L9：优先队列与堆

吉林大学计算机学院
谷方明
fmgu2002@sina.com

学习目标

- 掌握优先队列的定义、操作和实现
- 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

背景

- 队列的特性是先进先出。先进入队列的先获得服务，是一种自然公平的方式，但未必总是最好的做法。
  - ✓ 医院就诊：挂号顺序；紧急患者优先处理
  - ✓ 打印调度：特别重要的打印作业，优先处理
  - ✓ 银行的VIP服务，……

优先队列

- 优先队列（**Priority Queue**）：一类特殊的队列，根据用户定义的优先级（**Priority**）决定服务的顺序。

- 优先队列可分为两类
  - ✓ 最小优先队列：先服务优先权最小
  - ✓ 最大优先队列：先服务优先权最大

优先队列与最值问题

- 如果把优先权当成关键字(**Key**)，最小优先队列又可称为最小者先出队列，最大优先队列又可称为最大者先出队列。
- 优先队列就可以应用到取最值的场景中，应用范围大大扩展。

优先队列的操作（以最小为例）

1. 取最小（**getMin** ）： 获得队列中优先权最小的元素；
2. 删最小（ **deleteMin** ）： 删除队列中优先权最小的元素；
3. 插入 （ **insert** ）： 向队列中插入元素**x**。

4. 建队（make）：创建空的优先队列Q。

5. 删除（delete）：删除队列中x位置的元素;

6. 减值（decreaseKey）：队列中x位置的优先权减少为k;

7. 合并（merge）: 把两个优先队列Q1和Q2合成一个优先队列Q

优先队列的实现

- 使用线性结构实现？
  - 例如顺序表、链表等。
  - 如果表中的元素无序时，取最小或删最小的时间复杂度将达到O(N)，N为表中元素的个数；
  - 如果表中数据全有序时，那么插入操作的时间时间复杂度将达到O(N)。

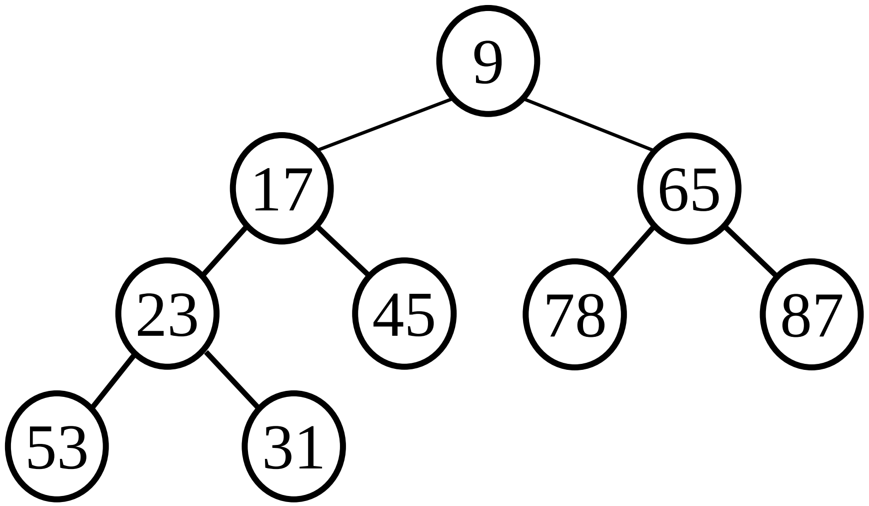
- 使用树形结构实现？（高效）
  - 在树形结构中，**数据维持部分序关系（Partical Order）**，要求每个结点的关键字小于其儿子结点的关键字。
  - **这样的树形结构被称为堆，维持的部分序也被称为堆序**。
  - 堆中最常用的是二叉堆，特别是完全二叉堆。一般所说的堆通常也指完全二叉堆。

