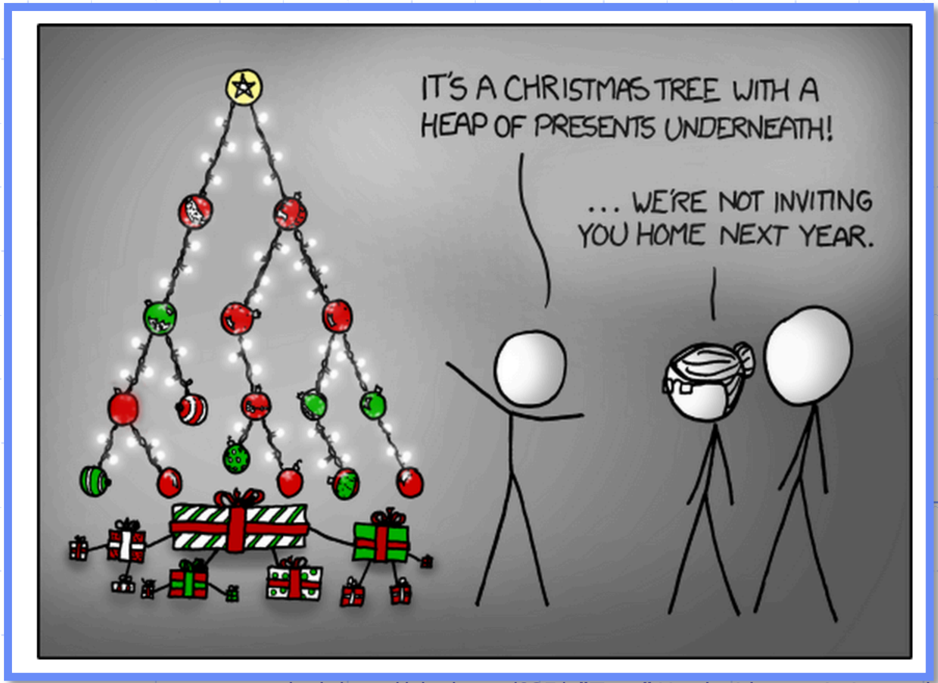
Presentation for use with the textbook Algorithm Design and Applications, by M. T. Goodrich and R. Tamassia, Wiley, 2015

Heaps



Recall Priority Queue Operations

- A priority queue stores a collection of entries
- Each entry is a pair (key, value)
- Main methods of the Priority Queue ADT
  - insert(k, v) inserts an entry with key k and value v
  - removeMin() removes and returns the entry with smallest key
- Additional methods
  - min() returns, but does not remove, an entry with smallest key
  - size(), isEmpty()
- Applications:
  - Standby flyers
  - Auctions
  - Stock market enigines

Recall PQ Sorting

- We use a priority queue
  - Insert the elements with a series of insert operations
  - Remove the elements in sorted order with a series of removeMin operations
- The running time depends on the priority queue implementation:
  - Unsorted sequence gives selection-sort: $O(n^2)$ time
  - Sorted sequence gives insertion-sort: $O(n^2)$ time
- Can we do better?

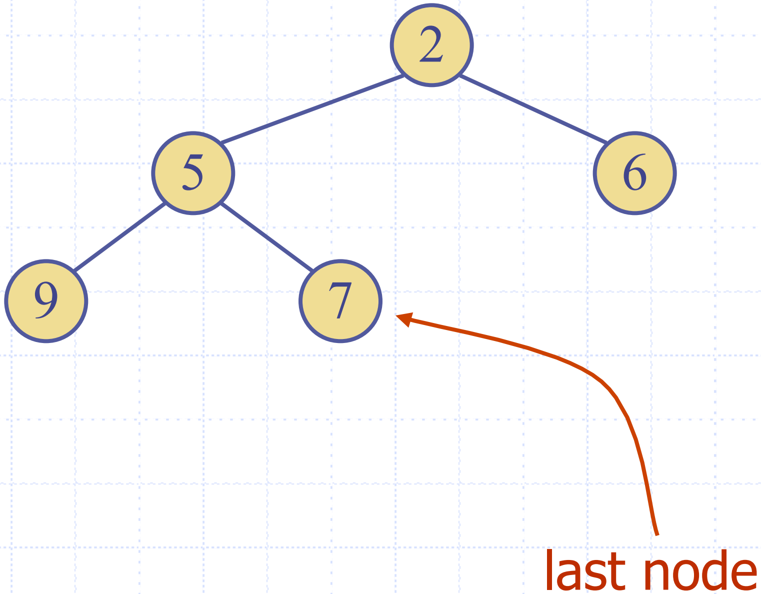
Algorithm PQ-Sort(C, P):
    Input: An n-element array, C, index from 1 to n, and a priority queue P that compares keys, which are elements of C, using a total order relation
    Output: The array C sorted by the total order relation

```
for i ← 1 to n do
    e ← C[i]
    P.insert(e, e)        // the key is the element itself
for i ← 1 to n do
    e ← P.removeMin()     // remove a smallest element from P
    C[i] ← e
```

Heaps

- A heap is a binary tree storing keys at its nodes and satisfying the following properties:
- Heap-Order: for every internal node v other than the root, $key(v) \geq key(parent(v))$
- Complete Binary Tree: let h be the height of the heap
  - for $i = 0, \dots, h-1$, there are 2^i nodes of depth i
  - at depth $h-1$, the internal nodes are to the left of the external nodes

