Priority Queues

CPE 103

Theresa Migler-VonDollen

The Priority Queue Abstract Data Type

Definition

A **priority queue** is similar to a regular queue or stack data structure, but where each element has a *priority* associated with it.

- An element with high priority is served before an element with low priority.
- If two elements have the same priority, they are served according to their order in the queue.

We will be using the minimum binary heap (array) implementation.

- We will use the convention that the smaller the priority number, the higher the priority.
  - An element with priority 3 will be served before an element with priority 8.

Priority Queue Operations

A priority queue must support the following operations:

- *insert (with priority):* adds an element to the queue with associated priority.
- *deleteMin:* removes the element from the queue that has the highest priority, and returns it.

In addition there are often the following two operations:

- *peek:* returns the highest-priority element but does not modify the queue.
- *isEmpty:* returns true if the queue is empty.

Binary Heap Implementation

Definition

A *binary heap* is a binary tree with two additional properties:

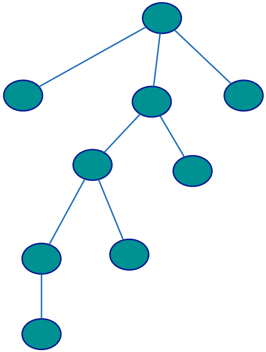
- The structure property
  - The binary tree is complete.
- The heap-order property

What is a complete binary tree?

Trees

Definition

A *tree* is a connected graph with no cycles.

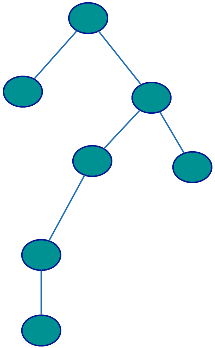


Terminology: root, parent, child, sibling, ancestor, descendant, leaf.

Binary Trees

Definition

A *binary tree* is a tree in which each vertex has at most two children.



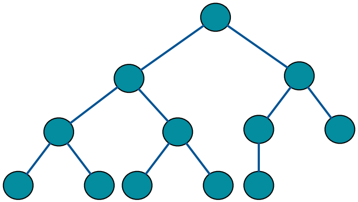
Terminology: root, parent, left child, right child.

Complete Binary Trees

Definition

A *complete binary tree* is a binary tree in which every level (except possibly the last) is completely filled.

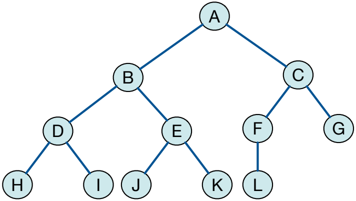
- The last level may be partially filled from left to right.



The height of a complete binary tree with n elements is $\lfloor \log_2 n \rfloor$

Storing Complete Binary Trees

Complete binary trees can easily be stored in an array.



A	B	C	D	E	F	G	H	I	J	K	L				

Notice that for an element at position x:

- The left child of the element at x is at position $2x$.
- The right child of the element at x is at position $2x + 1$
- The parent of the element at 2 is at position $\lfloor \frac{x}{2} \rfloor$

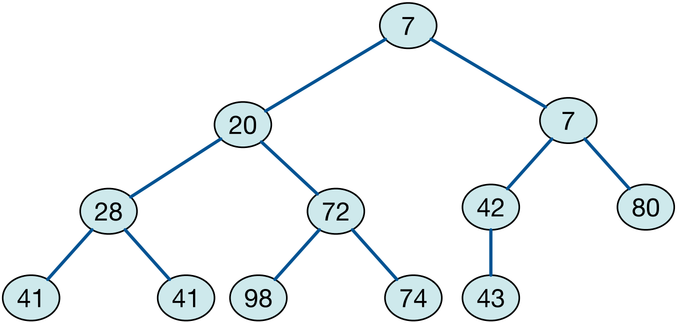
Binary Heap Order Property

For every vertex, v (except the root):

- key(parent(v)) $\leq$ key(v)
- Thus, the minimum key is at the root.

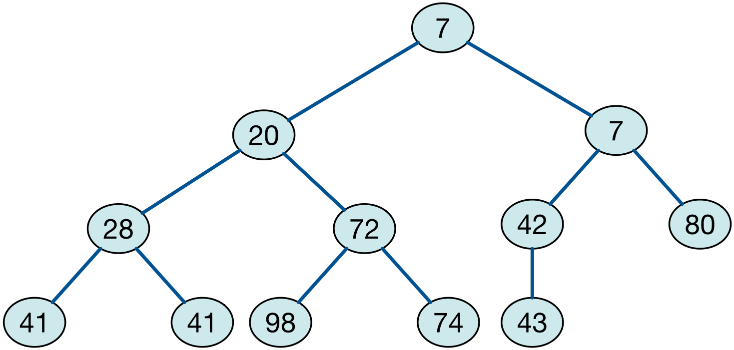
Any operation on the heap (insert, deleteMin) must maintain the order property.