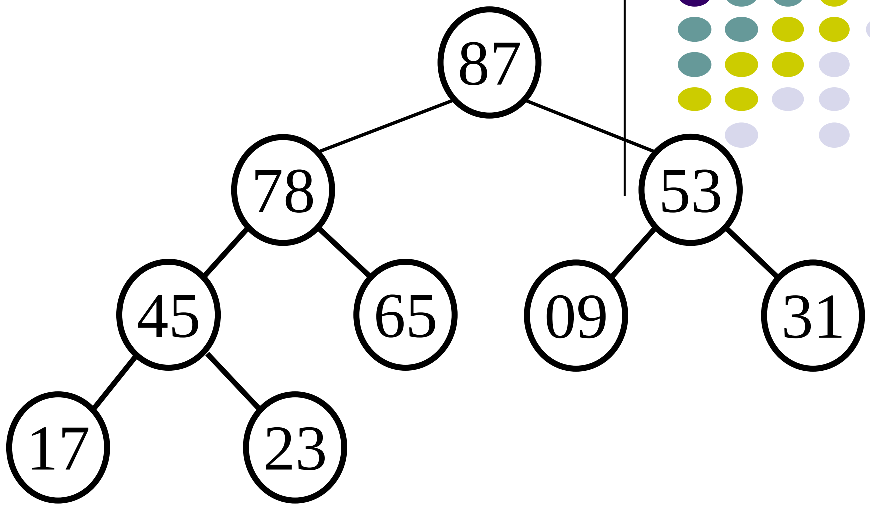
堆（Heap）

- 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。

- 大根 (极大)堆
- 小根 (极小)堆

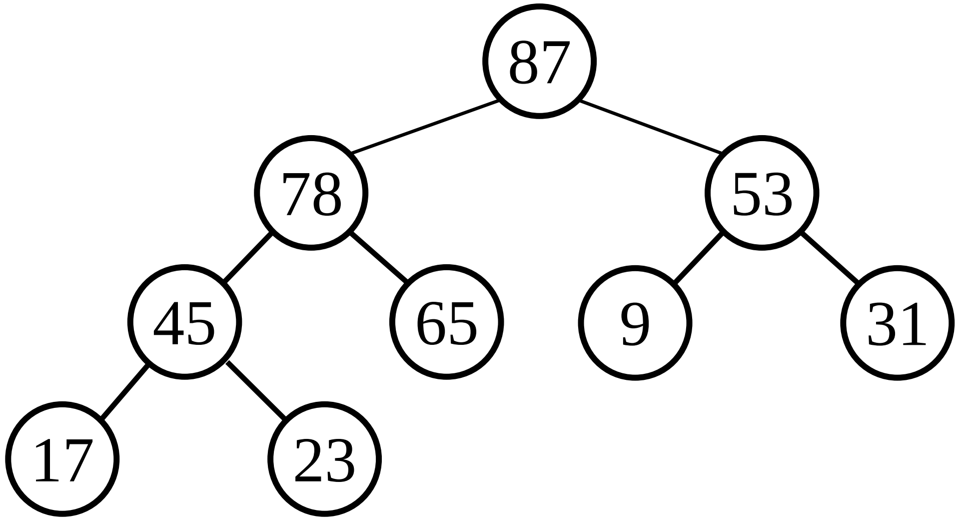
堆的性质1





- 堆序（有序性）
  - ✓ 小根堆： $K_i \le K_{2i}$ 且 $K_i \le K_{2i+1}$
  - ✓ 大根堆： $K_i \ge K_{2i}$ 且 $K_i \ge K_{2i+1}$

堆的性质2

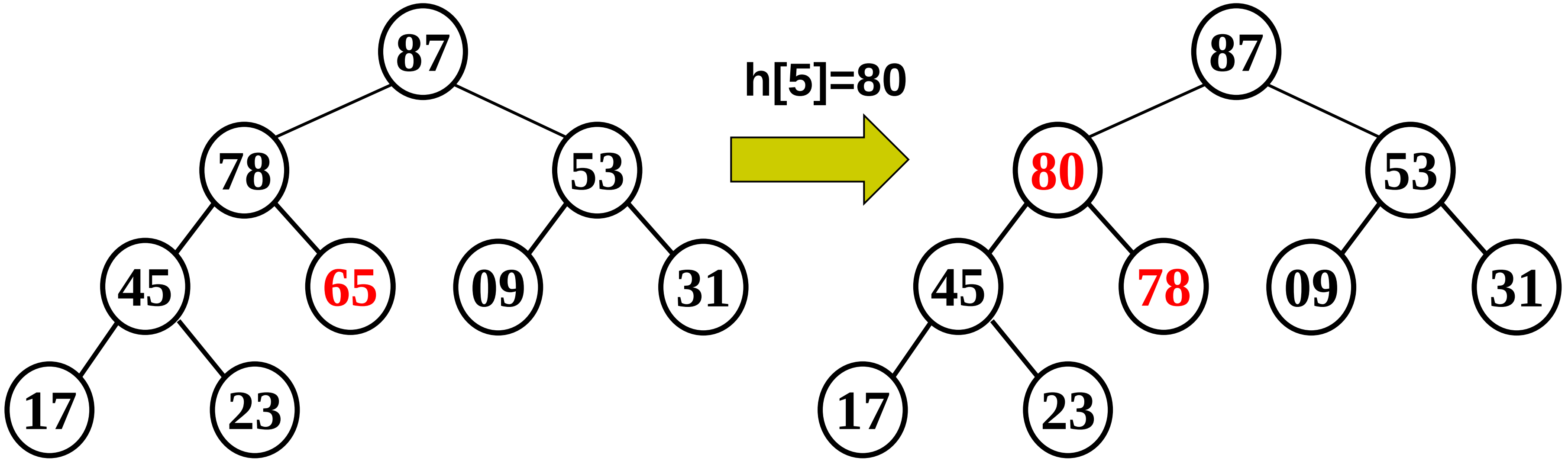


87	**78**	**53**	**45**	**65**	**9**	**31**	**17**	**23**
01	**02**	**03**	**04**	**05**	**06**	**07**	**08**	**09**

