堆

吉林大学计算机学院
谷方明
fmgu2002@sina.com

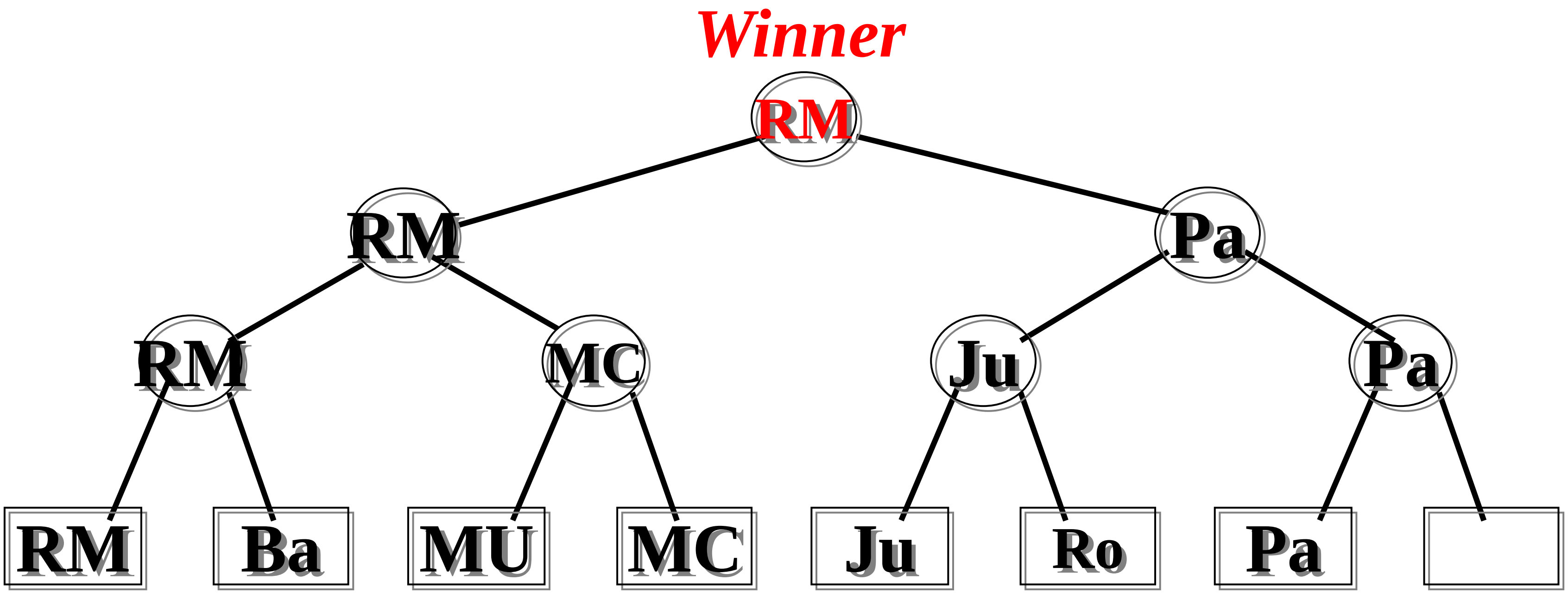
学习目标

- 掌握堆的定义和特性
- 掌握堆的基本操作操作、实现及效率分析
- 掌握堆排序
- 了解优先级队列

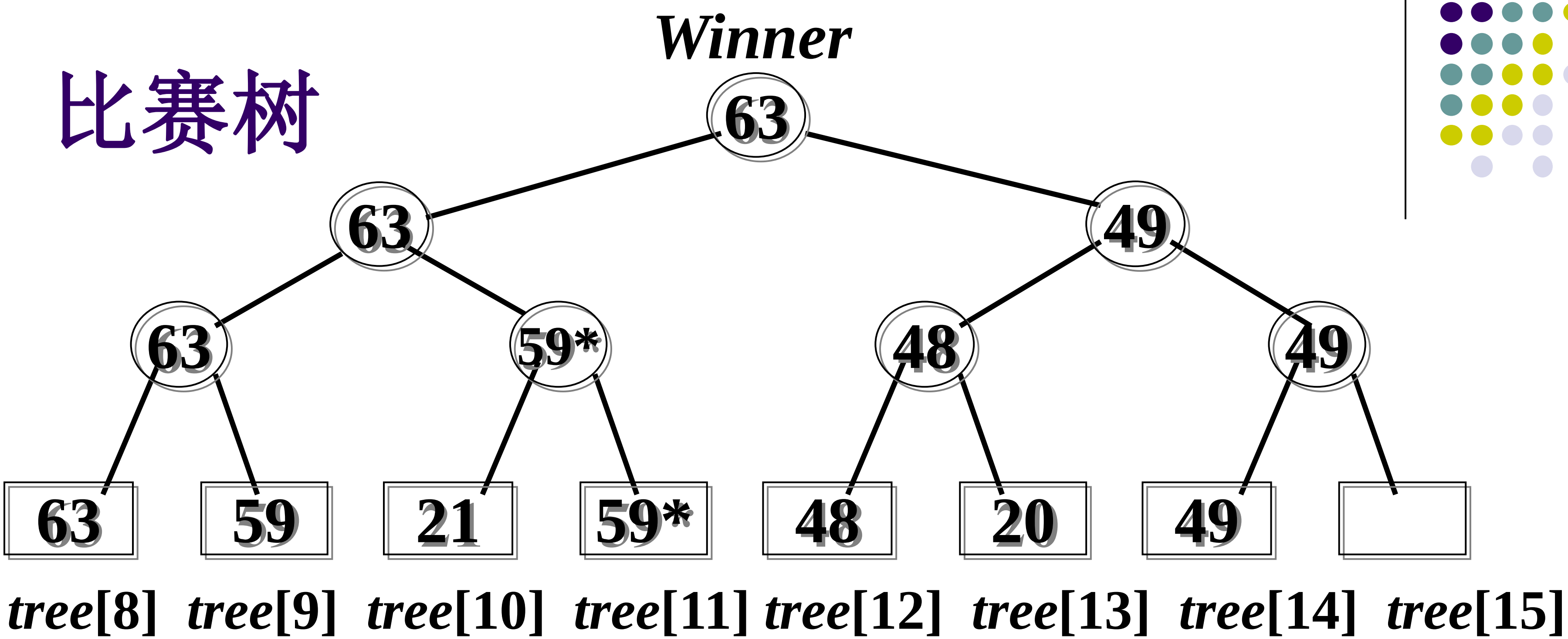
求最大（小）值

- 问题：在n个给定的整数中找最大值
- 直接查找： O(n)
- 多次查找，效率低
  - 选择排序：第1大，第2大，…

淘汰赛的启发

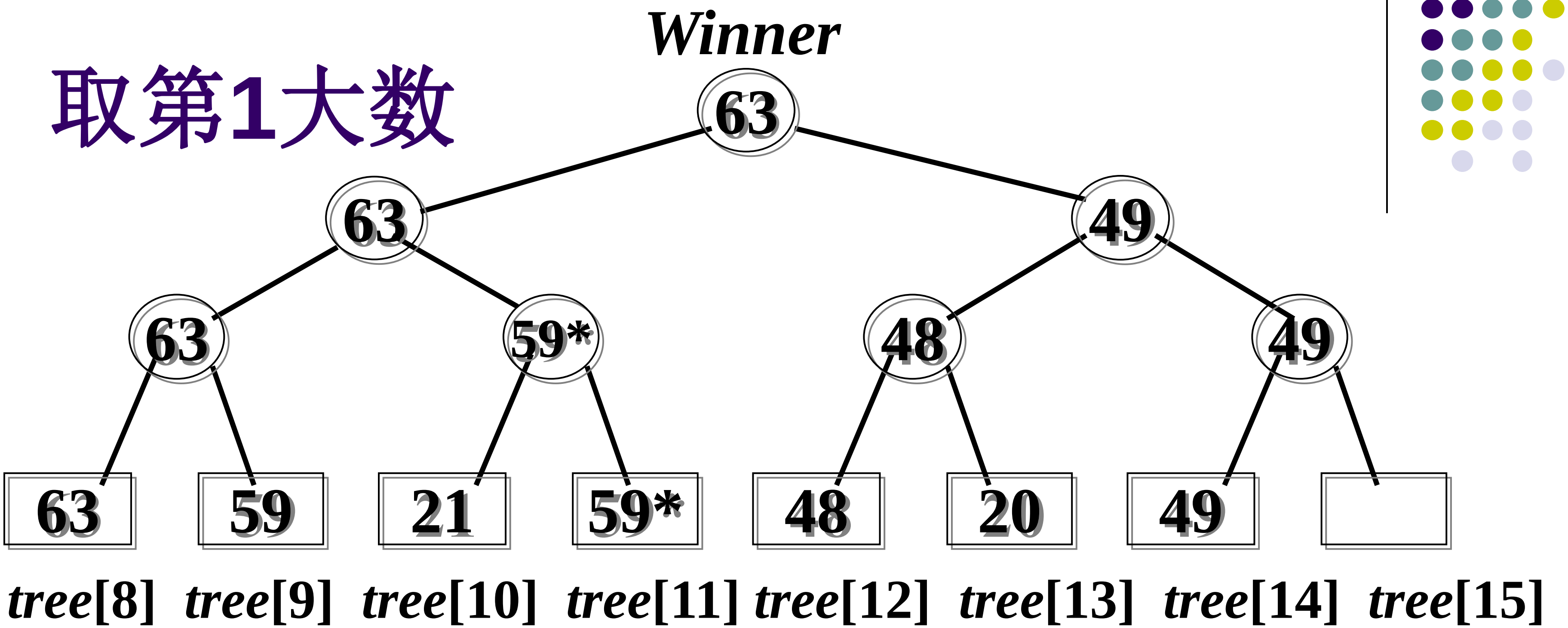


比赛树



