CSE 250: Priority Queues, Heaps

Lecture 22

Oct 25, 2023

Reminders

- PA2: Implement **Map Routing**
  1. Create an adjacency list (discussed today)
  2. Find a path from A to B with the fewest intersections
  3. Find a path from A to B with the shortest distance
- PA2 implementation due Sun, Nov 5 at 11:59 PM

New ADT: Priority Queue

`PriorityQueue<E>` (`E` must be `Comparable`Comparable)

- `public void add(E e)`: Add `e` to the queue.
- `public E peek()`: Return the *least*least element added.
- `public E remove()`: Remove and return the *least*least element added.

Lazy Priority Queue

Base Data Structure: Linked List

- `public void add(T v)` $O(1)$
  Append `v` to the end of the linked list.
- `public T remove()` $O(N)$
  Traverse the list to find the least value and remove it.

Proactive Priority Queue

Base Data Structure: Linked List

- `public void add(T v)` $O(N)$
  Traverse the list to insert `v` in sorted order.
- `public T remove()` $O(1)$
  Remove the head of the list.

Sorting with a Priority Queue

```
1   public List<T> prioritySort(List<T> items,
2                               PriorityQueue<T> pqueue)
3   {
4     T[] out = new T[items.size];
5     for( item : items ){ pqueue.add(item) }
6     for( int i = 0; i < items.size; i++ )
7     {
8       out[i] = items.remove()
9     }
10    return Arrays.asList(out)
11  }
```

With a lazy, or proactive queue, this is $O(N^2)$?

Priority Queues

Operation	Lazy	Proactive
`add`	$O(1)$	$O(N)$
`remove`	$O(N)$	$O(1)$
`peek`	$O(N)$	$O(1)$

Can we do better?

Priority Queues

- **Lazy**
  Fast Enqueue, Slow Dequeue
- **Proactive**
  Slow Enqueue, Fast Dequeue
- **???**
  Fast(ish) Enqueue, Fast(ish) Dequeue

Priority Queues

Idea: Keep the priority queue "kinda" sorted.

- Keep larger items closer to the frontroot of the listtree.
- The closer we are to the front of the listtree, the more sorted it gets.

Challenge: How do we keep track of which items are sorted?

Trees

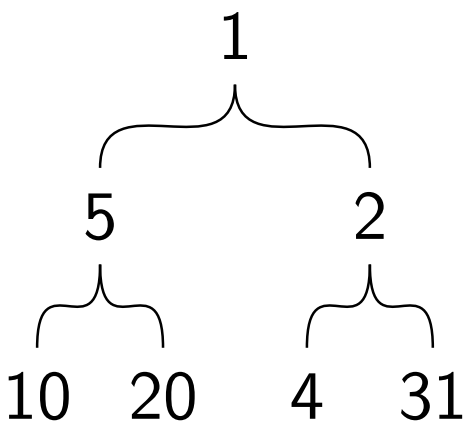
- **Child**
  An adjacent node connected by an out-edge
- **Leaf**
  A node with no children
- **Depth** of a node
  The number of edges from the root to the node
- **Depth** of a tree
  The maximum depth of any node in the tree
- **Level** of a node
  The depth + 1