- 完全二叉树（结构性）
  - ✓ h[MAXN], hlen

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；

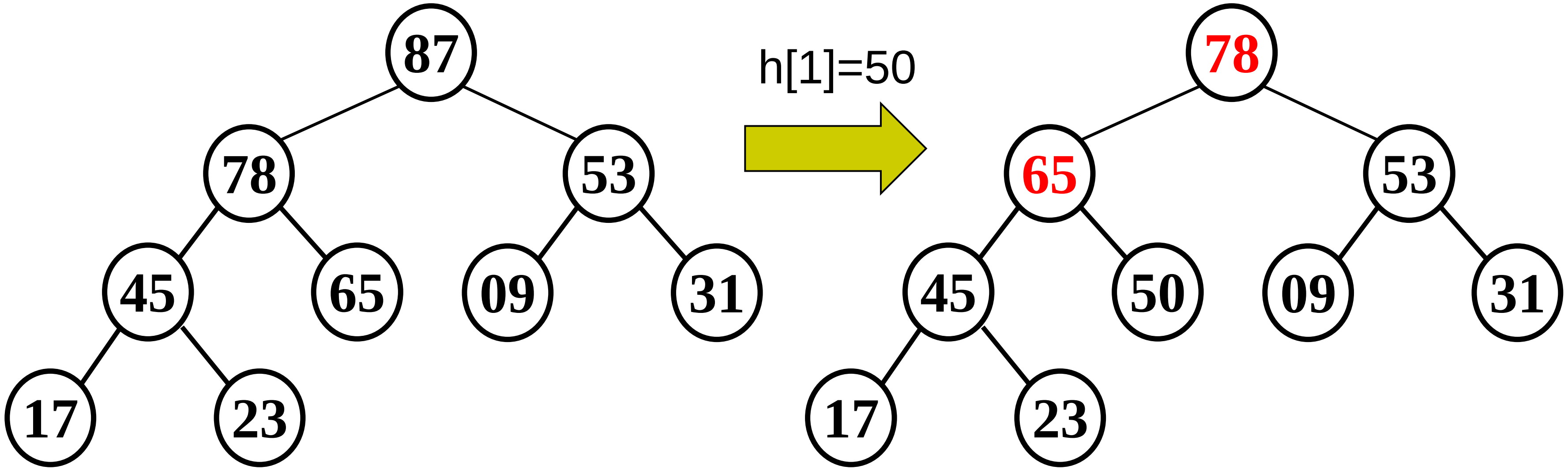


up 实现

```
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1

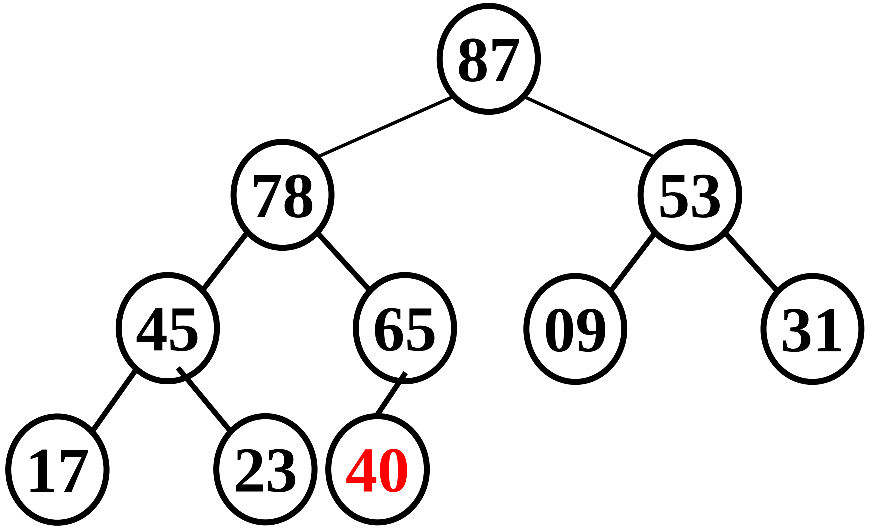
```
inline void down( int x )   // h[x]下沉
{
    int i = x, y ;
    while ( 2*i <= hlen && h[ 2*i ] > h[ i ] ||
               2*i+1<=hlen && h[ 2*i + 1 ]>h[ i ] ) {
          y = 2 * i;
          if ( 2*i+1<=hlen && h[ 2*i+1 ]>h[ 2*i ] ) y++;
          swap(h[i], h[y]);
          i = y;
     }
}   //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