Binary Heap Order Property



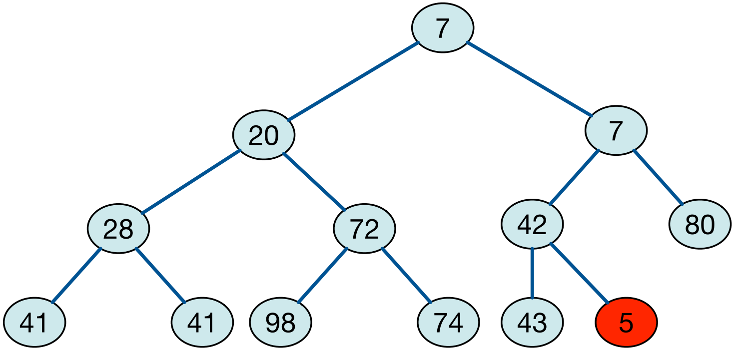
Binary Heap Insert Operation

Insert the new element into the heap at the next available leaf.



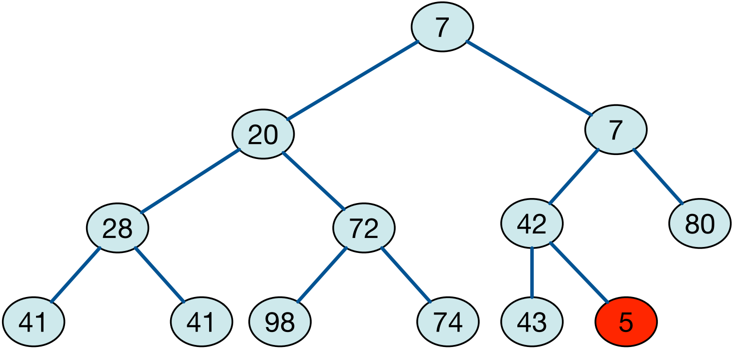
Binary Heap Insert Operation

Insert the new element into the heap at the next available leaf.



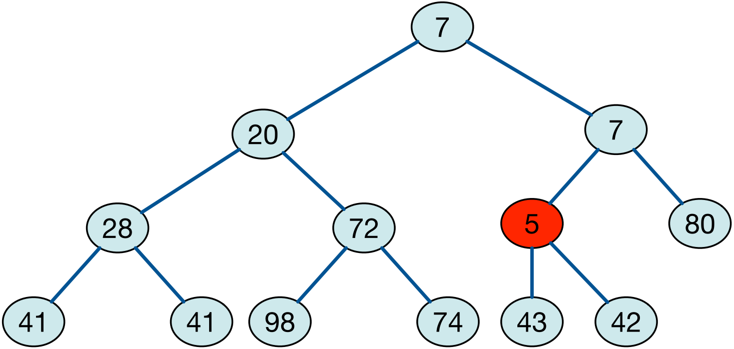
Binary Heap Insert Operation

As long as the heap order is not satisfied, “percolate” up.



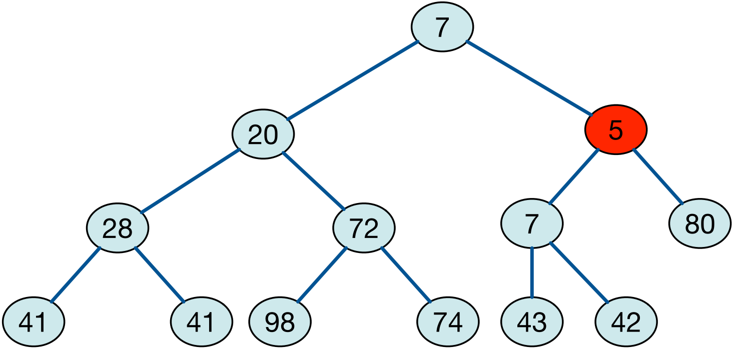
Binary Heap Insert Operation

As long as the heap order is not satisfied, "percolate" up.



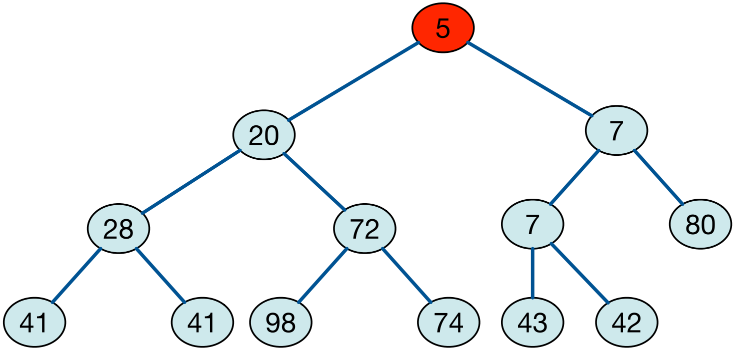
Binary Heap Insert Operation

As long as the heap order is not satisfied, “percolate” up.



