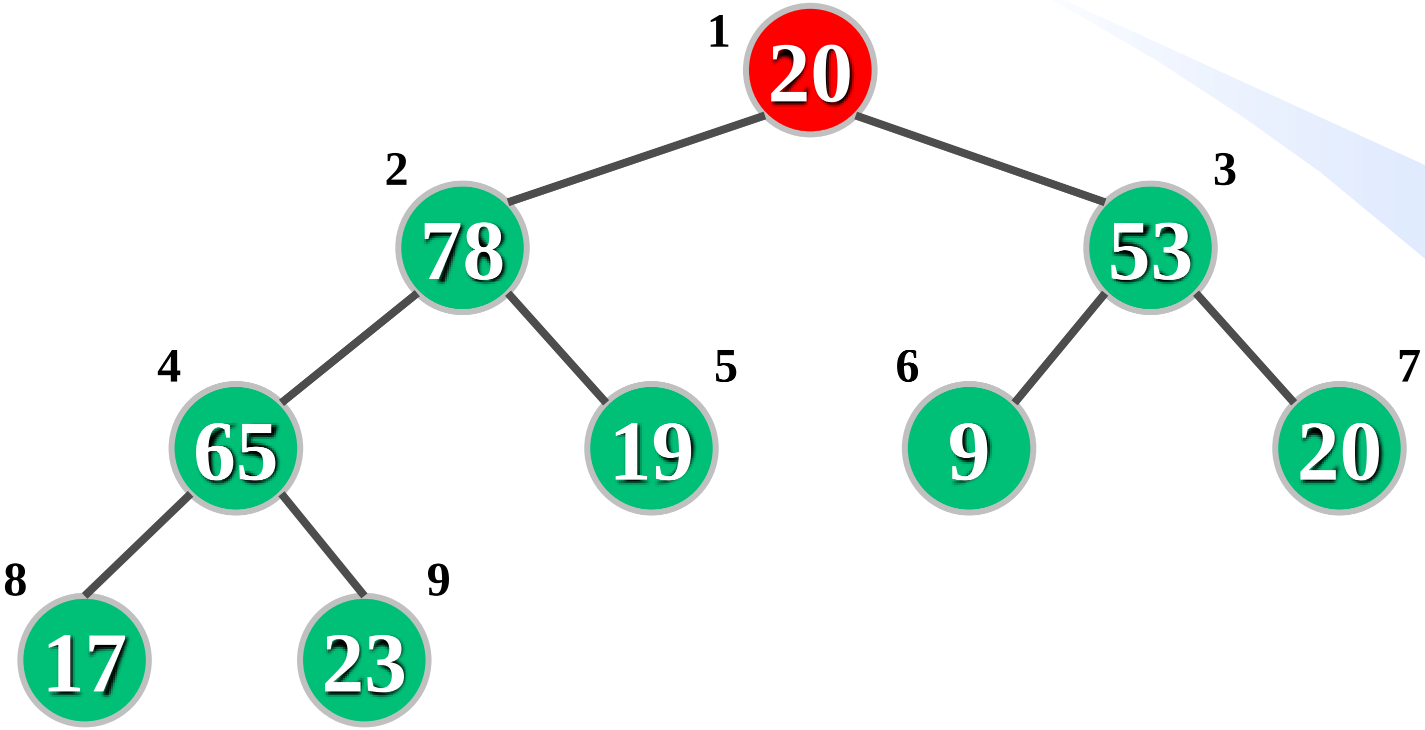
将R[1]修改为20

87	78	53	65	19	9	20	17	23
1	2	3	4	5	6	7	8	9

堆的下沉操作

A

当大根堆的元素值R[*i*]变小时，该结点**可能会**下沉；

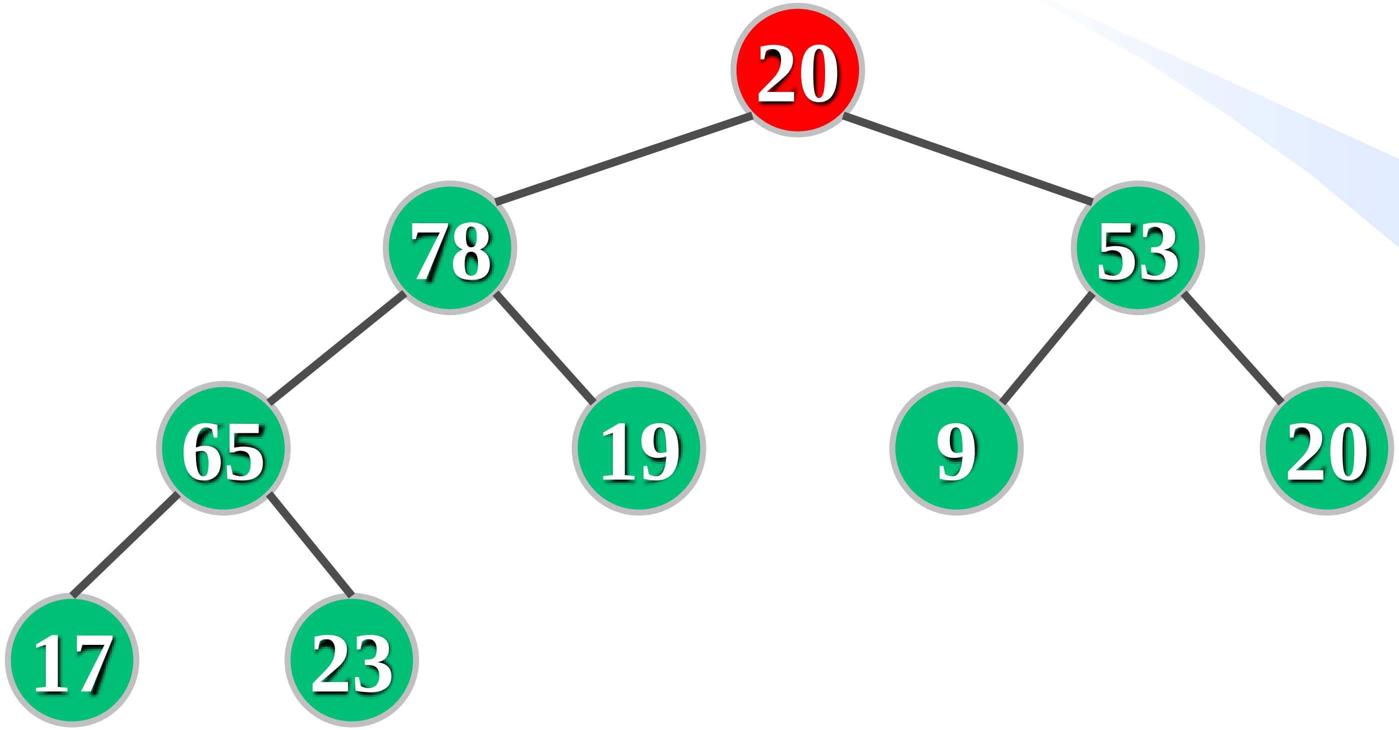


将R[1]修改为20

20	78	53	65	19	9	20	17	23
1	2	3	4	5	6	7	8	9

堆的下沉操作

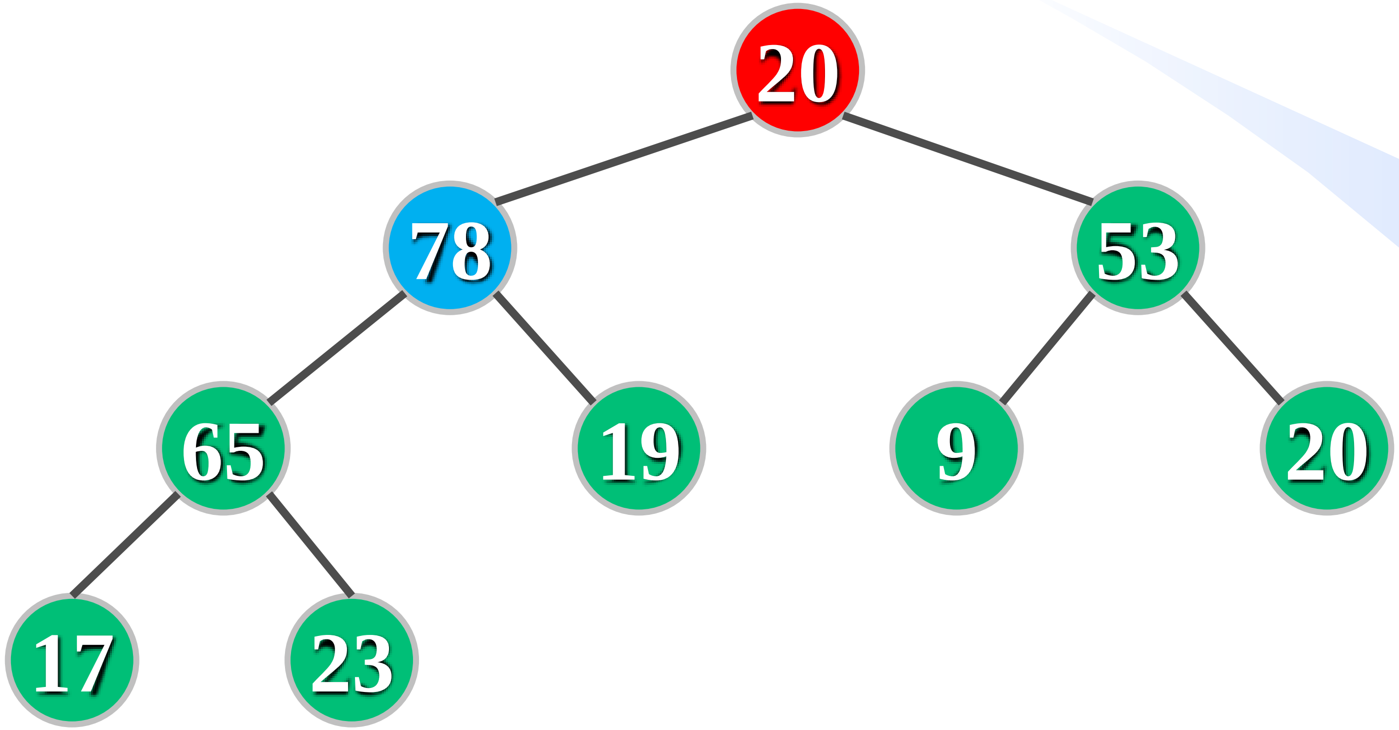
当大根堆的元素值R[*i*]变小时，该结点**可能会**下沉；



堆的下沉操作

A

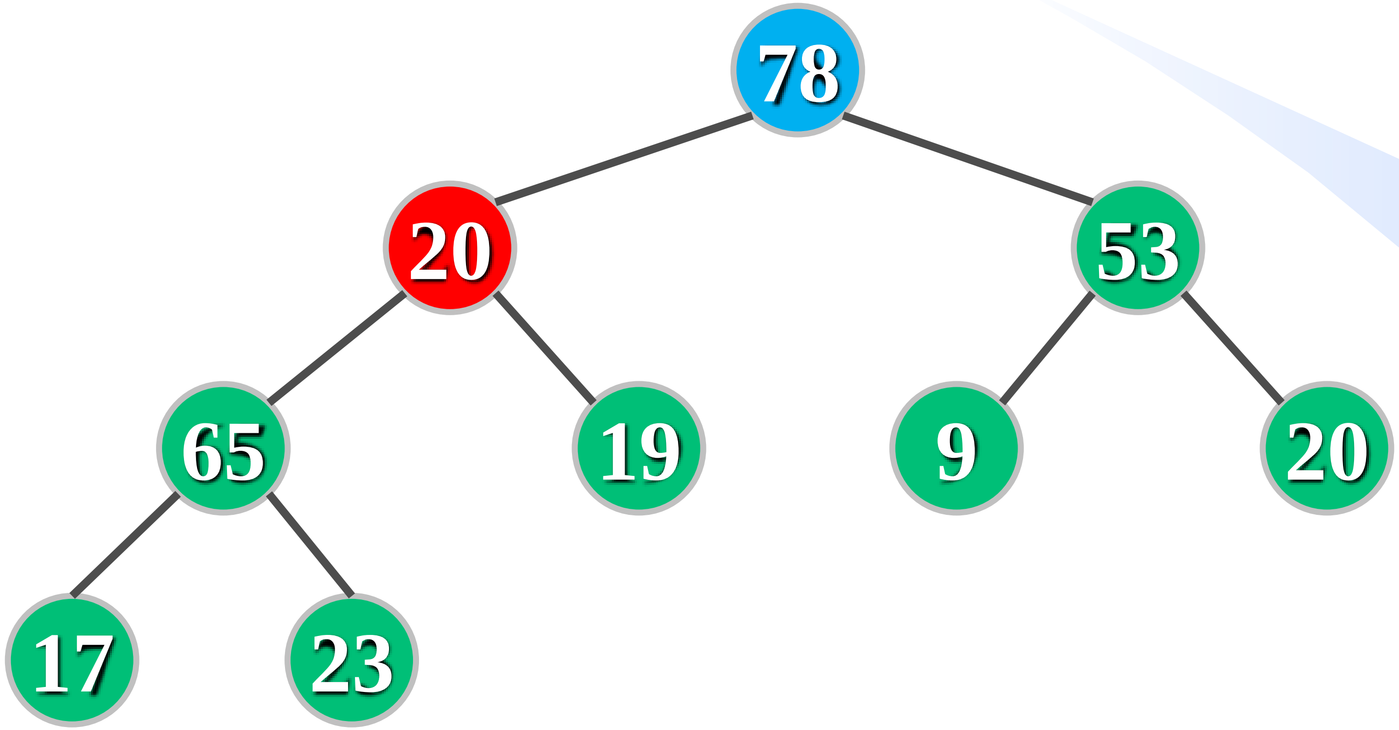
当大根堆的元素值R[*i*]变小时，该结点可能会下沉；



堆的下沉操作

A

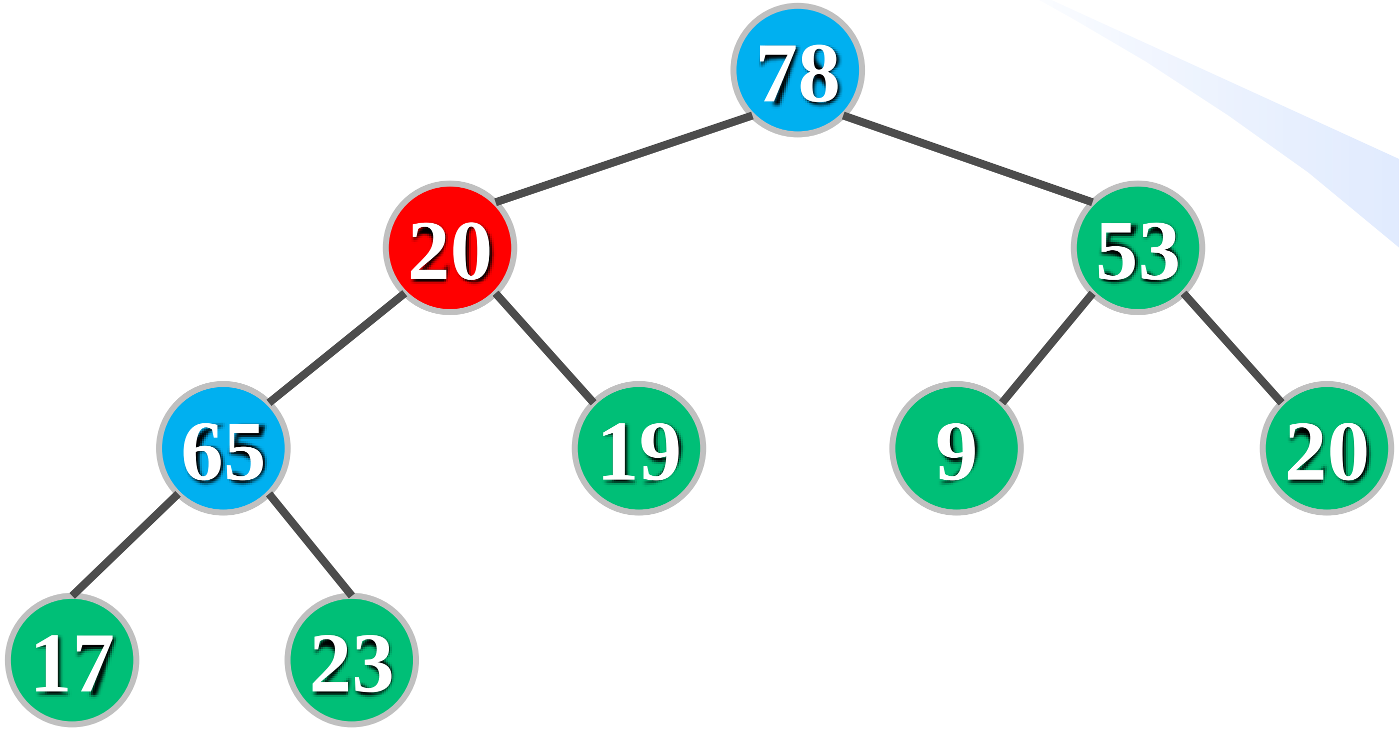
当大根堆的元素值R[i]变小时，该结点可能会下沉；



堆的下沉操作

A

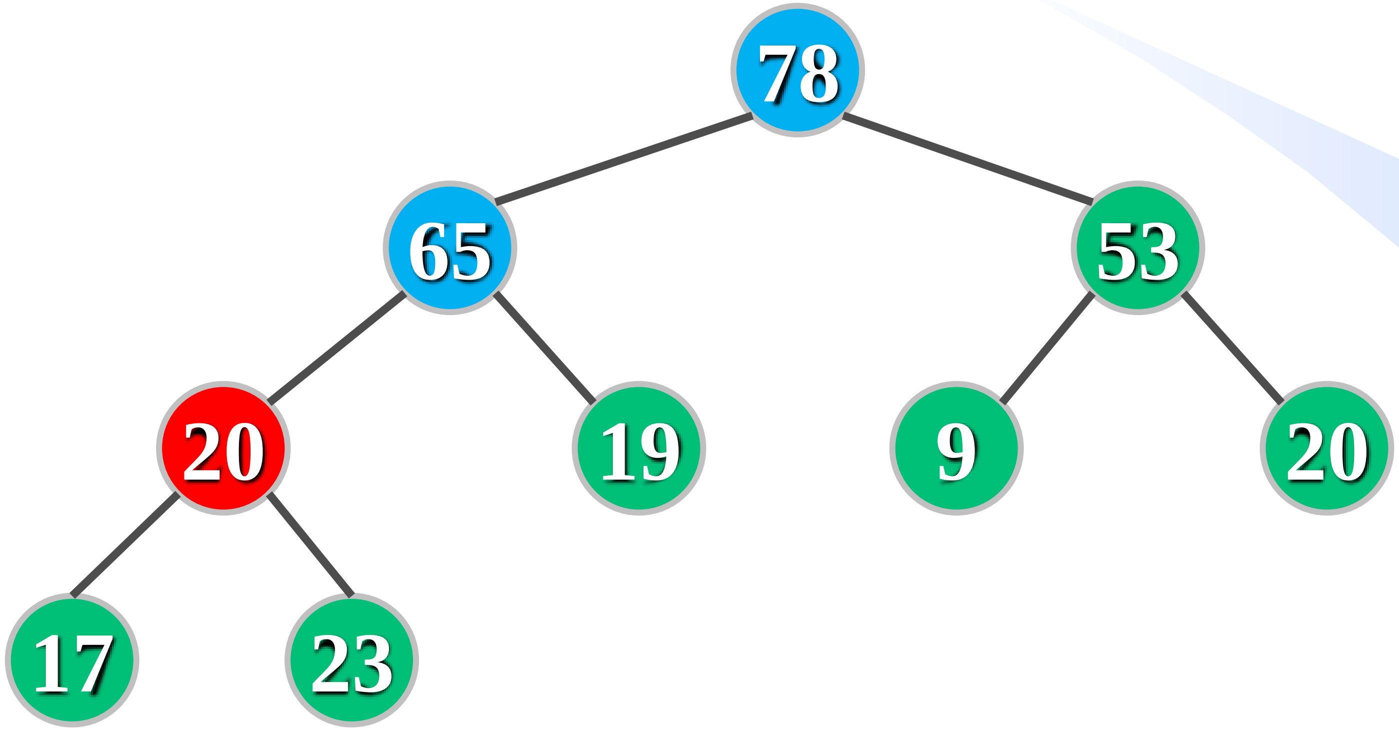
当大根堆的元素值R[*i*]变小时，该结点可能会下沉；



堆的下沉操作

A

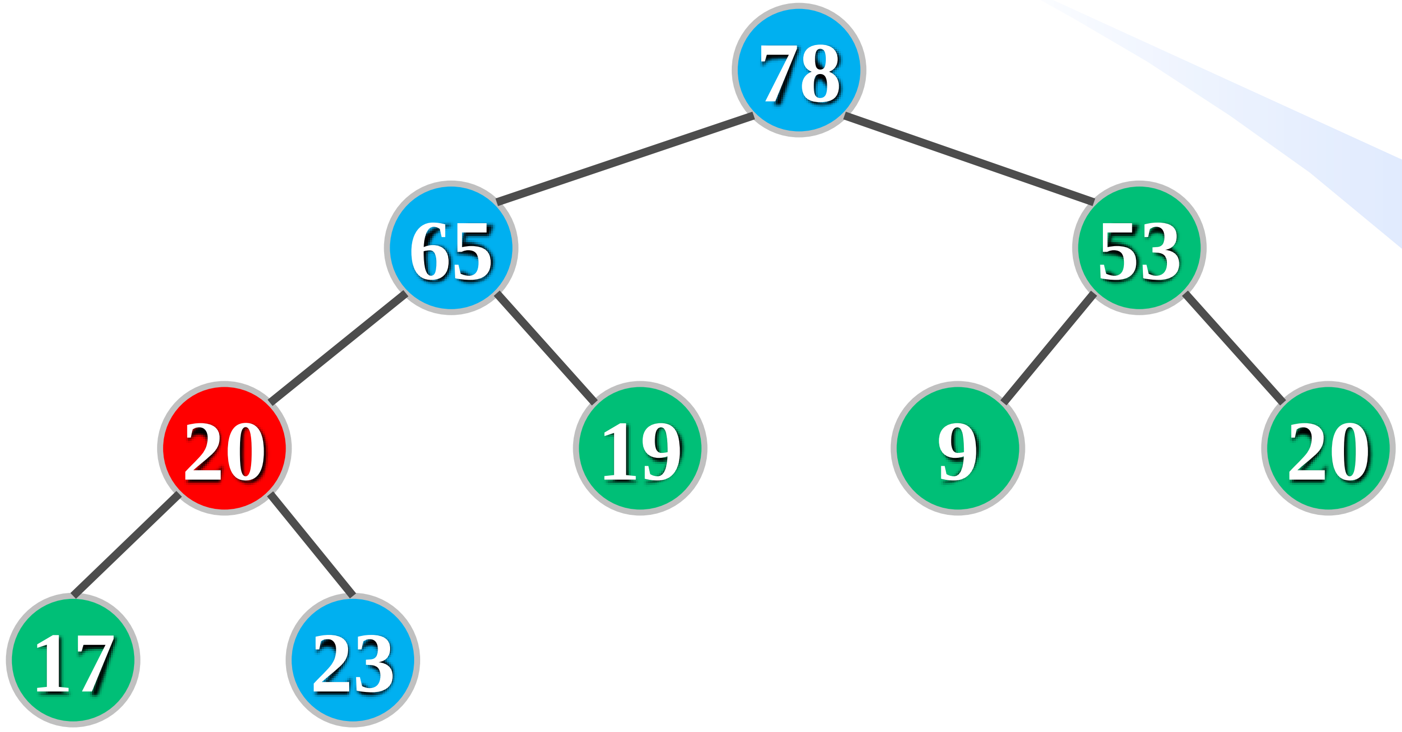
当大根堆的元素值R[*i*]变小时，该结点可能会下沉；



堆的下沉操作

A

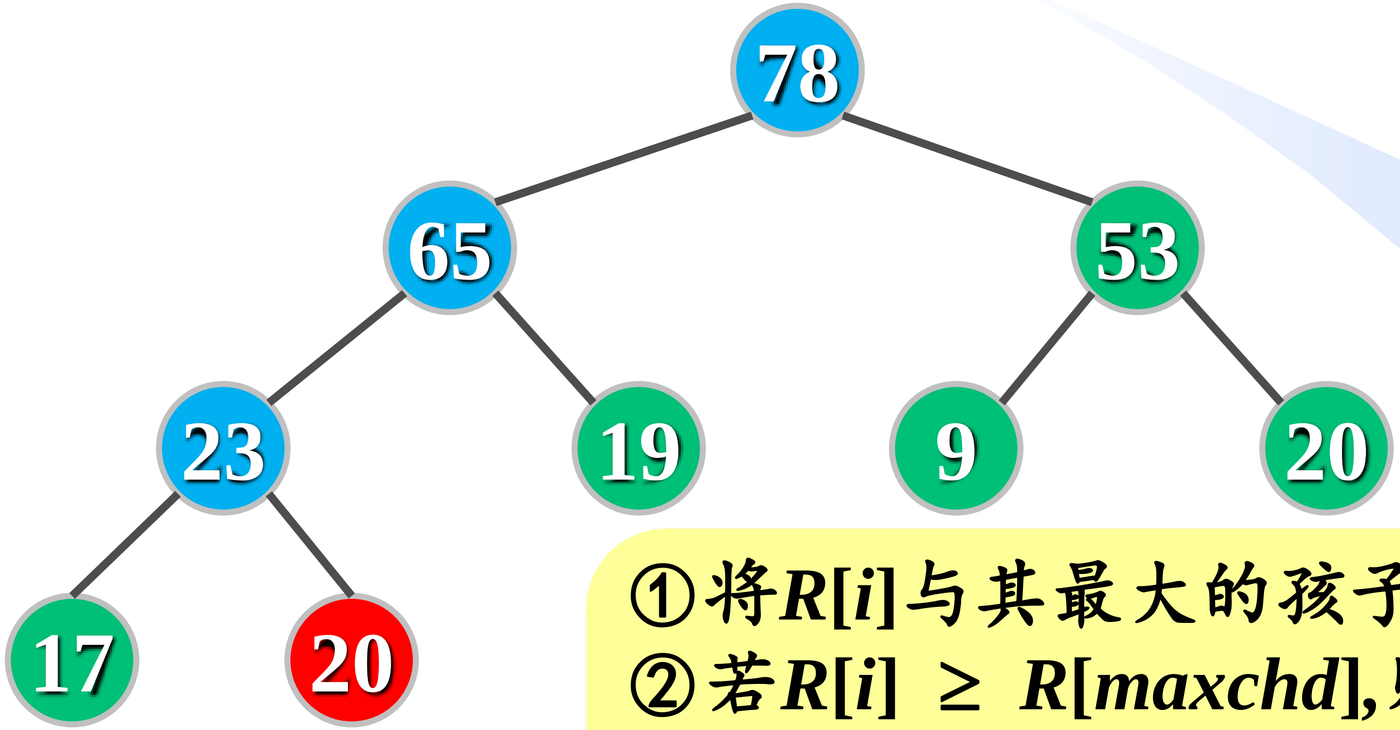
当大根堆的元素值R[*i*]变小时，该结点可能会下沉；



堆的下沉操作

A

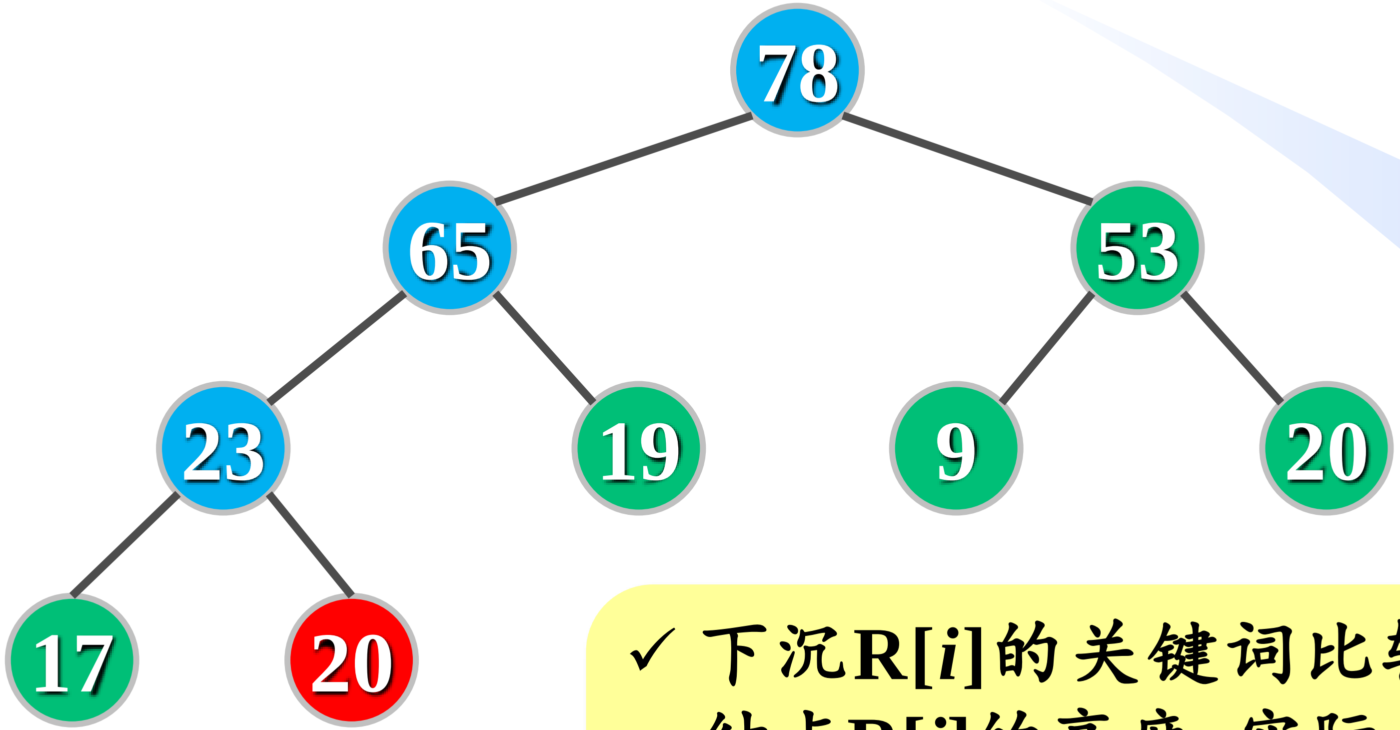