down 实现2

```
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。



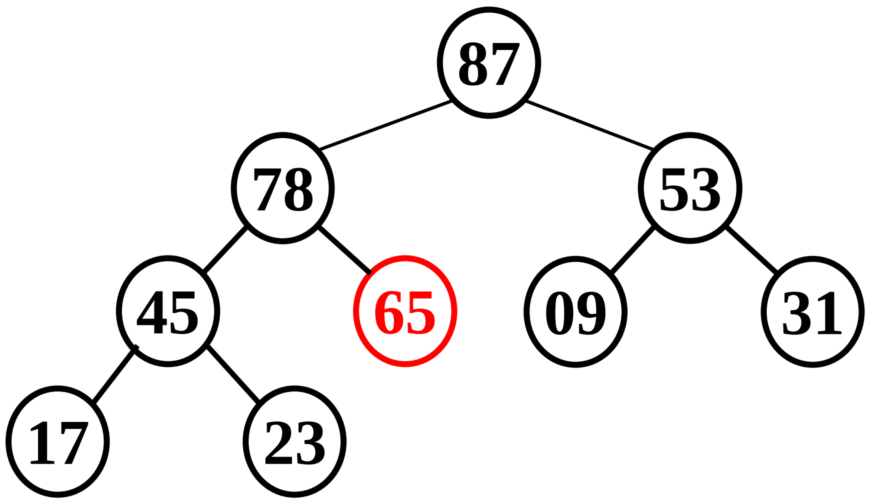
insert实现