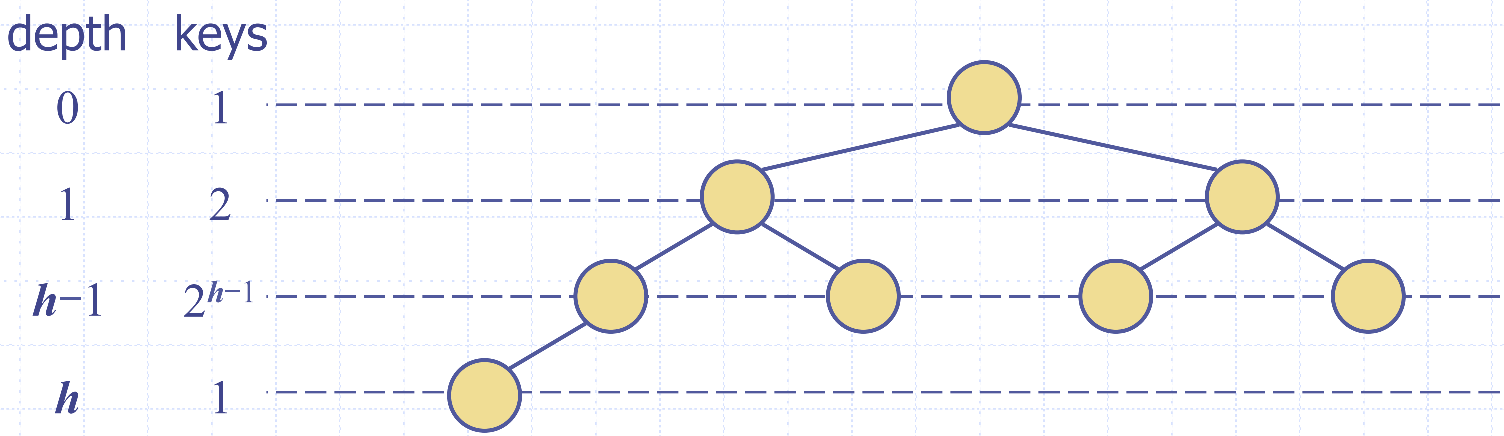
- The last node of a heap is the rightmost node of maximum depth



Height of a Heap

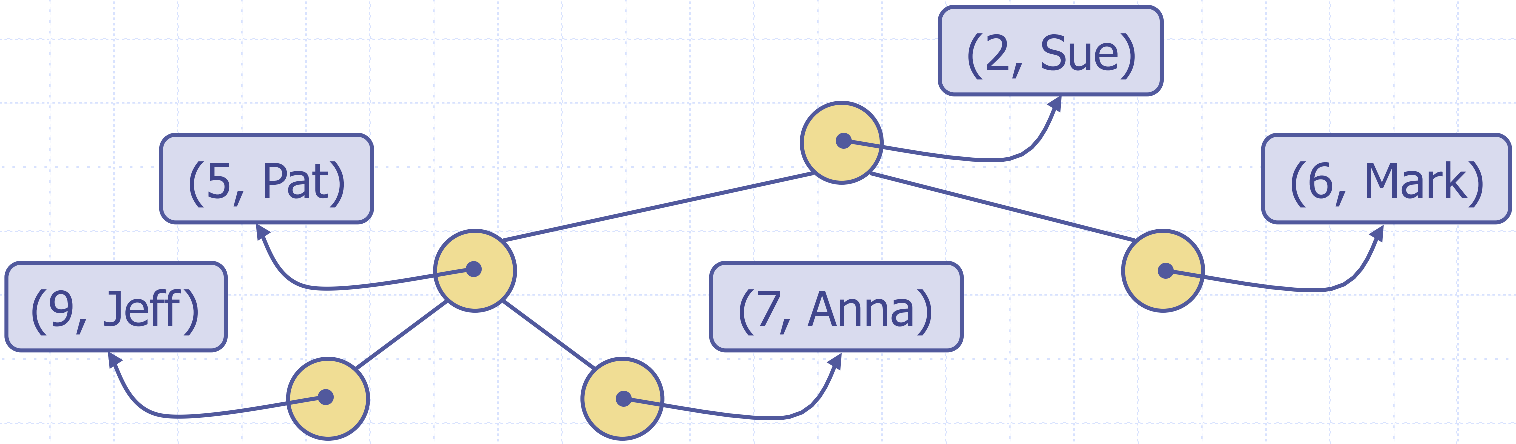
- **Theorem:** A heap storing n keys has height $O(\log n)$

  Proof: (we apply the complete binary tree property)
  - Let h be the height of a heap storing n keys
  - Since there are 2^i keys at depth $i = 0, \ldots, h-1$ and at least one key at depth h, we have $n \geq 1 + 2 + 4 + \ldots + 2^{h-1} + 1$
  - Thus, $n \geq 2^h$, i.e., $h \leq \log n$