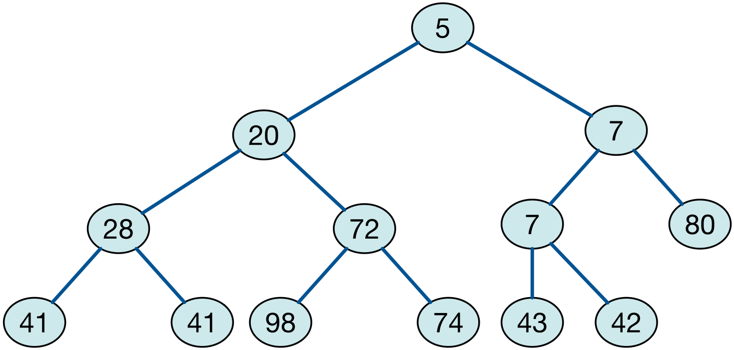
Binary Heap Insert Operation

As long as the heap order is not satisfied, “percolate” up.



Binary Heap Insert Operation

As long as the heap order is not satisfied, “percolate” up.



Insert Pseudocode

HeapInsert(H, x)

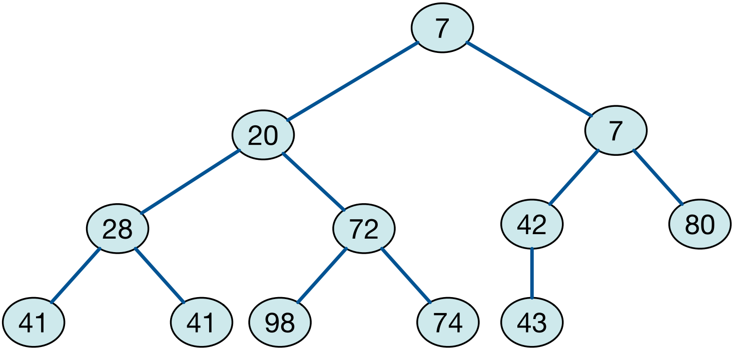
Input: A binary heap, H, and an element with value x.
Output: A new binary heap containing the element with value x.

Add a vertex with value x to the right of the farthest right leaf (on the last level) in H.
while $x <$ the value of x's parent **do**
    Swap the values of the respective vertices.

Can we (should we?) make this more formal?

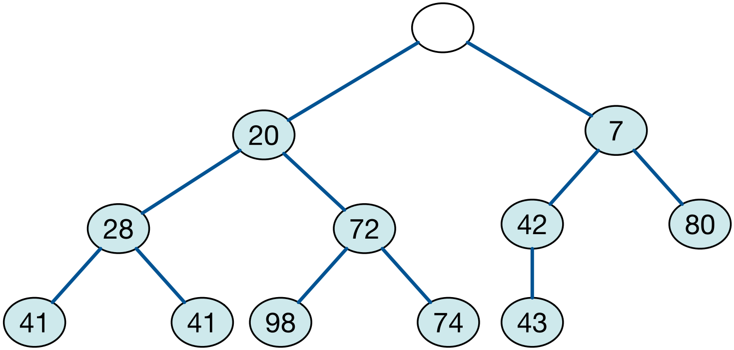
Binary Heap DeleteMin Operation

Move the last leaf element into the empty position at root.



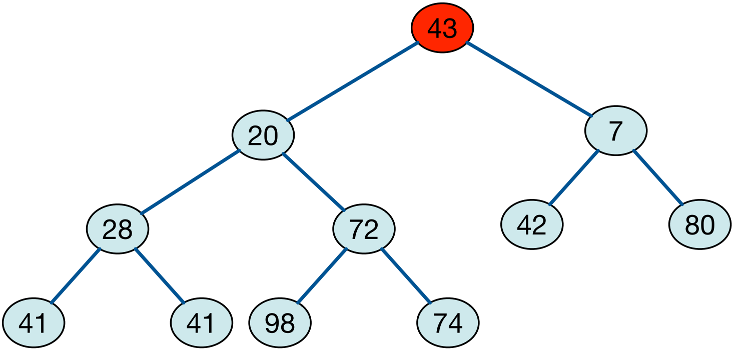
Binary Heap DeleteMin Operation

Move the last leaf element into the empty position at root.



