上机作业（4+1）

一行用空格分隔的整数，以层序遍历表示树的结构。非叶结点用0表示，叶结点给出其权重（大于0），空结点用-1表示。
你可以认为输入总是合法的：根节点不为空；空结点的左右子树也一定为空（输入为对应的完全二叉树）。
前4道题目必做 （布尔表达式、二叉树带权路径、快排（稳定）、平衡二叉树）
第5题（最短路径）选做（2分作为平时选做分）
平时分30分封顶

思路：如何创建一颗二叉树？

数据结构

教师：姜丽红 jianglh@sjtu.edu.cn

IST 实验室 http://ist.sjtu.edu.cn

助教：芮召普 ruishaopu@qq.com 江嘉晋 IST实验室 软件大楼5号楼5308

《软件基础实践》教师：杜东 IPADS 实验室 软件大楼3号楼4层

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

第7章 优先级队列 priority queue

- **基本的优先级队列**
- **二叉堆**
- **排队系统的模拟**

本章学习目标
1.优先级队列设计与实现。对不同的实现方式进行效率对比分析。
2. 优先级队列应用。

优先级队列

- **结点之间的关系是由结点的优先级决定的，而不是由入队的先后次序决定。**
- **有两种简单方法可以应对优先级队列情形**
  - **方式一：入队时，按照优先级数值在队列（线性表）中寻找合适的位置，将新入队的元素插入在此位置。出队操作的实现保持不变。**
  - **方式二：入队时将新入队的元素直接放在队尾。但出队时，在整个队列中查找优先级最高的元素，让它出队。**