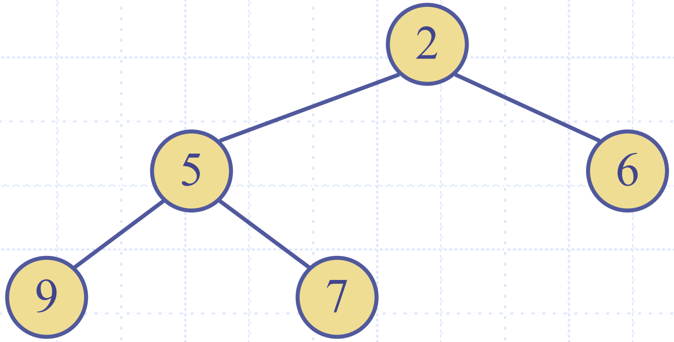
Heaps and Priority Queues

- We can use a heap to implement a priority queue
- We store a (key, element) item at each internal node
- We keep track of the position of the last node



Array-based Heap Implementation

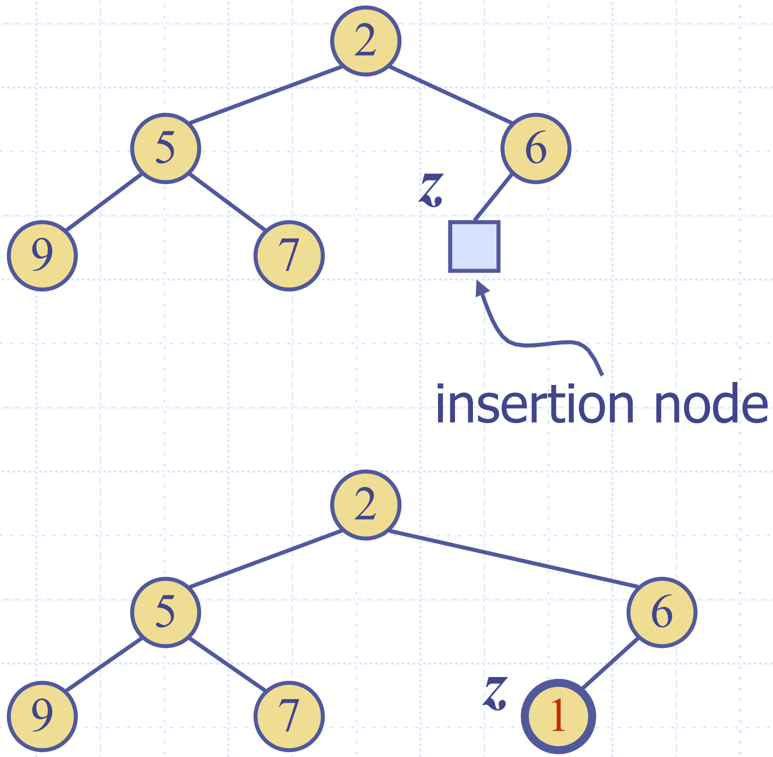
- We can represent a heap with n keys by means of an array of length n
- For the node at rank i
  - the left child is at rank $2i$
  - the right child is at rank $2i + 1$
- Links between nodes are not explicitly stored
- Operation add corresponds to inserting at rank $n + 1$
- Operation remove_min corresponds to removing at rank n
- Yields in-place heap-sort



2	5	6	9	7
1	2	3	4	5

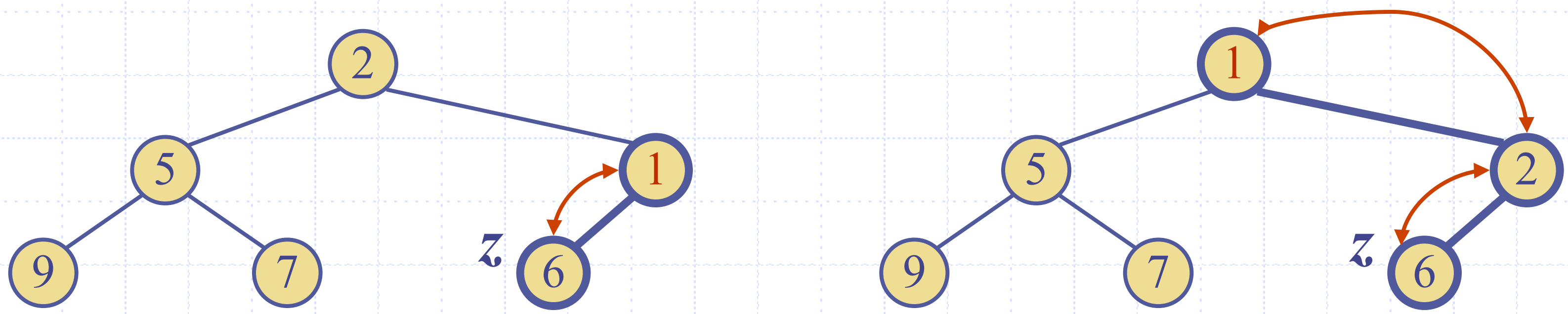
Insertion into a Heap

- Method insertItem of the priority queue ADT corresponds to the insertion of a key k to the heap
- The insertion algorithm consists of three steps
  - Find the insertion node z (the new last node)
  - Store k at z
  - Restore the heap-order property (discussed next)



Upheap

- After the insertion of a new key k, the heap-order property may be violated
- Algorithm upheap restores the heap-order property by swapping k along an upward path from the insertion node
- Upheap terminates when the key k reaches the root or a node whose parent has a key smaller than or equal to k
- Since a heap has height $O(\log n)$, upheap runs in $O(\log n)$ time



Insertion Pseudo-Code

- Assumes an array-based heap implementation.