```
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。

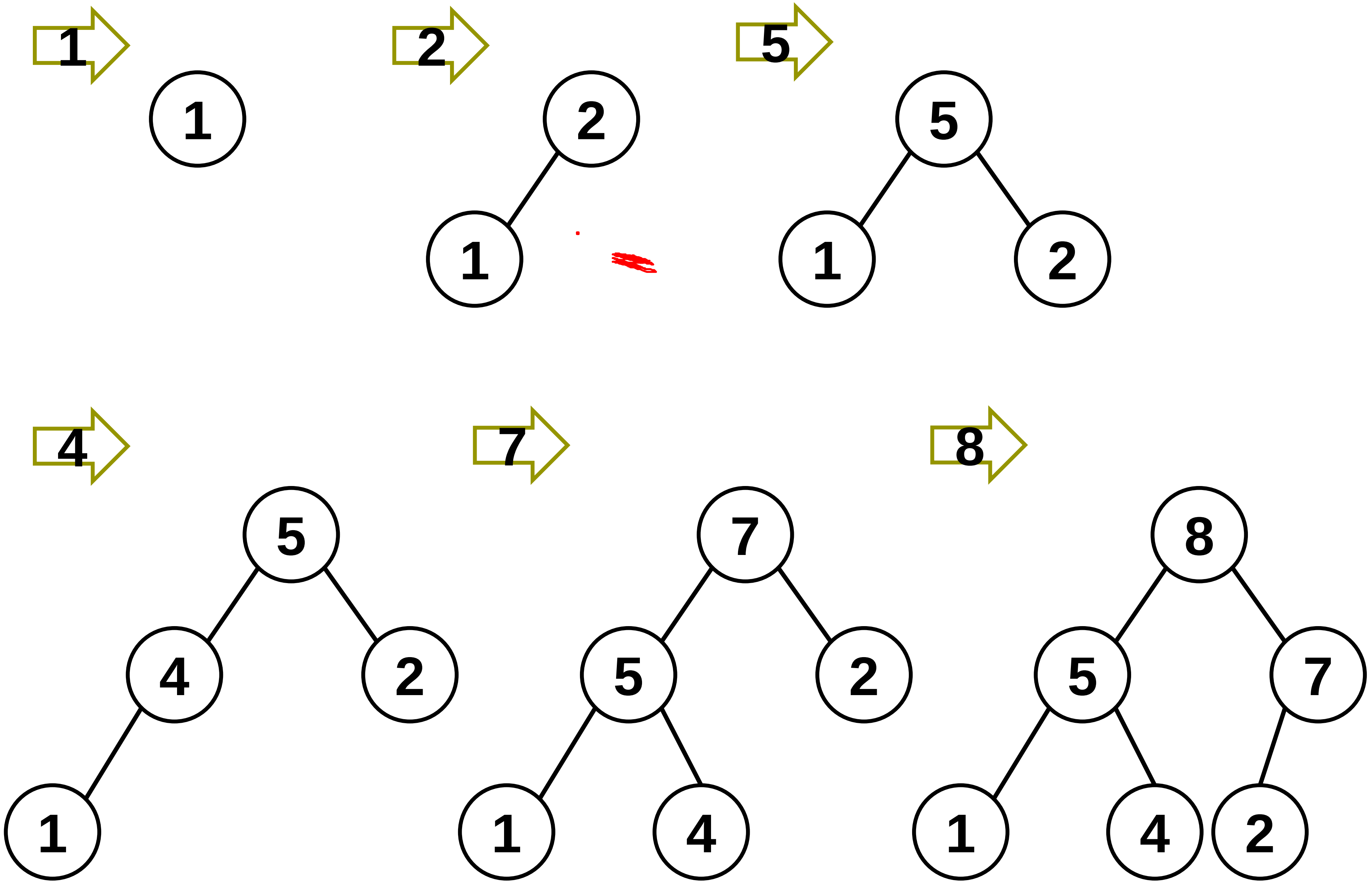
- 例: {1,2,5,4,7,8 }

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
2
5
4
7
8

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

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

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

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

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

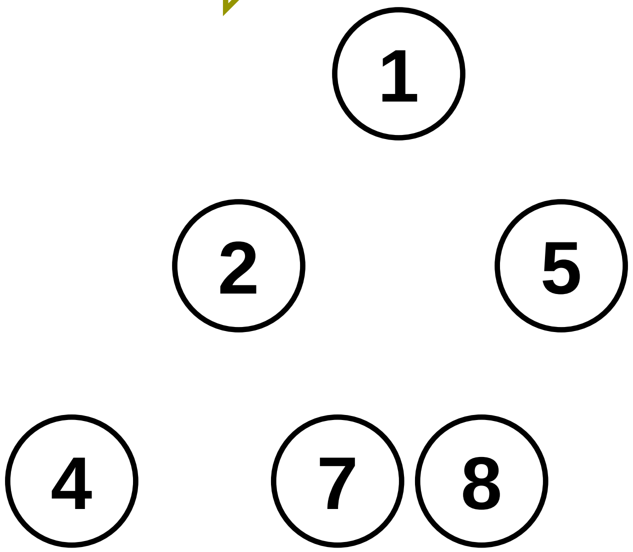
方法二(筛选法，Floyd)

- 调整：执行n/2次down操作。
- 参考代码

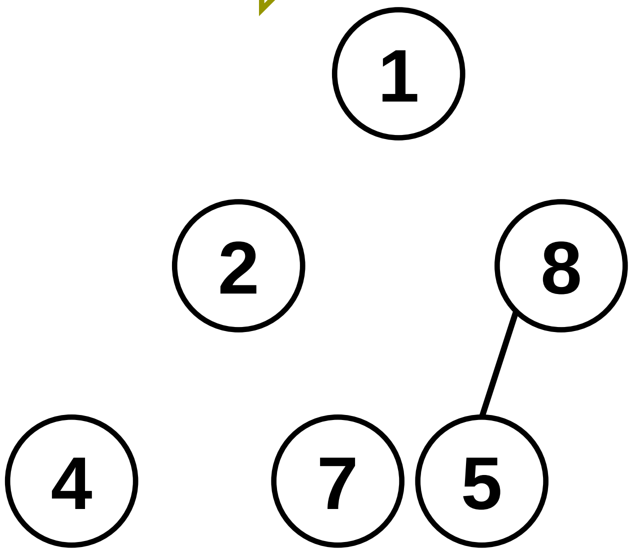
```
void build() {
    hlen = n;
    for(int i=1 ; i<=n ; i++) h[ i ]=a[ i ];
    for(int i=n/2 ; i>=1 ; i--) down( i );
}
```