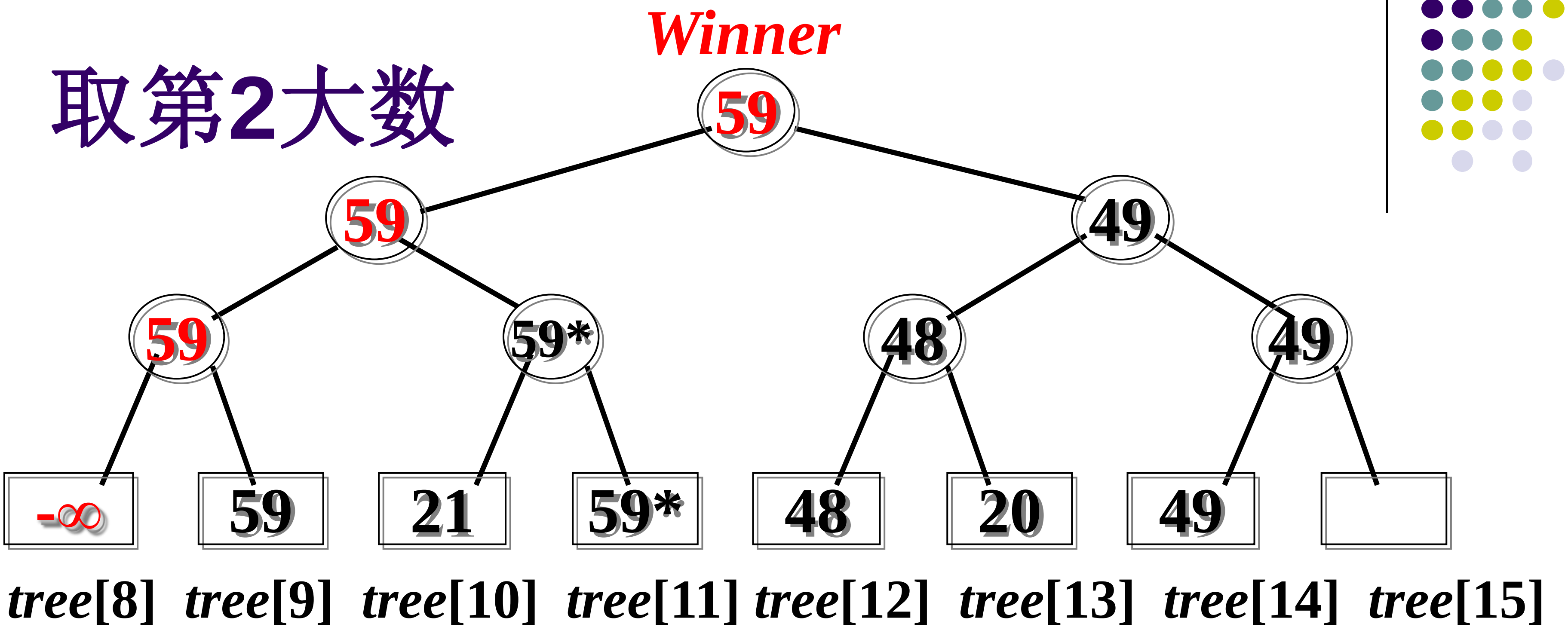
- 满二叉树；
- 叶结点存放的关键词；
- 分支结点存放关键词两两比较的结果
- n 非 2 的 幂时，叶结点补足到满足2^k，$2^{k-1} < n \leq 2^k$

取第1大数



- 形成初始比赛树（最大关键词上升到根）
- 得到最大值**tree[1]**

取第2大数



- 将剩余6个整数，调整为新的比赛树
- 得到第2大记录tree[1]

比赛树相关概念和问题

- 相关概念
  - 比赛树、竞赛树、选择树
  - 内结点保存比赛的赢家：赢者树(winner tree)；
  - 内结点保存比赛的输家：输者树(loser tree)；

