HEAP

一种有效实现PQ的一种数据结构

这个数据结构允许我们insertion 或removal 用Log时间，对于基于list的 PQ来说是一种很大的进步

从根本上说，通过放弃在list中存储entry的想法，堆实现了这种改进;相反，它将entry存储在二叉树中。

换句话说，heap就是二叉树存entry作为Node

Heap的几个性质：

对于除了root的internal node v，他的key一定大于等于parent

Complete binary tree，

假设二叉树深度为h， 从0到h-1层都是满的每层都是2^I node

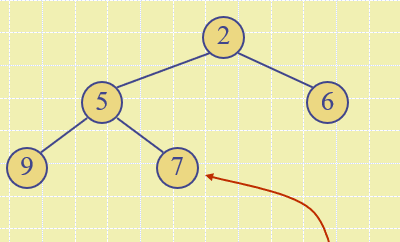
但是在第h层，所有node尽量往左

COMPLETE BINARY TREE的性质

所有的leaf集中在第h层与第h-1层

对于任意Node,如果右子树的最大层次为L，左边最大层次为L OR L+1

最多只有一个Node有一个CHILD,而且这个child必是左child



Heap-order property

对于每一个internal node，key(v)>=key(parent(v))

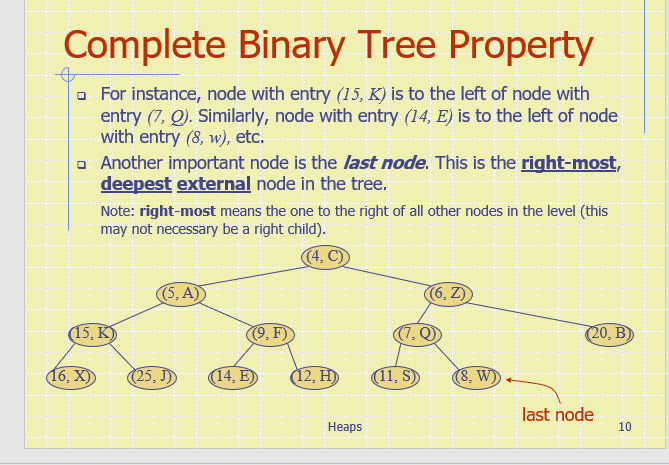
这条性质导致的结果就是：从root到external node，一路的Key越来越大

另外，最小的key永远被存储在root中

Complete Binary Tree property

这是一个结构性质，我们要尽量确保heap的height越小越好

在complete binary tree中，如果Node v在w的左侧并且他们同一高度，那么就先visit v再visit w，也就是inorder traversal



比如15K再7Q的左边，

Last node:最后的Node<最深最右的Node，不一定是right child

Height of a heap:

* 一个heap存储n 个key 的height 是***O***(log ***n***)