当大根堆的元素值R[*i*]变小时，该结点**可能会**下沉；



①将*R*[*i*]与其最大的孩子*R*[*maxchd*]比较；
②若*R*[*i*] ≥ *R*[*maxchd*],则已满足堆序性，算法结束；
③否则交换*R*[*i*]和*R*[*maxchd*]，令*i*下行 *i*←*maxchd*，返回①

堆的下沉操作

当大根堆的元素值R[*i*]变小时，该结点**可能会**下沉；



- ✓ 下沉R[*i*]的关键词比较次数：取决于结点R[*i*]的高度, 实际是高度×2;
- ✓ 每个非叶结点对应2次关键词比较;
- ✓ 时间复杂度O(log*n*)

堆的下沉操作

A

```
void ShiftDown(int R[], int n, int i) {
   //堆元素R[i]下沉, 数组R[ ]存储堆, n为堆包含的元素个数
   while(i <= n/2){          //i最多下行至最后一个非叶结点
      int maxchd = 2*i;   // 假定最大孩子为左孩子
      if(maxchd+1<=n && R[maxchd]<R[maxchd+1])
         maxchd++;          //i的右孩子是最大孩子
      if(R[i] >= R[maxchd]) return;
      swap(R[maxchd],R[i]); // R[i]的最大孩子比R[i]大
      i = maxchd;               // 结点i继续下沉
   }
}
```

若i有右孩子

i的右孩子比左孩子大

也称为Sink或PercolateDown

时间复杂度 O(log*n*)

