数据结构与算法

Data Structures and Algorithms

张岩

海量数据计算研究中心

哈工大计算机科学与技术学院

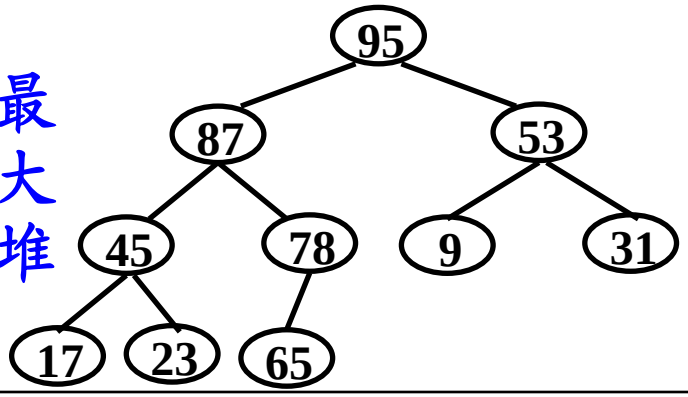
3.3 堆（Heap）

一、ADT堆

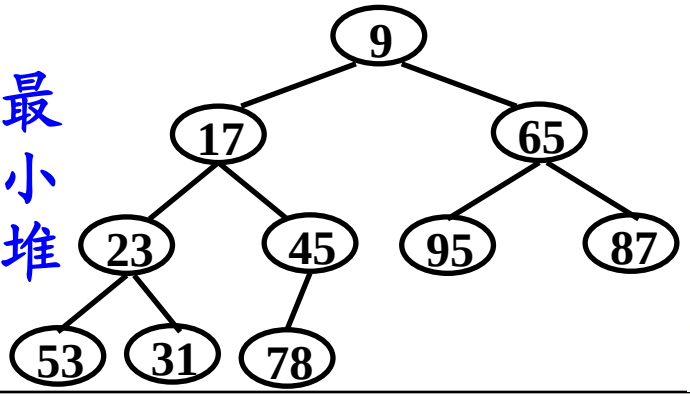
堆的定义

- 如果一棵完全二叉树的任意一个非终结结点的元素都不小于其左儿子结点和右儿子结点（如果有的话）的元素，则称此完全二叉树为最大堆。
- 如果一棵完全二叉树的任意一个非终结结点的元素都不大于其左儿子结点和右儿子结点（如果有的话）的元素，则称此完全二叉树为最小堆。
- 结构特点：根结点的元素是最大（小）的。。。。。。

最大堆



最小堆



3.3 堆（Heap）

一、ADT堆

ADT堆的基本操作

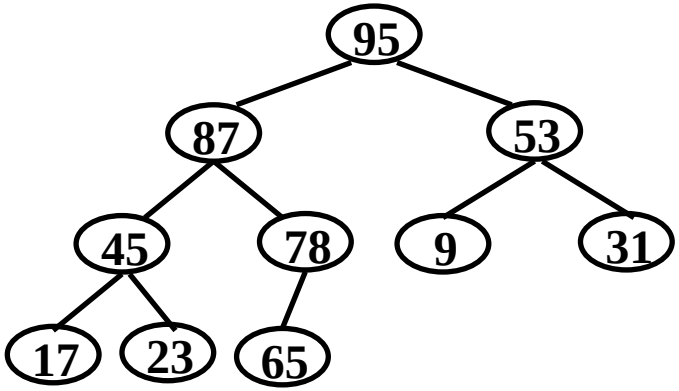
- **MaxHeap(maxsize)**：创建一个空堆，最多可容纳maxsize个元素
- **HeapFull(heap，n)**：判断堆是否为满。若堆中元素个数n达到最大容量maxsize，则返回TRUE，否则返回FALSE;
- **Insert(heap，item，n)**：插入一个元素。若堆不满，则将item 插入heap，否则不能插入;
- **HeapEmpty(heap，n)**：判断堆是否为空。若堆中元素个数n大于0，则返回TRUE，否则返回FALSE;
- **DeleteMax(heap，n)**：删除最大元素。若堆为不空，则返回堆中最大元素，并将其删除，否则，返回一个特定值，表明不能进行删除。

3.3 堆（Heap）

二、ADT堆的实现—最大堆的实现

- ADT堆的存储结构
  - 由于堆是一个完全二元树，所以可以采用完全二元树的数组表示。
- 堆的存储结构定义

```
#define Maxsize  200
typedef struct {
    int key;
    /* other fields*/
} ElemType;
typedef struct {
    ElemType data[Maxsize];
    int n;
} HEAP;
```



