Data Structure

Lecture#12: Binary Trees 3 (Chapter 5)

U Kang
Seoul National University

In This Lecture

- Motivation of Priority Queue data structure
- Main ideas and implementations of Heap data structure
- Analysis of Heap data structure

Priority Queues (1)

- Problem: We want a data structure that stores records as they come (insert), but on request, releases the record with the greatest value (removemax)

- Example: scheduling jobs in a multi-tasking operating system.

Priority Queues (2)

- Possible Solutions:
  - **Unsorted** array or linked list?

  - **Sorted** array or linked list?

Priority Queues (3)

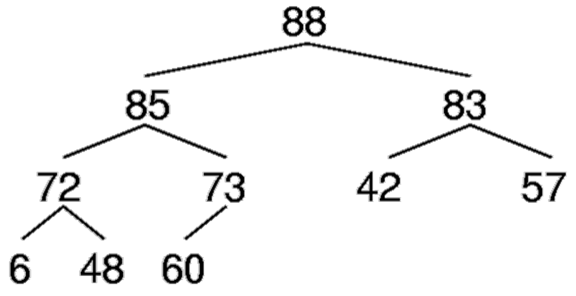
- Possible Solutions:
  - Insert appends to an **unsorted** array or a linked list (O(1)) and then removemax determines the maximum by scanning the list (O(n))
  - A **sorted** array or a linked list is used, and is in increasing order; insert places an element in its correct position (O(n)) and removemax simply removes the end of the list (O(1)).
  - Use a *heap* – both insert and removemax are O(log n) operations

Heaps

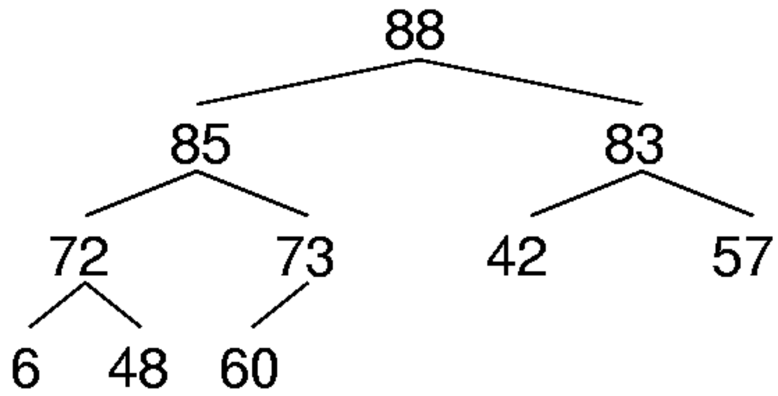
- Heap: complete binary tree with the heap property:
  - Min-heap: All values less than child values.
  - Max-heap: All values greater than child values.

- The values are partially ordered (parent-child)
  - ⇔ Binary Search Tree

- Heap representation: normally the array-based complete binary tree representation.



Max Heap Example



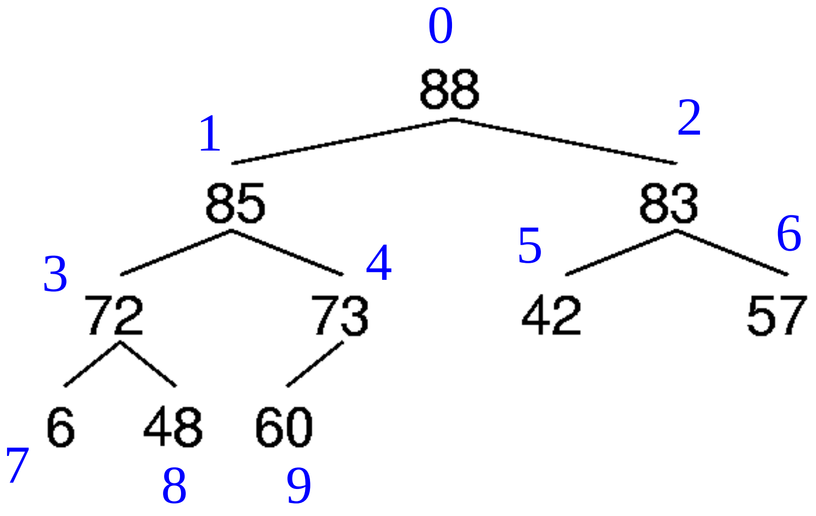
88	85	83	72	73	42	57	6	48	60

Pos: 0 1 2 3 4 5 6 7 8 9

Max Heap Property

- Positions of leaf nodes in a max heap with n nodes: $\left\lfloor \frac{n}{2} \right\rfloor \sim n-1$