从0到H-1层，有2i，最后一层至少有一个

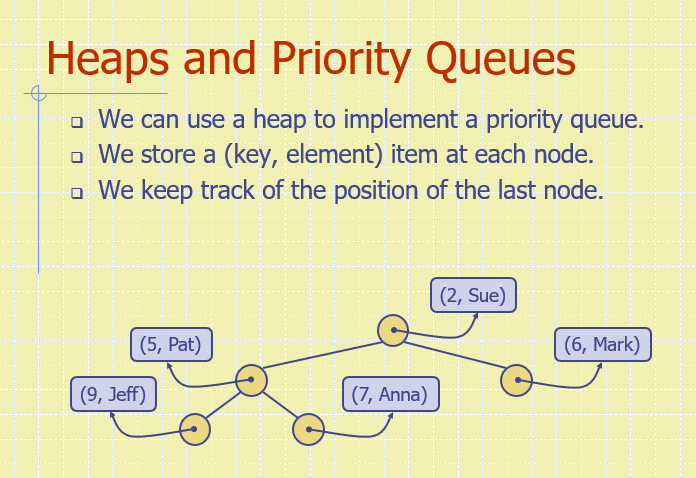




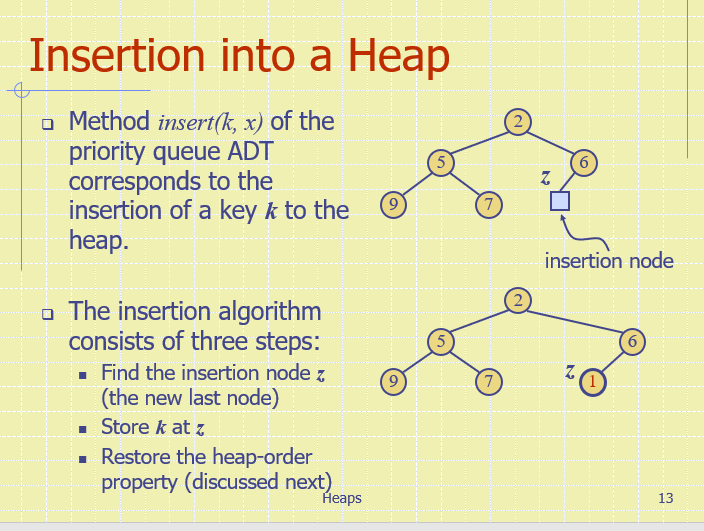
我们可以用heap来构建一个PQ

我们在每个node存储一个(key,element)

我们track最后一个node的位置



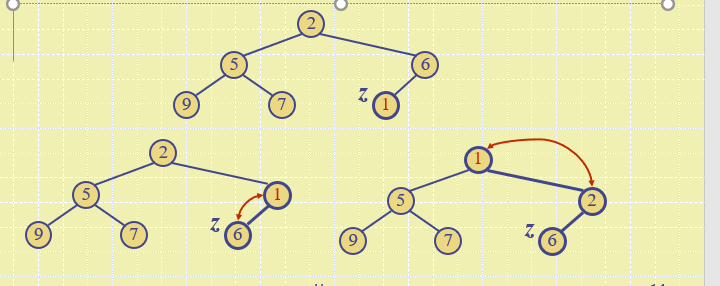
heap中的Insertion

1. 找到要插入的Node//新last node
2. 把key k存在z里
3. 重新重置heap-order property
4. 

