Priority Queues:
Binary Min Heaps

CSE 373
Data Structures and Algorithms

Today's Outline

- **Announcements**
  - Assignment #2 due Fri, Oct 12 at the BEGINNING of lecture
  - Midterm #1, Fri, Oct 19.
  - Assignment #3 coming soon, due Thurs, Oct 25.
- **Today's Topics:**
  - **Dictionary**
    - **Balanced Binary Search Trees - (AVL Trees)**
  - **Priority Queues**
    - **Binary Min Heap**

Priority Queue ADT

1. **PQueue data** : collection of data with priority
2. **PQueue operations**
   - insert
   - deleteMin

   (also: create, destroy, is_empty)
3. **PQueue property**: for two elements in the queue, *x* and *y*, if *x* has a **lower** priority value than *y*, *x* will be deleted before *y*

Applications of the Priority Q

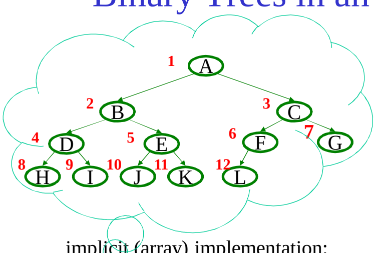
- Select print jobs in order of decreasing length
- Forward packets on network routers in order of urgency
- Select most frequent symbols for compression
- Sort numbers, picking minimum first

- **Anything *greedy***

Implementations of Priority Queue ADT

	insert	deleteMin
Unsorted list (Array)		
Unsorted list (Linked-List)		
Sorted list (Array)		
Sorted list (Linked-List)		
Binary Search Tree (BST)		

Representing Complete Binary Trees in an Array



From node **i**:

left child:
right child:
parent:

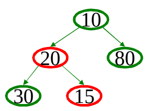
implicit (array) implementation:

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**	**J**	**K**	**L**	
0	1	2	3	4	5	6	7	8	9	10	11	12	13

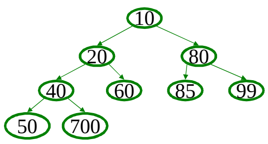
Why better than tree with pointers?

Heap **Order** Property

Heap order property: For every non-root node X, the value in the parent of X is less than (or equal to) the value in X.

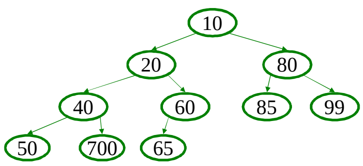


not a heap



Heap Operations

- findMin:
- insert(val): percolate up.
- deleteMin: percolate down.



Heap – Insert(val)

Basic Idea:

1. Put val at “next” leaf position
2. Repeatedly exchange node with its parent if needed

Insert pseudo Code (optimized)

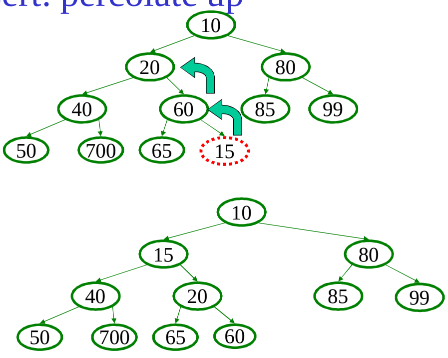
```
void insert(Object o) {
  assert(!isFull());
  size++;
  newPos =
    percolateUp(size,o);
  Heap[newPos] = o;
}
```

```
int percolateUp(int hole,
                Object val) {
  while (hole > 1 &&
         val < Heap[hole/2])
    Heap[hole] = Heap[hole/2];
    hole /= 2;
  }
  return hole;
}
```

runtime:

(Java code in book)

Insert: percolate up



Heap – Deletemin

Basic Idea:

1. Remove root (that is always the min!)
2. Put "last" leaf node at root
3. Find smallest child of node
4. Swap node with its smallest child if needed.
5. Repeat steps 3 & 4 until no swaps needed.

