Heaps
Recall Priority Queue ADT

- A priority queue stores a collection of entries
- Each entry is a pair (key, value)
- Main methods of the Priority Queue ADT
  - `insert(k, x)` inserts an entry with key k and value x
  - `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
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(S, C)`

- **Input** sequence $S$, comparator $C$ for the elements of $S$
- **Output** sequence $S$ sorted in increasing order according to $C$

$$P \leftarrow \text{priority queue with comparator } C$$

while $\neg S.isEmpty()$

$$e \leftarrow S.remove(S.first())$$

$P.insertItem(e, e)$

while $\neg P.isEmpty()$

$$e \leftarrow P.removeMin().getKey()$$

$S.addLast(e)$

© 2010 Goodrich, Tamassia
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, \( \text{key}(v) \geq \text{key}(\text{parent}(v)) \)
  - **Complete Binary Tree:** let \( h \) be the height of the heap
    - for \( i = 0, \ldots, h - 1 \), 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.

© 2010 Goodrich, Tamassia
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

Diagram: A heap with nodes labeled with (key, element) pairs such as (2, Sue), (5, Pat), (6, Mark), (9, Jeff), and (7, Anna).
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.
Removal from a Heap (§ 7.3.3)

- 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.
Updating the Last Node

- The insertion node can be found by traversing a path of $O(\log n)$ nodes
  - Go up until a left child or the root is reached
  - If a left child is reached, go to the right child
  - Go down left until a leaf is reached
- Similar algorithm for updating the last node after a removal
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
Vector-based Heap Implementation

- We can represent a heap with $n$ keys by means of a vector of length $n + 1$
- 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
- The cell of at rank 0 is not used
- Operation insert corresponds to inserting at rank $n + 1$
- Operation removeMin corresponds to removing at rank $n$
- Yields in-place heap-sort
Merging Two Heaps

- We are given 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.
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.