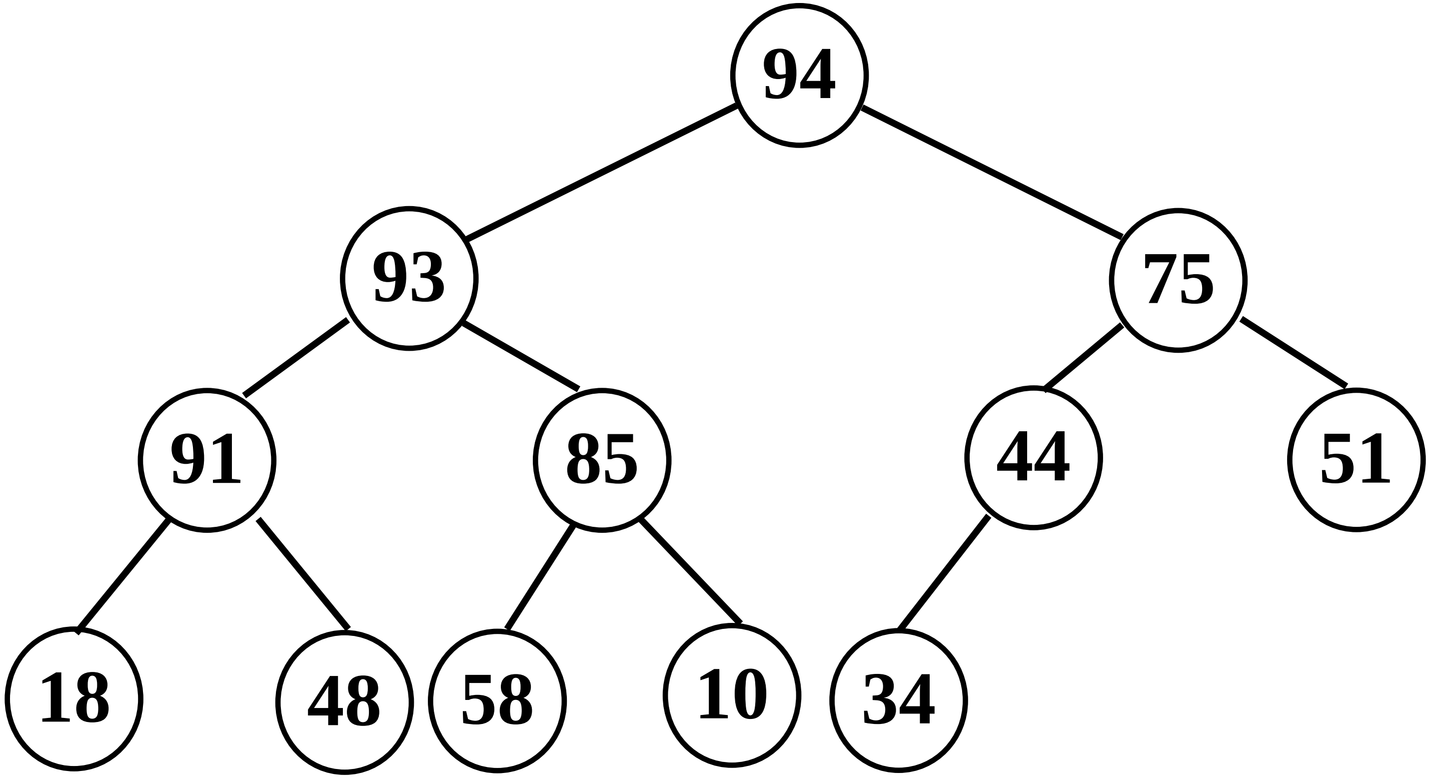
- 问题：
  - 重复保存
  - 满二叉树的限制强

堆（Heap）

- 在一棵完全二叉树中，如果任意结点的关键词大于等于它的两个孩子结点的关键词，那么被称为极大堆。简称堆。

- 极大(大根)堆
- 极小(小根)堆

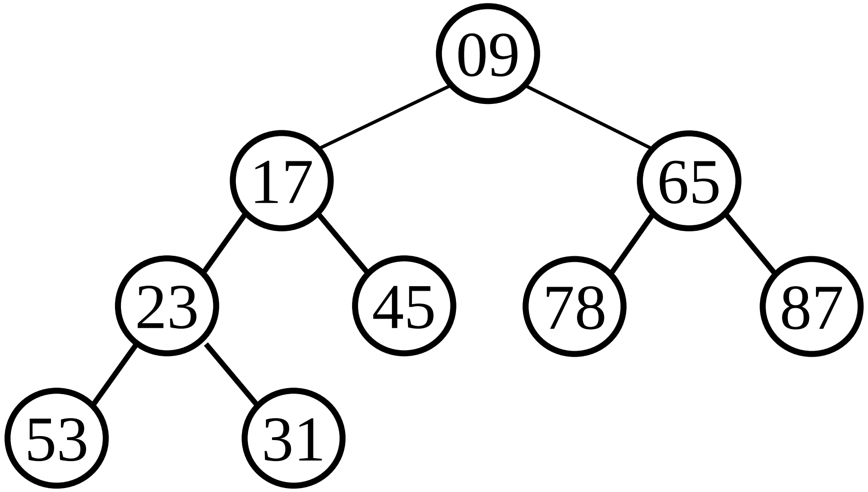
例

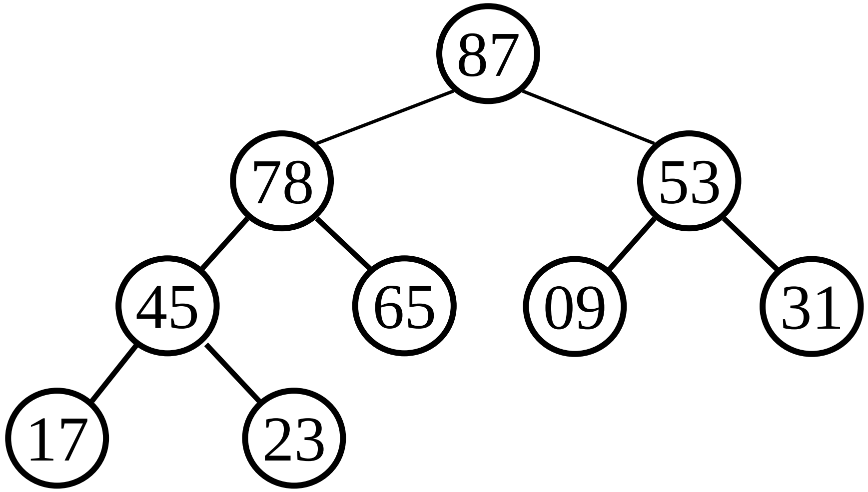


01	02	03	04	05	06	07	08	09	10	11	12
94	93	75	91	85	44	51	18	48	58	10	34

堆的性质

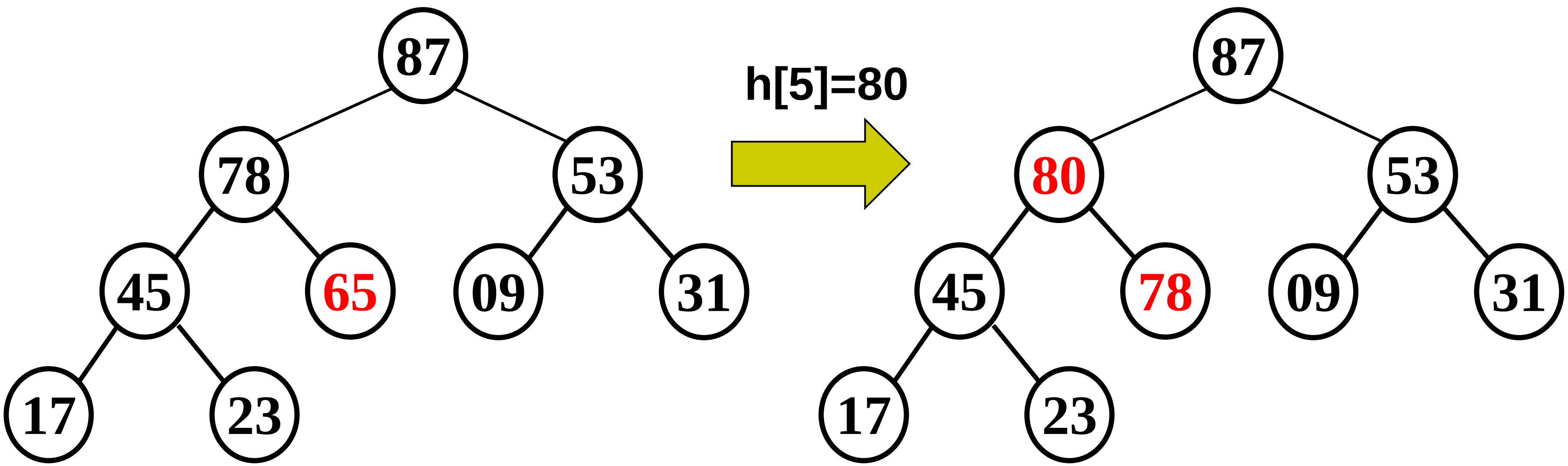
- 完全二叉树
  - ✓ h[MAXN], hlen
- 堆有序
  - ✓ 小根堆： $K_i \le K_{2i}$ 且 $K_i \le K_{2i+1}$
  - ✓ 大根堆： $K_i \ge K_{2i}$ 且 $K_i \ge K_{2i+1}$





上浮操作

- 当大根堆的元素值h[x]变大时，该结点可能会上浮；

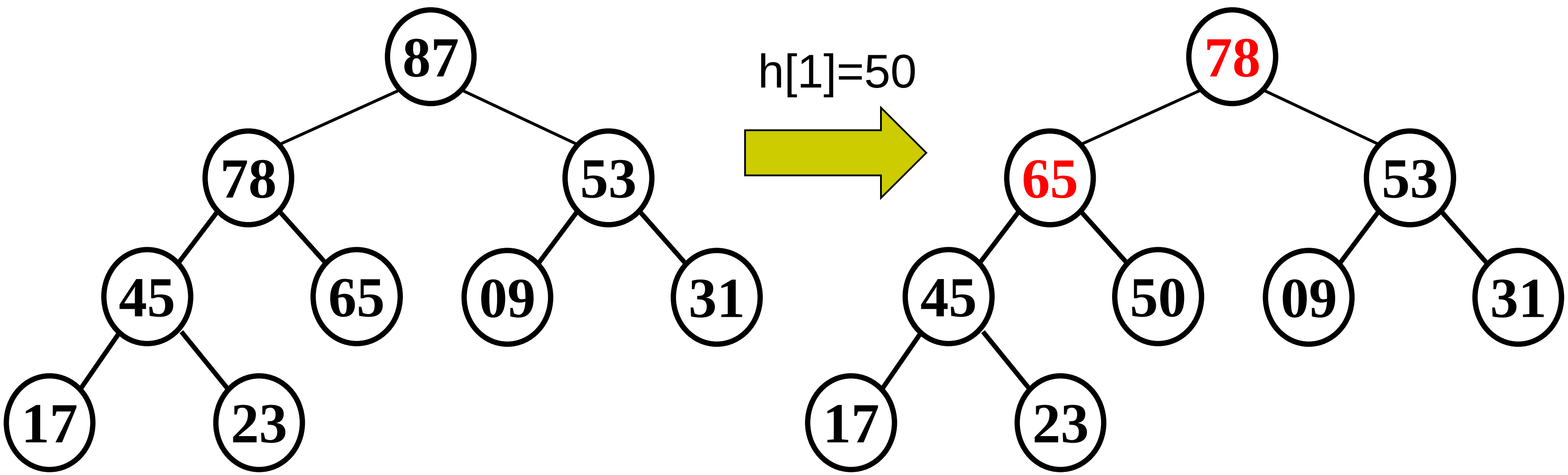


up 实现

```
inline void up(int x)   // h[x]上浮
{
    int i = x ;
    while( i > 1 && h[ i ] > h[ i / 2 ] ) {
        swap( h[ i ], h[  i / 2 ] );
        i /= 2;
    }
}