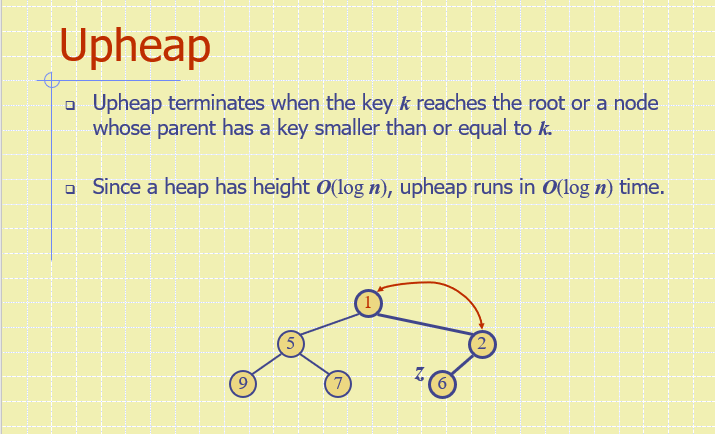
Upheap增加一个节点

当Insertion一个new key k后，heap-order property有可能被侵犯

算法 unheap通过交换k沿着向上的路径来恢复heap-order property



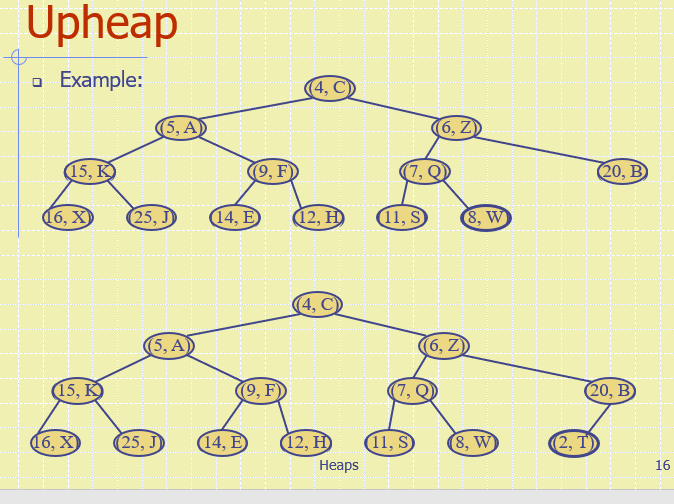
heap-order property就是保证越往上越小



UNHEAP一直会重复直到k到了root或到了一个node小于等于他的地方

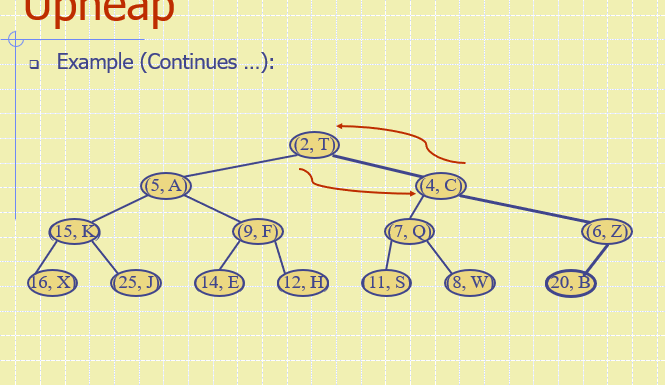
heap的height是LOGN所以Unheap就是logn

例子



加了一个2T，

2T和20B换，和6Z换，和4C换



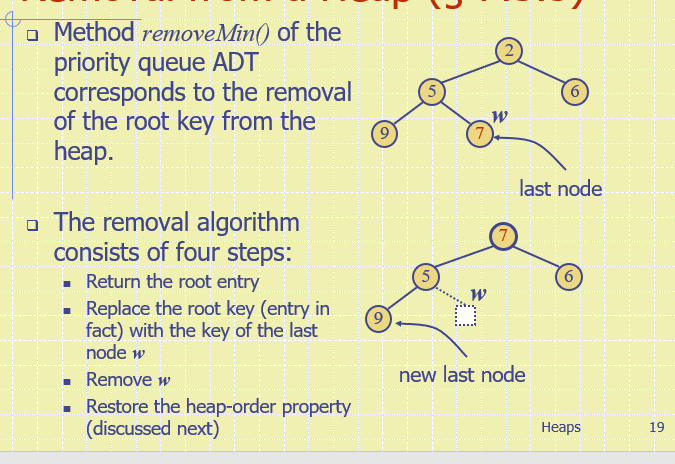
Removal