时间复杂度各是多少？

二叉堆

堆是一棵完全二叉树，且满足下述关系：

$k_i \le k_{2i}$ 且 $k_i \le k_{2i+1}$ $(i=1,2,\ldots,\lfloor n/2 \rfloor)$

或者：

$k_i \ge k_{2i}$ 且 $k_i \ge k_{2i+1}$ $(i=1,2,\ldots,\lfloor n/2 \rfloor)$

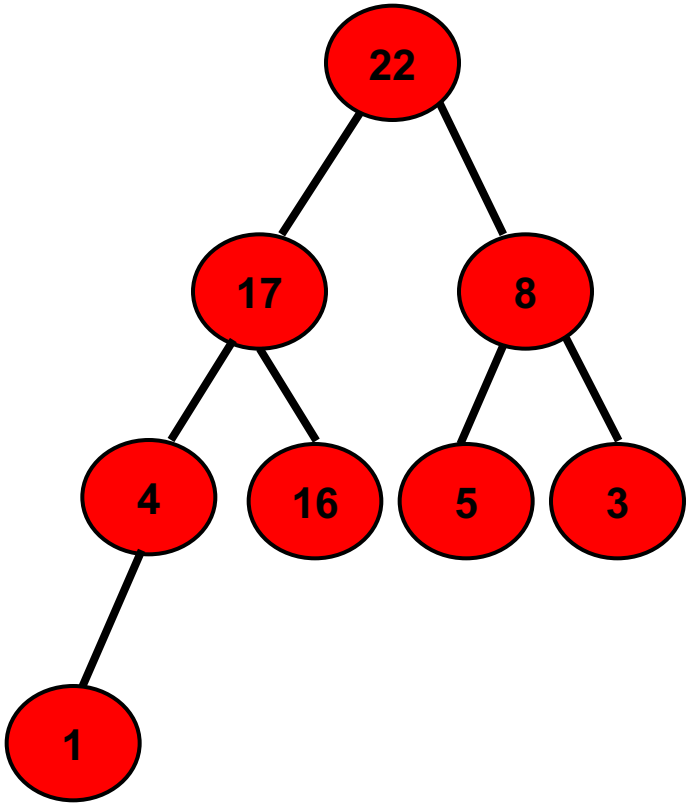
~~其中，下标是树按层次遍历的次序~~

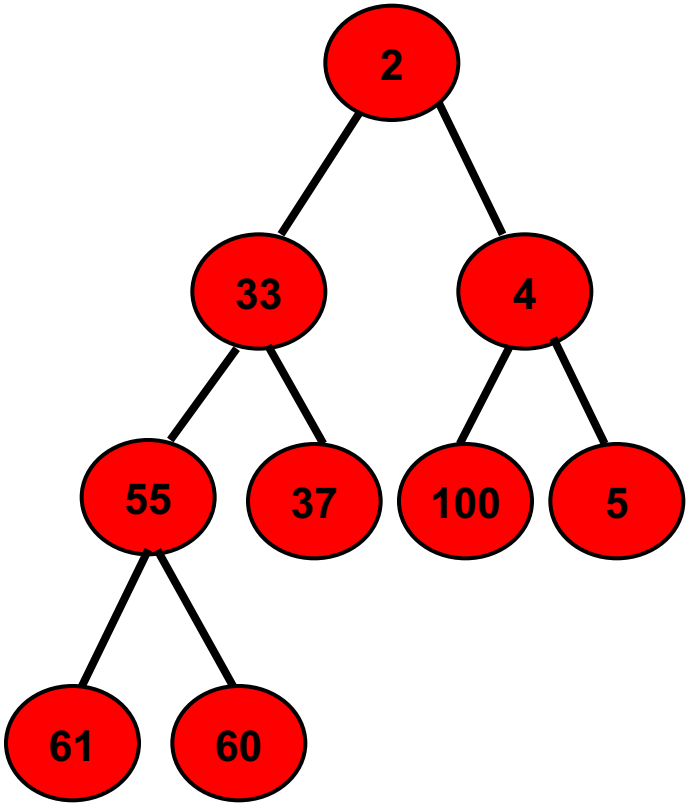
最大化堆/大顶堆或最小化堆/小顶堆

例如:序列 { 2,3,4,5,7,10,23,29,60 } 是最小化堆

序列 { 12,7,8,4,6,5,3,1} 是最大化堆

教材默认最小化堆





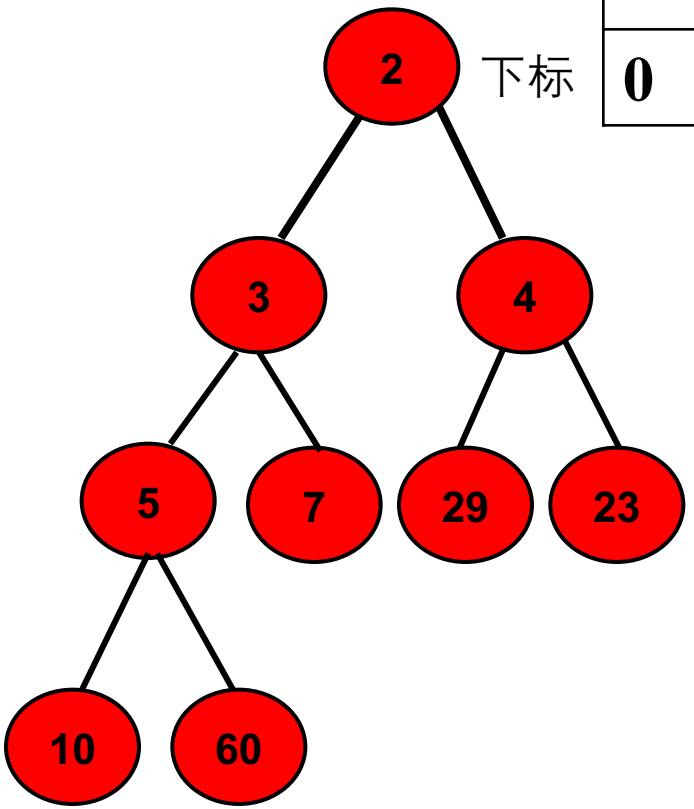
最大化堆

最小化堆

二叉堆的存储

- 采用顺序存储(因为定义上是完全二叉树）逻辑视角
- 二叉堆的有序性可以很容易地通过下标来反映

元素值		2	3	4	5	7	29	23	10	60
下标	0	1	2	3	4	5	6	7	8	9



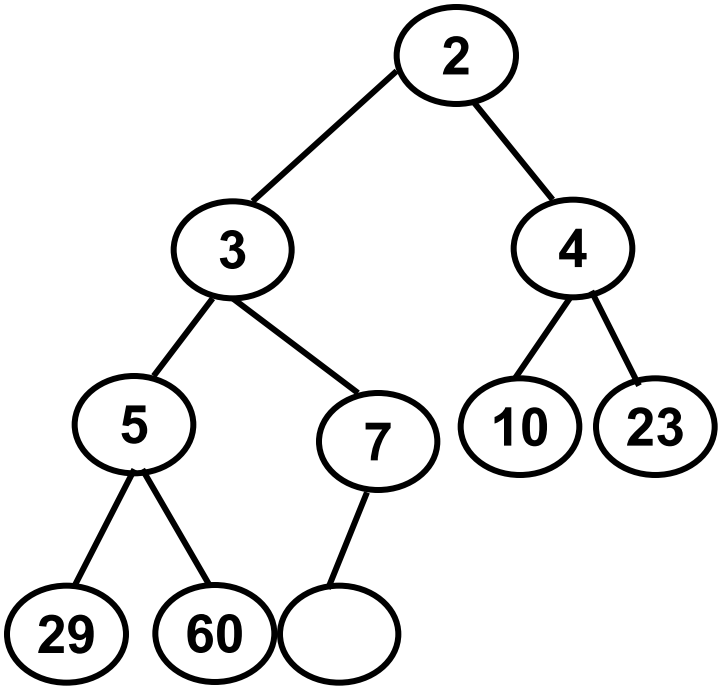
用堆实现优先级队列

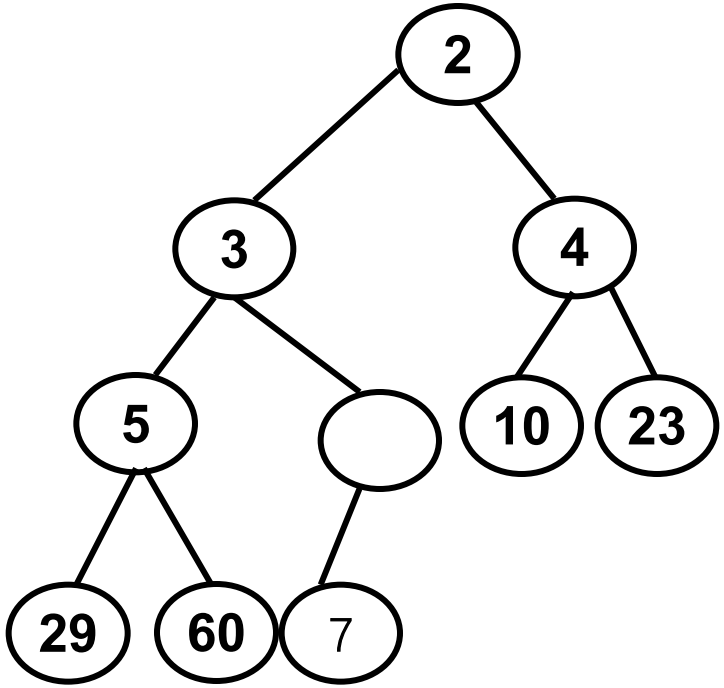
```
template <class Type>
class priorityQueue:public queue<Type>
{private:
    int currentSize;
    Type *array;
    int maxSize;  //同顺序表定义

    void doubleSpace();
    void buildHeap( );//建堆
    void percolateDown( int hole );  //堆调整
```

用堆实现优先级队列

```
public:
    priorityQueue( int capacity = 100 )
     { array = new Type[capacity];
         maxSize = capacity;
         currentSize = 0;}//顺序表初始化
    priorityQueue( const Type data[], int size );//建堆过程
   ~priorityQueue() { delete [] array; }
    bool isEmpty( ) const { return currentSize == 0; }
    void enQueue( const Type & x );//入队
    Type deQueue();
    Type getHead() { return array[1]; }//array[0]做了哨兵
};
```