	95	87	53	45	78	09	31	17	23	65			
0	1	2	3	4	5	6	7	8	9	10			

3.3 堆（Heap）

二、ADT堆的实现—最大堆的实现

- 堆的基本操作的实现
  - ①创建空堆

```
void MaxHeap（HEAP heap）
{
      heap.n=0;
}
```

  - ②判空

```
bool HeapEmpty（HEAP heap）
{
      return (!heap.n);
}
```

  - ③判满

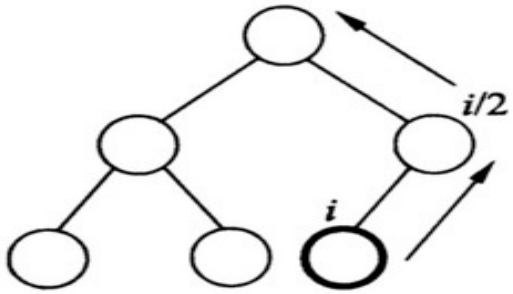
```
bool HeapFull（HEAP heap）
{
      return (heap.n==MaxSize);
}
```

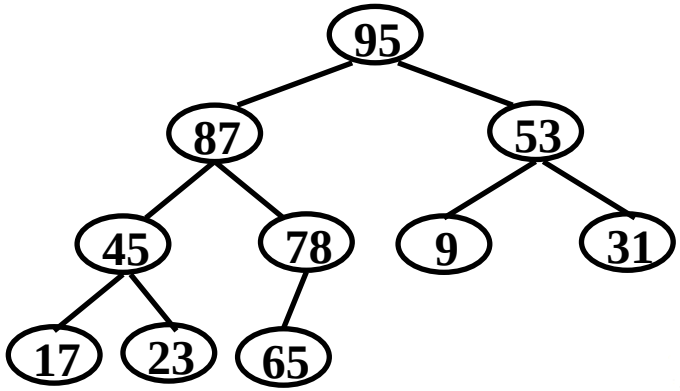
3.3 堆（Heap）

二、ADT堆的实现—最大堆的实现

- 堆的基本操作的实现
  - ④插入

```
void Insert(HEAP& heap，ElemType elem)
{
    int i;
    if (!HeapFull(heap)){
       i=heap.n+1;
       while((i!=1)&&(elem >heap.data[i/2])){
           heap.data[i]=heap.data[i/2];
           i/=2;
       }
    }
    heap.data[i]= elem;
}//时间复杂性O（logn）
```

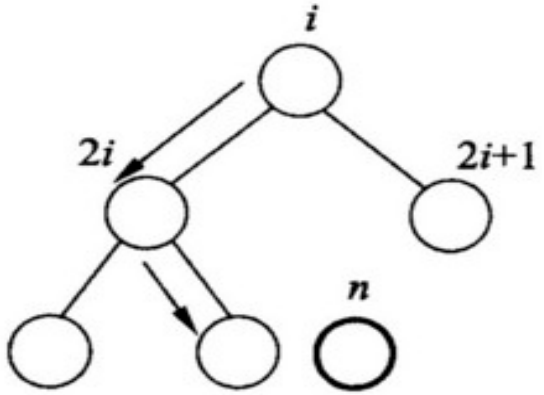


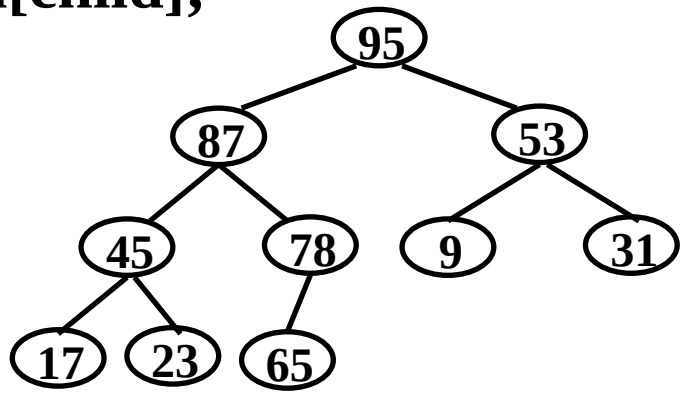


3.3 堆（Heap）

⑤删除最大元素

```
ElemType DeleteMax( HEAP &heap )
{    int parent=1, child=2;
     ElemType elem, tmp;
     if (!HeapEmpty(heap)){
        elem=heap.data[1];
         tmp=heap.data[heap.n--];
        while (child<=heap.n){
            if ((child< heap.n)&&
                 (heap.data [child]<heap.data [child+1]) )
               child++;  //找最大子结点
            if (tmp>= heap.data[child])  break;
            heap.data[parent]= heap.data[child];
            parent=child;
            child*=2;
        }
        heap[parent]=tmp;
        return elem;
     }
}//时间复杂性O（logn）
```