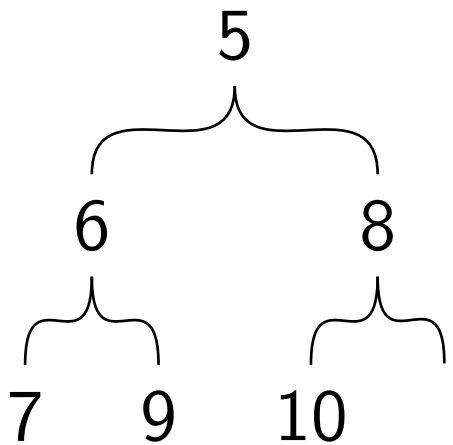
Priority Queue as a Tree

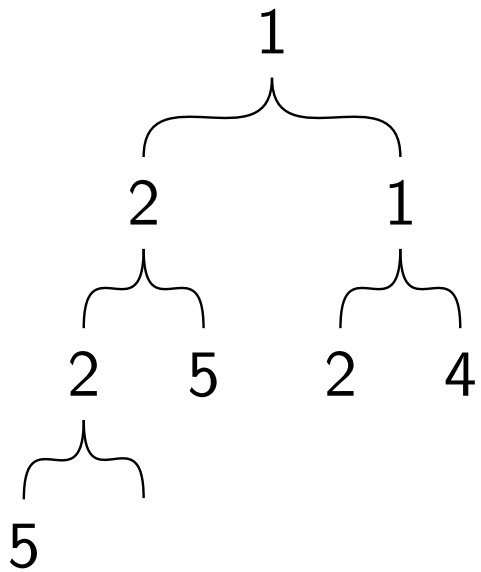
- **Directed** A directed edge in the tree means $\leq$
- **Binary** (max 2 children, easy to reason about)
- **Complete** (every 'level' except last is full)
  - For consistency, keep all nodes in the last level to the left.

This is a **Min-Heap**

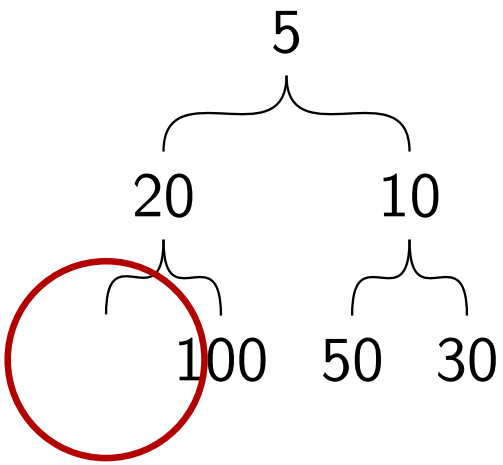
Valid Min-Heaps

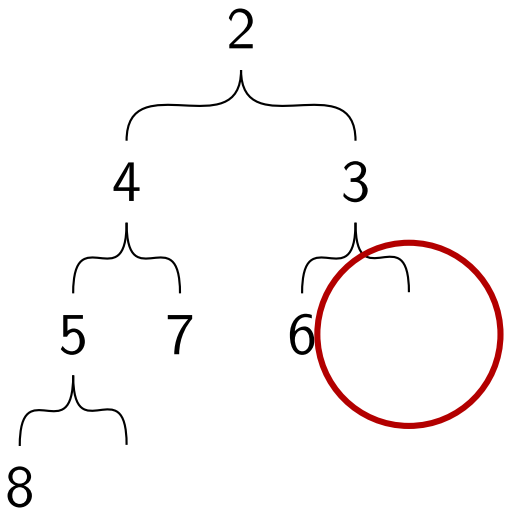


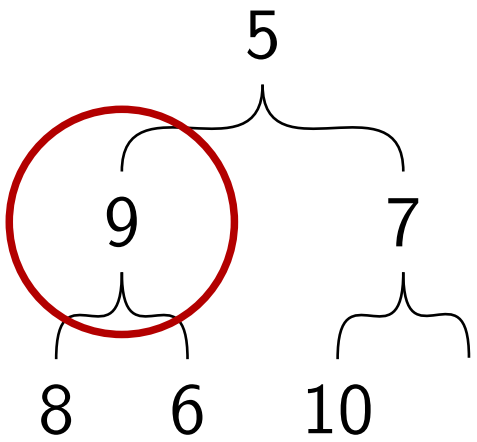




Invalid Min-Heaps







Heaps

What is the depth of a **binary** heap containing N items?

- Level 1: up to 1 item
- Level 2: up to 2 items
- Level 3: up to 4 items
- Level 4: up to 8 items
- Level 5: up to 16 items

  ...
- Level ℓ: up to 2^ℓ items

$$N \leq 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + \ldots + 2^{\ell_{max}}$$

What is the smallest allowable value of ℓ_{max}?