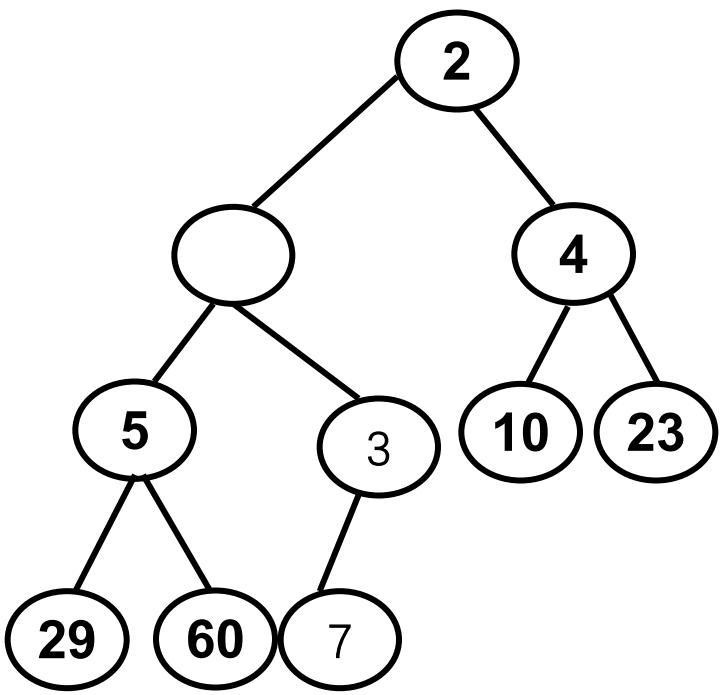
enQueue过程//以小顶堆为例

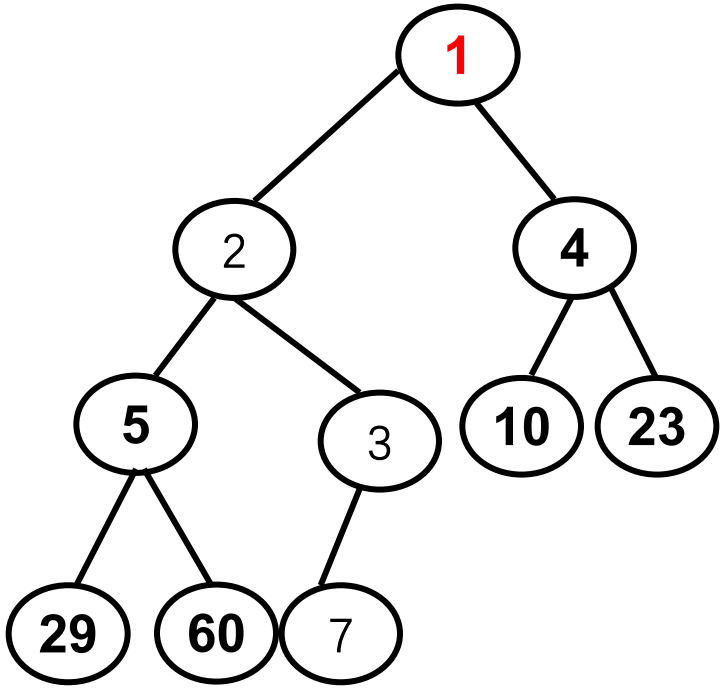




以在小顶堆中插入元素1为例

enQueue过程//以小顶堆为例





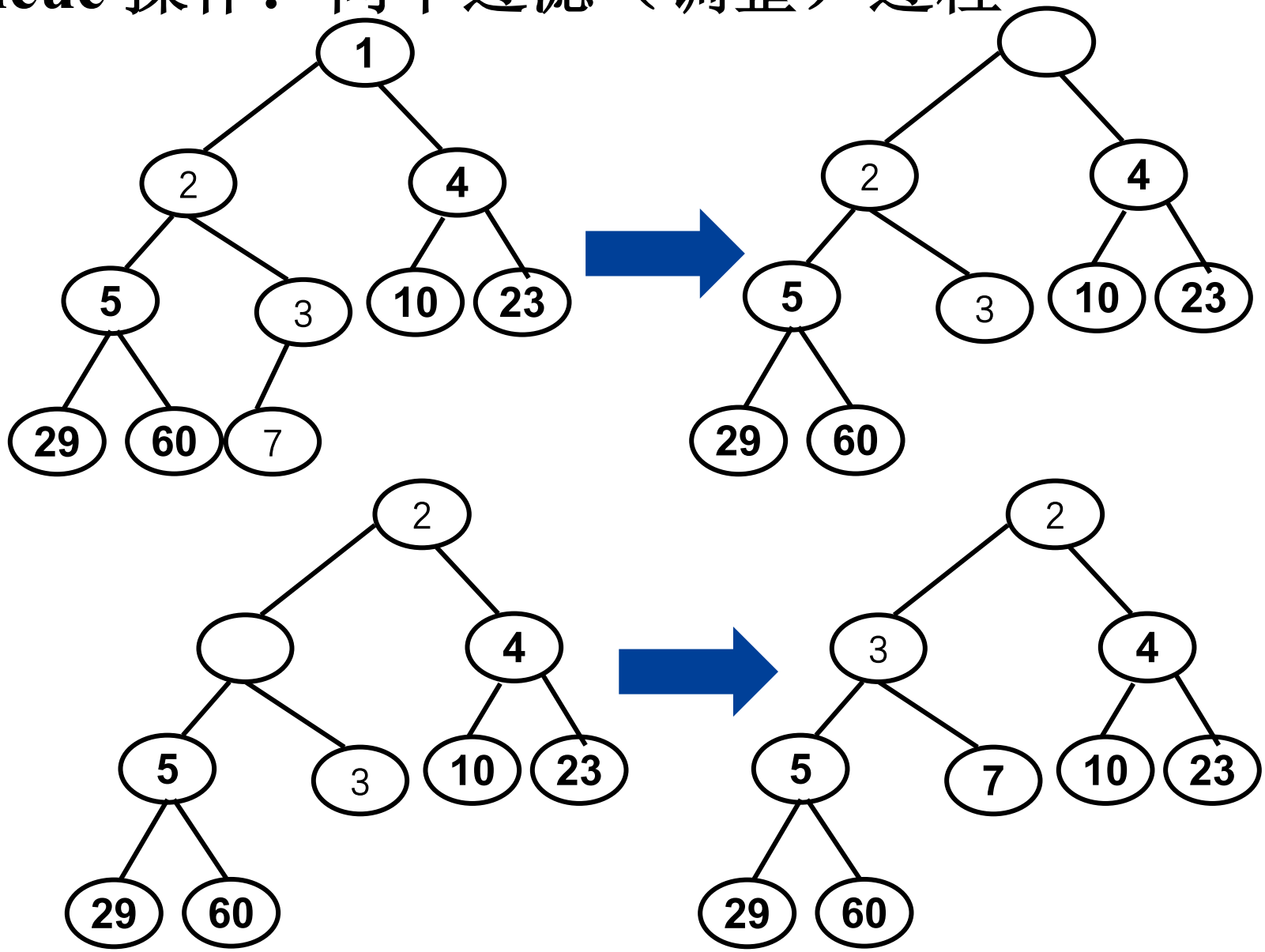
以在小顶堆中插入元素1为例

enQueue过程//以小顶堆为例

```
template <class Type>
void priorityQueue<Type>::enQueue( const Type & x )
{  if( currentSize == maxSize - 1 )  doubleSpace();

  // 向上过滤（调整）
  int hole = ++currentSize;
  for( ; hole > 1 && x < array[ hole / 2 ]; hole /= 2 )//注意边界及下标的含义
          array[ hole ] = array[ hole / 2 ];
  array[ hole ] = x;
}
```

（不考虑堆满扩容）最坏情况下一次完整调整过程的时间复杂度是O(logn)：为什么？？？

DeQueue 操作：向下过滤（调整）过程



deQueue()

```
template <class Type>
Type priorityQueue<Type>::deQueue()