- 例: {1,2,5,4,7,8 }

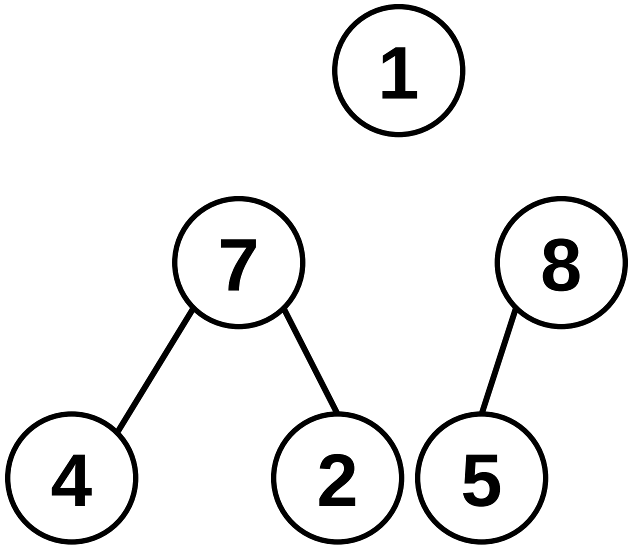
初始



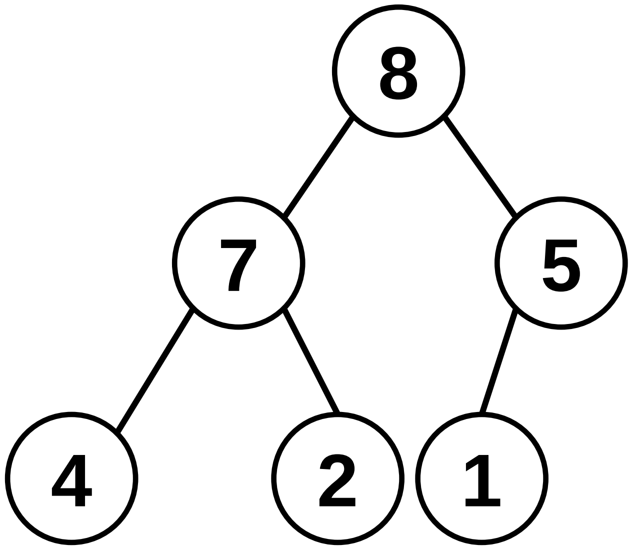
K=n/2=3



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

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

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

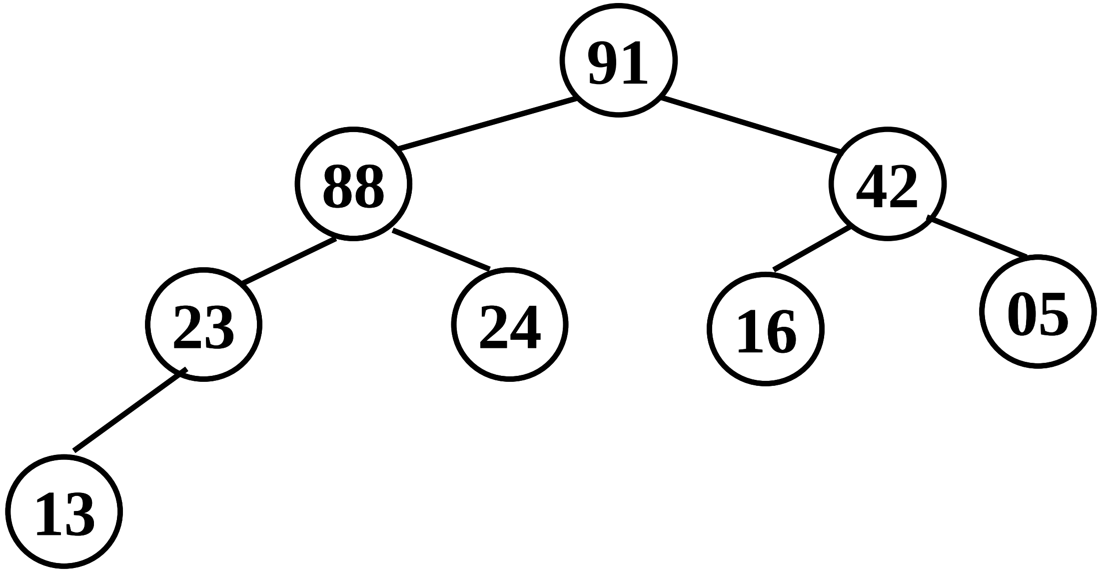
$$T_2(n) = 2^{k+1} - k - 2 = O(n)$$

堆的应用1：堆排序

- **用堆把n个数从小到大排序。**
- **思想：每次选择第i大的放到 n+1-i 处（选择排序）**
- **描述**
  - 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n+1-i 处，重复n-1次