3.3 堆（Heap）

三、堆与优先级队列（Priority Queue）

- 优先队列在计算机操作系统任务调度以及日常工作安排等领域具有十分广泛的应用。
- 在优先队列中，被删除的是优先级最高的元素，而在任何时刻可插入任意优先级的元素。支持这两种操作的数据结构称为最大优先队列。
- 如果被删除的是优先级最低的元素，则相应的数据结构称为最小优先队列。

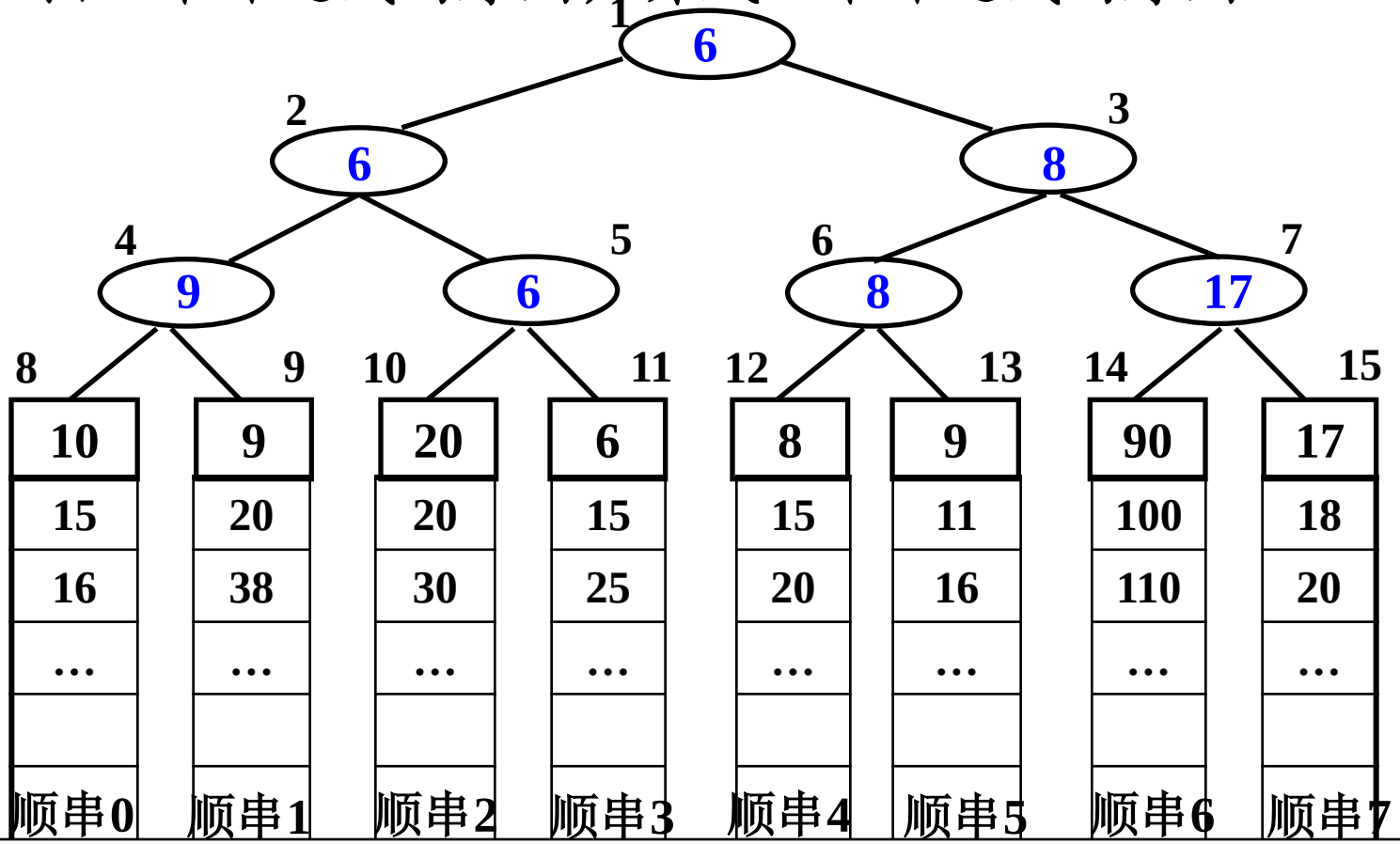
- 优先级队列的各种存储表示和代价分析

存储表示	插入操作	删除操作
无序数组	Θ（1）	Θ（n）
无序单向链表	Θ（1）	Θ（n）
有序数组	O（n）	Θ（1）
有序单向链表	O（n）	Θ（1）
最大堆	O（$\log_2 n$）	O（$\log_2 n$）