- I.e., a max heap with n nodes contains n $- \left\lfloor \frac{n}{2} \right\rfloor$ leaf nodes

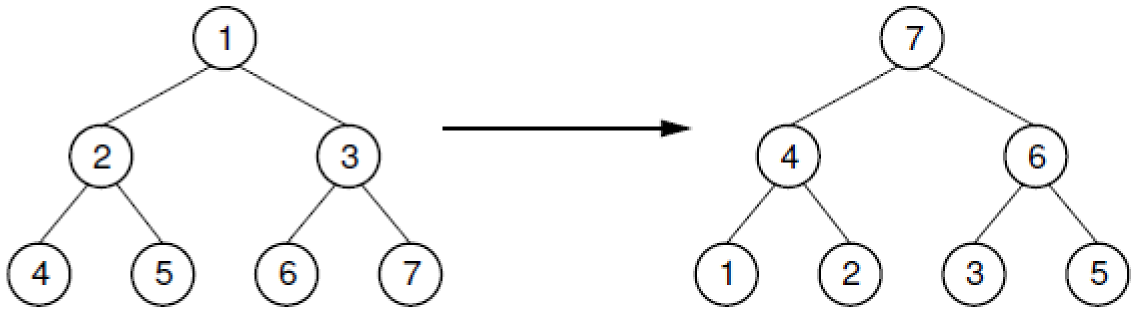


Max Heap Implementation (1)

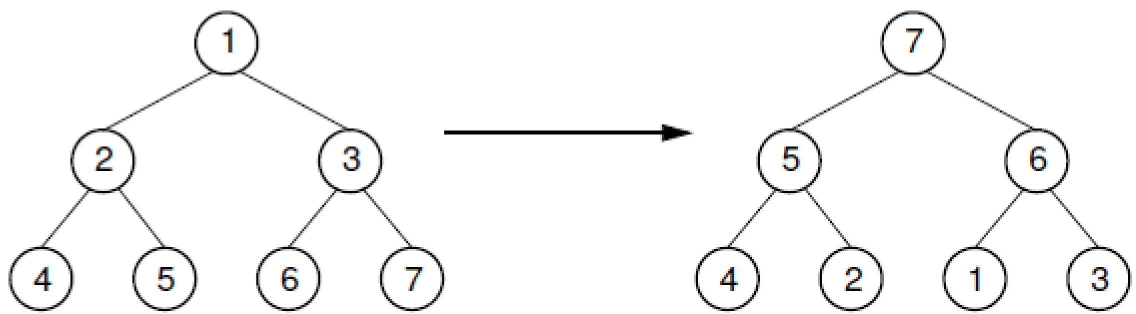
```
public class MaxHeap<K extends Comparable<? super K>, E> {
  private E[] Heap;     // Pointer to heap array
  private int size;     // Maximum size of heap
  private int n;        // # of things in heap

public MaxHeap(E[] h, int num, int max)
{ Heap = h;  n = num;  size = max;  buildheap(); }

public int heapsize() { return n; }

public boolean isLeaf(int pos) // Is pos a leaf position?
{ return (pos >= n/2) && (pos < n); }

public int leftchild(int pos) { // Leftchild position
  assert pos < n/2 : "Position has no left child";
  return 2*pos + 1;
}

public int rightchild(int pos) { // Rightchild position
  assert pos < (n-1)/2 : "Position has no right child";
  return 2*pos + 2;
}
public int parent(int pos) {
  assert pos > 0 : "Position has no parent";
  return (pos-1)/2;
}
```

Building Heaps

- Binary tree to heap



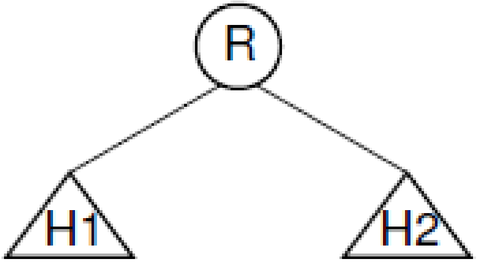
(4-2), (4-1), (2-1), (5-2), (5-4), (6-3), (6-5), (7-5), (7-6)



(5-2), (7-3), (7-1), (6-1)