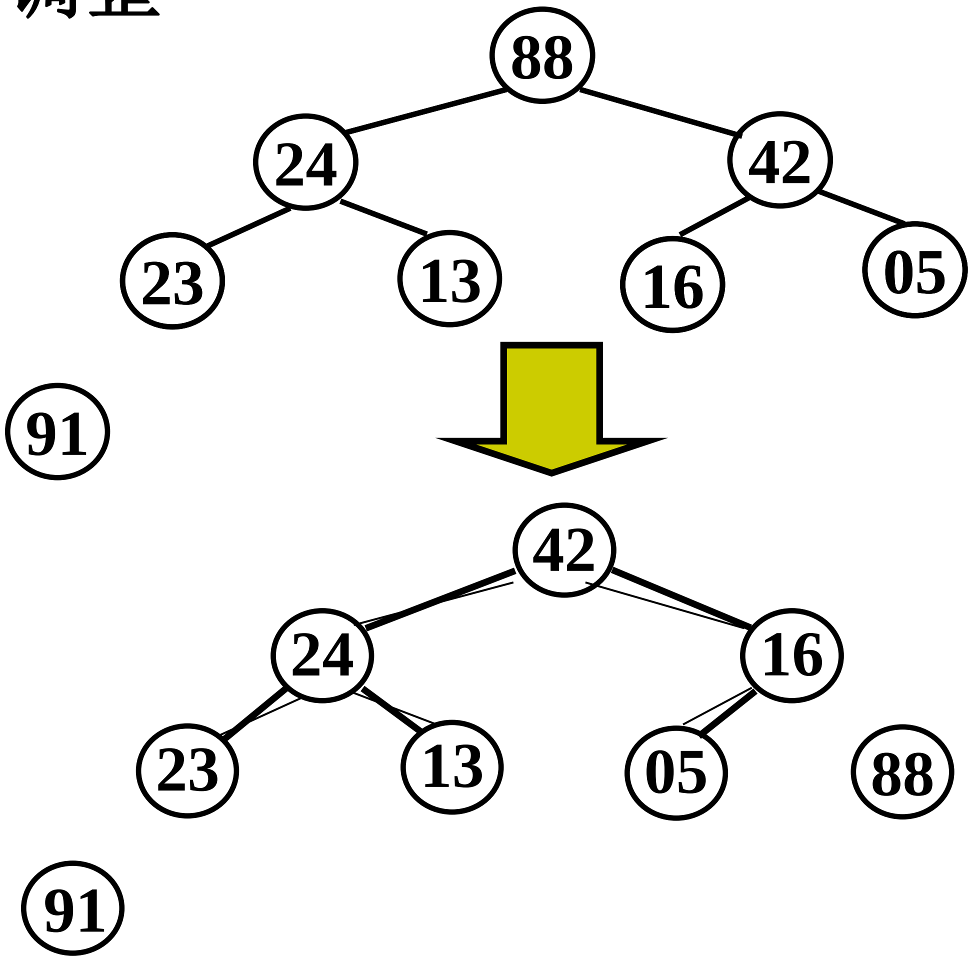
堆排序过程

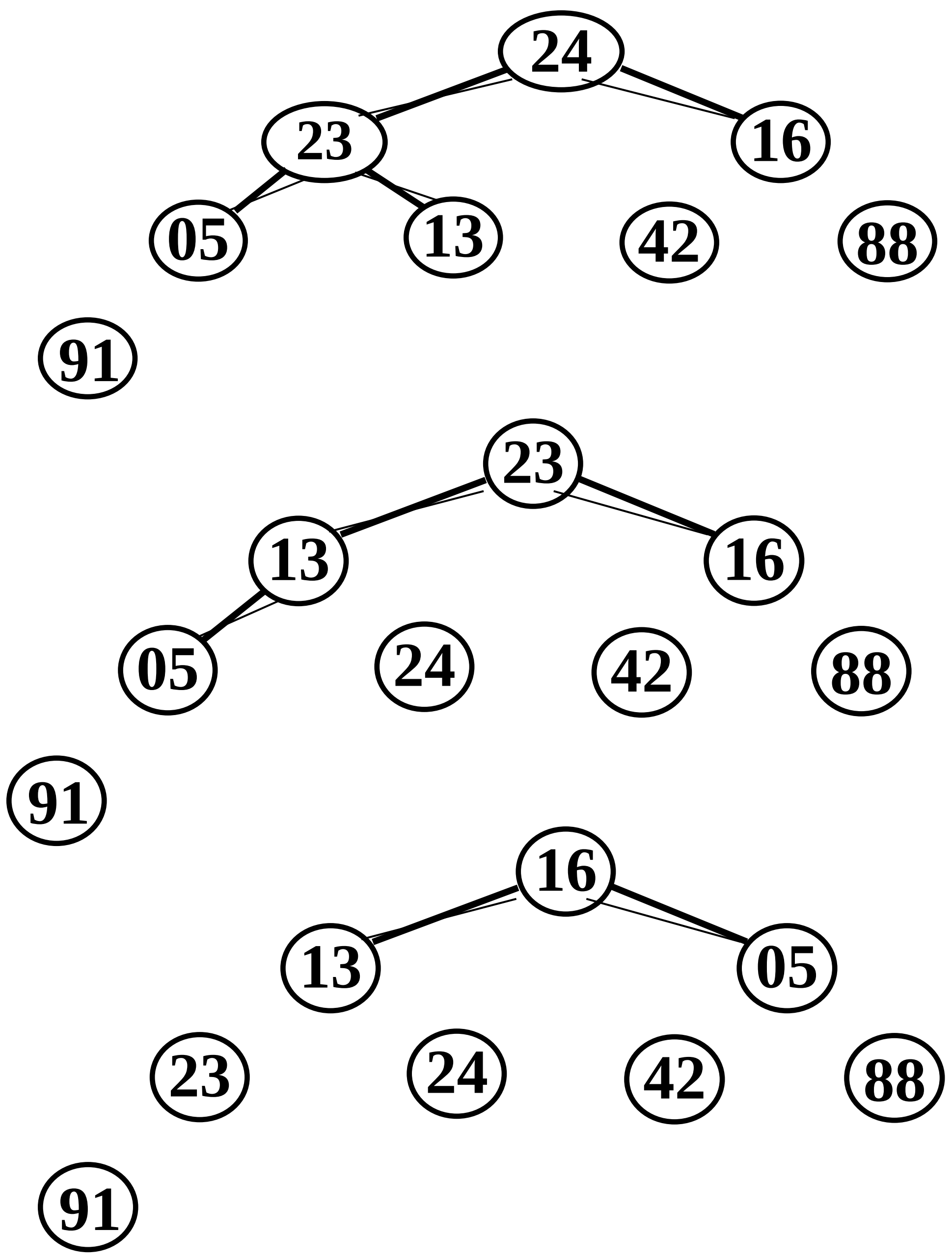
- 关键字序列(42, 13, 91, 23, 24, 16, 05, 88)
- 第1步：初始建堆