return root entry

把root key 替换成last node key

移除last node

重置heap-order property

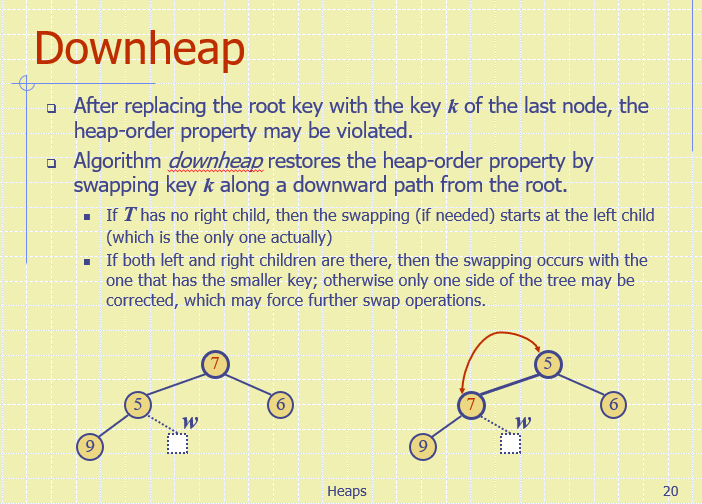


Downheap

downheap通过把k从root开始一直往下换来恢复heap-order property

如果T没有right child，就和left child换

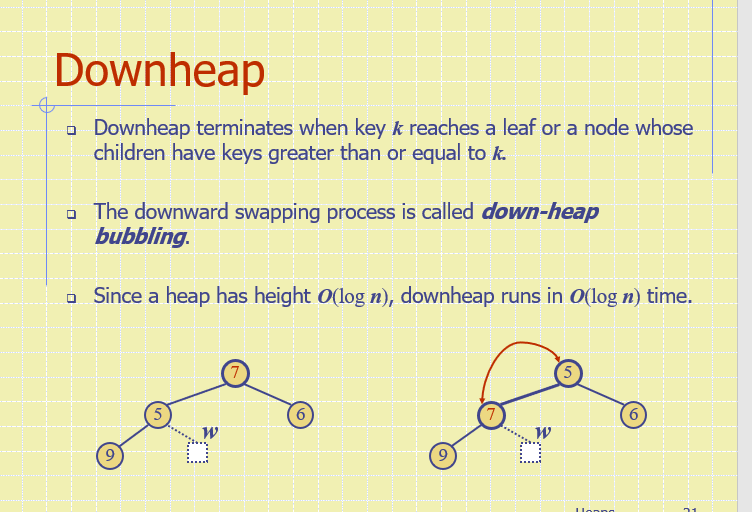
如果都有，就与key较小的换

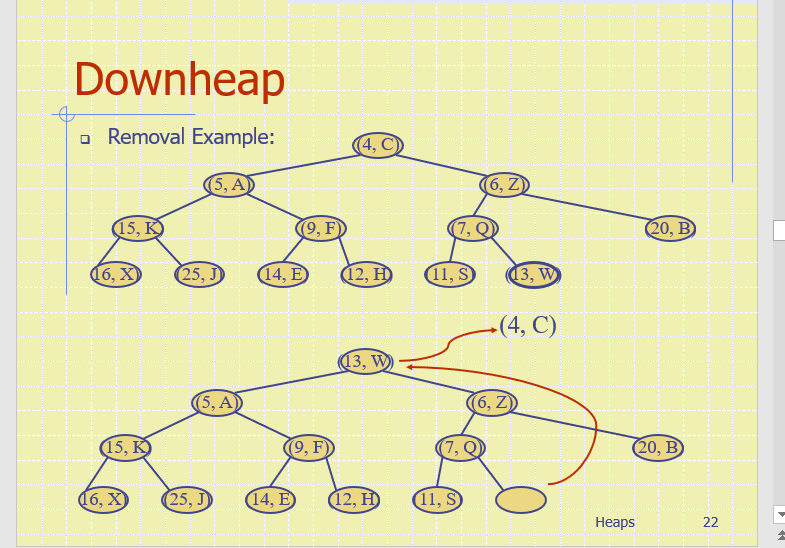


downheap终止直到他到了leaf或children node大于等于他

downward swapping process叫做 down-heap bubbling

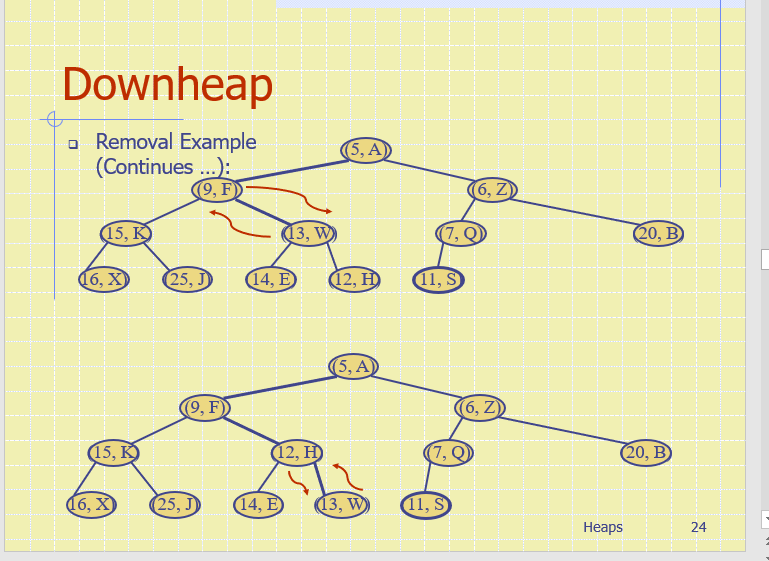
用时logn





第一步removal，交换

然后13和5换，13和9换，13和12换



update the last node

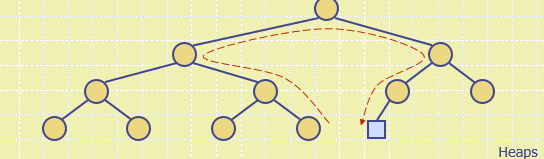
怎么更新last node

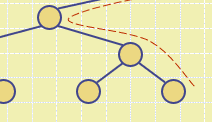
我们需要创造一个新last node以供使用

我们可以通过遍历O logn个nodes找到insertion node

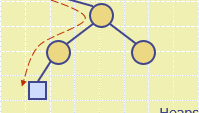
例如，现在last node是left child那么就直接到father然后右下创造新lastnod

如果现在lastnode是right child，一直往上直到遇到一个left child或者root，如果遇到了left child就去往right child，一直往左往下直到遇到leaf，