Algorithm HeapInsert(k, e):

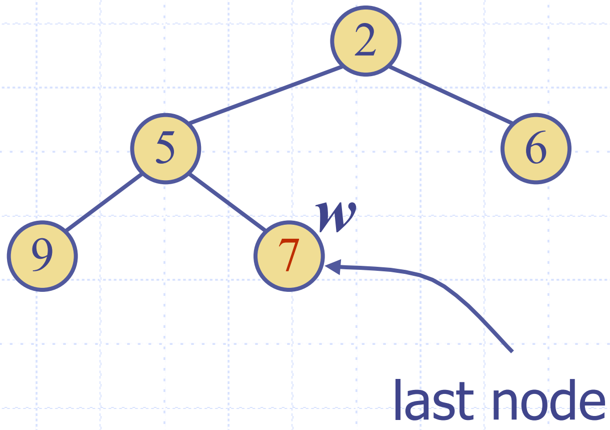
__Input:__ A key-element pair

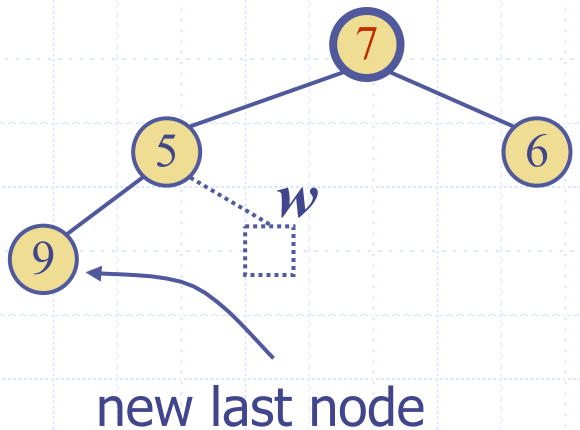
__Output:__ An update of the array, A, of n elements, for a heap, to add (k, e)

$n \leftarrow n + 1$
$A[n] \leftarrow (k, e)$
$i \leftarrow n$
while $i > 1$ **and** $A[\lfloor i/2 \rfloor] > A[i]$ **do**
  Swap $A[\lfloor i/2 \rfloor]$ and $A[i]$
  $i \leftarrow \lfloor i/2 \rfloor$

Removal from a Heap

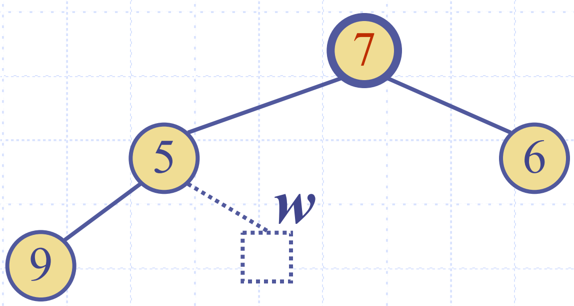
- Method removeMin of the priority queue ADT corresponds to the removal of the root key from the heap
- The removal algorithm consists of three steps
  - Replace the root key with the key of the last node w
  - Remove w
  - Restore the heap-order property (discussed next)

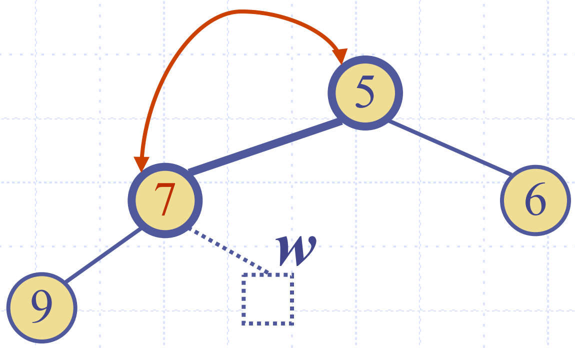




Downheap

- After replacing the root key with the key k of the last node, the heap-order property may be violated
- Algorithm downheap restores the heap-order property by swapping key k along a downward path from the root
- Upheap terminates when key k reaches a leaf or a node whose children have keys greater than or equal to k
- Since a heap has height $O(\log n)$, downheap runs in $O(\log n)$ time





RemoveMin Pseudo-code

- Assumes heap is implemented with an array.

Algorithm HeapRemoveMin():

Input: None

Output: An update of the array, A, of n elements, for a heap, to remove and return an item with smallest key

```
temp ← A[1]
A[1] ← A[n]
n ← n − 1
i ← 1
while i < n do
    if 2i + 1 ≤ n then   // this node has two internal children
        if A[i] ≤ A[2i] and A[i] ≤ A[2i + 1] then
            return temp       // we have restored the heap-order property
        else
            Let j be the index of the smaller of A[2i] and A[2i + 1]
            Swap A[i] and A[j]
            i ← j
    else   // this node has zero or one internal child
        if 2i ≤ n then   // this node has one internal child (the last node)
            if A[i] > A[2i] then
                Swap A[i] and A[2i]
        return temp       // we have restored the heap-order property
return temp       // we reached the last node or an external node
```

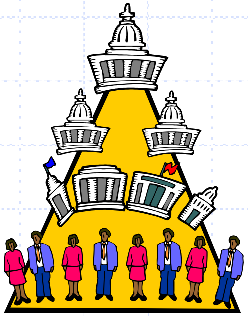
Performance of a Heap

- A heap has the following performance for the priority queue operations.