初始建堆

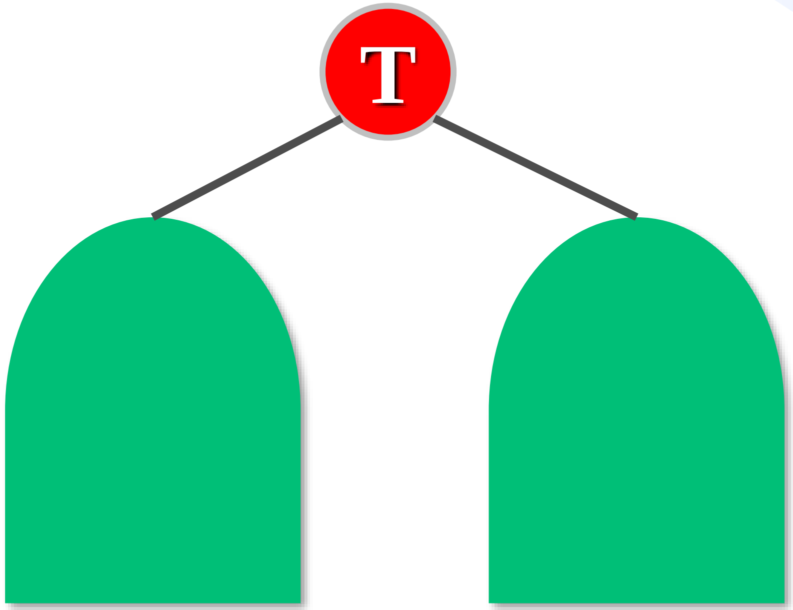
自顶向下

递归思路

① 建立T的左子堆

② 建立T的右子堆

③ 结点T下沉。



自底向上

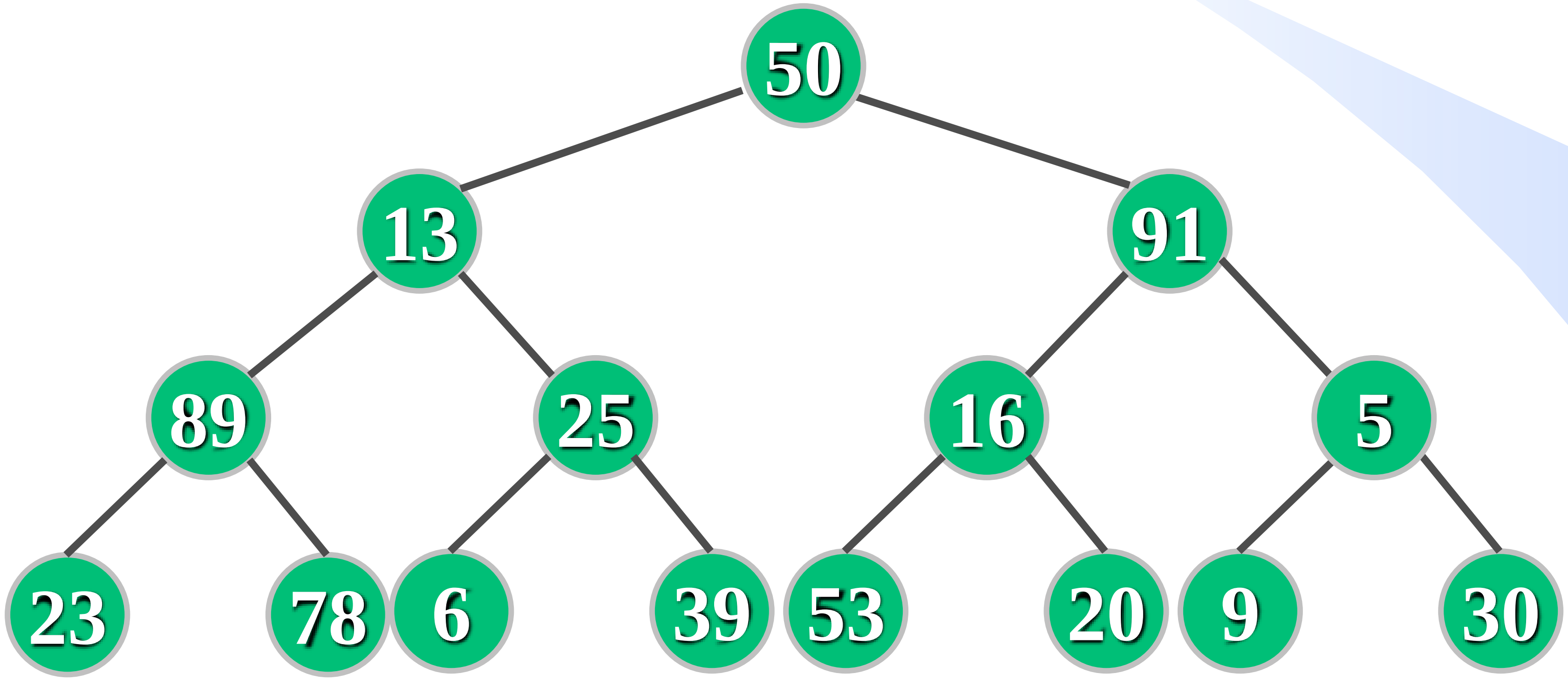
动态规划思路

从最后一个非叶结点开始，依次建立以每个非叶结点为根的子堆。

初始建堆

A

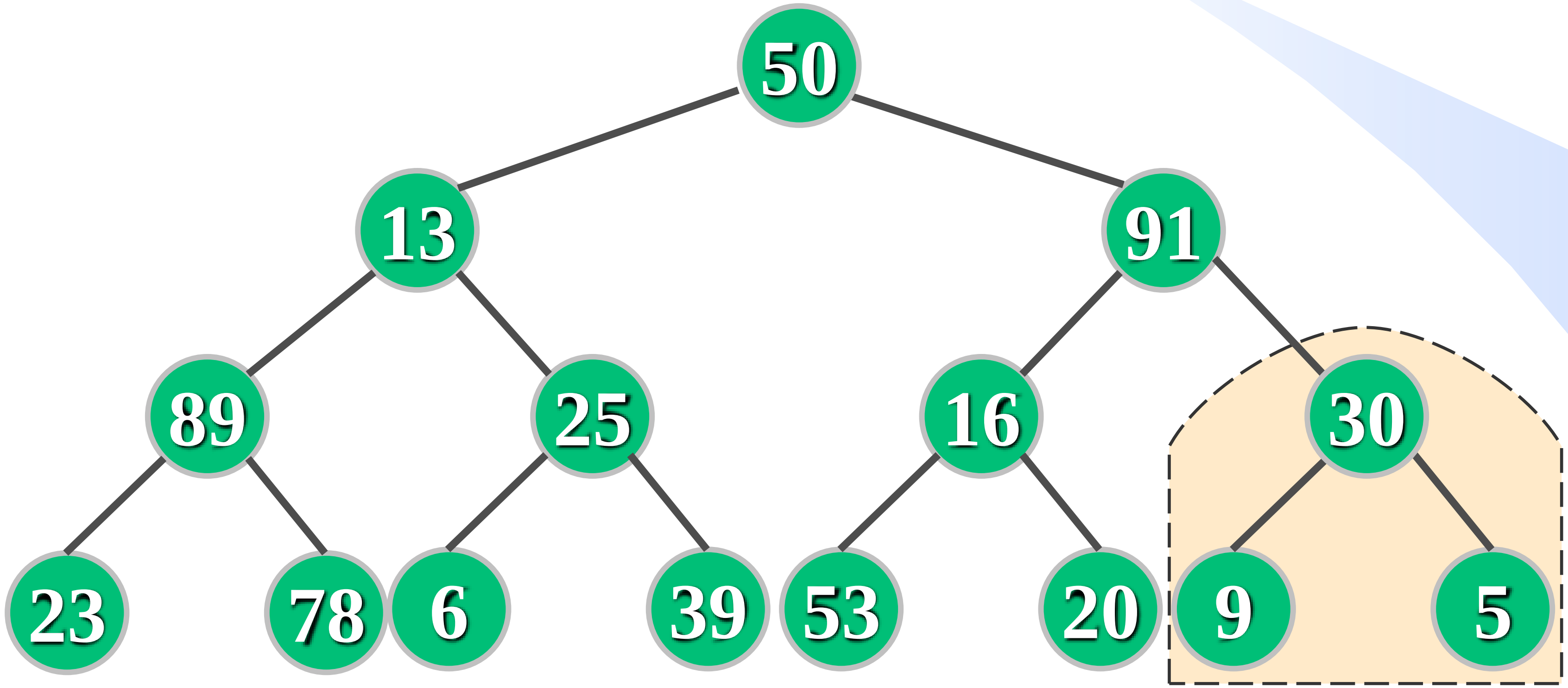
从最后一个非叶结点开始，依次建立以每个非叶结点为根的子堆。



初始建堆

A

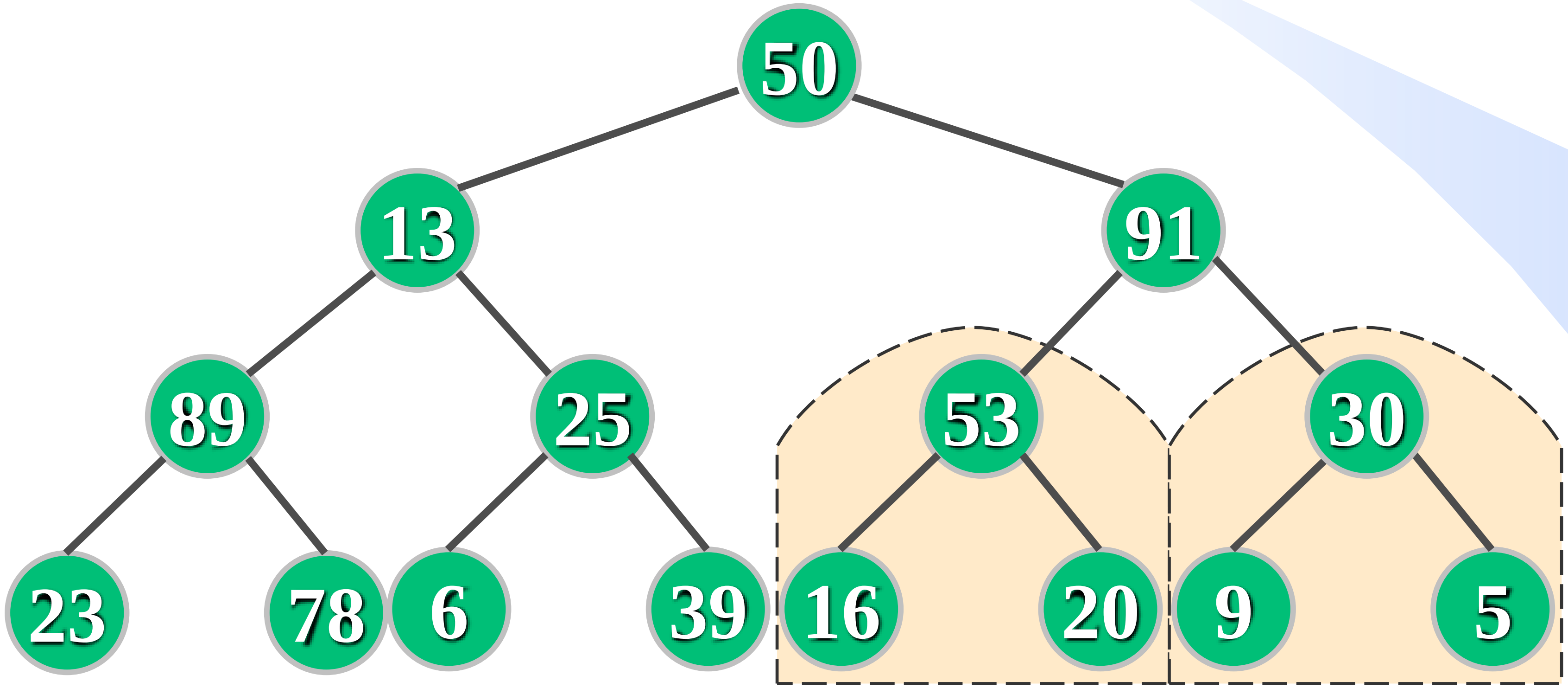
从最后一个非叶结点开始，依次建立以每个非叶结点为根的子堆。



初始建堆

A

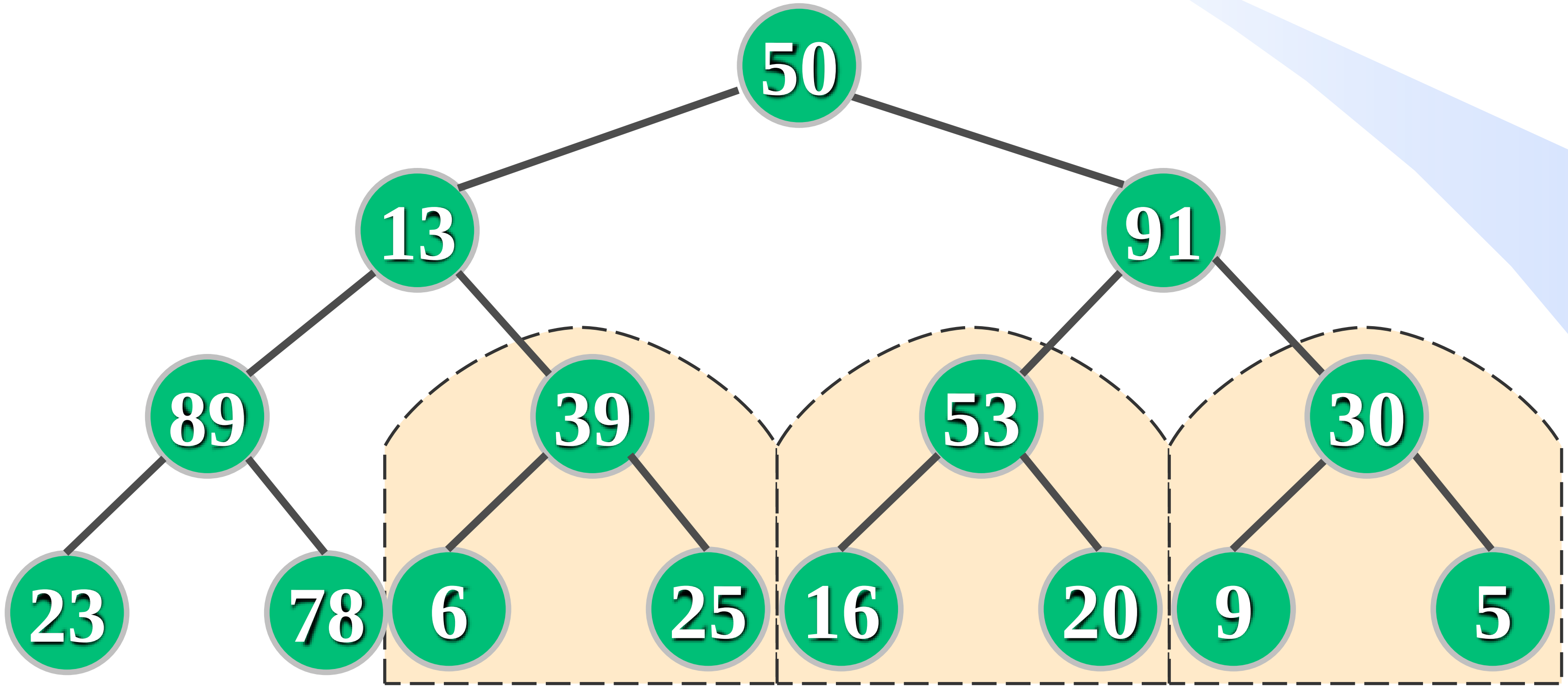
从最后一个非叶结点开始，依次建立以每个非叶结点为根的子堆。



初始建堆

A

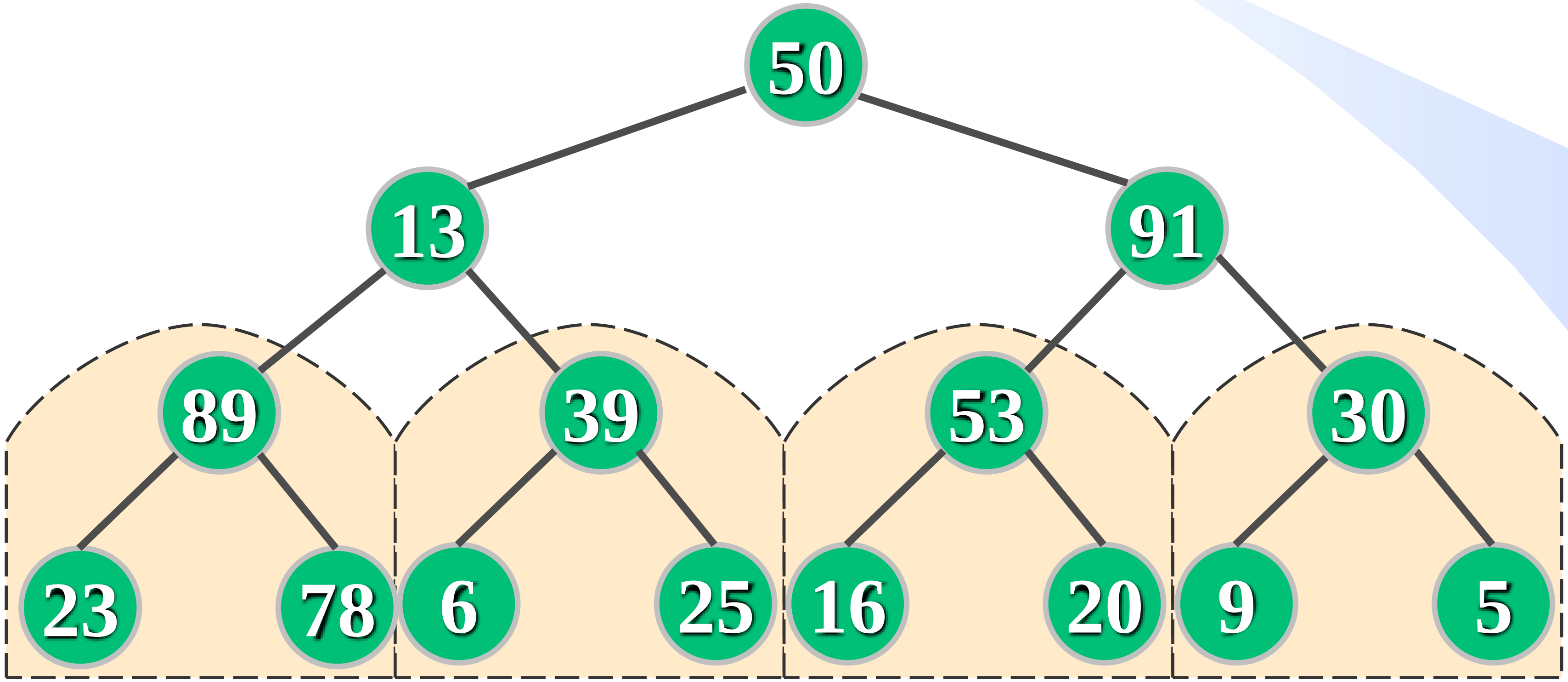
从最后一个非叶结点开始，依次建立以每个非叶结点为根的子堆。



初始建堆

A

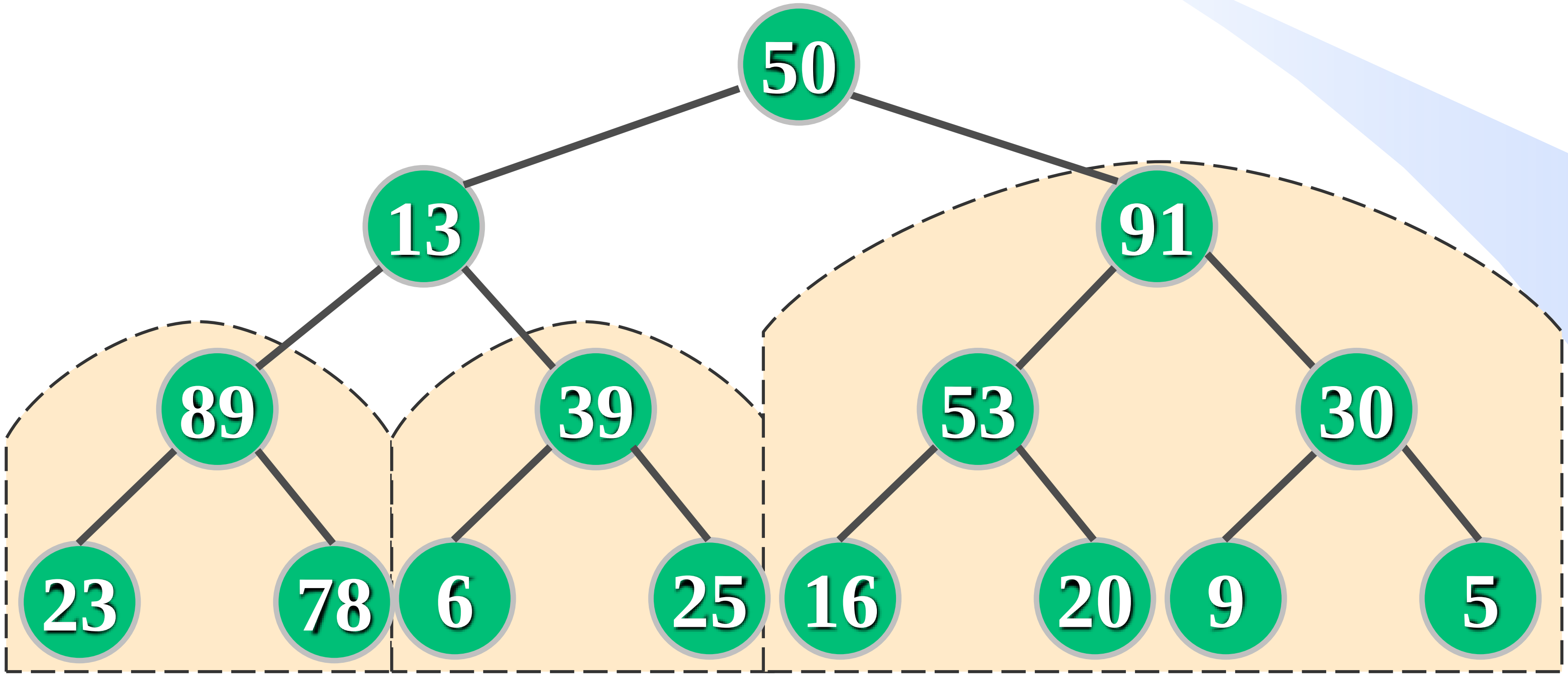
从最后一个非叶结点开始，依次建立以每个非叶结点为根的子堆。



初始建堆

A

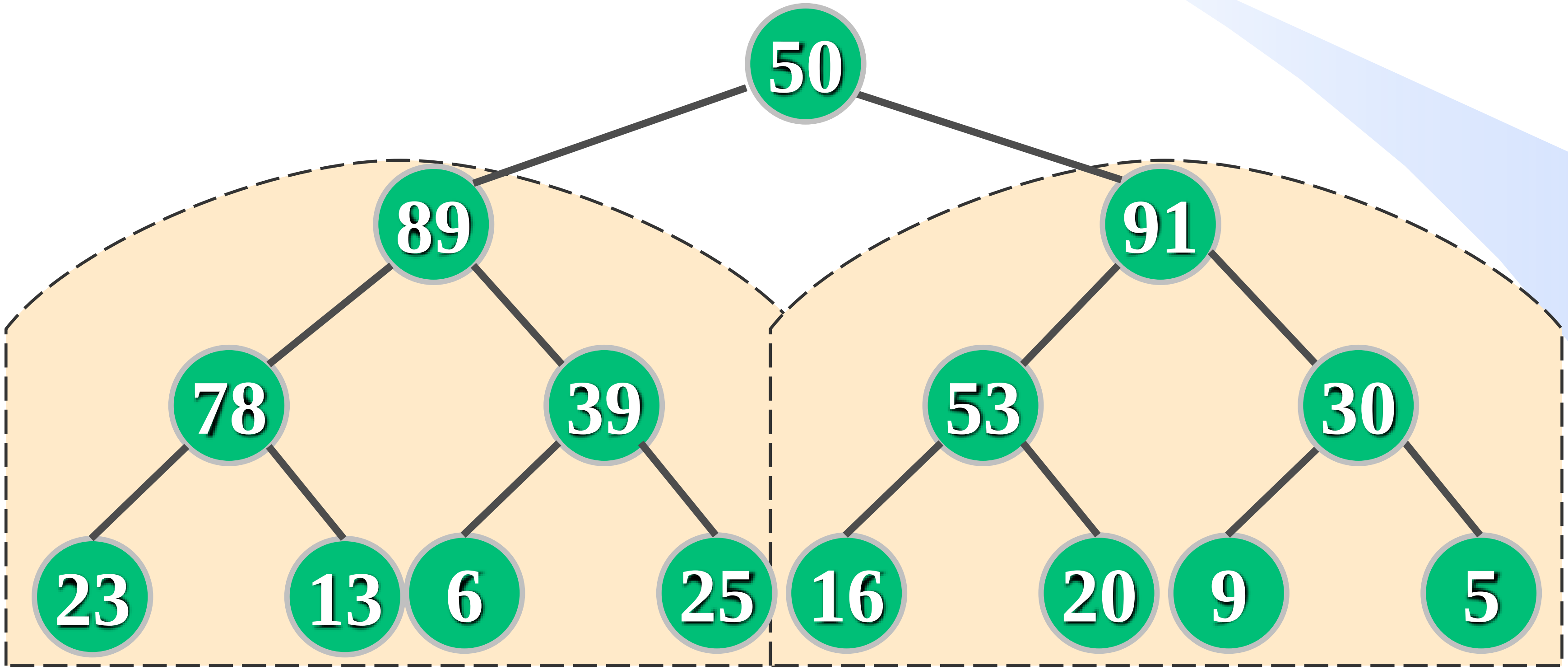