Building Heaps

- How to build heap? “sift down” method: move small nodes down the heap

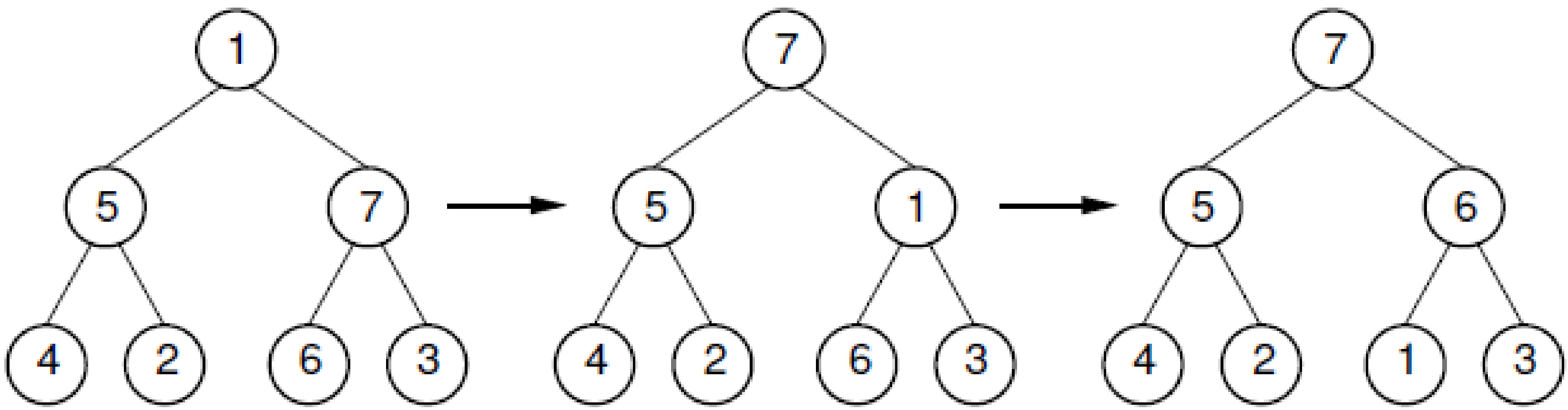


- Both H1 and H2 are heaps. Push R down properly

Sifting down flour

Building Heaps

- Siftdown operation



Sift Down

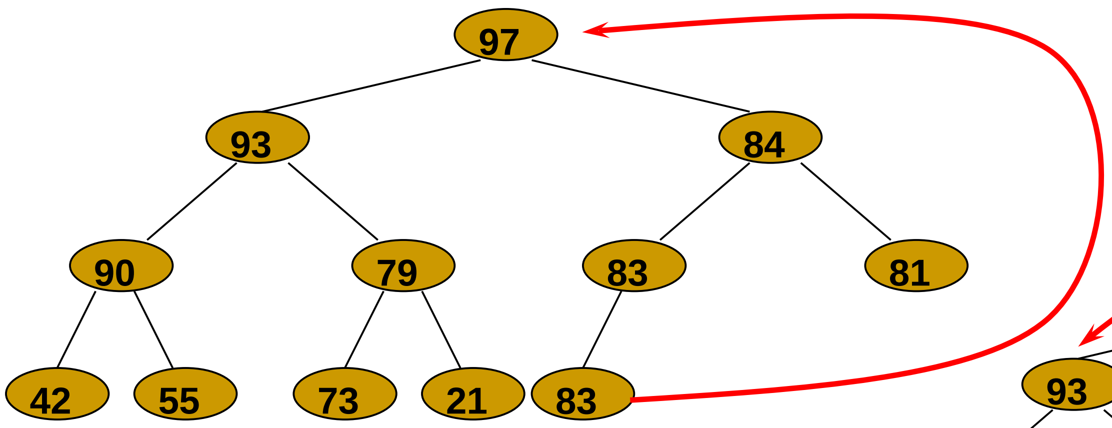
```
public void buildheap() // Heapify contents
  { for (int i=n/2-1; i>=0; i--) siftdown(i); }

private void siftdown(int pos) {
  assert (pos >= 0) && (pos < n) :
         "Illegal heap position";
  while (!isLeaf(pos)) {
    int j = leftchild(pos);
    if ((j<(n-1)) &&
        (Heap[j].compareTo(Heap[j+1]) < 0))
      j++; // index of child w/ greater value
    if (Heap[pos].compareTo(Heap[j]) >= 0)
      return;
    DSutil.swap(Heap, pos, j);
    pos = j;  // Move down
  }
}
```