// 时间复杂度  O(log n) ;   使用位运算 i  >> 1代替除2
```

下沉操作

- 当大根堆的元素值**h[x]**变小时，该结点可能会下沉；



down 实现

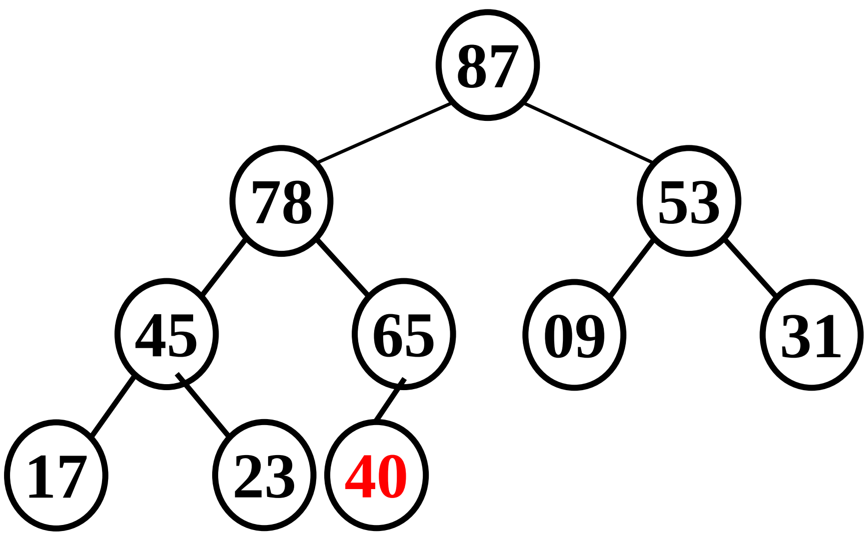
```
inline void down( int x )   // h[x]下沉
{
     int i = x, y ;
     while ( 2*i <= hlen && h[ 2*i ] > h[ i ] ||
               2*i+1<=hlen && h[ 2*i + 1 ]>h[ i ] ) {
          y = 2 * i;
          if ( 2*i+1<=hlen && h[ 2*i+1 ]>h[ 2*i ] ) y++;
          swap(h[i], h[y]);
          i = y;
     }
}   //  时间复杂度  O(log n)
```

down 实现

```
inline void down( int x )   // h[x]下沉
{
    int i = x, y ;
    while ( 2*i <= hlen) {
            y=2*i;
            if (2*i+1<=hlen && h[ 2*i+ 1 ]>h[ 2*i] ) y++;
            if (h[ y ] > h[ i ] ) { swap(h[i], h[y]); i = y; }
            else break;
    }
}   //  时间复杂度  O(log n)
```