从最后一个非叶结点开始，依次建立以每个非叶结点为根的子堆。



初始建堆

A

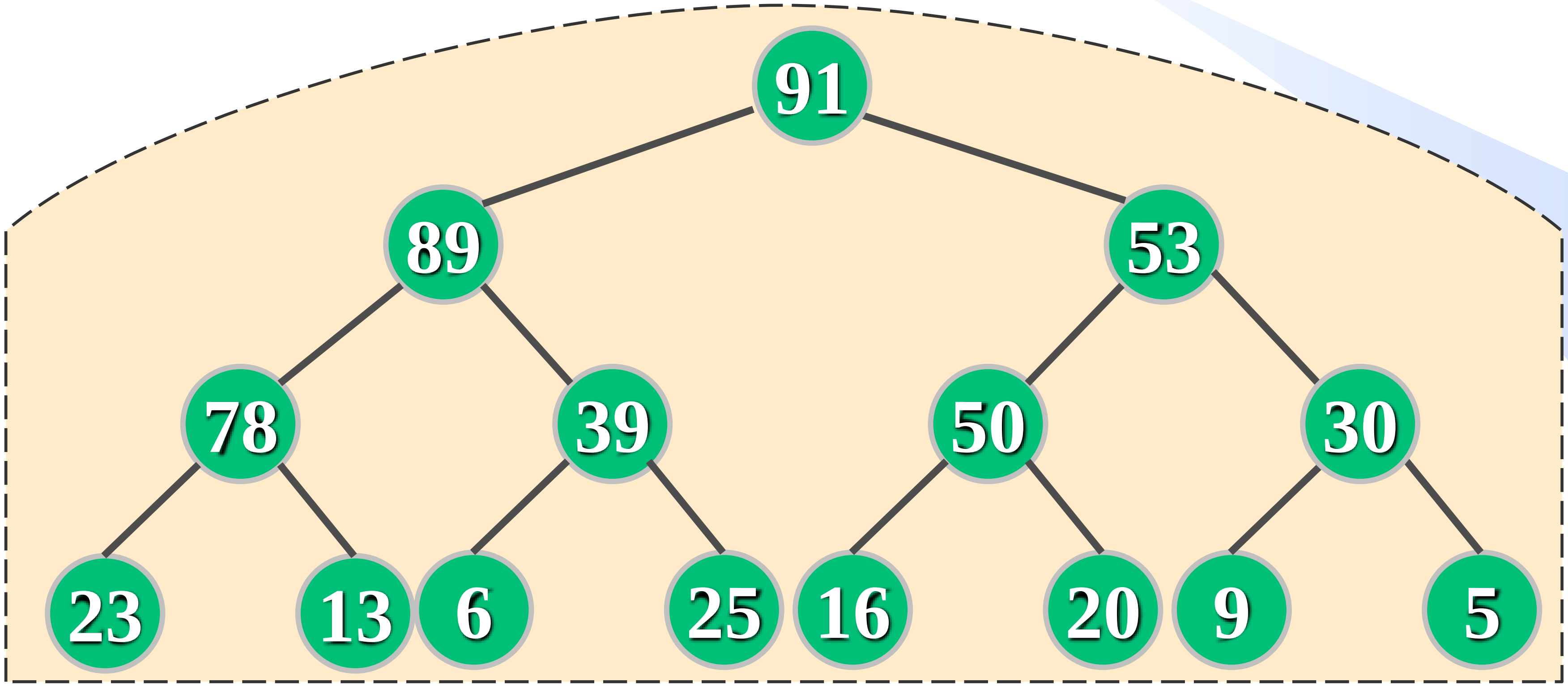
从最后一个非叶结点开始，依次建立以每个非叶结点为根的子堆。



初始建堆

A

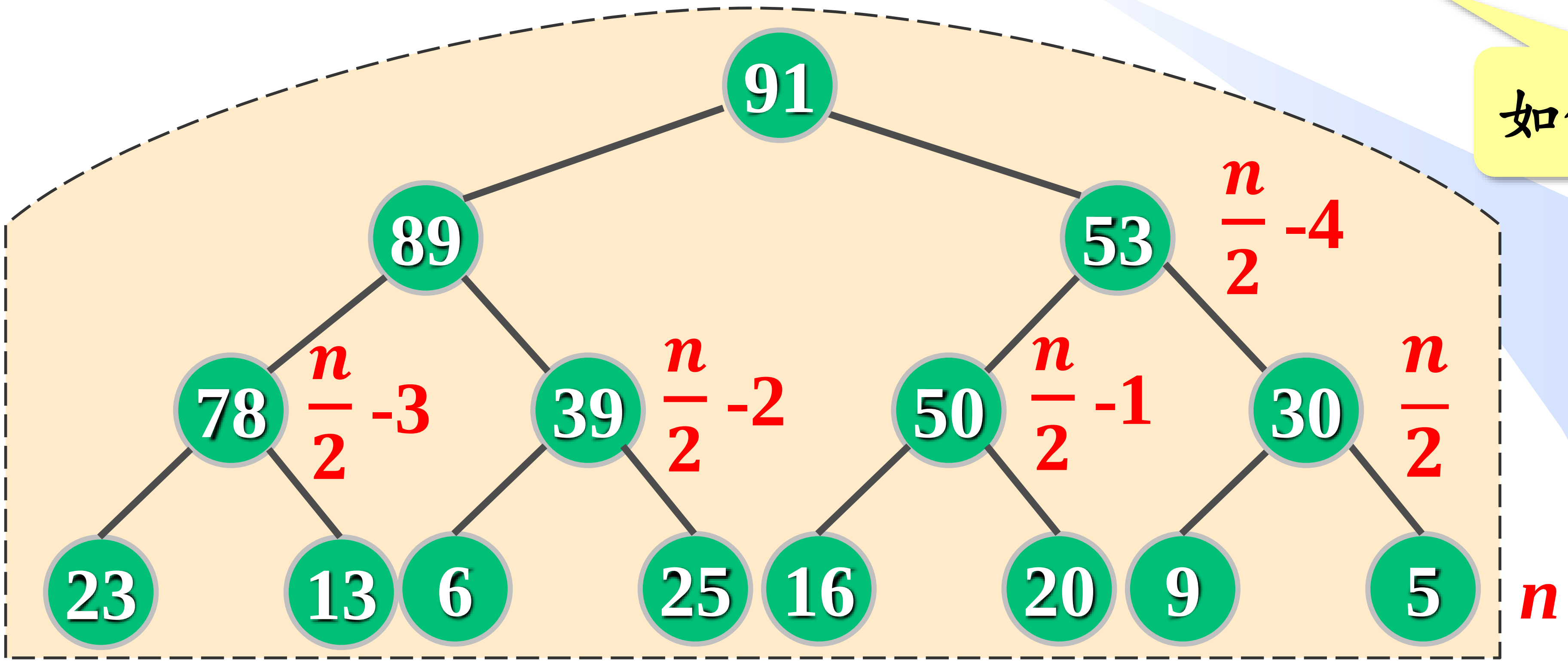
从最后一个非叶结点开始，依次建立以每个非叶结点为根的子堆。



初始建堆

A

从最后一个非叶结点开始，依次建立以每个非叶结点为根的子堆。



如何实现?

从最后一个非叶结点开始，依次下沉$\lfloor n/2\rfloor$, $\lfloor n/2\rfloor$-1, …, 1。

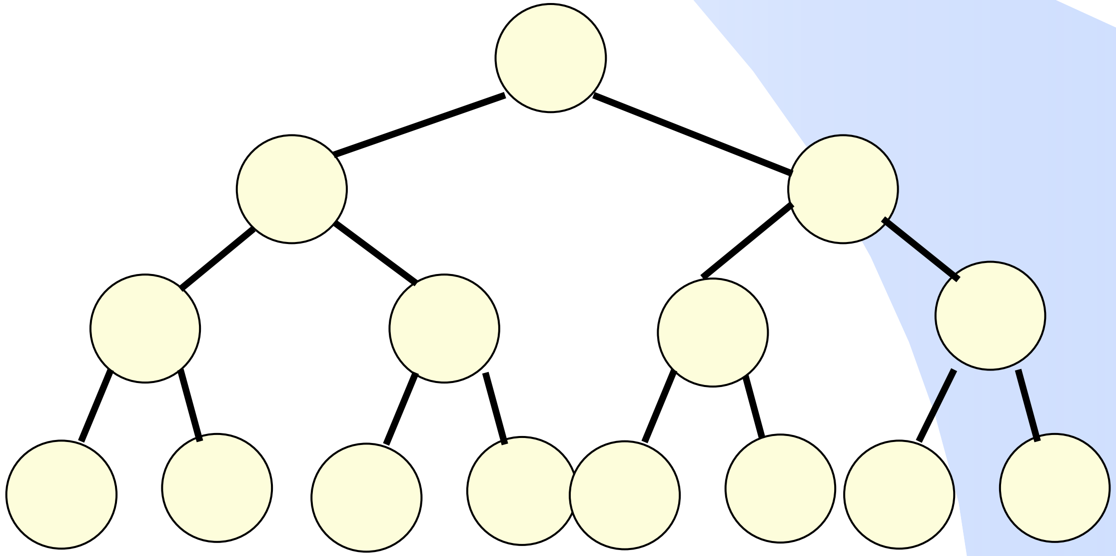
初始建堆算法

从最后一个非叶结点开始，依次下沉结点$\lfloor n/2 \rfloor$, $\lfloor n/2 \rfloor$-1, ..., 1。

```
void BuildHeap(int R[],int n){
   for(int i=n/2; i>=1; i--)
      ShiftDown(R,n,i); //建立以i为根的堆，即下沉i
}
```

Floyd建堆算法
自底向上建堆

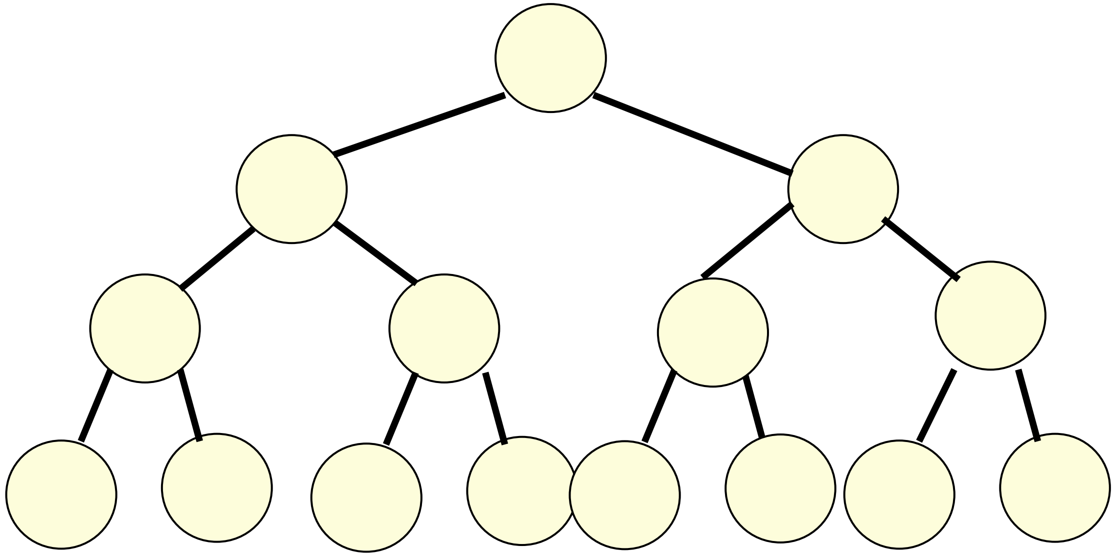
最坏时间复杂度
O(*n*log*n*)?