{ Type minItem;
  minItem = array[ 1 ];//堆顶下标为1或者0，可自行定义
  array[ 1 ] = array[ currentSize-- ];
  percolateDown( 1 );
  return minItem;
}
```

时间复杂度：O(logn)

向下过滤（调整）：O(logn) （小顶堆为例）

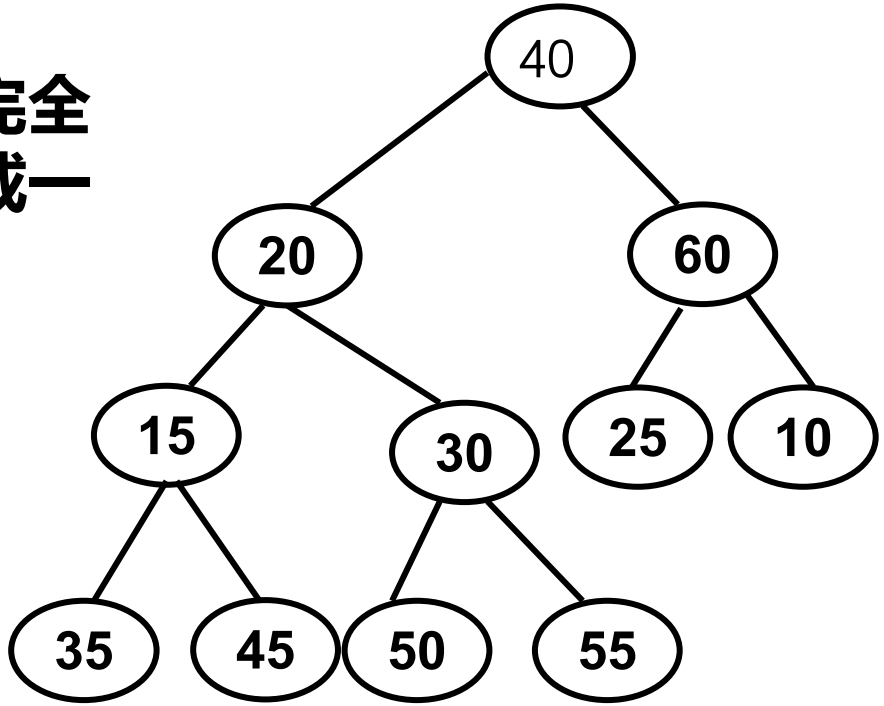
```
template <class Type>
void priorityQueue<Type>::percolateDown( int hole )
{  int child;
   Type tmp = array[ hole ];
   for( ; hole * 2 <= currentSize; hole = child )
      {  child = hole * 2;
         if( child != currentSize && array[ child + 1 ] < array[ child ] )
               child++;
         if( array[ child ] < tmp )   array[ hole ] = array[ child ];
               else    break;
      }
   array[ hole ] = tmp;
}
```