3.4 选择树（Selection Tree）

一、背景

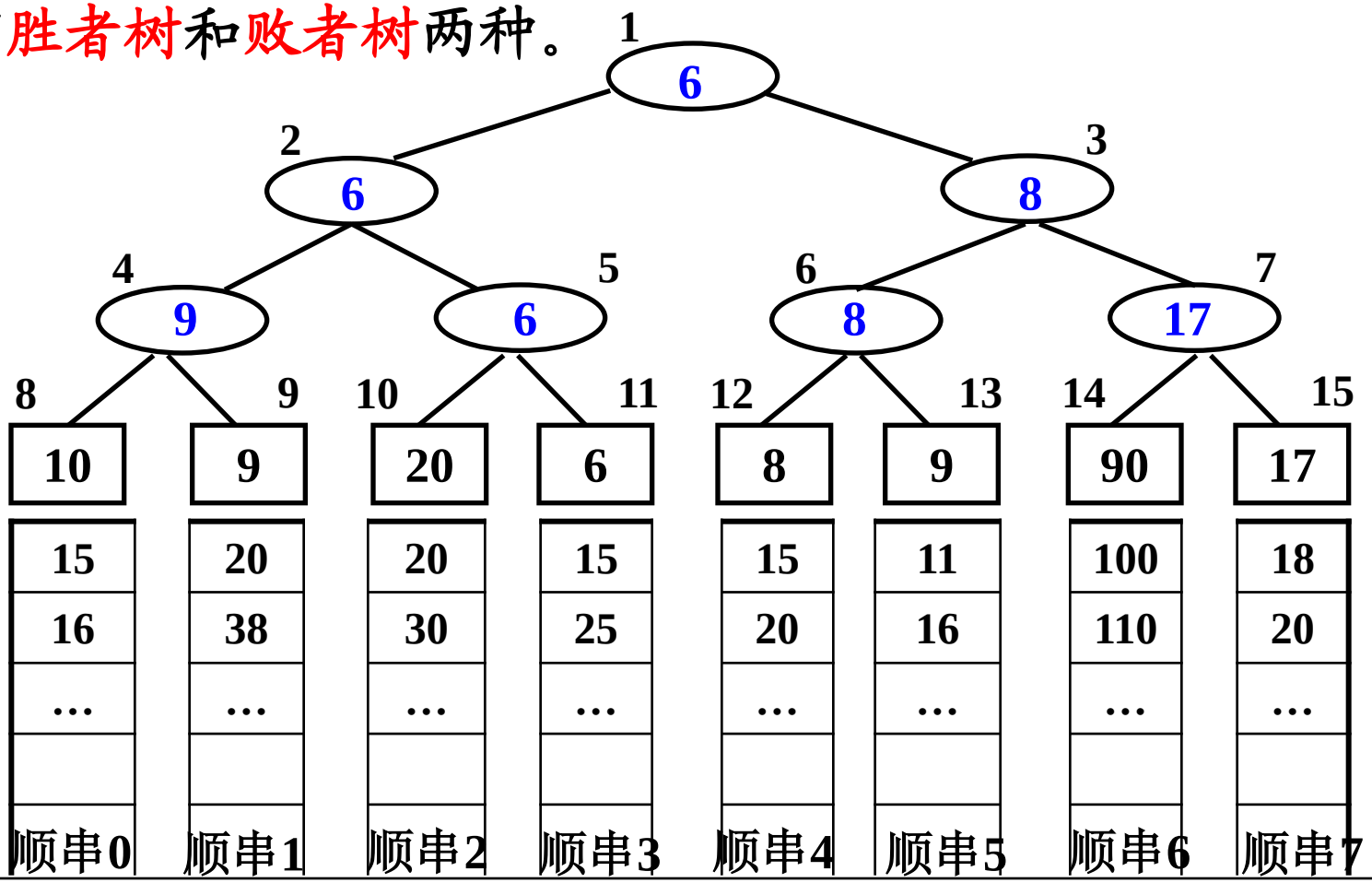
- 如何从n各元素中选择最小的，进而给n各元素排序？
- 如何把K个非递减的序列归并成一个非递减的序列？



