lecture 28

heaps

- priority queue ADT,
  - add(key)
  - removeMin()

Priority Queue

- like a queue, but it allows a more general definition of which element to remove next
  - e.g. hospital emergency room
  - airline ticket counter

Priority Queue ADT

- holds set of comparable keys
- main operations:
  - add(key)
  - removeMin()

highest priority = number 1 priority

Java class PriorityQueue<T>

methods
- offer(T key) ≈ add(key)
- poll() ≈ removeMin()
- peek() ≈ findMin()
- remove(Object key) * overrides method from super class (allow “duplicates” i.e. keys that are equal to each other)

Priority Queue

How to implement add(), removeMin()?

- sorted list
  - linked list
  - array

- binary search tree
- heap

Complete Binary Tree (definition)

- binary tree of height h
- every level ≤ h less than h is full (2^h nodes)
- all nodes at level h are as far to the left as possible
Heap (min Heap)

- complete binary tree, with a key at each node
- keys are comparable
- parent key ≤ left child key
- parent key ≤ right child key

How many heaps from letters a, b, c, d, e, f, g?

a
\ / \
 b e
\ / \ / \ / 
c d f g

any more?
(no)

Example:

\ a \ /
 \ b /
 \ e /
 f l u k
 m

Example: add(c)

\ a \ /
 \ b /
 \ e /
 f l u k
 m c

How many heaps from letters a, b, c, d, e, f, g?

a
\ / \
 b c
\ / \
 d e f g
 c f e g

Note that if you swap the positions of siblings (swap the nodes, not the keys) you automatically preserve the heap property.

Example: add(c)

\ a \ /
 \ b /
 \ e /
 f l u k
 m c

Swap c with f
```
add(key)
create new node at next available leaf position
cur = node
cur.key = key
while (cur != root) & & (cur.key < parent.key)
    swap key (cur, parent)
    cur = cur.parent
}
```
remove Min()
{
  remove last leaf and put its key in root
  node = root
  while ((node.key > left child.key)
          or (node.key > right child.key))
  {
    c = child with smaller key
    SwapKey(node, c)
    node = c
  }
}