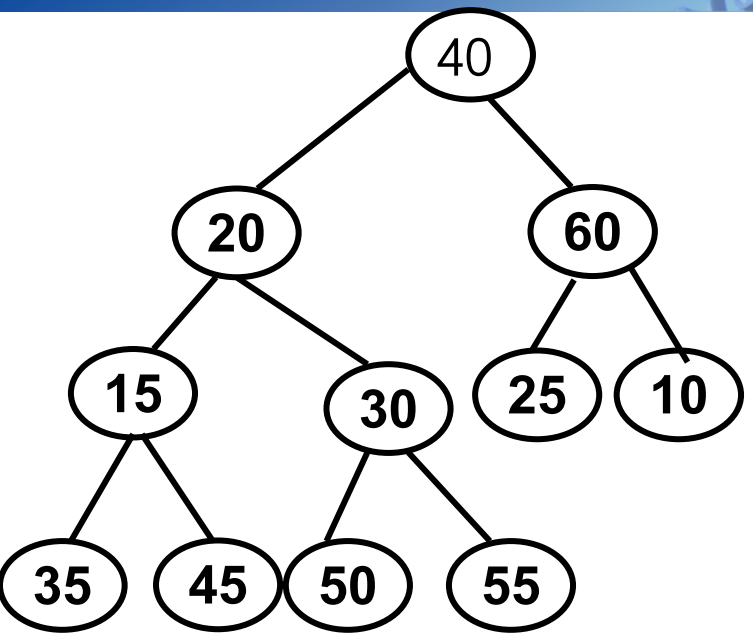
建堆

- 可以采取N次连续插入的方式，但是，其时间复杂度是O(NlogN)//拓展思考：你可以证明吗？。因此，不采纳。
- 事实上，构造堆的时间复杂度为O(N)

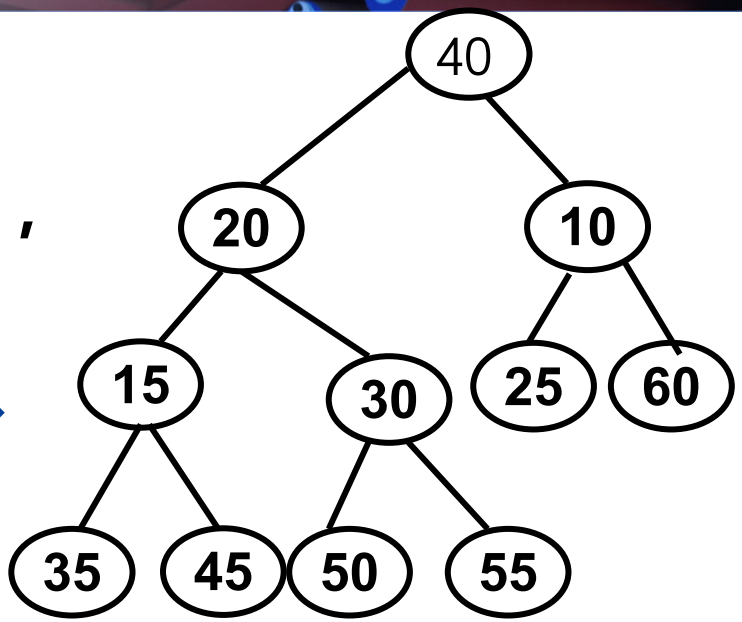
例如，给出的数据初值为40，20，60，15，30，25，10，35，45，50，55，构造一个最小化堆