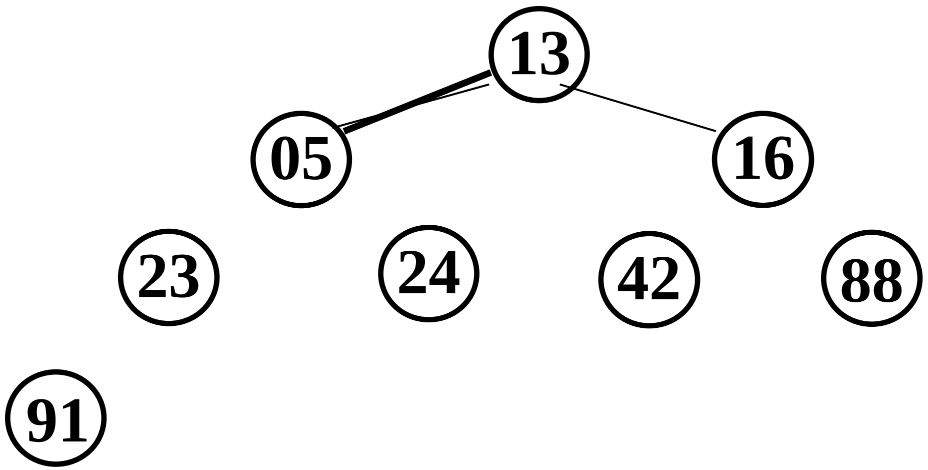


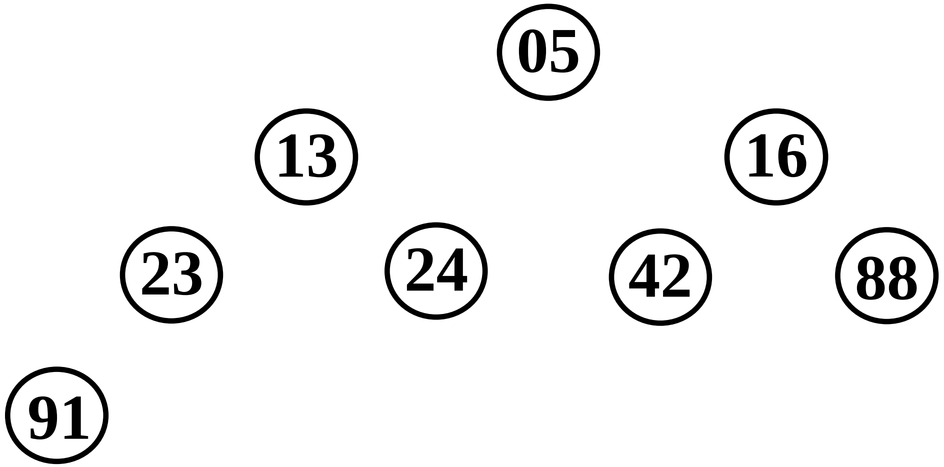
第2步 堆排序

- 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

堆的应用2：实现优先队列

- 优先级队列通常用堆来实现
  - insert：堆的insert操作， O(logn)
  - getMin: 堆顶， O(1)
  - deletMin：堆的delete操作， O(logn)
  - decreaseKey(S,x,k)：堆的上浮操作，O(logn)
  - make: 堆的build操作，O(n)
  - delete: 堆的delete操作，，O(logn)

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）**

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）