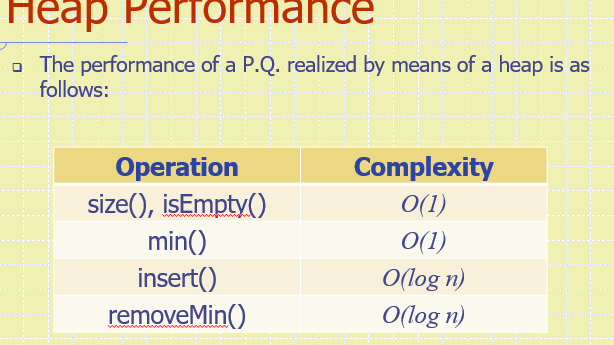


一直往上遇到left node，

去往right nnode

往左往下

HEAP PERFORMANCE

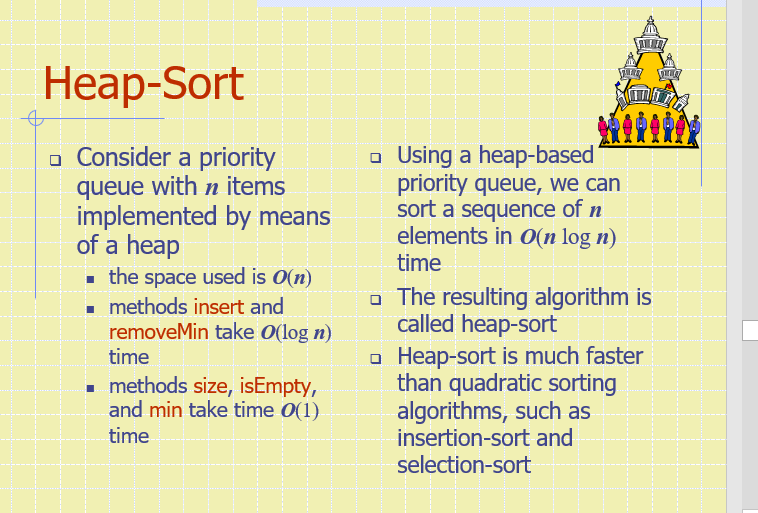


在基于heap的PQ中，每个Insertion用时为O（logn）每个removal用时O（logn）

作为结果总用时为O（nlogn）因为第一个Loop有N步，第二个Loop n removal也要O（nlogn）

总算法为O（Nlogn）

这个实现PQ的算法叫做Heap-sort

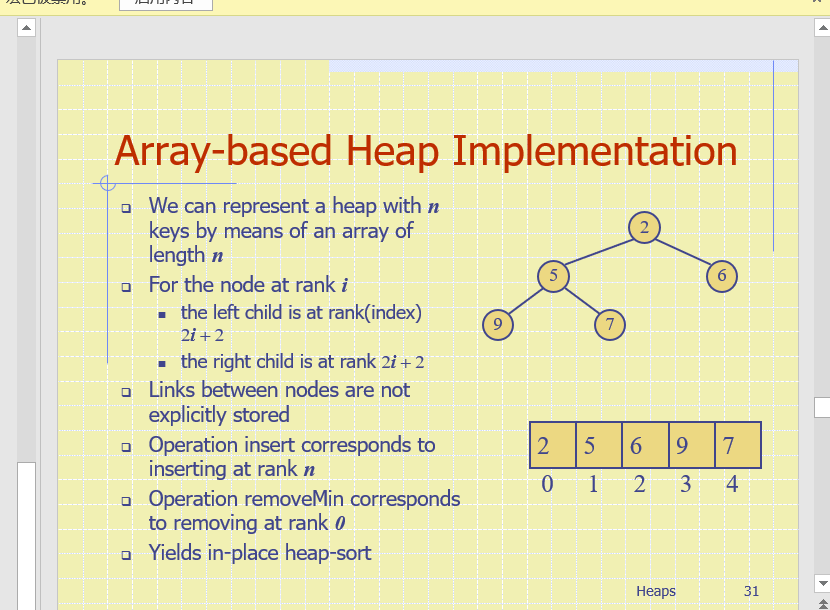


我们可以用array实现heap

left child在2i+1,right child在2i+2

INSERT就相当于INSERT在 INDEX N//创造了一个新格子