3.4 选择树（Selection Tree）

二、选择树（也称Tournament Tree）

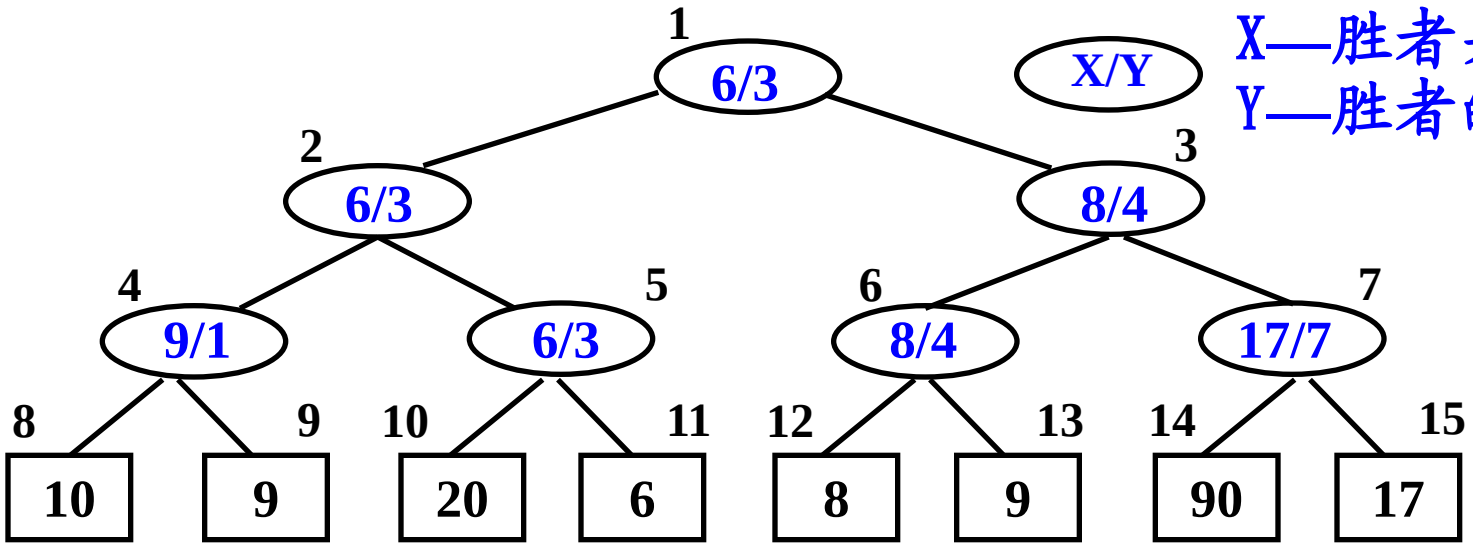
- 选择树就是能够记载上一次比较获得的知识的完全二叉树，有胜者树和败者树两种。



3.4 选择树（Selection Tree）

三、胜者树(Winner Trees)