插入操作

- 插入一个元素，把该元素放在最后，再做**up**操作。



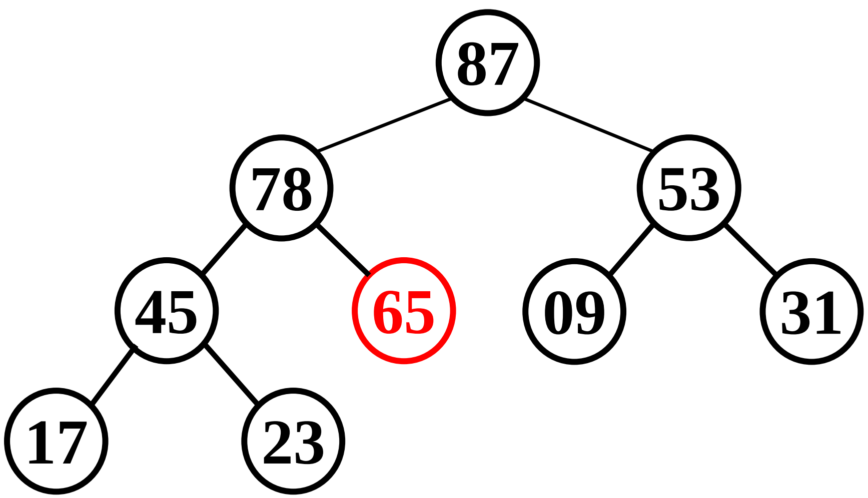
insert实现

```
inline void insert( int x )   // 插入x
{
    hlen++;
    h[hlen] = x;
    up (hlen);
}

// 时间复杂度  O(log n)
```

删除操作

- 删除第x个元素；为了不破坏堆的性质，把h[hlen]移到x处，堆元素个数减一，再判断做up(x)还是down(x)。



delete实现

```
inline void delete( int x )   // 删除h[x]
{
    int t = h[x];
    h[x] = h[hlen];
    hlen--;
    if( h[x] > t ) up (x); else down(x);
}
// 时间复杂度  O(log n)
```

初始建堆

- 目标：建立一个n个元素的堆。

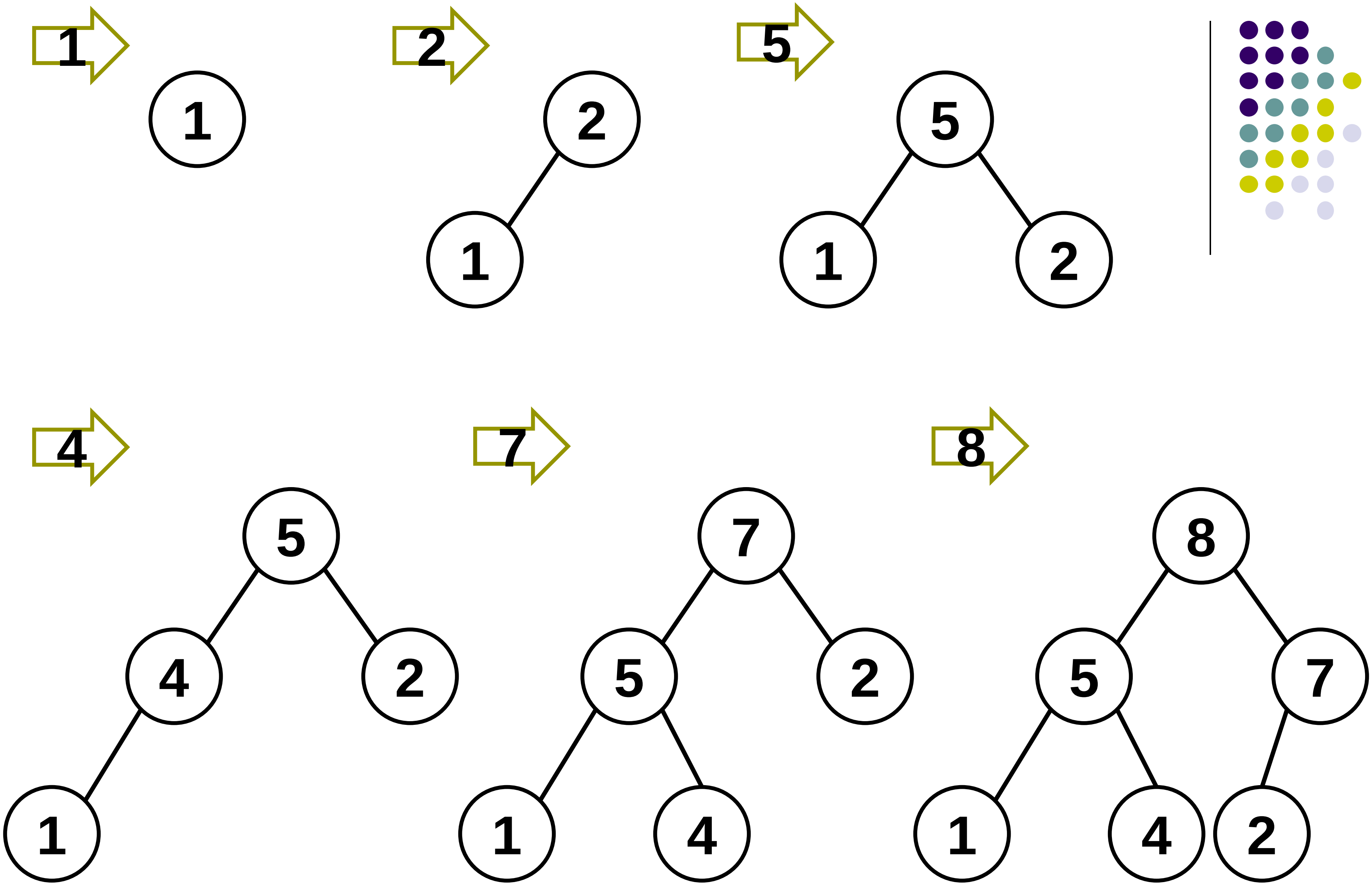
- 例: {1,2,5,4,7,8 }

方法一

- 执行n次insert操作。
- 参考代码

```
void build()
{
    for( int i=1 ; i<=n ; i++ ) insert( a[i] );
}
```

