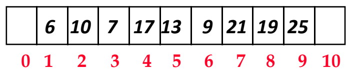
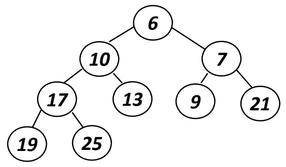
CompSci 100e
Program Design and Analysis II





March 31, 2011

Prof. Rodger

Announcements

- Written linked lists/trees due today
- APT BSTCount due Tuesday, April 2
- Boggle assignment due in one week
  - Will discuss more in lab

Abstract Data Type

- Stack (LIFO)
  - Push (add), pop (remove)
- Queue (FIFO)
  - Enqueue (add), dequeue (remove)
- Priority queue – queue, but best item dequeued (example: delete and return the minimum each time)
  - Enqueue (add), deleteMin (remove)

Priority queue implementation

- Operations: add and delete min
- Want to do these operations efficiently

Priority Queue sorting

code below sorts, complexity?

```
String[] array = {...}; // array filled with data
PriorityQueue<String> pq = new
  PriorityQueue<String>();
for(String s : array) pq.add(s);
for(int k=0; k < array.length; k++){
   array[k] = pq.remove();
}
```

Priority Queue top-M sorting

- What if we have *lots and lots and lots* of data
  - code below sorts top-M elements, complexity?

```
Scanner s = … // initialize;
PriorityQueue<String> pq =
      new PriorityQueue<String>();
while (s.hasNext()) {
    pq.add(s.next());
    if (pq.size() > M) pq.remove();
}
```

- What's advantageous about this code?
  - Store everything and sort everything?
  - Store everything, sort first M?
  - What is complexity of sort: O(n log n)

`PriorityQueue.java (Java 5+)`

- What about objects inserted into pq?
  - Comparable, e.g., essentially sortable
  - How can we change what *minimal* means?
  - Implementation uses *heap*, tree stored in an array
- Use a Comparator for comparing entries we can make a min-heap act like a max-heap, see PQDemo
  - Where is class Comparator declaration? How used?
  - What if we didn't know about Collections.reverseOrder?
    - How do we make this ourselves?

Priority Queue implementation

- Heap data structure is fast and reasonably simple
  - Why not use inheritance hierarchy as was used with Map?
  - Trade-offs when using HashMap and TreeMap:
    - Time, space, ordering properties, TreeMap support?
- Changing comparison when calculating priority?
  - Create object to replace, or in lieu of `compareTo`
    - `Comparable` interface compares `this` to passed object
    - `Comparator` interface compares two passed objects
  - Both comparison methods: `compareTo()` and `compare()`
    - Compare two objects (parameters or self and parameter)
    - Returns –1, 0, +1 depending on <, ==, >

Creating Heaps

- Heap: array-based implementation of binary tree used for implementing priority queues:
  - add/insert, peek/getmin, remove/deletemin, O(???)

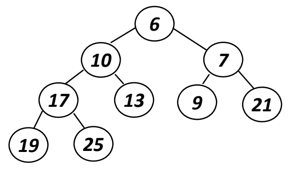
- Array minimizes storage (no explicit pointers), faster too, contiguous (cache) and indexing
- Heap has *shape* property and *heap/value* property
  - shape: tree filled at all levels (except perhaps last) and filled left-to-right (complete binary tree)
  - each node has value smaller than both children

Array-based heap – one implementation for priority queue

- store "node values" in array beginning at index 1
- for node with index k
  - left child: index `2*k`
  - right child: index `2*k+1`

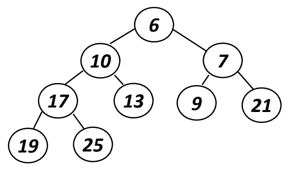
- why is this conducive for maintaining heap shape?
- what about heap property?
- is the heap a search tree?
- where is minimal node?
- where are nodes added? deleted?