初始建堆算法的时间复杂度

B

- 下沉*R*[*i*]的关键词比较次数取决于*R*[*i*]的高度。
- 建堆算法是下沉了每个非叶结点，故总时间取决于各结点的高度之和。



初始建堆算法的时间复杂度

B

$$T=\frac{n}{4}+2\cdot\frac{n}{8}+3\cdot\frac{n}{16}+4\cdot\frac{n}{32}+\cdots$$

$$2T=\frac{n}{2}+2\cdot\frac{n}{4}+3\cdot\frac{n}{8}+4\cdot\frac{n}{16}+\cdots$$

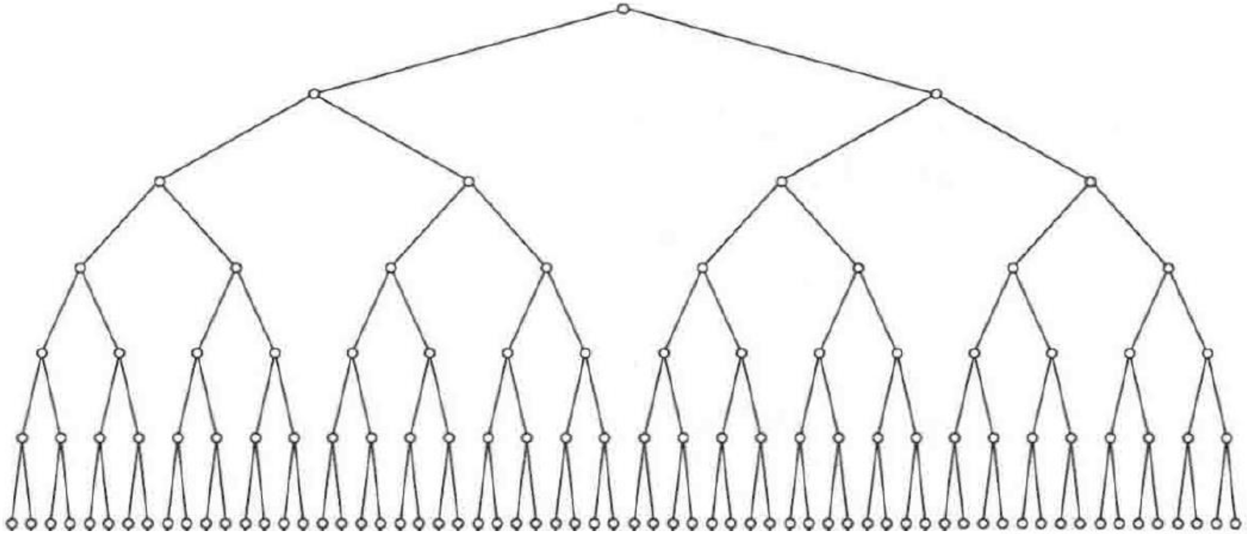
$$2T-T=\frac{n}{2}+\frac{n}{4}+\frac{n}{8}+\frac{n}{16}+\frac{n}{32}+\cdots$$

$$T=n(\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\cdots)$$

$$=n\left(\frac{\frac{1}{2}\left(1-\left(\frac{1}{2}\right)^k\right)}{1-\frac{1}{2}}\right)=n\left(1-\left(\frac{1}{2}\right)^k\right)<n$$

各结点的高度之和小于*n*

算法最坏情况时间复杂度O(*n*)



课下思考

B

若规定sum++为基本运算，以下选项中_______能最精确的反映下面算法的时间复杂度。【吉林大学21级期末考试题】

```
void f(int n){
    int t=0, sum=0, i, j;
    for(i=n; i>1; i/=2) {
        t++;
        for(j=0; j<t*i; j++)
            sum++;
    }
}
```

A. $O(n^3)$　　B. $O(n\log n)$　　C. $O(n^2)$　　D. $O(n)$

课下思考

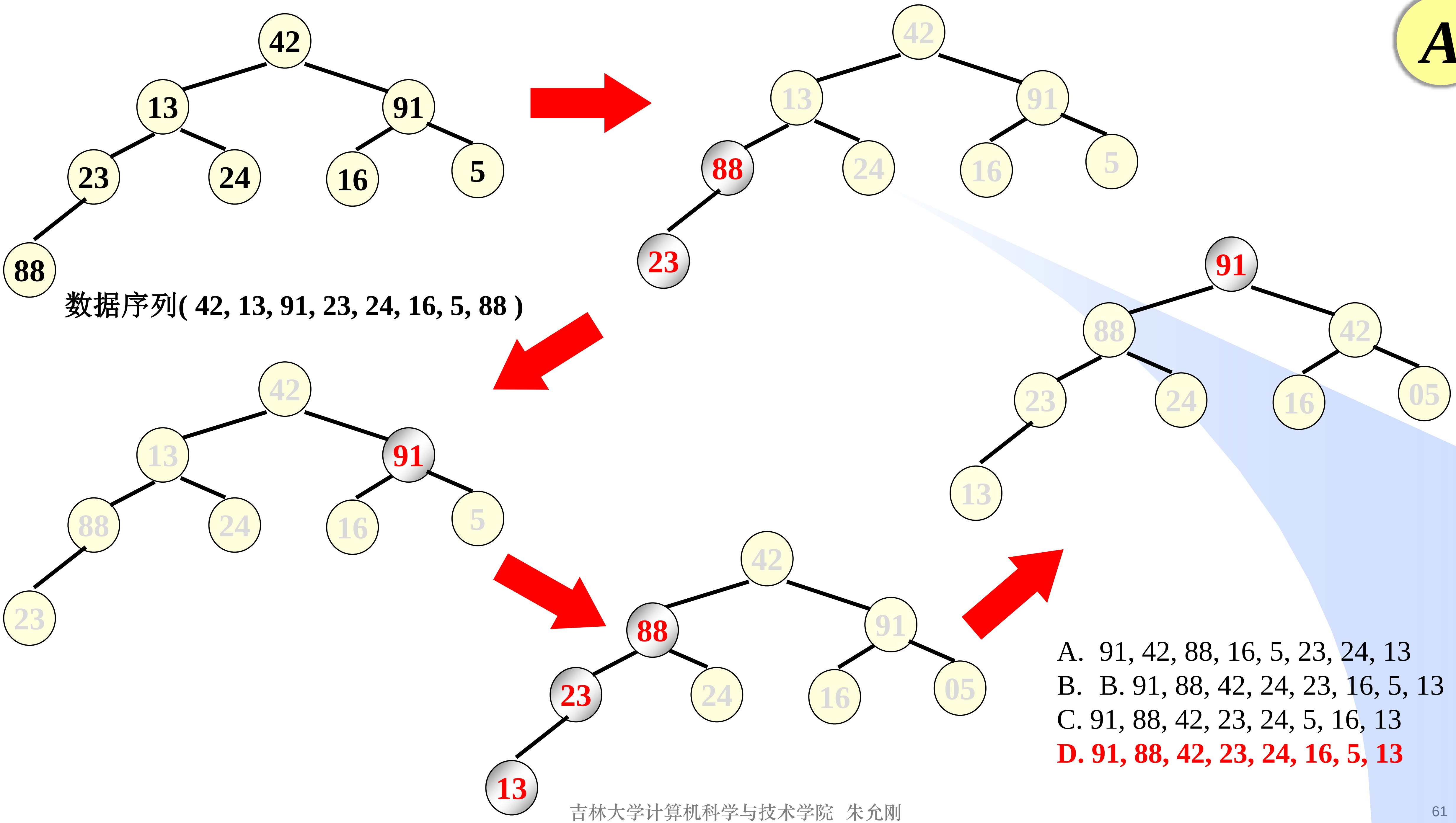
A

判断题：将n个元素建成一个堆，至少需要O($n\log n$)时间。【清华大学考研题】

使用初始建堆算法，将数据序列(42, 13, 91, 23, 24, 16, 5, 88)建成的大根堆为_______。【吉林大学21级期末考试题】

A. 91, 42, 88, 16, 5, 23, 24, 13　　B. 91, 88, 42, 24, 23, 16, 5, 13

C. 91, 88, 42, 23, 24, 5, 16, 13　　D. 91, 88, 42, 23, 24, 16, 5, 13



数据序列(42, 13, 91, 23, 24, 16, 5, 88)

A. 91, 42, 88, 16, 5, 23, 24, 13
B. B. 91, 88, 42, 24, 23, 16, 5, 13
C. 91, 88, 42, 23, 24, 5, 16, 13
D. 91, 88, 42, 23, 24, 16, 5, 13

课下思考

A

使用初始建堆算法，将数据序列(6, 4, 3, 5, 8, 9, 12, 10)建成的大根堆为______。【2022级吉林大学期末考试题】

A. 12, 10, 6, 4, 8, 9, 3, 5

B. 12, 10, 9, 5, 8, 6, 3, 4

C. 12, 9, 10, 6, 8, 3, 4, 5

D. 6, 4, 8, 3, 5, 9, 12, 10

课下思考

A

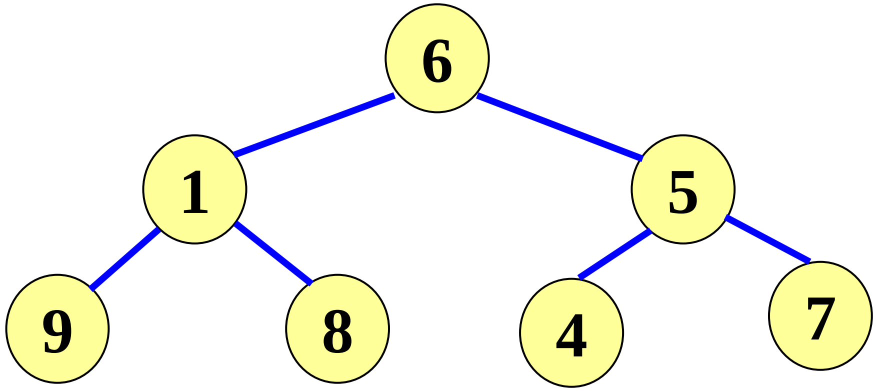
将数据序列 (6, 1, 5, 9, 8, 4, 7)建成大根堆，序列变化过程为______。【2018年考研题全国卷】

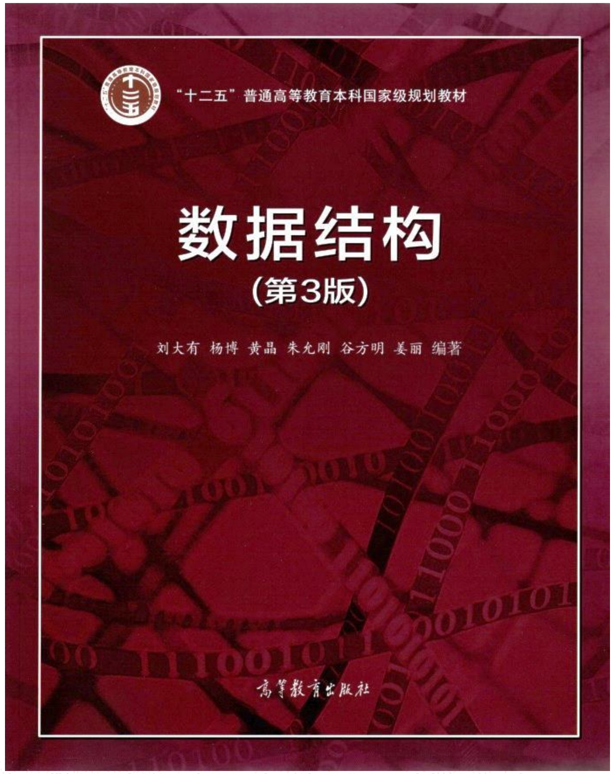
A. 6 1 7 9 8 4 5 → 6 9 7 1 8 4 5 → 9 6 7 1 8 4 5 → 9 8 7 1 6 4 5