- 胜者树

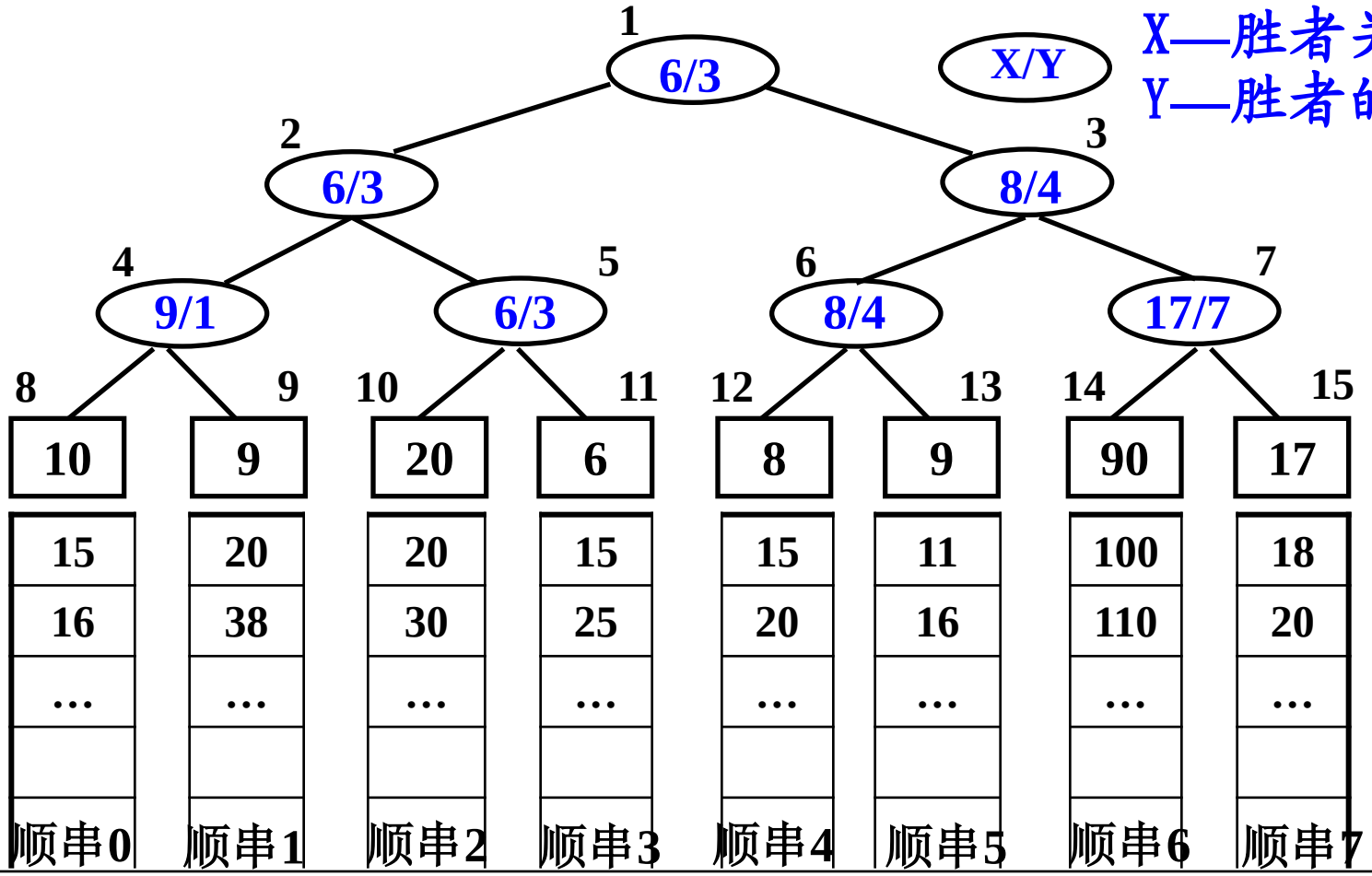


- 具有n个外结点和n−1个内结点
- 外结点为比赛选手，内结点为一次比赛，每一层为一轮比赛
- 比赛在兄弟结点间进行，胜者保存到父结点中
- 根结点保存最终的胜者

3.4 选择树（Selection Tree）

三、胜者树(Winner Trees)