Heaps

What is the smallest allowable value of ℓ_{max}?

$$N \leq \sum_{i=1}^{\ell_{max}} 2^i$$

$$N \leq 2^{\ell_{max}+1} - 1$$

$$\frac{N}{2} + 1 = 2^{\ell_{max}}$$

$$\log\left(\frac{N}{2} + 1\right) = \ell_{max}$$

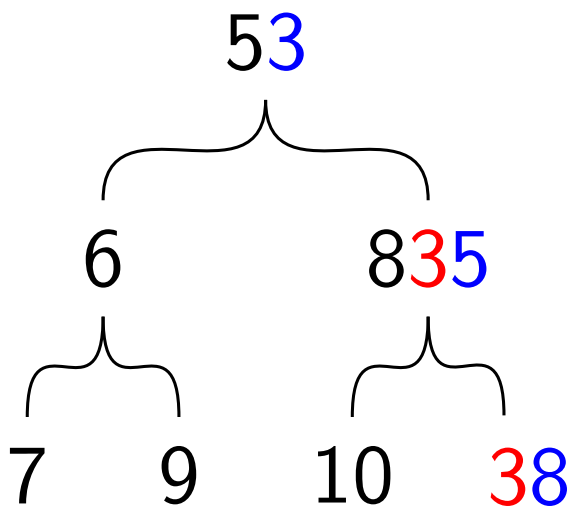
$$\ell_{max} = O(\log(N))$$

The Heap Data Structure

- `public void pushHeap(T element)`
  Place an item onto the heap.
- `public T popHeap()`
  Remove the least item from the heap.
- `public T peekHeap()`
  Peek at the least element on the heap.

pushHeap

Idea: Insert element at next available spot, then fix



How many steps did fixing it take?
(2)

pushHeap

```
1     public void fixUp(Node current)
2     {
3       Node parent = current.parent;
4
5       while(current.value > parent.value){
6
7         swap( current.value, parent.value );
8
9         current = parent; parent = current.parent;
10      }
11    }
```

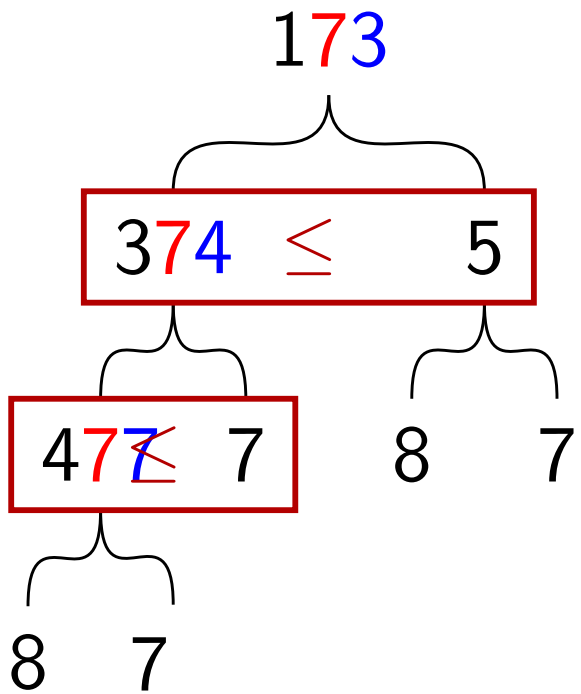
What's the complexity? (how many swaps are required?)

swaps = O(depth) =

$O(\log(N))$, because the tree is complete

popHeap

Idea: Replace root with 'last' element.



popHeap

```
1  public void fixDown(Node current)
2  {
3    Node left = current.leftChild;
4    Node right = current.rightChild;
5
6    while(current.value < left.value OR
7          current.value < right.value){
8
9      Node nodeToSwap = min(left, right);
10
11     swap( current.value, nodeToSwap.value );
12
13     current = nodeToSwap; left = current.left;
14                           right = current.right
15   }
16 }
```

What's the complexity? $(O(depth))$ $(= O(\log(N)$ for this tree)