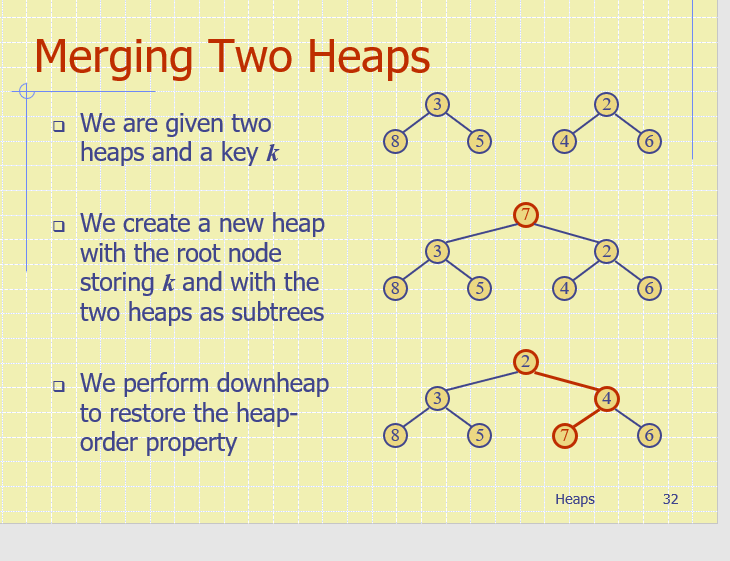
REMOVE就相当于remove rank 0



Merging two heaps合并两个heap

给你两个heap和一个key k

我们把这个key作为root node，然后downheap

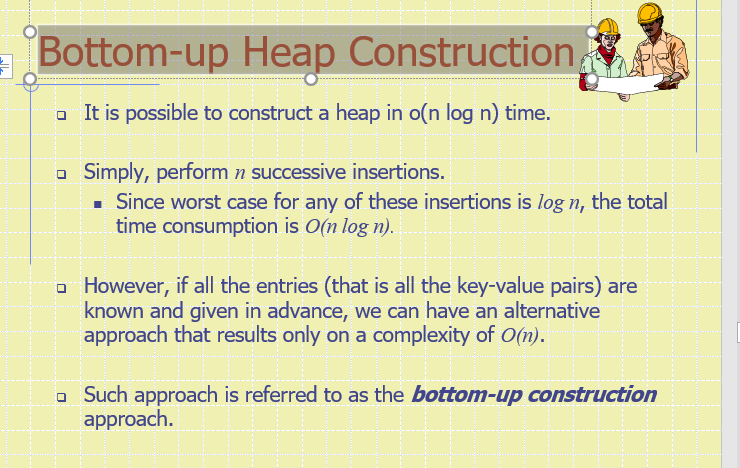


Bottom-up heap construction 从底向上建设heap

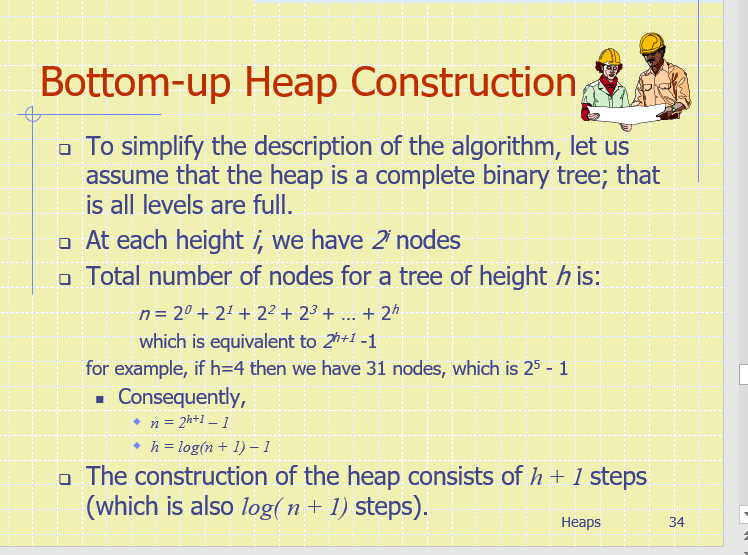
用时nlogn

只要进行n次insertion

如果每一个entry都是提前给好的，我们可以O（n）就构建好，这就是bottom-up heap construction



为了简化，我们创造一个每层都满的binary tree，