B. 6 9 5 1 8 4 7 → 6 9 7 1 8 4 5 → 9 6 7 1 8 4 5 → 9 8 7 1 6 4 5

C. 6 9 5 1 8 4 7 → 9 6 5 1 8 4 7 → 9 6 7 1 8 4 5 → 9 8 7 1 6 4 5

D. 6 1 7 9 8 4 5 → 7 1 6 9 8 4 5 → 7 9 6 1 8 4 5 → 9 7 6 1 8 4 5





堆排序

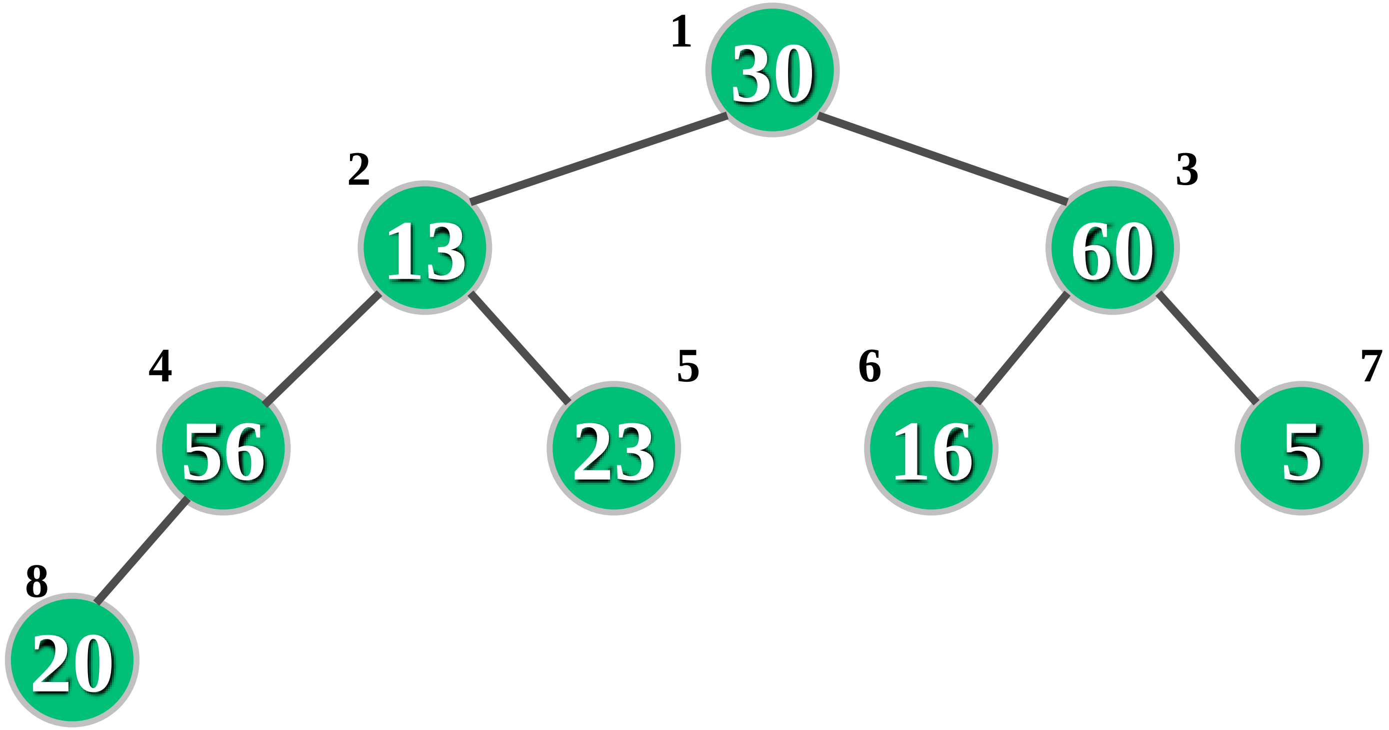
- 锦标赛排序
- 堆的概念及基本操作
- **堆排序算法**

数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

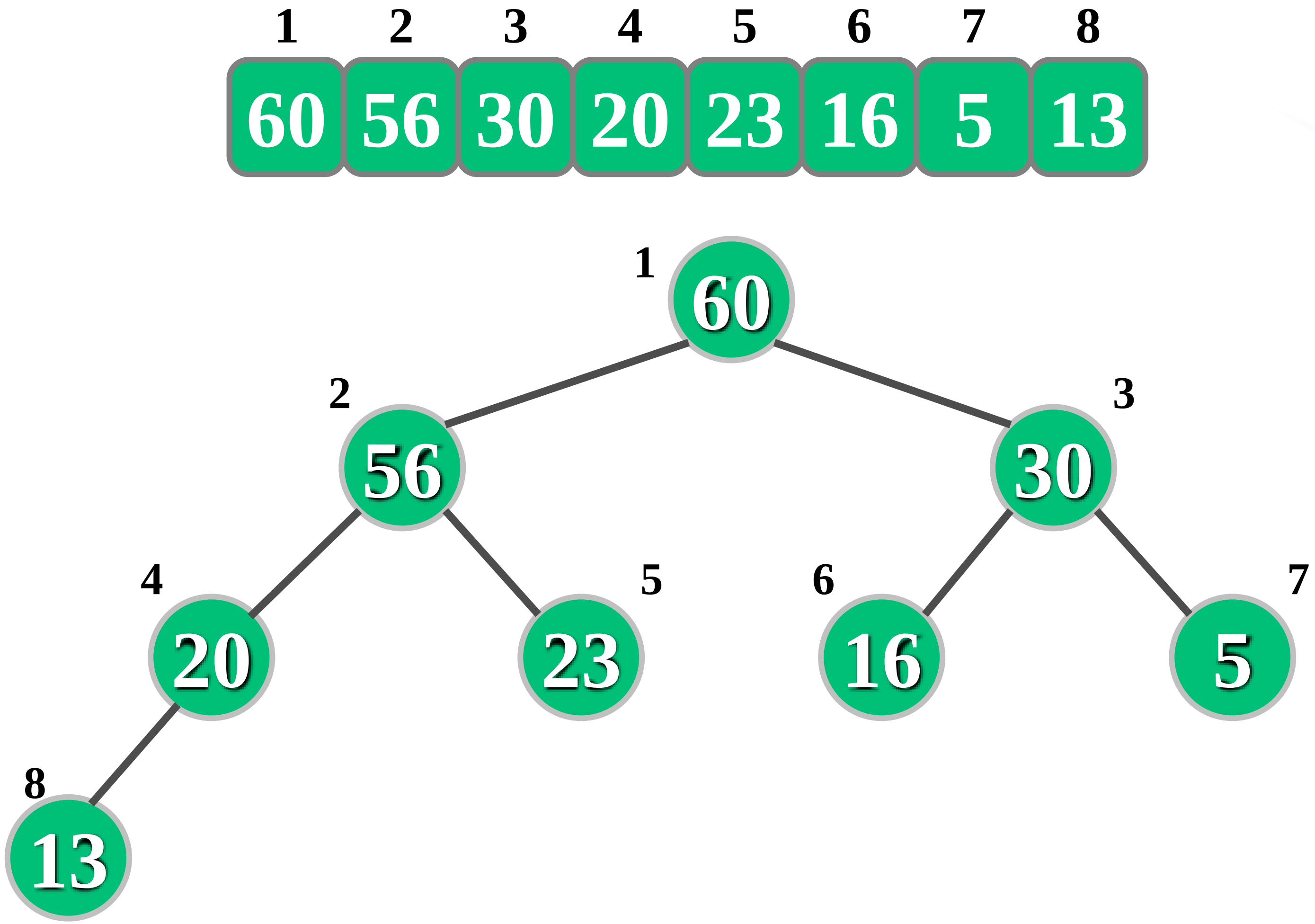
堆排序算法的思想

1	2	3	4	5	6	7	8
30	13	60	56	23	16	5	20



调用BuildHeap将数组R建为堆

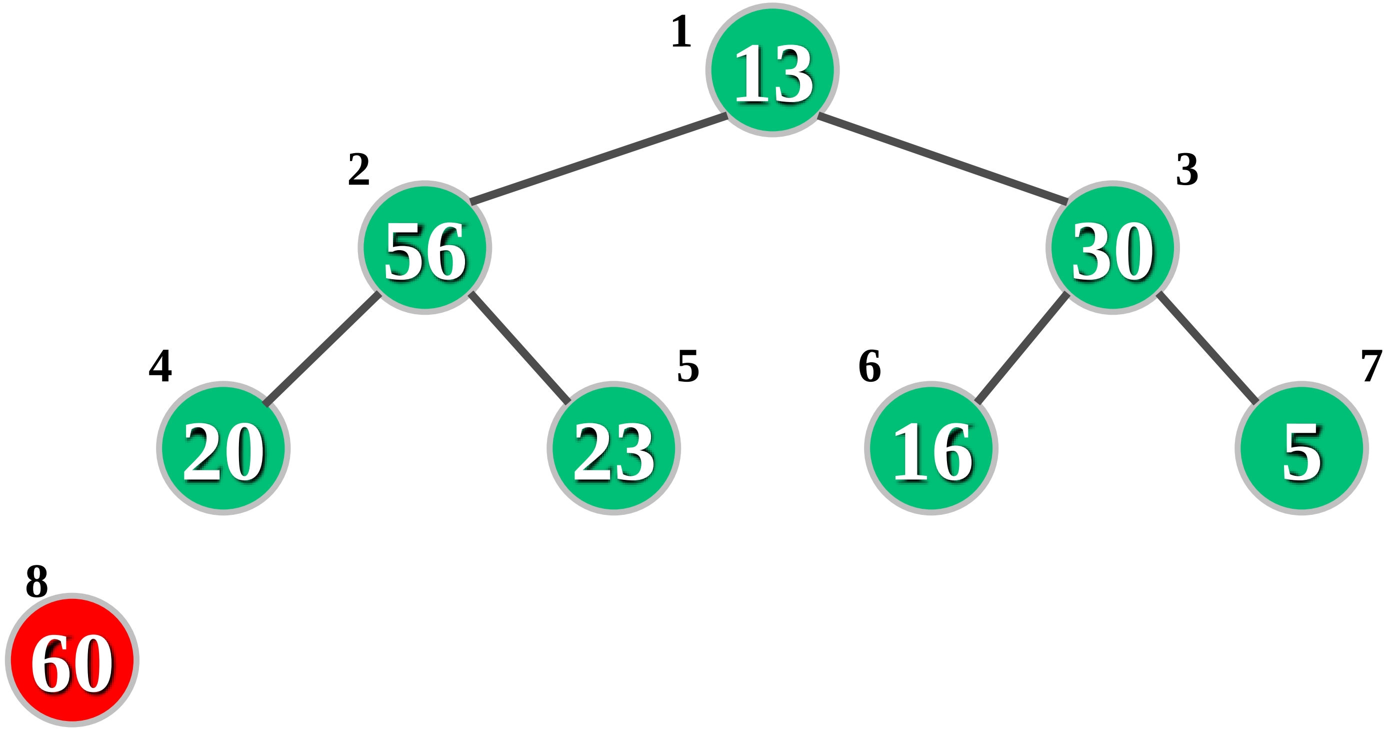
堆排序算法的思想



在前8个元素里找最大元素（即堆顶R[1]），与*R*[8]交换，使*R*[8]就位

堆排序算法的思想

1	2	3	4	5	6	7	8
13	56	30	20	23	16	5	60

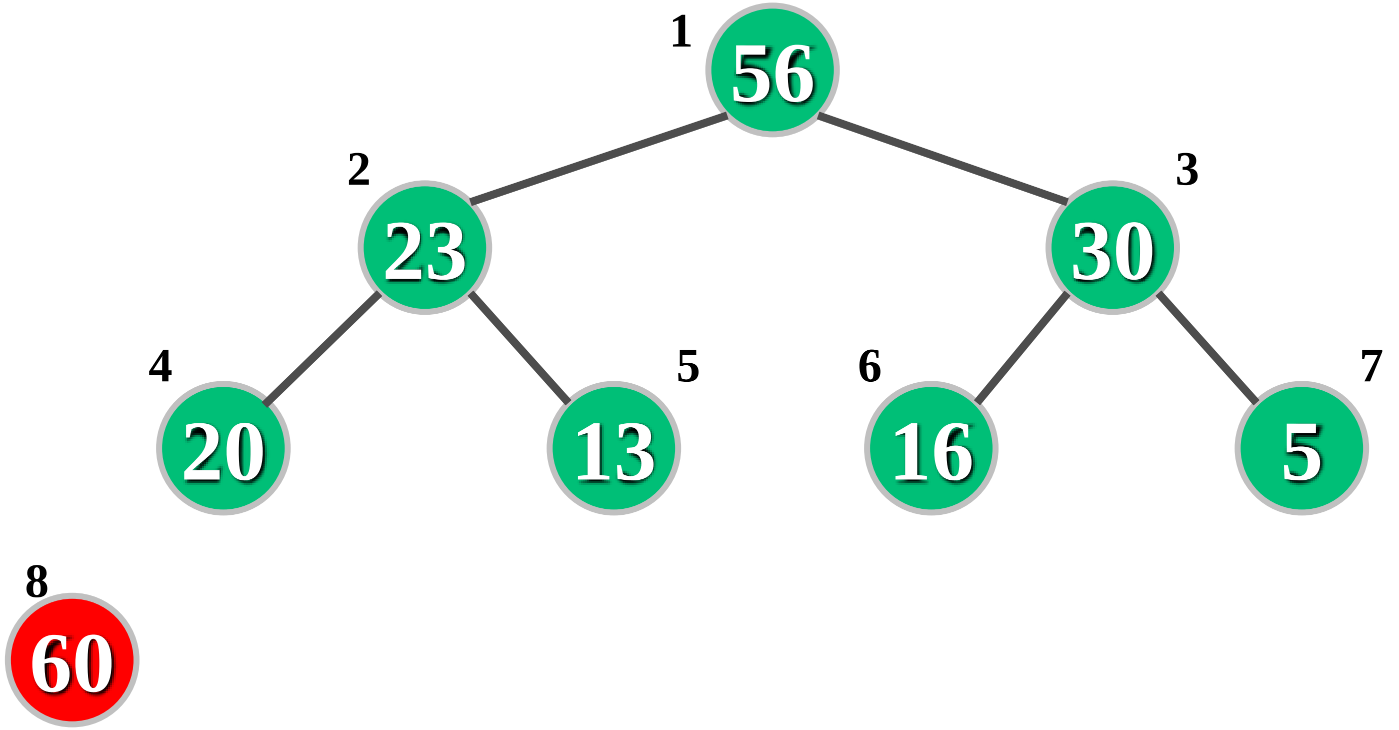


在前8个元素里找最大元素（即堆顶R[1]），与*R*[8]交换，使*R*[8]就位

堆排序算法的思想

A

1	2	3	4	5	6	7	8
56	23	30	20	13	16	5	60

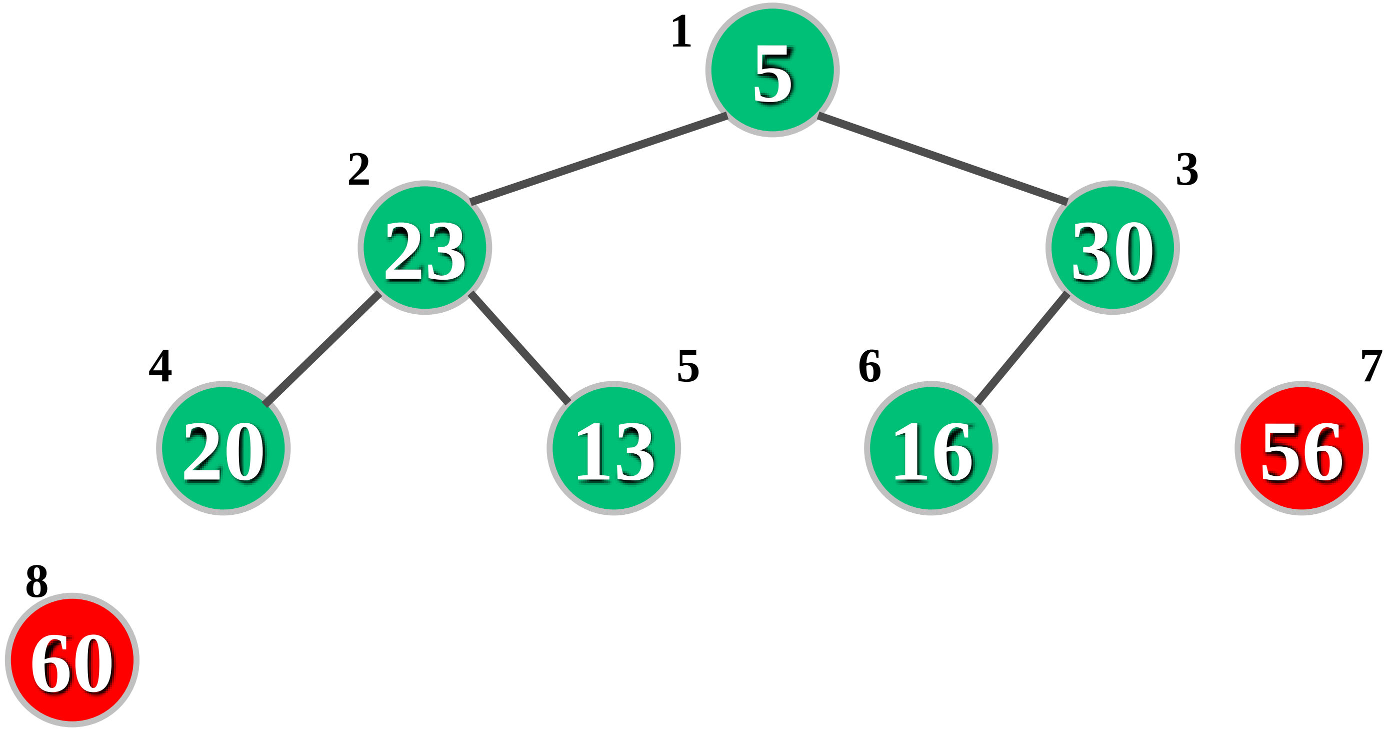


下沉根结点（堆顶R[1]）使前*R*[1]...*R*[7]重建为堆

A

堆排序算法的思想

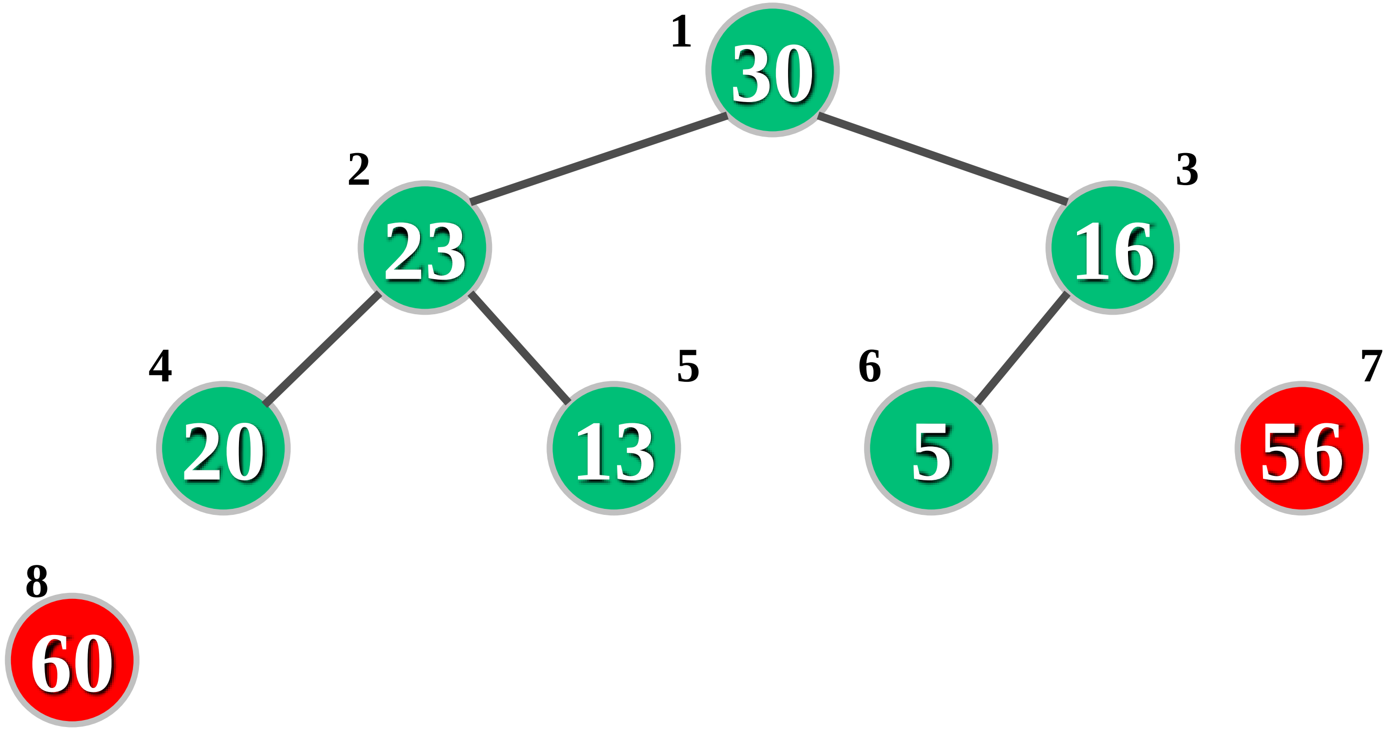
1	2	3	4	5	6	7	8
5	23	30	20	13	16	56	60



在前7个元素里找最大元素（即堆顶R[1]），与*R*[7]交换，使*R*[7]就位

堆排序算法的思想

1	2	3	4	5	6	7	8
30	23	16	20	13	5	56	60

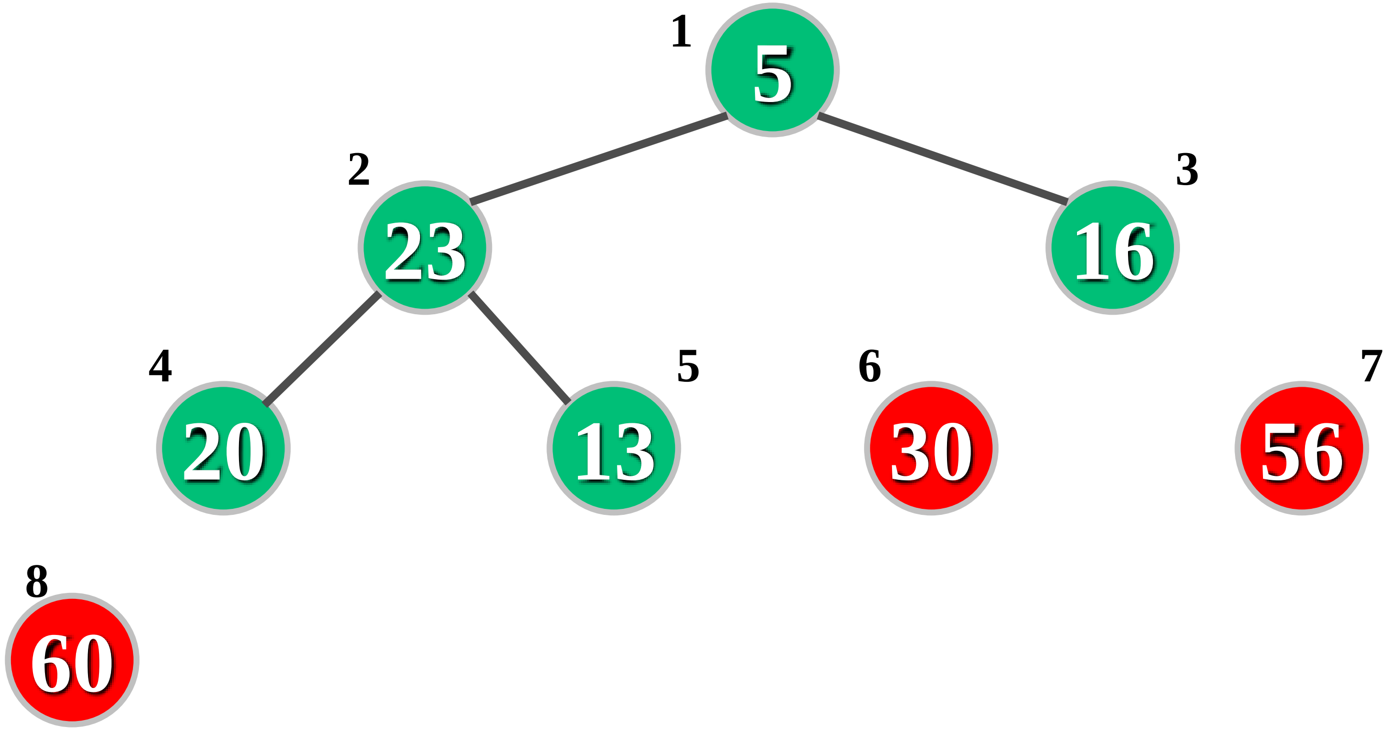


下沉根结点（堆顶 *R*[1]）使前*R*[1]…*R*[6]重建为堆

A