例: {1,2,5,4,7,8 }


1
1
2
2
1
5
5
1
2
4
5
4
2
1
7
7
5
2
1
4
8
8
5
7
1
4
2

方法一时间复杂度分析

- 堆结点个数为 n，高度为 *k*
- 方法一的元素移动次数最多

$$T_1(n)=\sum_{i=0}^{k-1} i*2^i+(n-2^k+1)*k$$

$$T=\sum_{i=0}^{k-1} i*2^i=k*2^k-2^{k+1}+2$$

$$T_1(n)=(n+1)k-2^{k+1}+2=O(n\log n)$$

T的计算

$$T=\sum_{i=0}^{k-1} i*2^{i}$$

$$2*T=\sum_{i=0}^{k-1} i*2^{i+1}=\sum_{i=0}^{k-1}(i+1)*2^{i+1}-\sum_{i=0}^{k-1}2^{i+1}$$

$$=\sum_{i=1}^{k} i*2^{i}-\sum_{i=1}^{k}2^{i}=T+k*2^{k}-2^{k+1}+2$$

$$T=k*2^{k}-2^{k+1}+2$$

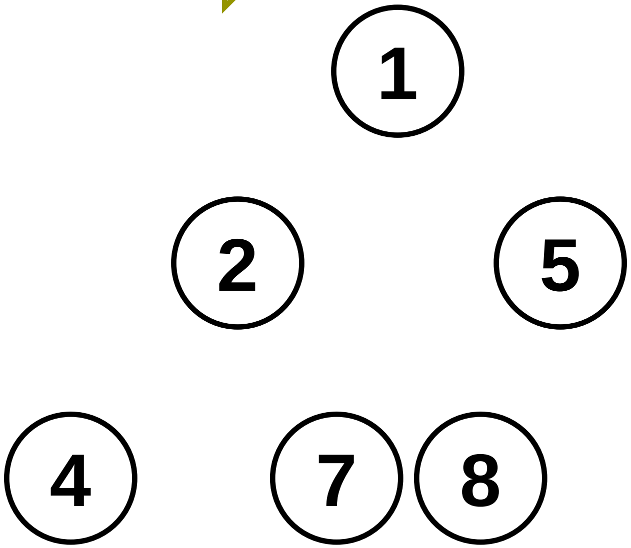
方法二(筛选法，Floyd)

- 调整：执行n/2次down操作。
- 参考代码

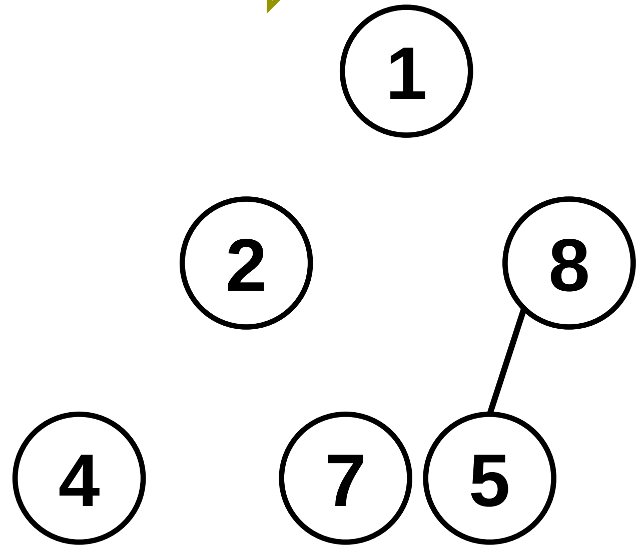
```
void build() {
    hlen = n;
    for(int i=1 ; i<=n ; i++) h[ i ]=a[ i ];
    for(int i=n/2 ; i>=1 ; i--) down( i );
}
```

- 例: {1,2,5,4,7,8 }

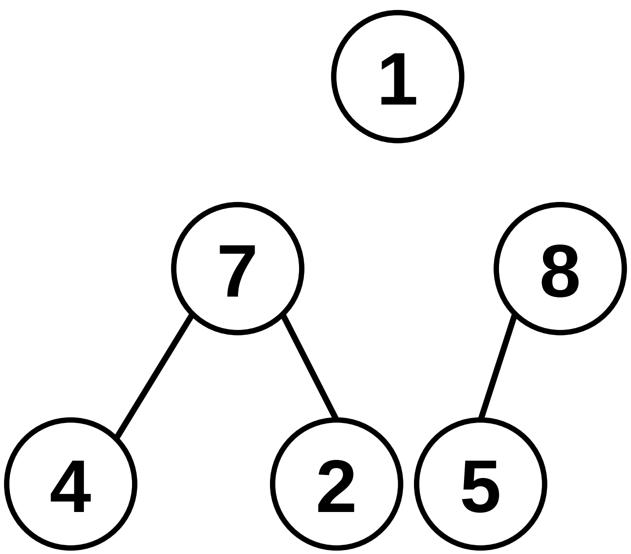
初始



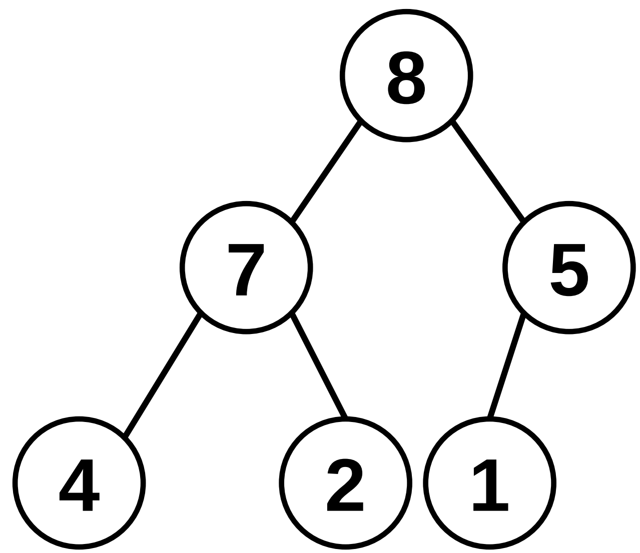
K=n/2=3



K=2



K=1



方法二时间复杂度分析

$$T = k*2^k - 2^{k+1} + 2$$

- 堆结点个数为 n，高度为 *k*
- 方法二的元素移动次数最多

$$T_2(n) = \sum_{i=0}^{k-1}(k-i)*2^i = \sum_{i=0}^{k-1}k*2^i - \sum_{i=0}^{k-1}i*2^i$$

