Priority Queues

IE 496 Lecture 12
Reading for This Lecture

- Horowitz and Sahni, Chapter 2
- Kozen, Lecture 8
Priority Queues

• A queue where each item has a specified *priority*.

• Additional operations for priority queues
  – find_min()
  – delete_min()

• Applications
  – sorting
  – greedy algorithms

• We will discuss these in future lectures
Heaps

- **A heap** is a scheme for implementing a priority queue.
- A heap is a binary tree in which the value at each node is at least as large as the values in each of its children.
- Hence, a largest element is always at the root.
- Heaps support the following operations
  - `insert()`
  - `delete_max()`
  - `make_heap()`
  - `adjust_heap()`
Inserting into a heap

- Insert the value into node $n+1$ and "bubble up".
- Compare the value to its parent and swap if necessary.
- Continue swapping until heap property is restored.
- One way to make a heap from $n$ elements is to simply insert them one at a time.

Analysis
- `insert()`
- `make_heap()`
Adjusting a heap

- If only the root of a heap is out of order, we can restore order by "bubbling down" (adjust()).
  - Swap the root with the larger child.
  - Continue swapping process until heap property is restored.

- Heapify (create a heap by iterative adjusting)

  For each node \( i = \lfloor n/2 \rfloor \rightarrow 1 \)
  adjust \( i \) w.r.t. the subtrees rooted at its children

- Analysis
Deleting from a heap

• To delete the root node,
  – exchange node 1 with node $n$.
  – adjust the heap.

• Heapsort
  – First heapify.
  – Iteratively delete the root node.

• Analysis
Binomial Trees

- The *binomial tree* of rank $i$ ($B_i$) is defined recursively.
- $B_i$ consists of a *root* with $i$ children $B_0, \ldots, B_{i-1}$.
Binomial Heaps

- A *binomial heap* is a collection of *heap ordered* binomial trees and a pointer to the overall max/min.
- No more than one tree of each rank is allowed.
- The children of each vertex are maintained in a *circular linked list*.
- The basic operation is *linking*.
- Two trees of rank $i$ can be combined into one tree of rank $i+1$ in constant time.
Eager Meld

- We can combine two heaps by performing a `meld()` reminiscent of binary addition.
- Successively link trees of equal rank and "carry" one if necessary.
- Must track the position of the new min/max element.
- This operation takes $O(\log n)$ time.
Inserting into a Binomial Heap

- **To insert() an element:**
  - Make a new heap from the single element to be inserted.
  - Meld the new heap with the old one.

- **To make_heap() from scratch, perform a sequence of inserts.**

- **To delete() the min/max element:**
  - The children of this element form a new binomial heap.
  - Meld the old heap and the new one.
Amortized Analysis

- `meld()` and `delete()` both take $O(\log n)$.
- We will use *amortized analysis* to show that `insert()` is constant time overall.
- Each time, we create a new tree, we charge an extra unit of time to that operation.
- We use those “credits” to “pay” for operations that link trees later on.
- In this way, we can justify that any sequence of inserts takes constant time *on average*. 