堆排序算法的思想

1	2	3	4	5	6	7	8
5	23	16	20	13	30	56	60



在前6个元素里找最大元素（即堆顶*R*[1]），与*R*[6]交换，使*R*[6]就位

堆排序算法的思想

A

① 将待排序数组R建成一个大根堆。

② 在前n个元素的堆里选最大元素$R[1]$，与$R[n]$交换使$R[n]$就位，下沉根结点$R[1]$使$R[1]...R[n-1]$重建为堆。

③ 在前n-1个元素的堆里选最大元素$R[1]$，与$R[n-1]$交换使$R[n-1]$就位，下沉根结点$R[1]$使$R[1]...R[n-2]$重建为堆。

④ 在前n-2个元素的堆里选最大元素$R[1]$，与$R[n-2]$交换使$R[n-2]$就位，下沉根结点$R[1]$使$R[1]...R[n-3]$重建为堆。

⑤ ....

上述操作反复进行，直到调整范围只剩下一个元素$R[1]$为止。此时，$R[1]$是n个元素中最小的，且数组R已按递增排列。

堆排序算法的粗略描述

A

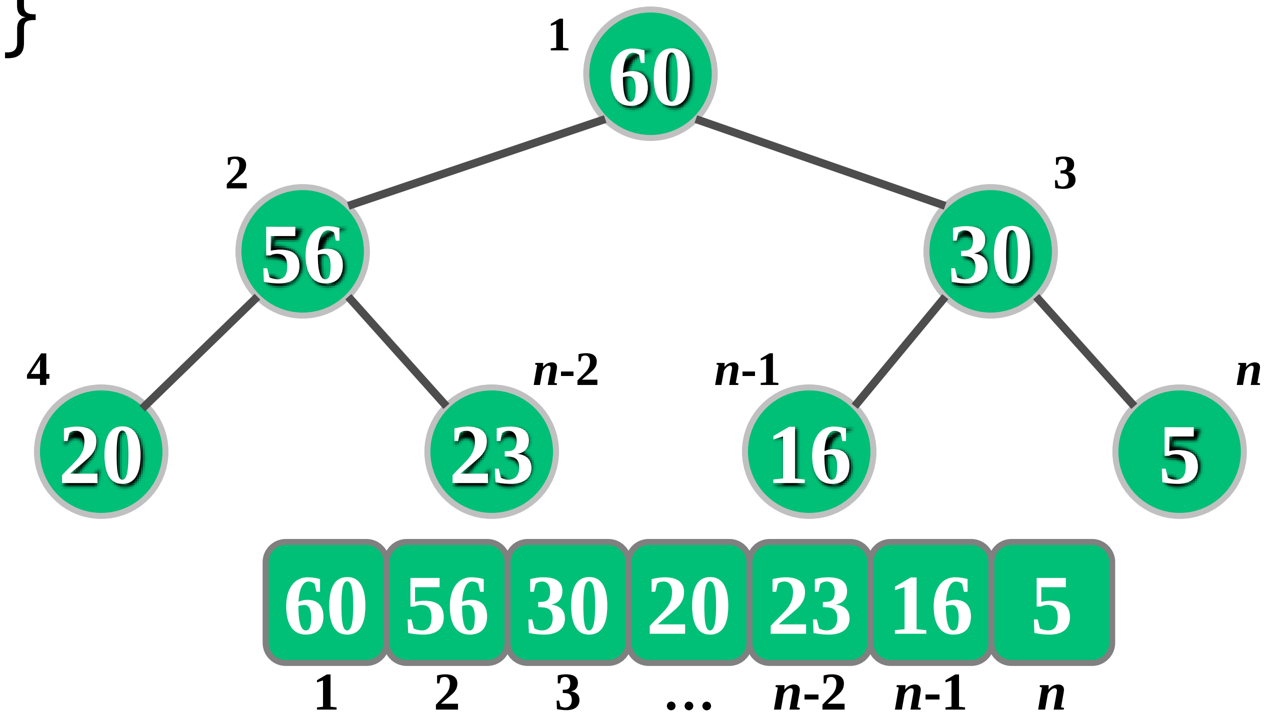
① 建立包含$R[1], R[2], \ldots, R[n]$的堆；

② `for(int i=n; i>1; i--){` //i标识当前处理的堆尾

$R[1] \leftrightarrow R[i]$; //根$R[1]$和堆尾交换

下沉$R[1]$使$R[1]\ldots R[i\text{-}1]$重建为堆

}



虽然操作的是数组，但背后隐藏的灵魂是二叉树

堆排序算法

```
void HeapSort(int R[],int n){  //堆排序R[1]...R[n]
    BuildHeap(R, n);           //将R建为堆
    for(int i=n; i>1; i--){    //i为当前堆的堆尾
        swap(R[1], R[i]);      //前i个元素的最大者R[1]与R[i]交换
        ShiftDown(R,i-1,1);    //下沉R[1]使R[1]...R[i-1]重建为堆
    }
}
```

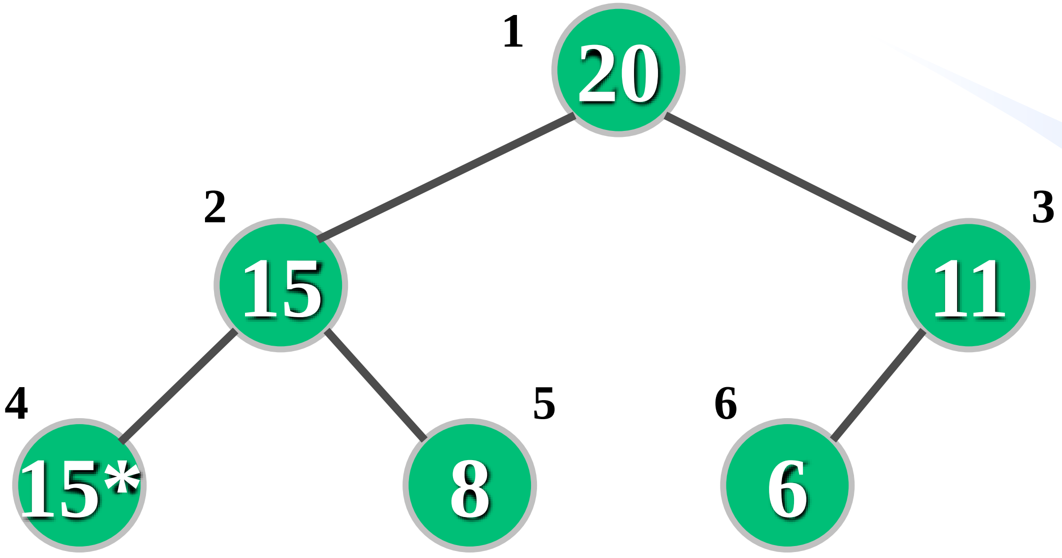
时间复杂度
O(*n*log*n*)

```
void swap(int &a, int &b){
    int temp = a;
    a = b;
    b = temp;
}
```