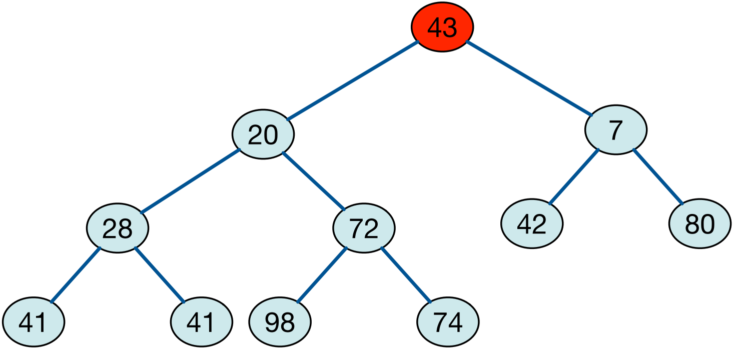
Binary Heap DeleteMin Operation

Move the last leaf element into the empty position at root.



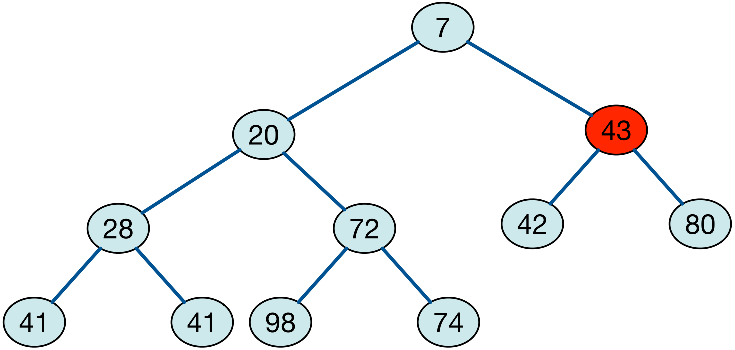
Binary Heap DeleteMin Operation

As long as heap order is not satisfied, "percolate" down (choose the min element to swap with).



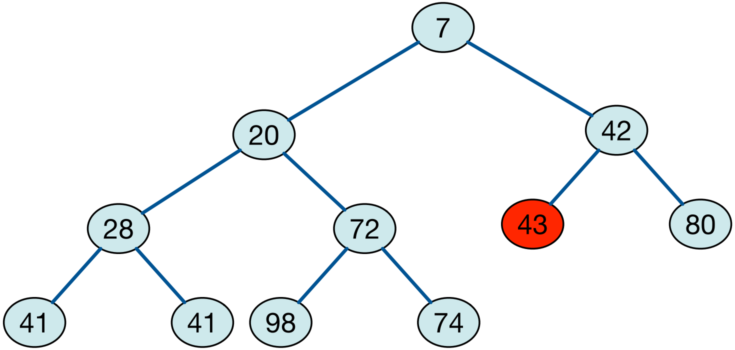
Binary Heap DeleteMin Operation

As long as heap order is not satisfied, "percolate" down (choose the min element to swap with).



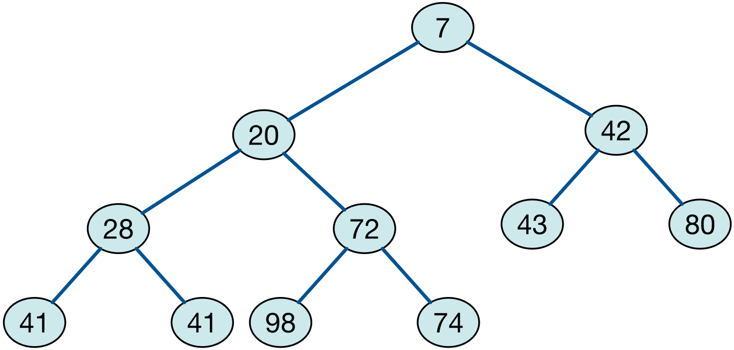
Binary Heap DeleteMin Operation

As long as heap order is not satisfied, "percolate" down (choose the min element to swap with).



Binary Heap DeleteMin Operation

As long as heap order is not satisfied, “percolate” down (choose the min element to swap with).



DeleteMin Pseudocode

HeapDeleteMin(H)

Input: A binary heap, H.

Output: A new binary heap with the original minimum element removed.

```
Delete the root vertex. (This creates a "hole".)
Replace the hole with the farthest right leaf (on the last level)
in H, x. (The tree is again complete binary tree.)
while value of x > value of x's children do
    Swap the values of x and the smaller of x's children.
```

Can we (should we?) make this more formal?

Heap Running Times

Note that the height of the heap is $\lfloor \log n \rfloor$

- *insert*: $O(\log n)$
- *deleteMin*: $O(\log n)$

Applications for Priority Queues

- Operating system scheduling.
- Prim's algorithm for minimum spanning tree.
- Huffman encoding.
- Bandwidth management.