Recall Priority Queue ADT (§ 7.1.3)

- A priority queue stores a collection of entries.
- Each entry is a pair of key and 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 Priority Queue Sorting (§ 7.1.4)

- We can use a priority queue to sort a set of comparable elements.
- 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) 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 ← priority queue with comparator C
while ¬S.isEmpty()
  e ← S.remove(S.first())
  P.insertItem(e, e)
while ¬P.isEmpty()
  e ← P.removeMin()
  S.insertLast(e)
```

Heaps (§7.3)

- 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) ≥ key(parent(v))`.
  - Complete Binary Tree: let `h` be the height of the heap.
    - For `i = 0, . . . , h − 1`, there are `2^i` nodes of depth `i`.
    - At depth `h − i`, the internal nodes are to the left of the external nodes.
- The last node of a heap is the rightmost node of depth `h`.
- The last node of a heap is the rightmost node of depth `h`.
**Height of a Heap (§ 7.3.1)**

**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
- For simplicity, we show only the keys in the pictures

**Insertion into a Heap (§ 7.3.3)**

- 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:
  1. Replace the root key with the key of the last node w.
  2. Remove w.
  3. 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:
  1. Go up until a left child or the root is reached.
  2. If a left child is reached, go to the right child.
  3. Go down left until a leaf is reached.
- Similar algorithm for updating the last node after a removal.

Heap-Sort (§2.4.4)

- Consider a priority queue with n items implemented by means of a heap:
  1. The space used is $O(n)$.
  2. Methods insert and removeMin take $O(\log n)$ time.
  3. 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 (§2.4.3)

- 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.

Bottom-up Heap Construction (§2.4.3)

- 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.)

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.