Operation	Time
insert	$O(\log n)$
removeMin	$O(\log n)$

- The above analysis is based on the following facts:
  - The height of heap T is O(log n), since T is complete.
  - In the worst case, up-heap and down-heap bubbling take time proportional to the height of T.
  - Finding the insertion position in the execution of insert and updating the last node position in the execution of removeMin takes constant time.
  - The heap T has n internal nodes, each storing a reference to a key and a reference to an element.

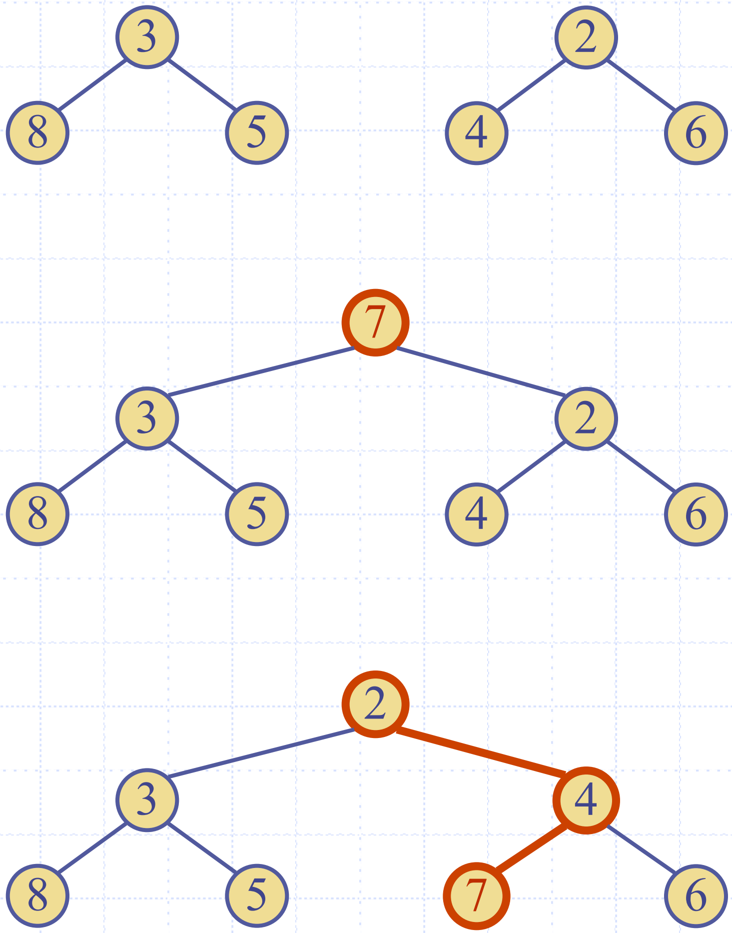
Heap-Sort



- Consider a priority queue with n items implemented by means of a heap
  - the space used is $O(n)$
  - methods insert and removeMin take $O(\log n)$ time
  - methods size, isEmpty, and min take time $O(1)$ time
- Using a heap-based priority queue, we can sort a sequence of n elements in $O(n \log n)$ time
- The resulting algorithm is called heap-sort
- Heap-sort is much faster than quadratic sorting algorithms, such as insertion-sort and selection-sort

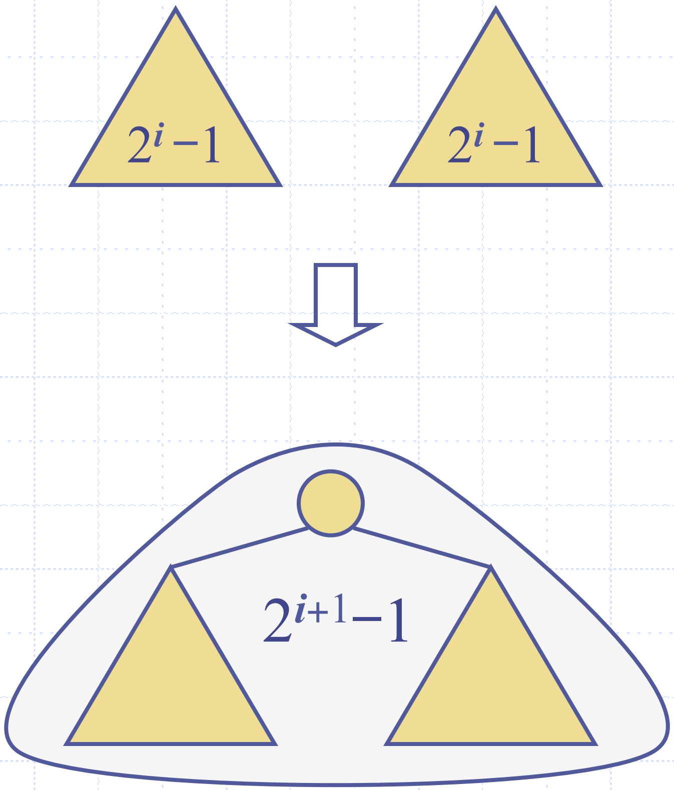
Merging Two Heaps

- We are given two two heaps and a key k
- We create a new heap with the root node storing k and with the two heaps as subtrees
- We perform downheap to restore the heap-order property