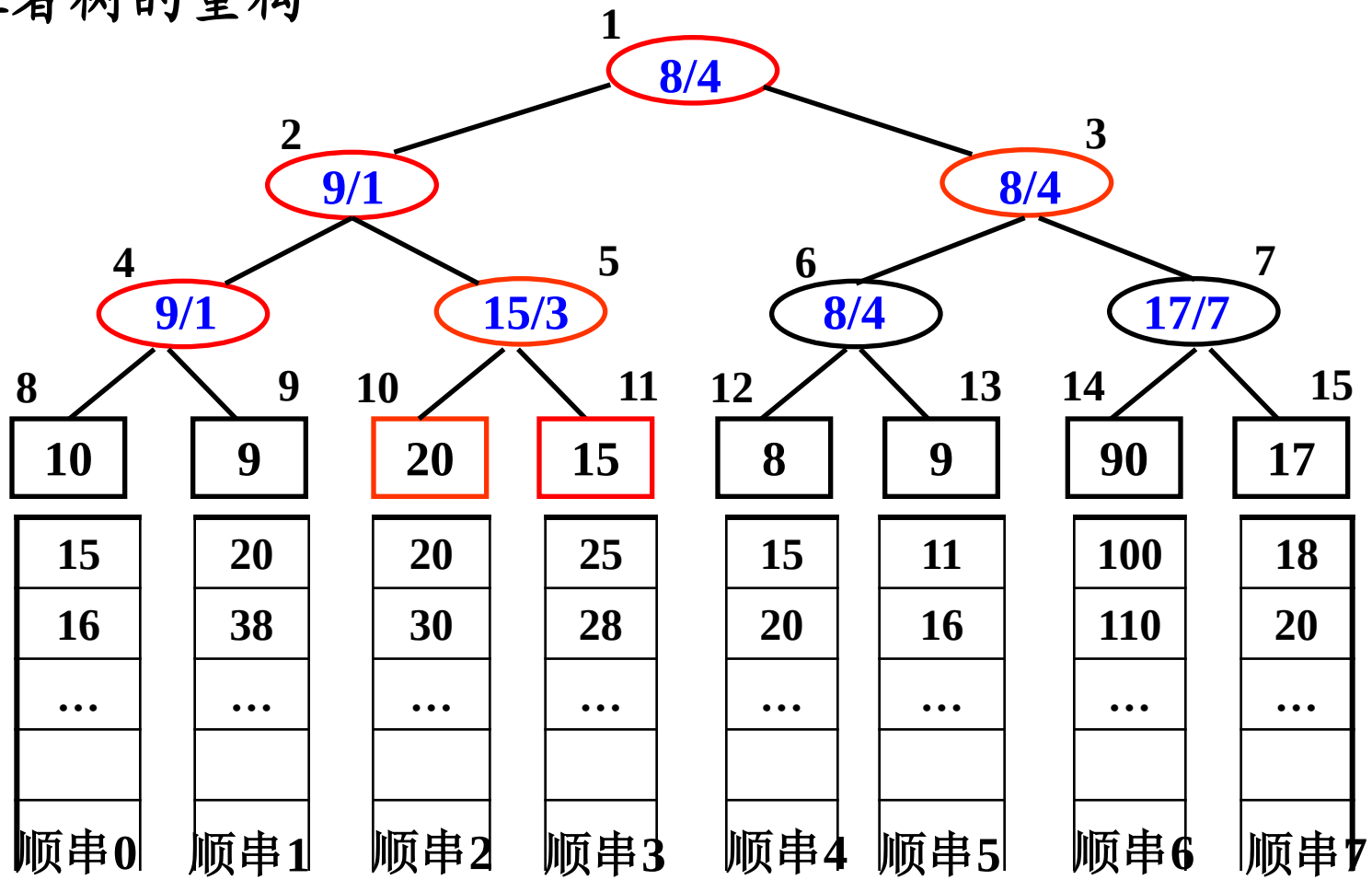
- 胜者树的构建



3.4 选择树（Selection Tree）

二、胜者树(Winner Trees)

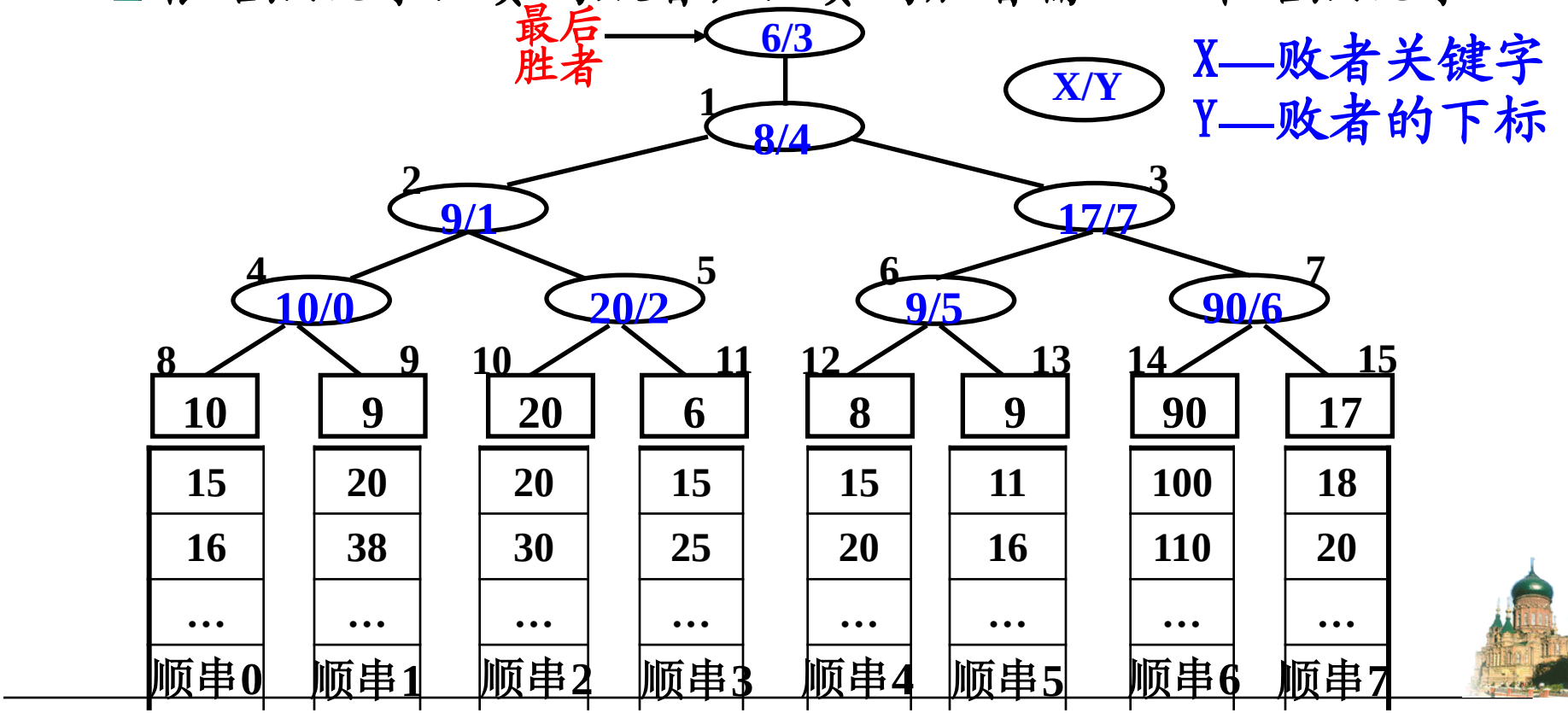
- 胜者树的重构



3.4 选择树（Selection Tree）

四、败者树(Loser Trees)