$$T_2(n) = k*\left(2^k - 1\right) - T$$

$$T_2(n) = 2^{k+1} - k - 2 = O(n)$$

堆的经典应用——堆排序

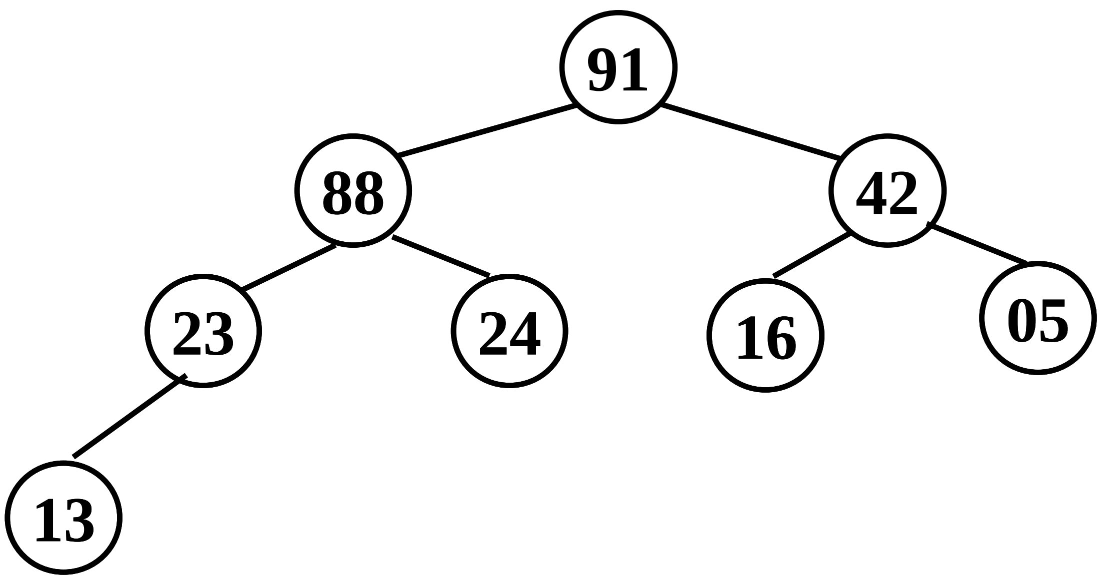
- 用堆把n个数从小到大排序。

- 先把输入数组A[1..n]用build建成一个大根堆。
- 每次取最大元素放到对应位置，重复n-1次
  - ✓ 数组中最大元素在根A[1]，把它与A[n]互换来达到最终正确的位置；
  - ✓ 把堆的元素个数减1；
  - ✓ 通过down(1)把余下n-1个元素调整成大根堆。

例：堆排序过程

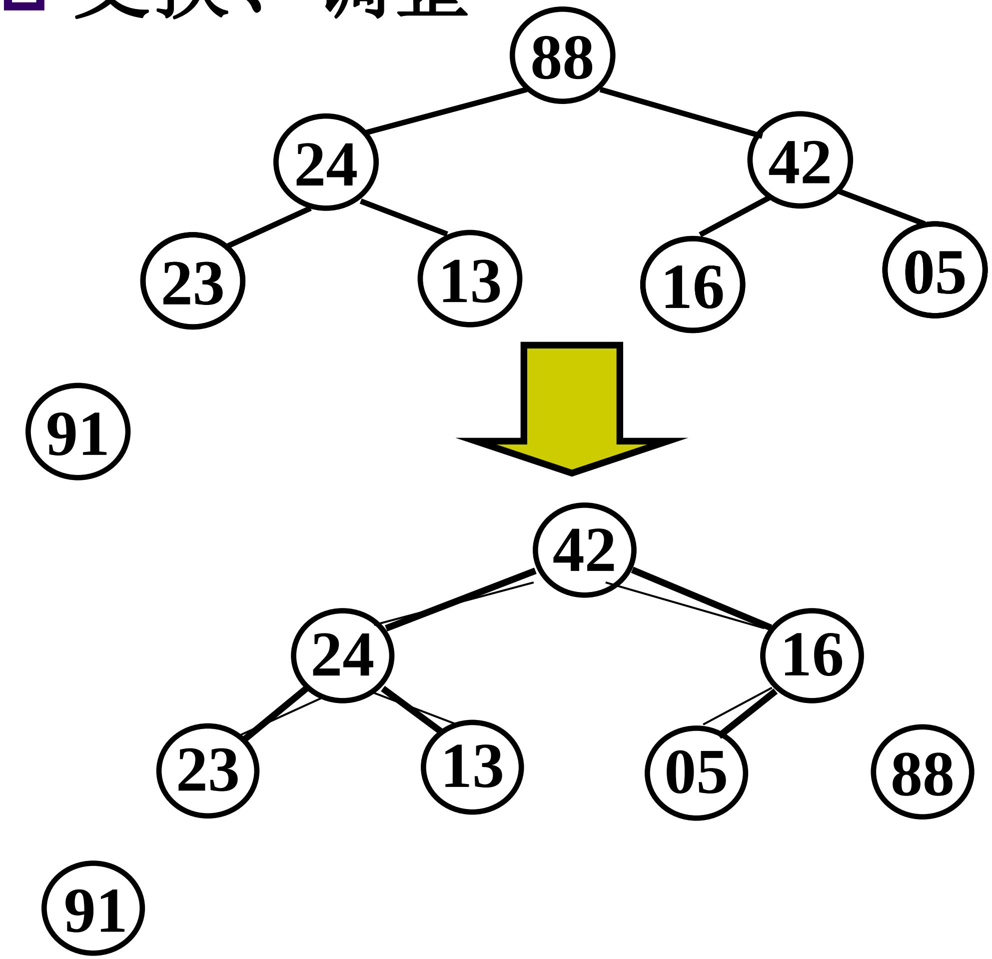
- 关键字序列(42, 13, 91, 23, 24, 16, 05, 88)