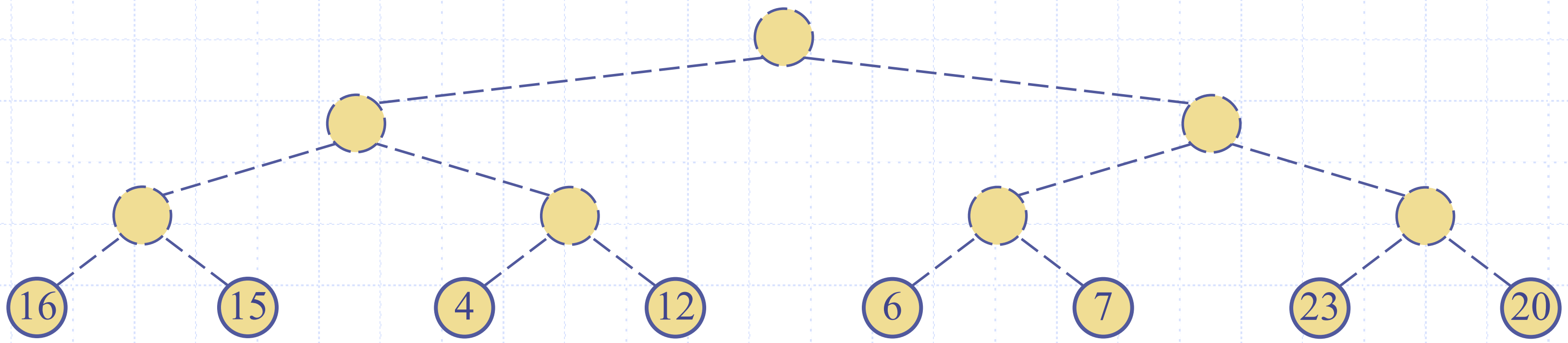


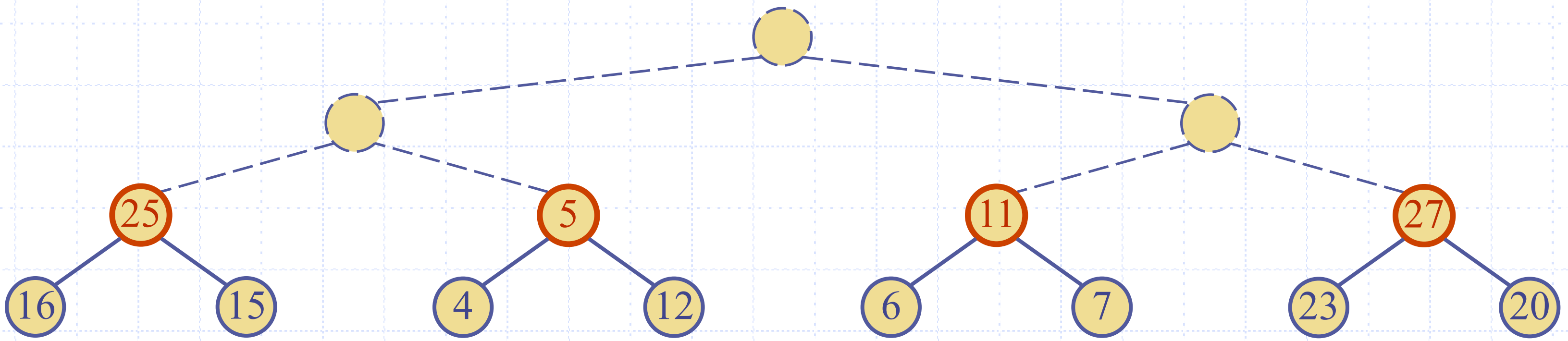
Bottom-up Heap Construction

- We can construct a heap storing n given keys in using a bottom-up construction with $\log n$ phases
- In phase i, pairs of heaps with $2^i - 1$ keys are merged into heaps with $2^{i+1} - 1$ keys