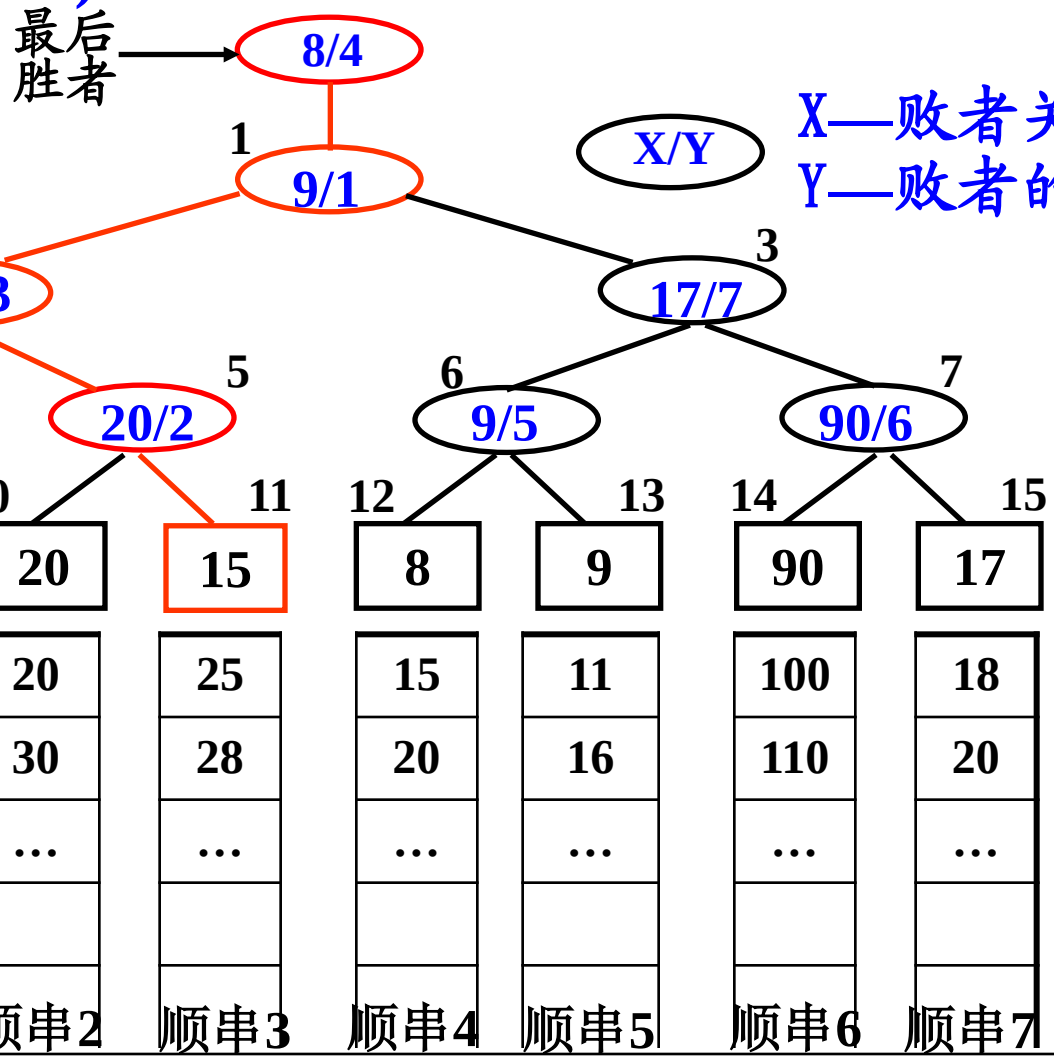
- 败者树的构建
  - 内部结点保存败者，胜者参加下一轮比赛
  - 根结点记录比赛的败者，比赛的胜者需加一个结点记录



3.4 选择树（Selection Tree）

四、败者树(Loser Trees)

- 败者树的重构



3.4 选择树（Selection Tree）

五、应用(Application)

- 锦标赛排序
- 外部归并排序
- ……