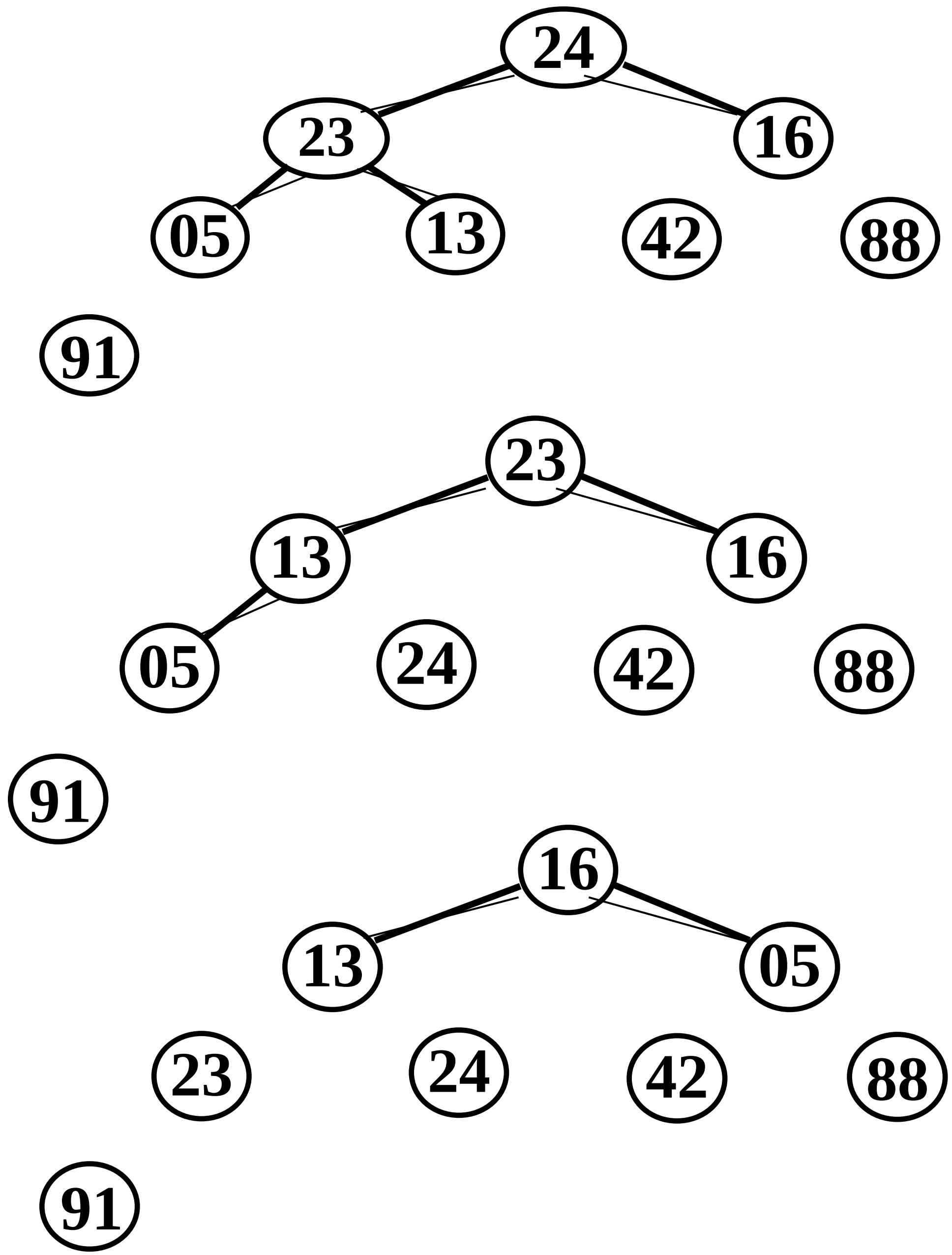
- 第1步：初始建堆

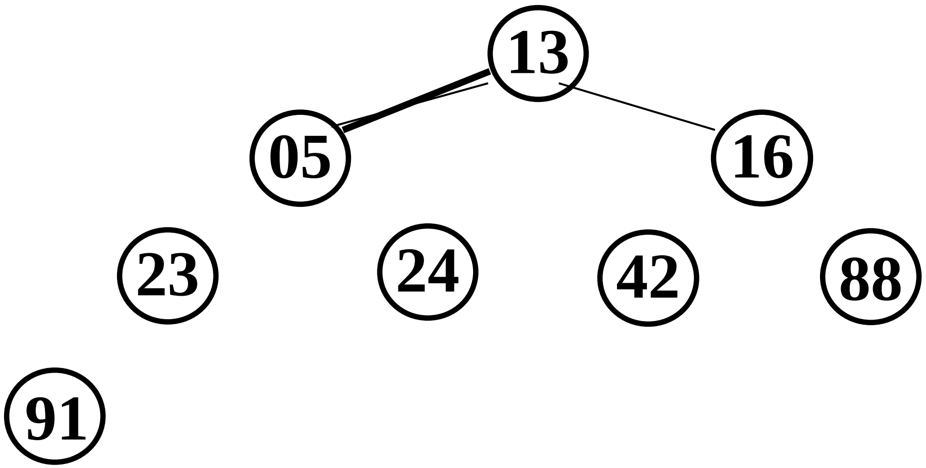


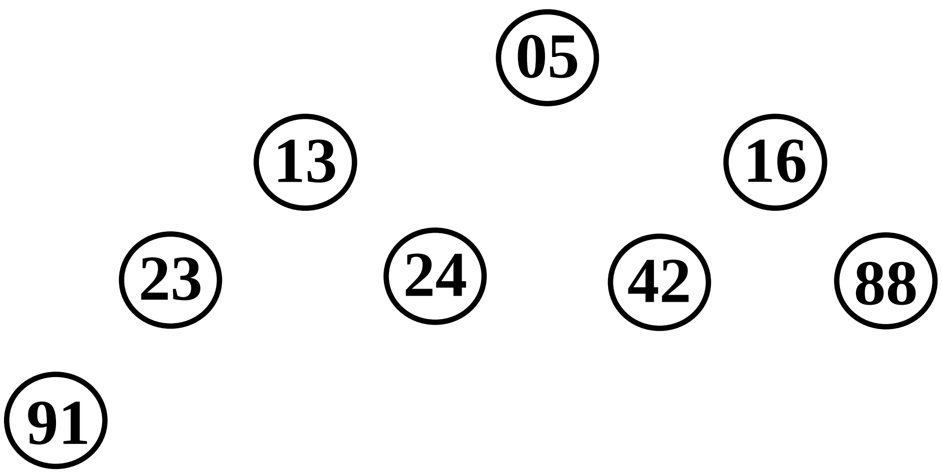
第2步 堆排序

- 交换、调整




24
23
16
05
13
42
88
91
23
13
16
05
24
42
88
91
16
13
05
23
24
42
88
91


13
05
16
23
24
42
88
91


05
13
16
23
24
42
88
91

算法HeapSort

H1. [初始建堆]

```
build();
```

H2. [n-1次选最大]

```
for(i=1 ; i<=n-1 ; i++){
    swap(1, hlen);
    hlen--;
    down(1);
}
```

堆排序时间复杂度

- 时间复杂度为**$O(n\log_2 n)$**
  - 初始建堆　$O(\ n\ )$;
  - n-1次选最大$O(n\log_2 n)$;
- **:最好、最坏和平均情况相同**

可合并堆(Mergeable Heap)

- **要维护多个堆**
  - 支持合并操作
  - UNION(H_1，H_2)：返回一个包含堆H_1和堆H_2中所有元素的新堆。

- **多种方案：Fibonacci堆、左氏堆、左偏树、斜堆等（一个研究点）**

优先队列

- 优先队列(priority queue)与普通队列不同，其中的元素被赋予优先级（或关键字）。当访问元素时，具有最高优先级的元素最先被访问。

- 例： 共享计算机系统的作业调度
  - 作业：job/task
  - 将要执行的作业放到作业队列，每个作业有一个优先级
  - 当一个作业完成或被中断，选择优先级最高的作业执行

优先队列的核心操作

- 两种形式：最大优先队列和最小优先队列
- 最大优先队列
  - INSERT(S,x): 把元素x 插入到S中；
  - MAXIMUM(S) : 返回S中优先级最高的元素；
  - EXTRACT-MAX(S)：去掉并返回S中优先级最高元素
  - INCREASE-KEY(S,x,k)：将元素x优先级增加到k(k≥x)
- 最小优先队列：**INSERT，MINIMUM，EXTRACT-MIN 和 DECREASE-KEY。**

优先队列的存储和实现

- 通常用堆来实现
  - INSERT(S,x): 堆的insert操作； O(logn)
  - MAXIMUM(S) : 堆顶； O(1)
  - EXTRACT-MAX(S)：堆的delete操作O(logn)
  - INCREASE-KEY(S,x,k)：堆的上浮操作O(logn)
- 也可用其它方式实现

STL Heap

- *STL*中的*Heap*是一个类属算法，包含在头文件*algorithm*中。
- *Heap*在STL中部不是一种容器组件，需要搭配容器使用。
  - make_heap: 将某区间内的元素转化成heap（默认大根堆；提供排序规范生成小根堆）
  - push_heap: heap增加一个元素，元素事先放堆尾
  - pop_heap: heap减少一个元素，元素结果在堆尾
  - sort_heap: heap转化为一个已序群集。

STL Priority Queue

- #include <queue>

- priority_queue<int,vector<int>,greater<int> > pq; //内部用到heap

- pq.push(x);
- pq.pop();
- pq.top()
- pq.empty()

总结

- 堆的定义和性质
- 堆的基本操作
  - 上浮、下沉、插入、删除、建堆
- 堆排序
- 堆的扩展和应用（可合并堆和优先级队列）