Priority Queues

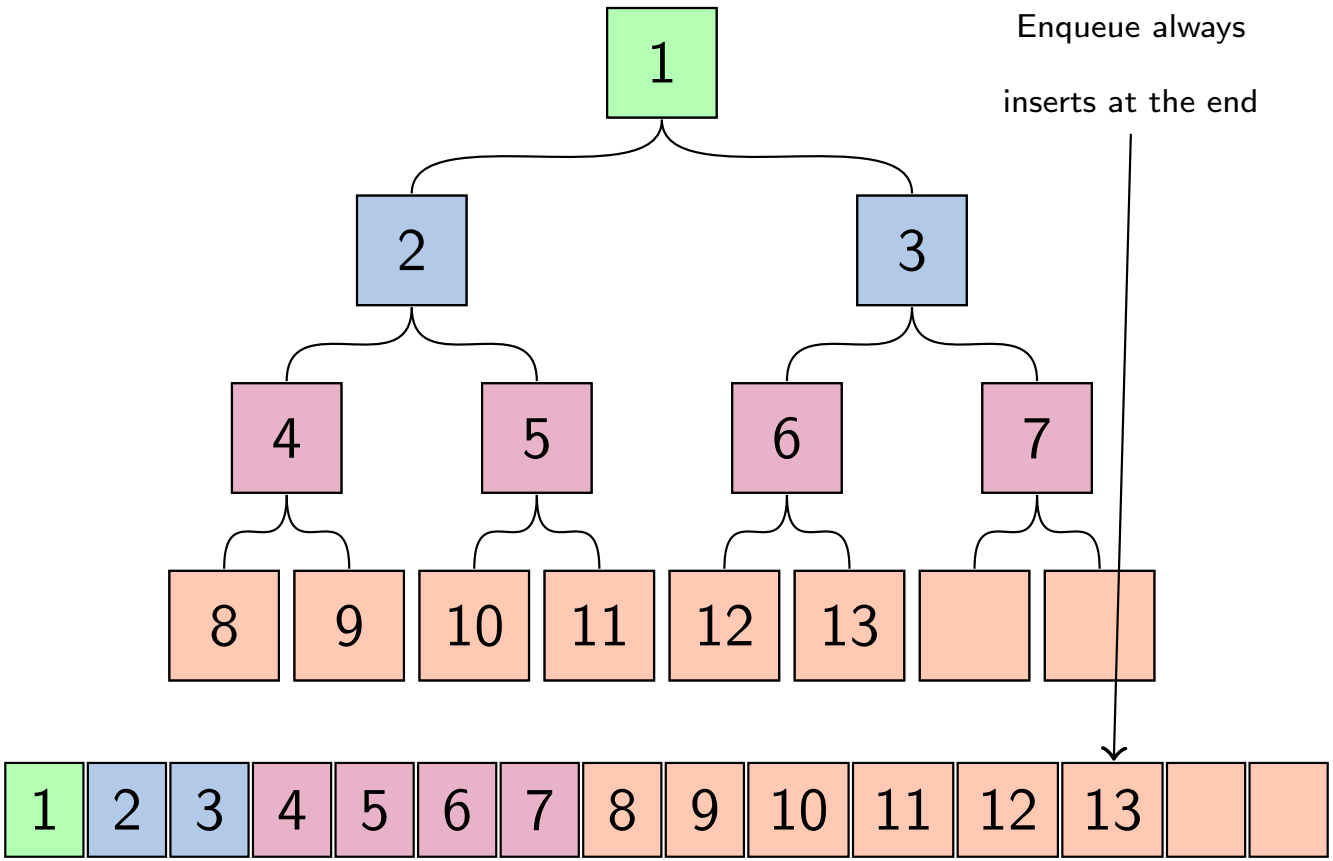
Operation	Lazy	Proactive	Heap
`add`	$O(1)$	$O(N)$	$O(\log(N))$
`remove`	$O(N)$	$O(1)$	$O(\log(N))$
`peek`	$O(N)$	$O(1)$	$O(1)$

Storing Heaps

- Each layer has a maximum size (2^ℓ)
- Each layer grows left-to-right
- Only the last layer grows

Idea: Use an `ArrayList` to store the heap.

Storing Heaps



Array Heap

- **pushHeap** Amortized $O(\log(N))$, Unqualified $O(N)$
  - Append to ArrayList $O(N)$; Amortized $O(1)$
  - `fixUp` $O(\log(N))$
- **popHeap** Unqualified $O(\log(N))$
  - Replace index 0 w/ Last Element $O(1)$
  - `fixDown` $O(\log(N))$

Heap Priority Queue

- `public void add(T elem)`
  - `pushHeap(elem)`
- `public T remove()`
  - `return popHeap();`
- `public T peek(T)`
  - Return the item at the top of the heap.