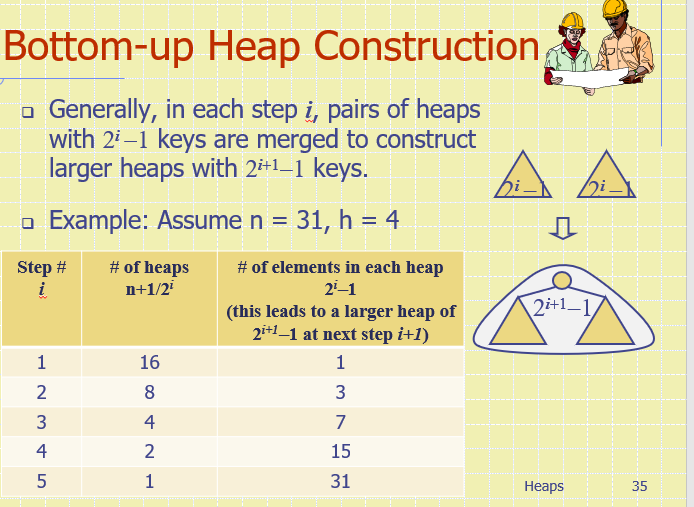


一共要插入的点是n=2^k+1-1

高度是Log(n+1)-1

构建这个heap一共需要h+1 step

在每一个step，我们把他左右子树merge起来

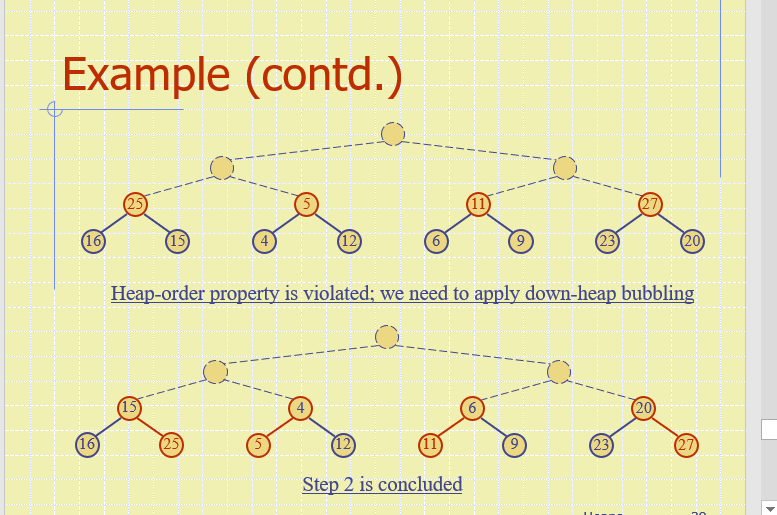


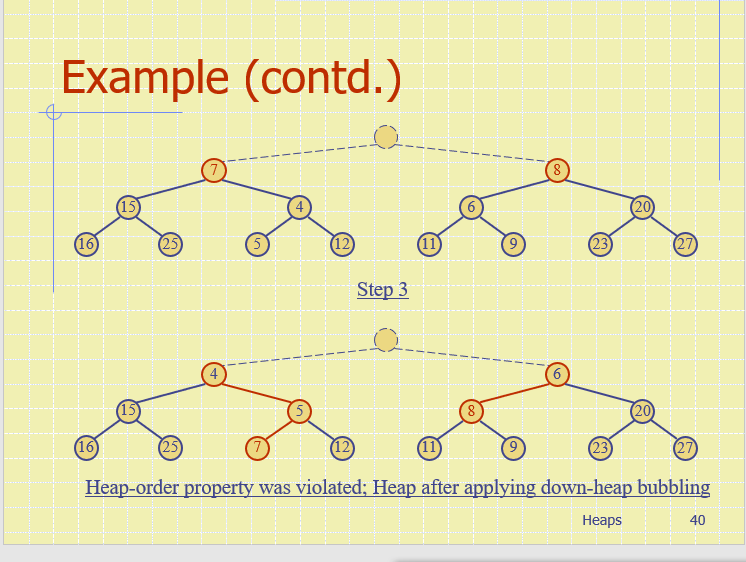
在每个step i我们把拥有2i-1个key的heap merge成2i+1 -1的大heap

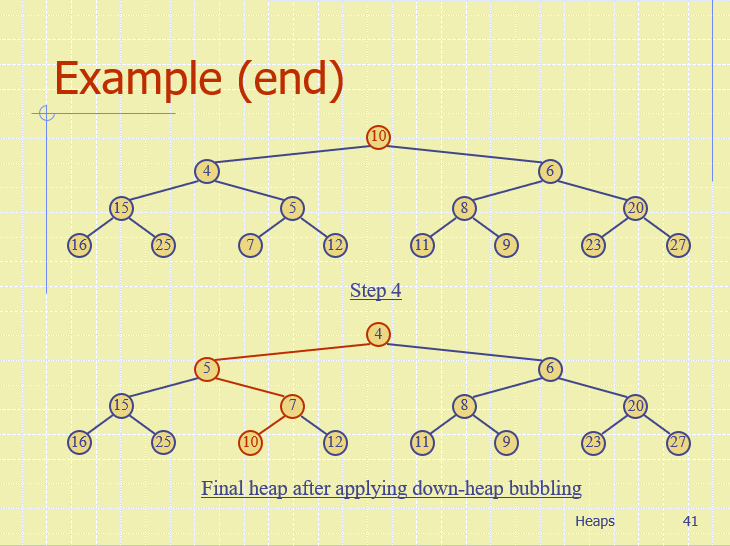
第一步，构建(n+1)/2个基础heap只有一个点

第二步构建(n+1)/4 个Heap，每个存储三个点‘

第h+1：创造(n+1)/2^(h+1)







每一轮都要downheap

总共需要O(n）时间