1	2	3	…	…	…	*n*-2	*n*-1	*n*

堆排序算法示例

A

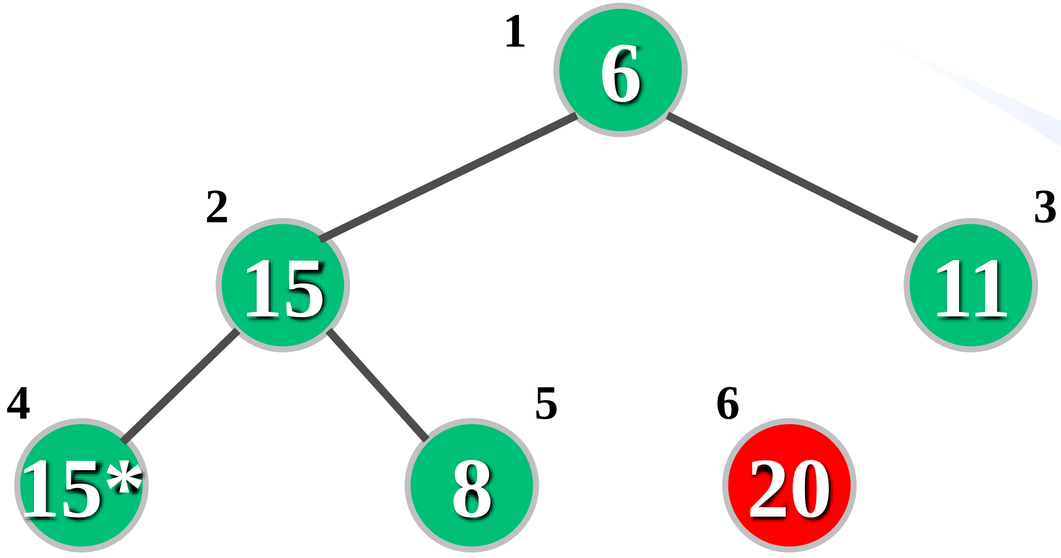


初始堆

20	15	11	15*	8	6
1	2	3	4	5	6

堆排序算法示例

A

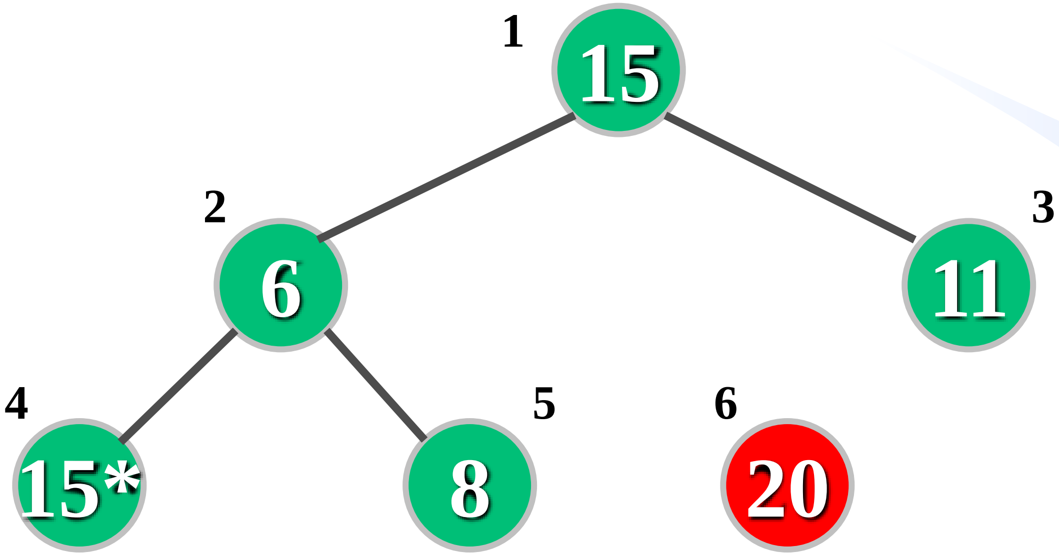


6	15	11	15*	8	20
1	2	3	4	5	6

交换R[1]与R[6],
使R[6]就位

A

堆排序算法示例

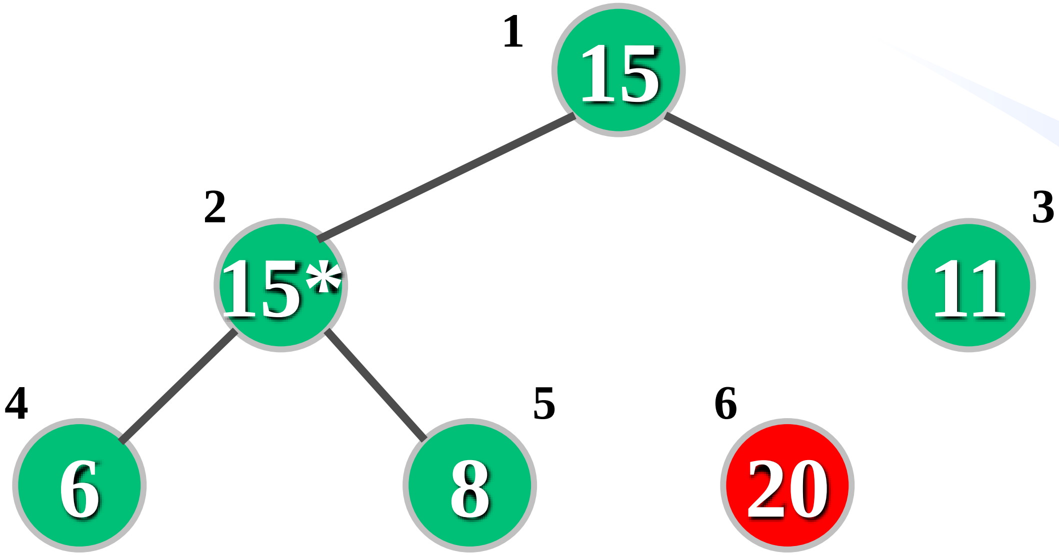


下沉R[1],使R[1]…R[5]重建为堆

15	6	11	15*	8	20
1	2	3	4	5	6

堆排序算法示例

A

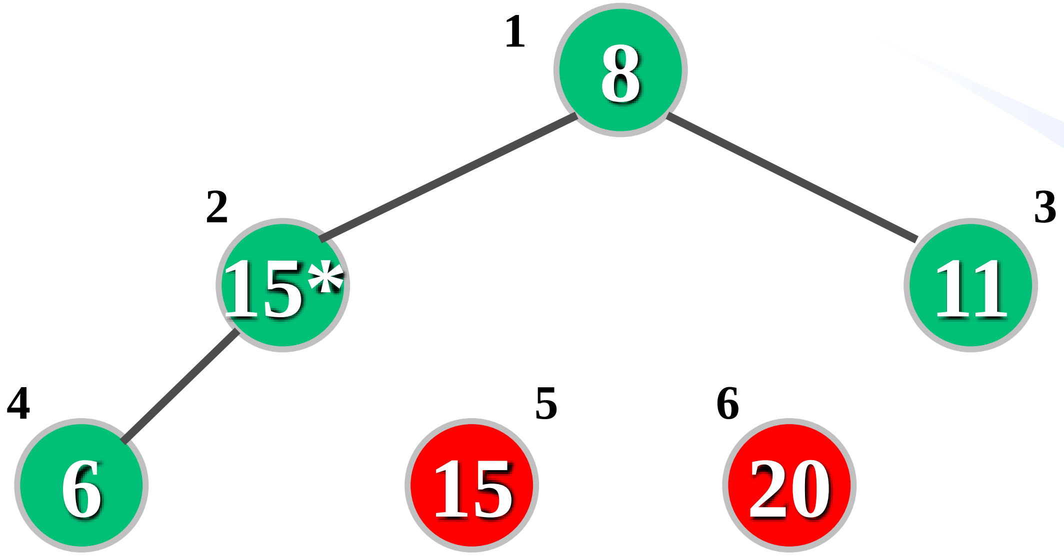


下沉R[1],使R[1]
...R[5]重建为堆

15	15*	11	6	8	20
1	2	3	4	5	6

A

堆排序算法示例

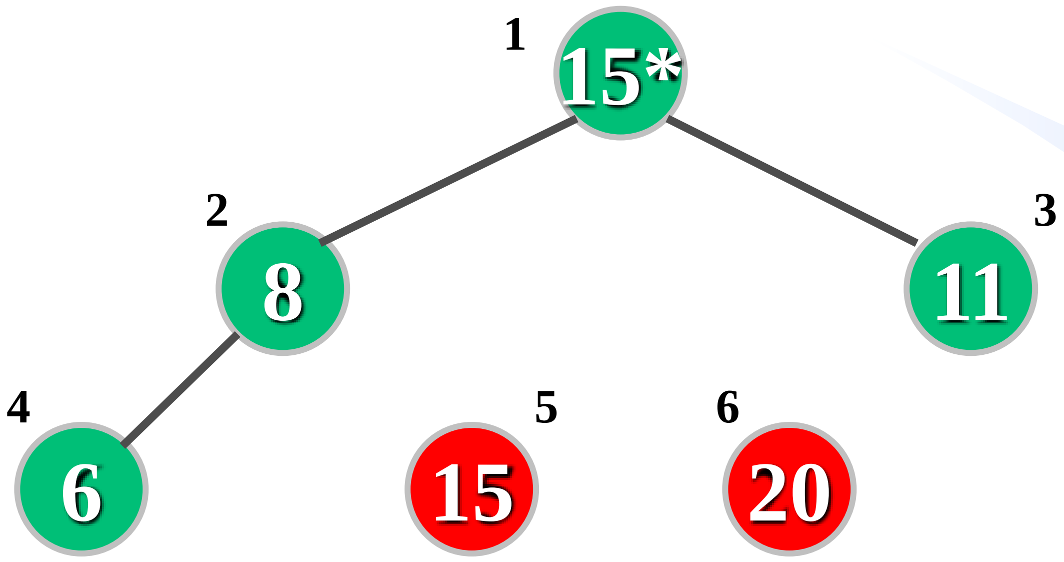


8	15*	11	6	15	20
1	2	3	4	5	6

交换R[1]与R[5]，
使R[5]就位

A

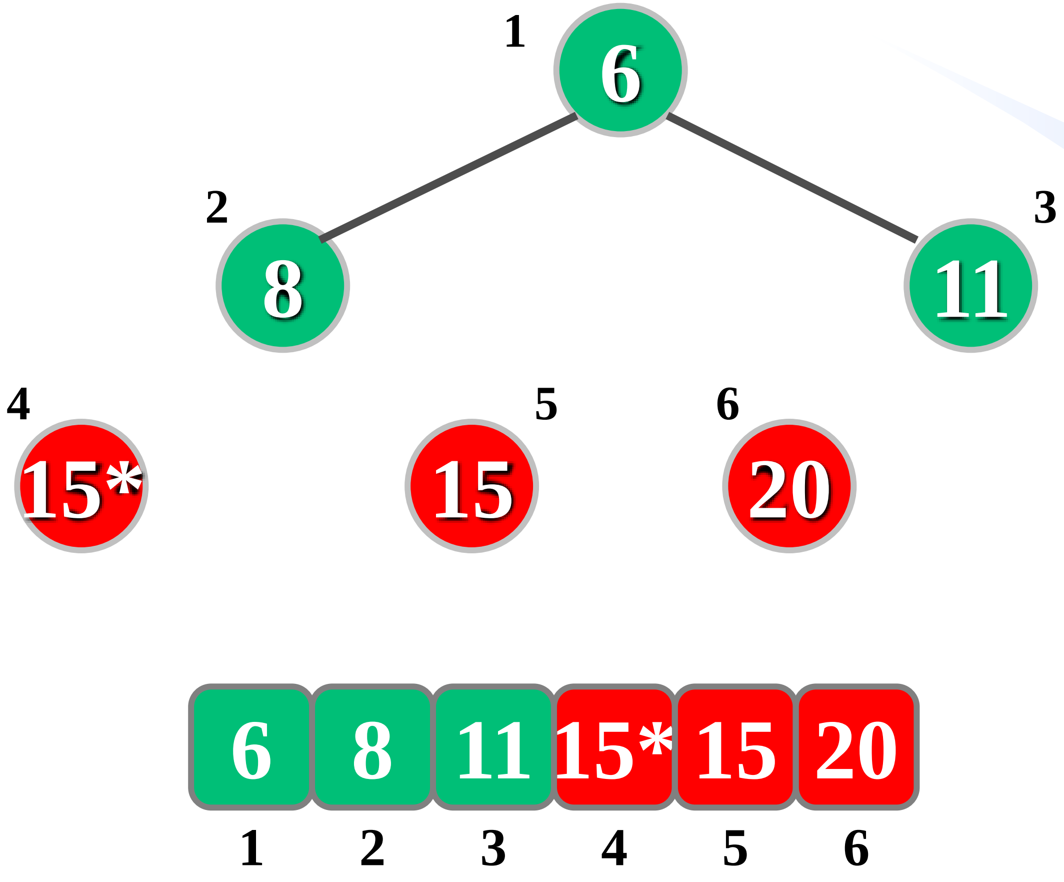
堆排序算法示例



15*	8	11	6	15	20
1	2	3	4	5	6

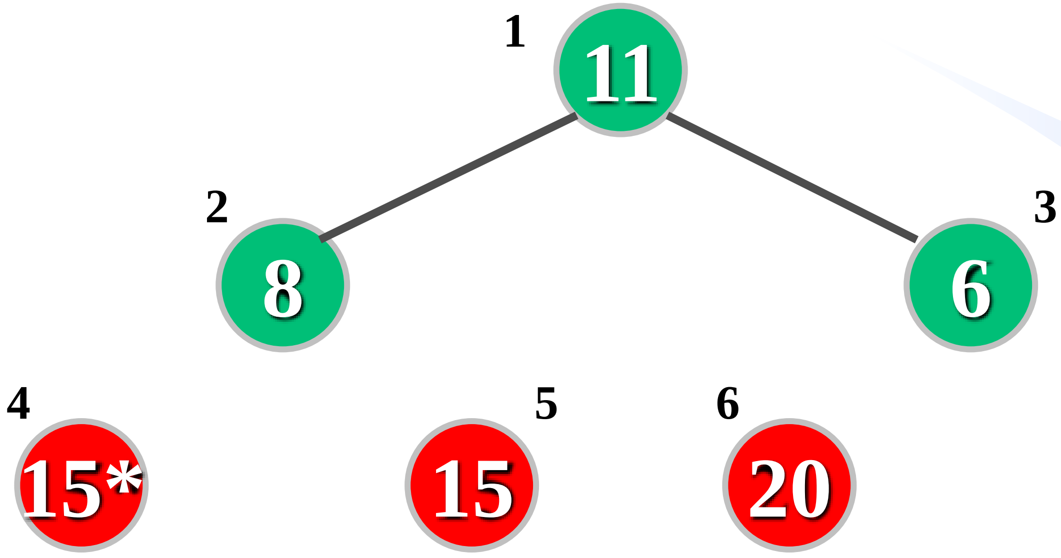
下沉R[1],使R[1]…R[4]重建为堆

堆排序算法示例



交换R[1]与R[4],
使R[4]就位

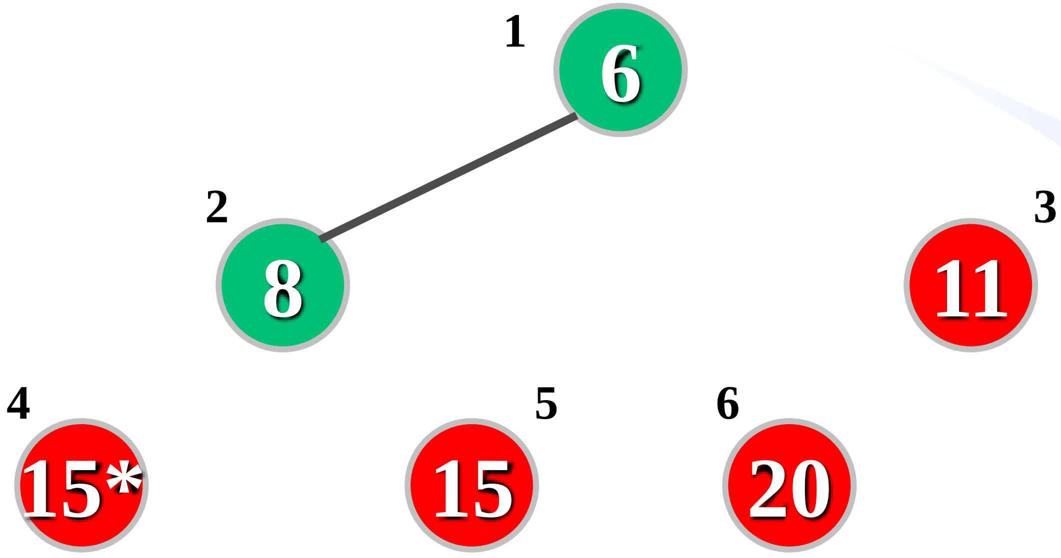
堆排序算法示例



11	8	6	15*	15	20
1	2	3	4	5	6

下沉R[1],使R[1]...R[3]重建为堆

堆排序算法示例

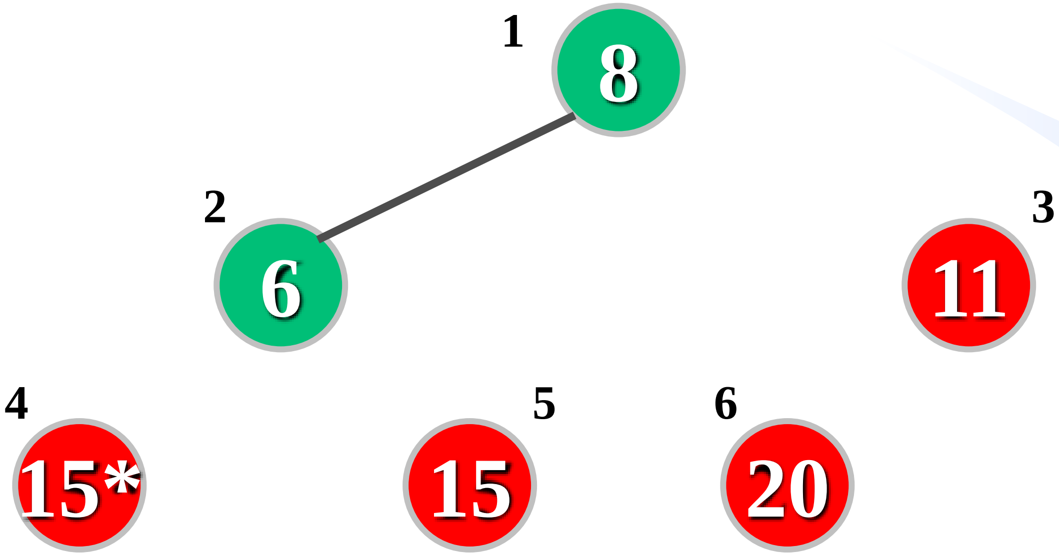


交换R[1]与R[3],使R[3]就位

6	8	11	15*	15	20
1	2	3	4	5	6

堆排序算法示例

A

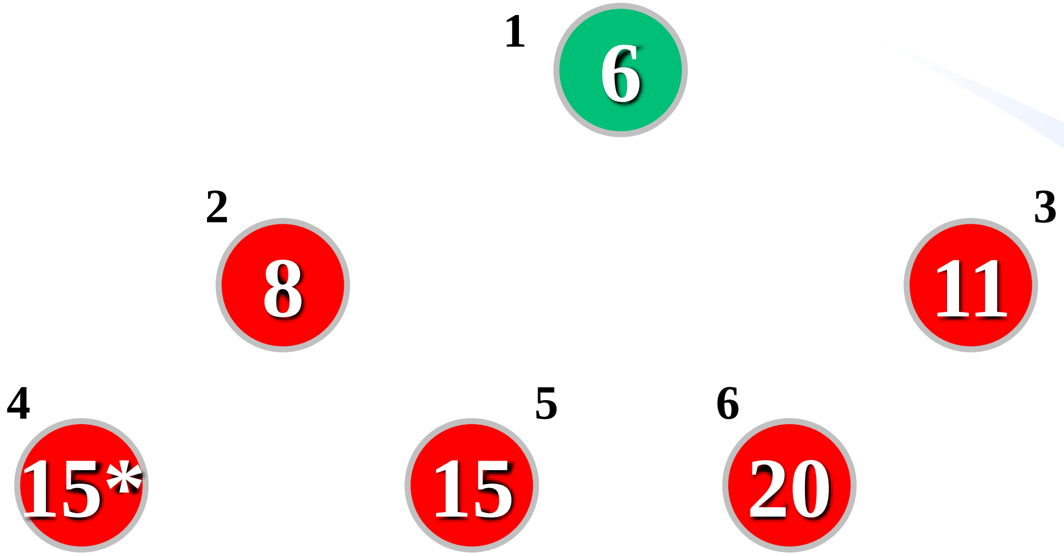


下沉R[1],使R[1]
…R[2]重建为堆

8	6	11	15*	15	20
1	2	3	4	5	6

A

堆排序算法示例



6	8	11	15*	15	20
1	2	3	4	5	6

交换R[1]与R[2],
使R[2]就位

不稳定

堆排序算法总结

排序算法	时间复杂度			空间复杂度	稳定性
	最好	平均	最坏		
堆排序	O(*n*log*n*)	O(*n*log*n*)	O(*n*log*n*)	O(1)	不稳定

对*n*个互异的随机关键词进行堆排序，关键词平均比较次数为2*n*log*n*-O(*n*loglog*n*)。

R. Schaffer, R. Sedgewick. The Analysis of Heapsort. *Journal of Algorithms*. 1993,15: 76-100.