DeleteMin pseudo Code (Optimized)

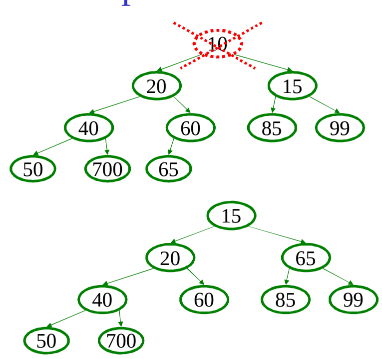
```
Object deleteMin() {
  assert(!isEmpty());
  returnVal = Heap[1];
  size--;
  newPos =
    percolateDown(1,
        Heap[size+1]);
  Heap[newPos] =
    Heap[size + 1];
  return returnVal;
}
```

runtime:

(Java code in book)

```
int percolateDown(int hole,
                  Object val) {
while (2*hole <= size) {
    left = 2*hole;
    right = left + 1;
    if (right ≤ size &&
        Heap[right] < Heap[left])
      target = right;
    else
      target = left;
    if (Heap[target] < val) {
      Heap[hole] = Heap[target];
      hole = target;
    }
    else
      break;
  }
  return hole;
}
```


DeleteMin: percolate down



Insert: 16, 32, 4, 69, 105, 43, 2

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**

Other Priority Queue Operations

- **decreaseKey**
  - given a pointer to an object in the queue, reduce its priority value

  Solution: change priority and ______________________________
- **increaseKey**
  - given a pointer to an object in the queue, increase its priority value

  Solution: change priority and _______________________________

Why do we need a *pointer*? Why not simply data value?

Other Heap Operations

decreaseKey(objPtr, amount): raise the priority of a object, percolate up

increaseKey(objPtr, amount): lower the priority of a object, percolate down

remove(objPtr): remove a object, move to top, them delete.
1) decreaseKey(**objPtr**, ∞)
2) deleteMin()

Worst case Running time for all of these:

FindMax?

ExpandHeap – when heap fills, copy into new space.

Binary Min Heaps (summary)

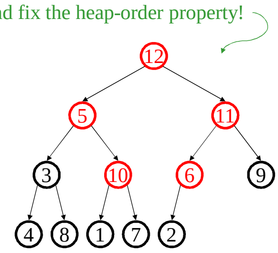
- **insert**: percolate up. Θ(log N) time.
- **deleteMin**: percolate down. Θ(log N) time.

- **Build Heap?**

BuildHeap: Floyd's Method

12	5	11	3	10	6	9	4	8	1	7	2

Add elements arbitrarily to form a complete tree.
Pretend it's a heap and fix the heap-order property!

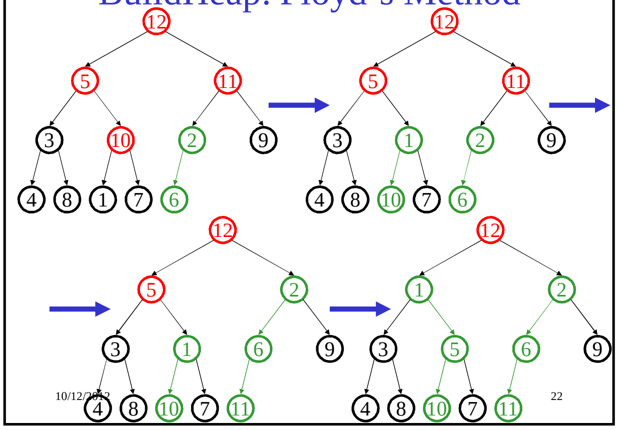


Buildheap pseudocode

```
private void buildHeap() {
  for ( int i = currentSize/2; i > 0; i-- )
      percolateDown( i );
}
```

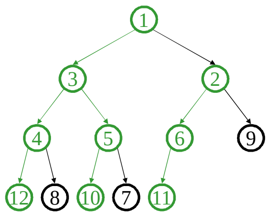
runtime:

BuildHeap: Floyd's Method



10/12/2012 22

Finally…



runtime:

10/12/2012 23