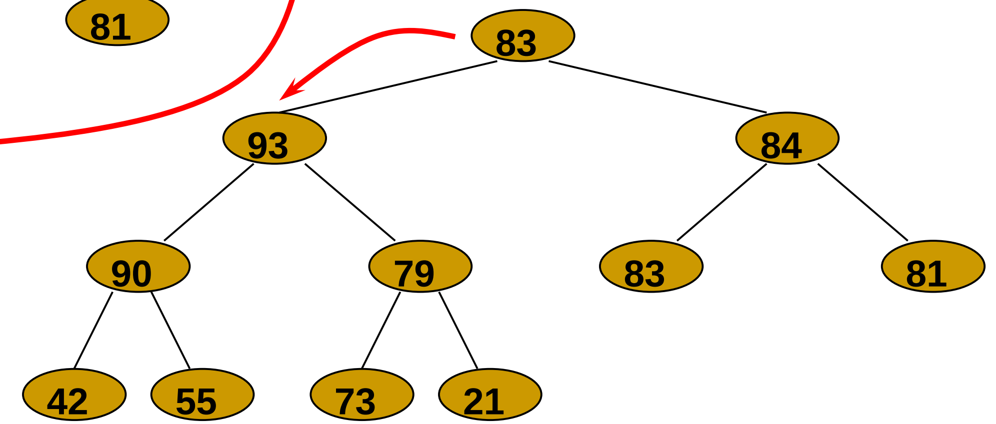
RemoveMax, Insert

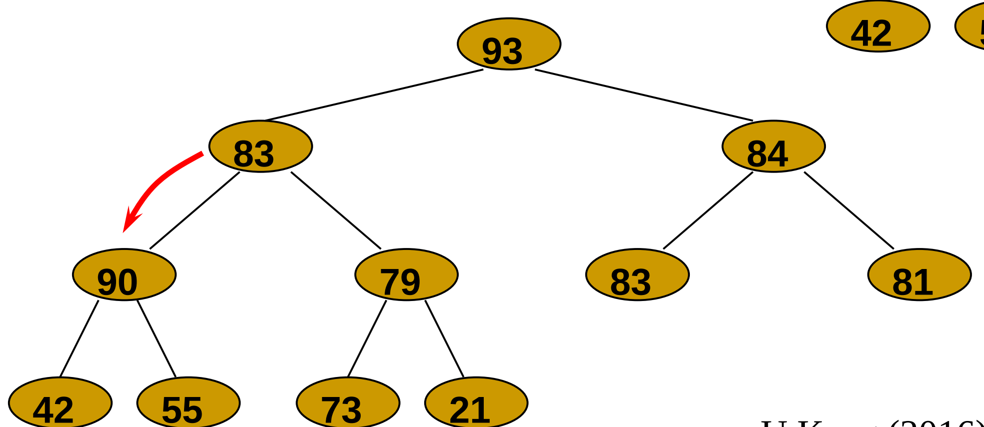
```
public E removemax() {
  assert n > 0 : "Removing from empty heap";
  DSutil.swap(Heap, 0, --n);
  if (n != 0) siftdown(0);
  return Heap[n];
}

public void insert(E val) {
  assert n < size : "Heap is full";
  int curr = n++;
  Heap[curr] = val;
  // Siftup until curr parent's key > curr key
  while ((curr != 0)  &&
        (Heap[curr].compareTo(Heap[parent(curr)])
          > 0)) {
    DSutil.swap(Heap, curr, parent(curr));
    curr = parent(curr);
  }
}
```

Example of RemoveMax

Given the initial heap:







In a heap of n nodes, the maximum distance the root can sift down would be floor(log n).

Heap Building Analysis

- Insert into the heap one value at a time:
  - Push each new value down the tree from the root to where it belongs
  - $\sum_i \log i = \theta(n \, log \, n)$
- Starting with full array, work from bottom up
  - Since nodes below form a heap, just need to push current node down (at worst, go to bottom)
  - Most nodes are at the bottom, so not far to go
  - When i is the level of the node counting from the bottom starting with 1, this is $\sum_{i=1}^{\log n}(i-1)\frac{n}{2^i} = \frac{n}{2}\sum_{i=1}^{\log n}\frac{i-1}{2^{i-1}} = \theta(n)$.

$$\boxed{\sum_{i=1}^{n}\frac{i}{2^i} = 2 - \frac{n+2}{2^n}.}$$

Heap with Complete Binary Tree

- Does Heap remains as complete binary tree after insert and removemax operations?
  - Yes!

- Thus, Heap can be implemented with an array

What you need to know

- Motivation of Priority Queue data structure; why list is not appropriate for Priority Queue
- Main ideas and implementations of Heap data structure
  - isLeaf, sift down, insert, remove max, …
  - Storage: complete binary tree using array
- Cost of building heap

Questions?