首先，将它看成是一棵完全二叉树，然后把它调整成一个堆。

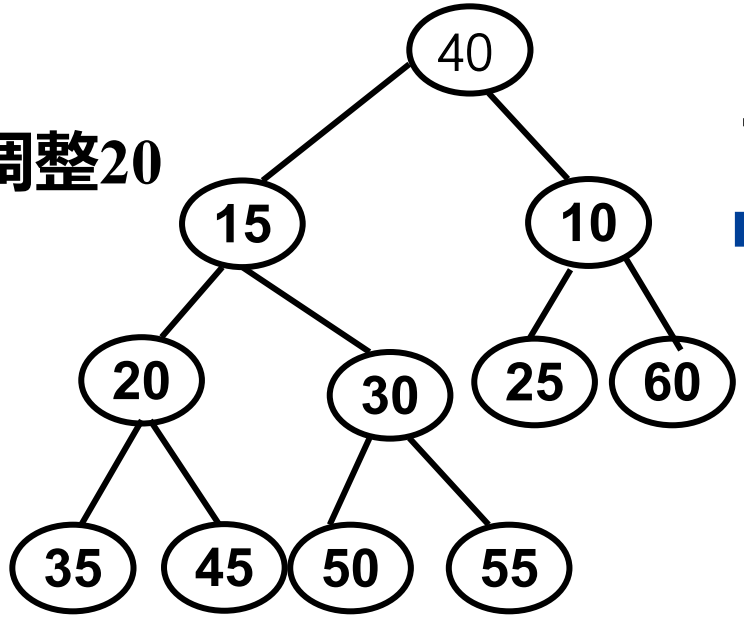




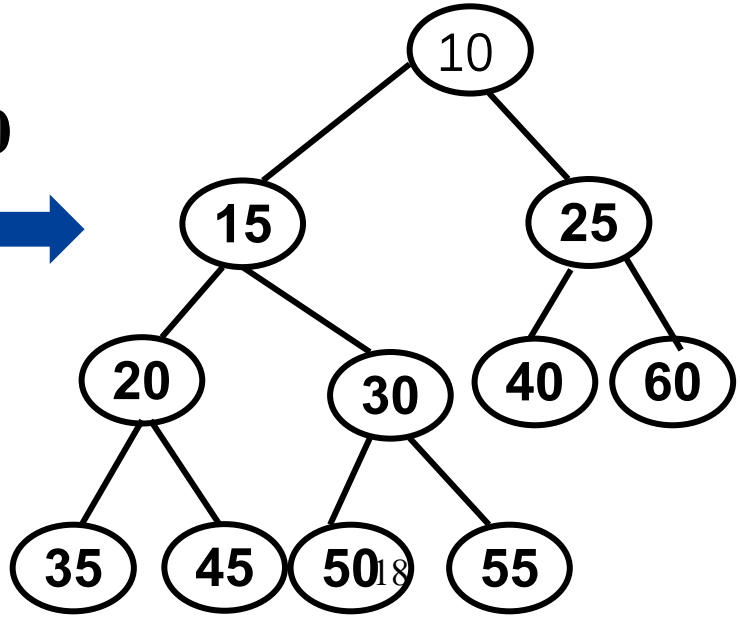
30和15没有违反堆定义，接着调整60



调整20



调整40



建堆

percolateDown(int hole)部分

```
Type tmp = array[ hole ];
  for( ; hole * 2 <= currentSize; hole = child )
    {  child = hole * 2;
       if( child != currentSize && array[ child + 1 ] < array[ child ] )
            child++;
       if( array[ child ] < tmp )   array[ hole ] = array[ child ];
            else    break;
    }
  array[ hole ] = tmp;
```

建堆的时间代价分析

- **建堆的时间复杂度是O(N)**

- **证明：高度为h的结点有一个，高度为h-1的结点有2个，高度为h-2的结点有2^2个，高度为h-i的节点有2^i个。第h层向下调整的最大层数为h-1。因此，调整层数最大和为：**

$$s=\sum_{i=0}^{h-1}2^i(h-i-1)$$

$$=2^0(h-1)+2^1(h-2)+2^2(h-3)+2^3(h-4)+\ldots 2^{h-2}(1)$$

$$2s=2^1(h-1)+2^2(h-2)+2^3(h-3)+2^4(h-4)+\ldots 2^{h-1}(1)$$

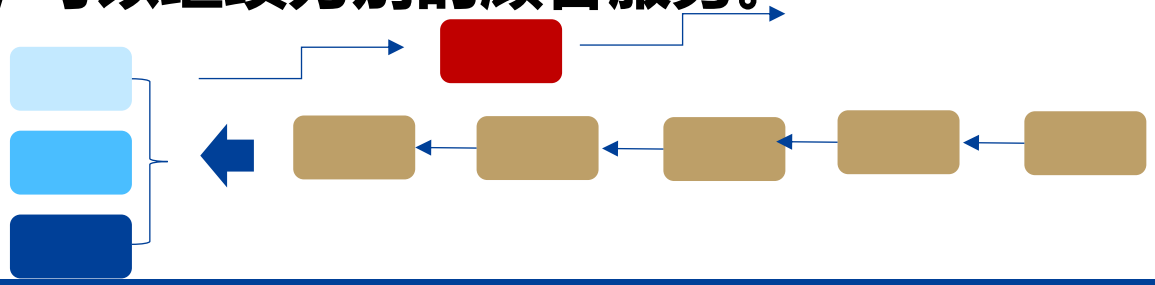
$$2s-s=-2^0(h-1)+[2^1+2^2+2^3+2^4+\ldots+2^{h-1}]$$