	6	10	7	17	13	9	21	19	25	
0	1	2	3	4	5	6	7	8	9	10



Thinking about heaps

- Where is minimal element?
- Where is maximal element?
- How many leaves are there in an N-node heap (big-Oh)?
- What is complexity of find max in a minheap? Why?
- Where is second smallest element? Why?



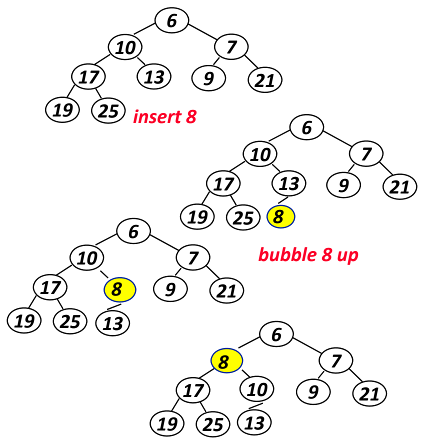
	6	10	7	17	13	9	21	19	25	
0	1	2	3	4	5	6	7	8	9	10

Adding values to heap

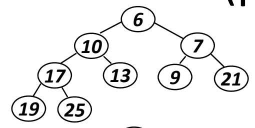
- to maintain heap shape, must add new value in left-to-right order of last level
  - could violate *heap property*
  - move value "up" if too small

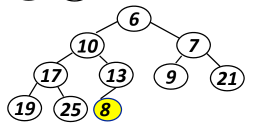
- change places with parent if heap property violated
  - stop when parent is smaller
  - stop when root is reached

- pull parent down, swapping isn't necessary (optimization)



Adding values, details (pseudocode)

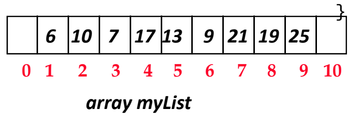




```
void add(Object elt)
{
  // add elt to heap in myList
  myList.add(elt);
  int loc = myList.size()-1;

  while (1 < loc &&
         elt < myList.get(loc/2)){
   myList.set(loc,myList.get(loc/2));
   loc = loc/2; // go to parent
  }
  // what's true here?

  myList.set(loc,elt);
}
```

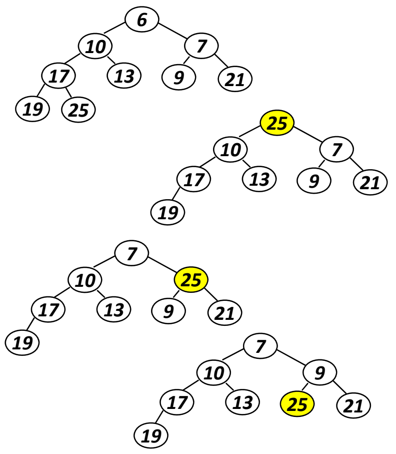


	6	10	7	17	13	9	21	19	25	
0	1	2	3	4	5	6	7	8	9	10

array myList

Removing minimal element

- Where is minimal element?
  - If we remove it, what changes, shape/property?
- How can we maintain shape?
  - "last" element moves to root
  - What property is violated?
- After moving last element, subtrees of root are heaps, why?
  - Move root down (pull child up) does it matter where?
- When can we stop "re-heaping"?
  - Less than both children
  - Reach a leaf



Priority Queue implementations

- Priority queues: average and worst case

	Insert average	Getmin (delete)	Insert worst	Getmin (delete)
Unsorted list				
Sorted list				
Search tree				
Balanced tree				
Heap				

- *Heap has O(n) build heap from n elements*

Anita Borg 1949-2003

- "Dr. Anita Borg tenaciously envisioned and set about to change the world for women and for technology. … she fought tirelessly for the development technology with positive social and human impact."
- "Anita Borg sought to revolutionize the world and the way we think about technology and its impact on our lives."
- http://www.youtube.com/watch?v=1yPxd5jqz_Q