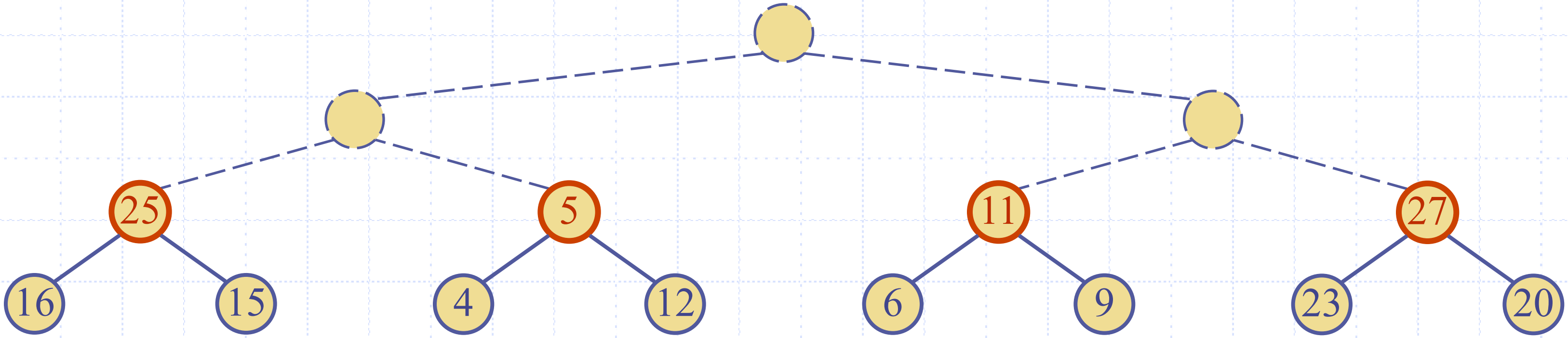


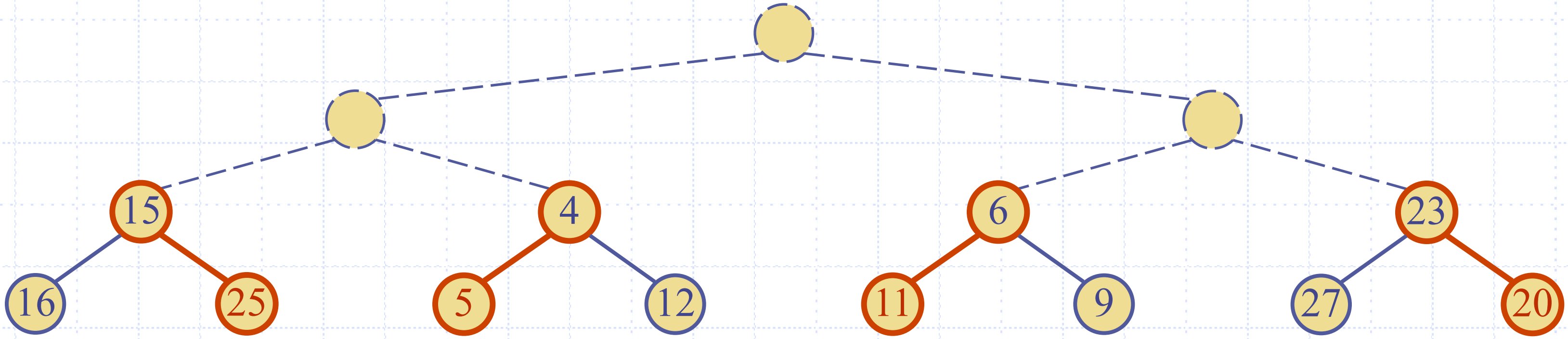
Example



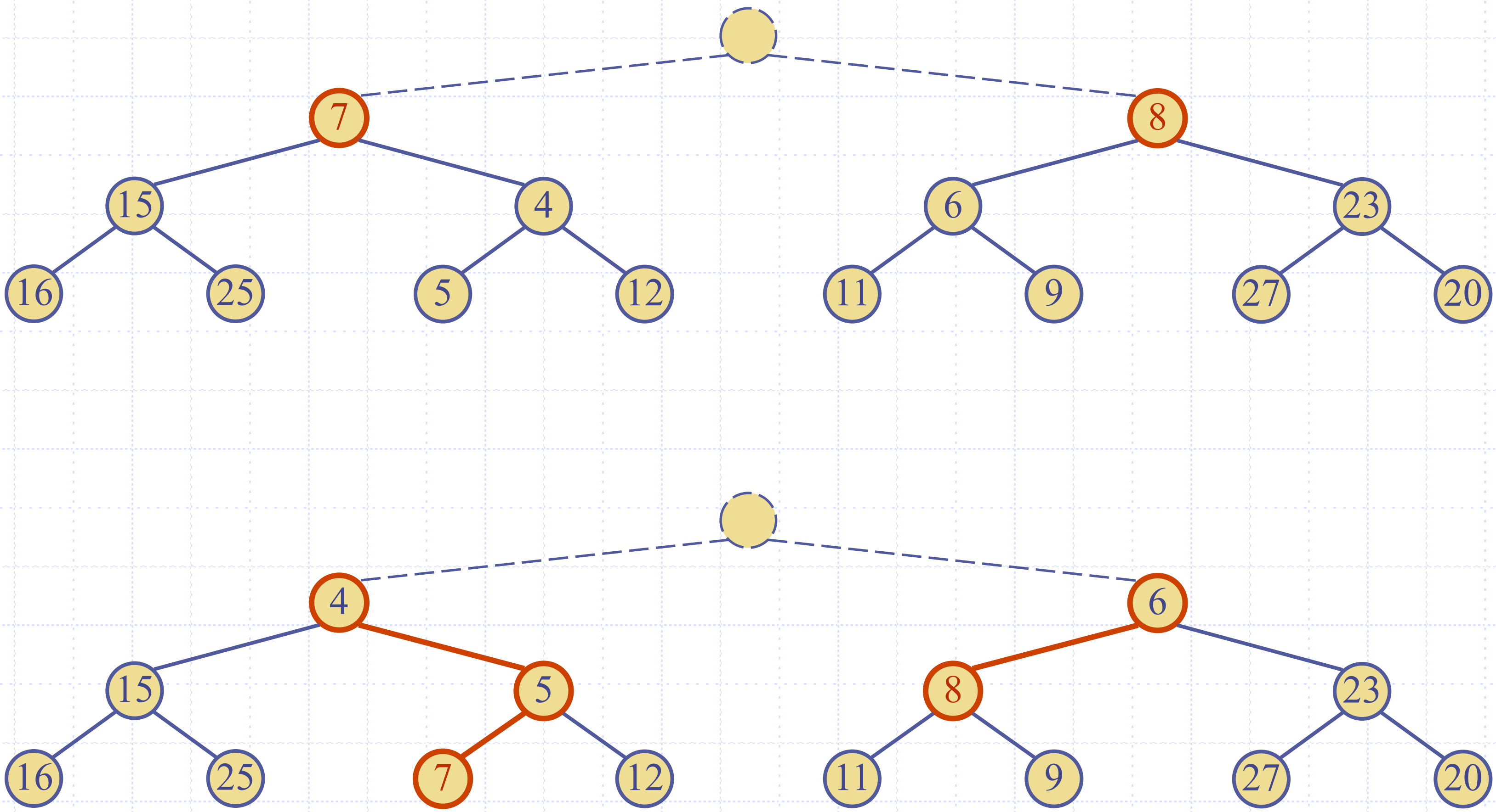


Example (contd.)

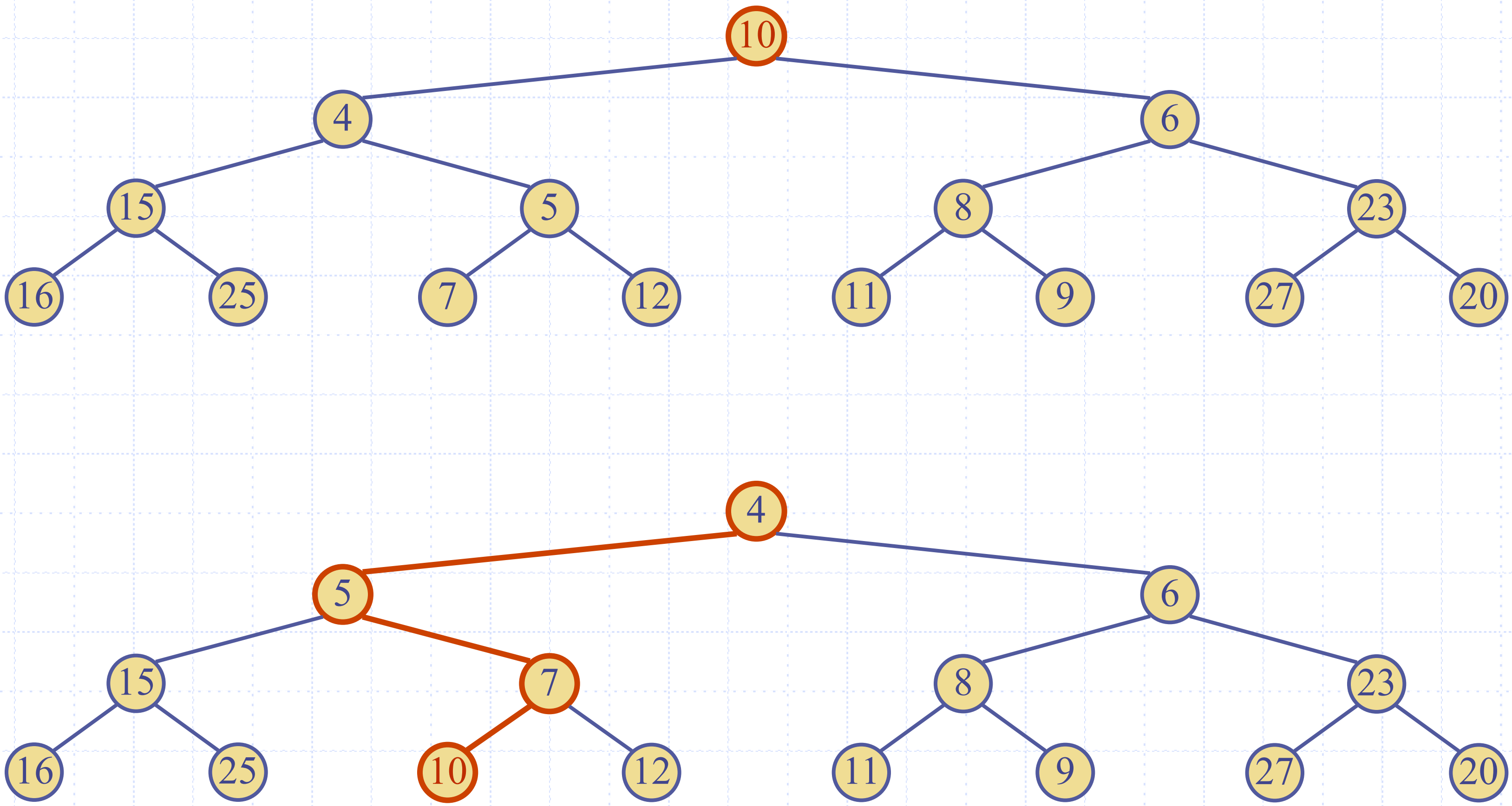




Example (contd.)



Example (end)



Analysis

- We visualize the worst-case time of a downheap with a proxy path that goes first right and then repeatedly goes left until the bottom of the heap (this path may differ from the actual downheap path)
- Since each node is traversed by at most two proxy paths, the total number of nodes of the proxy paths is $O(n)$
- Thus, bottom-up heap construction runs in $O(n)$ time
- Bottom-up heap construction is faster than n successive insertions and speeds up the first phase of heap-sort, which takes $O(n \log n)$ time in its second phase.