$$=-h+[1+2^1+2^2+2^3+2^4+\ldots+2^{h-1}]$$

$$=(2^h-1)-h=N-h$$

多服务台的排队系统模拟过程

- 模拟开始时，产生所有的到达事件，存入优先级队列。
- 模拟器开始处理事件：
  - 从队列中取出一个事件。这是第一个顾客的到达事件。生成所需的服务时间。当前时间加上这个服务时间就是这个顾客的离开时间。生成一个在这个时候离开的事件，插入到事件队列。
  - 从队列中取出的事件也可能是离开事件，这时只要将这个离开事件从队列中删去，为他服务的服务台变成了空闲状态，可以继续为别的顾客服务。



```
While （事件队列非空）{
    队头元素出队；    设置当前时间为该事件发生的时间；
   switch(事件类型）
     { case 到达：
            if (柜台有空)
              { 柜台数减1； 生成所需的服务时间 ；
       修改事件类型为“离开”；
       设置事件发生时间为当前时间加上服务时间；
       重新存入事件队列；}
            else 将该事件存入等待队列；
      case 离开：
            if （等待队列非空）
       { 队头元素出队； 统计该顾客的等待时间；
        生成所需的服务时间 ； 修改事件类型为“离开”
        设置事件发生时间为当前时间加上服务时间；
        存入事件队列；     }
            else 空闲柜台数加1；
     }
 }
计算平均等待时间； 返回；
}
```

avgWaitTime()

```
int simulator::avgWaitTime()
{ int serverBusy = 0;  // 正在工作的服务台数，初始化为0
  int currentTime ;    // 记录模拟过程中的时间
  int totalWaitTime = 0;
            //模拟过程中所有顾客的等待时间的总和
  linkQueue<eventT> waitQueue;  //顾客等待队列－普通队列
  priorityQueue<eventT> eventQueue;  //事件队列－优先级队列
  eventT currentEvent;
```

处理到达事件

```
if  (serverBusy != noOfServer)
      { ++serverBusy;//noOfServer为服务台的个数
        currentEvent.time += serviceTimeLow +  (serviceTimeHigh -
serviceTimeLow +1) * rand() / (RAND_MAX + 1);
        currentEvent.type = 1;
        eventQueue.enQueue(currentEvent);
            }
  else waitQueue.enQueue(currentEvent);
```

处理离开事件

```
if (!waitQueue.isEmpty())
 { currentEvent = waitQueue.deQueue();//从等待队列里出队一位顾客
        totalWaitTime += currentTime - currentEvent.time;
        currentEvent.time = currentTime + serviceTimeLow
              + (serviceTimeHigh - serviceTimeLow +1) *
               rand() / (RAND_MAX + 1);
       currentEvent.type = 1;
       eventQueue.enQueue(currentEvent);
     }
  else --serverBusy;//空闲服务柜台数加1/忙碌服务台减1
```

总结

- **本章介绍了一种优先级队列的实现方法。**
- **介绍了多服务台的排队系统的模拟。**
- **概念分辩：用非线性数据结构（堆）实现线性数据结构（队列），堆的存储结构为顺序实现方式。**

本章学习目标

1.可以设计实现高效率的优先队列。利用二叉堆创建O(n),插入和删除O(logn)；利用线性表插入或删除O(n)。理由分析？

2.可以利用优先队列进行多服务台模拟系统建模。

作业（交作业时间4月17日）

1、从最大化堆（30，26，13，17，11，8，7，10，3，4）中删除最大值后，得到的堆是什么？

2、画出在一个空堆中，依次插入8,6,4,3,2，生成的最小化堆的结果。

3、给定一棵顺序存储的完全二叉树，设计算法判断它是否为最小化堆。

教材P233练习7（推荐练习，不用交）

- 简答题2、3、4；程序设计题7
- 设计算法合并两个高度相同的堆。
- 以上均为推荐练习的题目，可查阅习题集寻找答案。

有下面键值的元素在大顶堆（最大化堆）的什么位置？

(a)第二大键值；

(b)第三大键值；

(c)最小键值

设计算法，在最小化堆中查找最大键值元素，给出算法时间复杂度分析。

Thanks! & QA