Admin

Pre-pre enrollment
- thanks all of you that are not potential CS majors for your patience!

Autocomplete assignment

---

Binary heap

A binary tree where the value of a parent is greater than or equal to the value of its children

Additional restriction: the tree must be complete!

Max heap vs. min heap

---

Binary heap - references

all nodes in a heap are themselves heaps

parent ≥ child

complete tree
What are heaps good for?

ExtractMax

Return and remove the largest element in the set. The rest of the data should stay as a heap.
ExtractMax

Remove the root

Sink

Fix a heap where the left/right are heaps, but the parent/child ordering might be violated at the parent node.
Sink
Fix a heap where the left/right are heaps, but the parent/child ordering might be violated at the parent node

Sink
Fix a heap where the left/right are heaps, but the parent/child ordering might be violated at the parent node

Sink: when are we done?
Fix a heap where the left/right are heaps, but the parent/child ordering might be violated at the parent node

sink/heapify/demote

```java
private void sink(int i) {
    // if we're not a leaf
    if( left(i) < heap.size() ) {
        // find the largest child
        int maxIndex = maxChildIndex(i);
        E current = heap.get(i);
        E maxChild = heap.get(maxIndex);
        if( maxChild.compareTo(current) > 0 ) {
            swap(i, maxIndex);
            sink(maxIndex);
        }
    }
}
```
What is the worst case runtime?

O(height of tree)

What is the worst case runtime?

O(log n)
What is the worst case runtime?

ExtractMax

```java
public E extractMax()
```
```
    { 
        E maxVal = data.get(1);
        data.set(1, data.get(data.size()-1));
        data.remove(data.size()-1);
        return maxVal;
    }
```

What is the worst case runtime? \( O(\log n) \)

Insert

How do we insert a value into a heap?

```
Insert
```
```
    Insert the value at the end of the array (or as the rightmost leaf)
```

<table>
<thead>
<tr>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>15</td>
</tr>
</tbody>
</table>
Insert

Swap the value up until it’s in the right place

25

Insert

Swap the value up until it’s in the right place

26

Insert

Swap the value up until it’s in the right place

27

Insert

Swap the value up until it’s in the right place

28
Swap the value up until it's in the right place

When do we stop?

Less than our parent or we're the root

```java
private void swim(int i) {
    if (i > 0) {
        T value = data.get(i);
        T parentVal = data.get(parent(i));
        if (value.compareTo(parentVal) > 0) {
            swap(i, parent(i));
            swim(parent(i));
        }
    }
}
```
What's the worst case runtime?  $O(\text{height of tree}) = O(\log n)$
What's the worst case runtime? $O(\log n)$

Heaps summarized

- Very good at extracting min/max (depending on heap ordering)

<table>
<thead>
<tr>
<th></th>
<th>best</th>
<th>worst</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>extractMax</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>insert</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>change node</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>

Heapsort

Could we sort data with a heap?

What would be the runtime (best, average, worst)?

<table>
<thead>
<tr>
<th></th>
<th>best</th>
<th>worst</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>extractMax</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>insert</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>change node</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>

Heapsort

- Build a heap out of the data (e.g., insert $n$ items into heap)
- Call extractMin $n$ times and add to answer
Heapsort runtime

Build a heap out of the data (e.g., insert n items into heap)

Call extractMin n times and add to answer

Best case?
Worst case?
Average case?

Heapsort runtime

Best case? $O(n)$ – when all items have the same value
Worst case? $O(n \log n)$
Average case? $O(n \log n)$

<table>
<thead>
<tr>
<th></th>
<th>best</th>
<th>worst</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>max</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>extractMax</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>insert</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>change node</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>

Heapsort

Build a heap out of the data (e.g., insert n items into heap)

Call extractMin n times and add to answer

Stable? No.
In-place? Not this implementation, but can be done without too much trouble

Sorting summarized

<table>
<thead>
<tr>
<th></th>
<th>in-place?</th>
<th>stable?</th>
<th>Best</th>
<th>Average</th>
<th>Worst</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selection</td>
<td>X</td>
<td></td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
<td>n swaps</td>
</tr>
<tr>
<td>Insertion</td>
<td>X</td>
<td>X</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
<td>use for partially ordered</td>
</tr>
<tr>
<td>Merge</td>
<td>X</td>
<td></td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>guaranteed, stable</td>
</tr>
<tr>
<td>Quick</td>
<td>X</td>
<td></td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>fastest in practice</td>
</tr>
<tr>
<td>Heap</td>
<td>X</td>
<td></td>
<td>$O(n)$</td>
<td>$O(n \log n)$</td>
<td>$O(n \log n)$</td>
<td>guaranteed, in-place</td>
</tr>
</tbody>
</table>
Priority Queues

Queues work well for keeping track of sequential ordering when everything is equivalent (e.g., waiting in line to get lunch!

Some queues everything is not equivalent (e.g., ER waiting room)

Priority queues support add/remove ordered by a weight/priority

Applications?

- process scheduling (e.g., ‘top’ command)
- network traffic scheduling
- Many algorithms
  - Search algorithms (A*)
  - Shortest paths algorithms (Dijkstra’s)
  - Minimum spanning trees (Prim’s)
  - Huffman codes

Priority queue interface

```java
public interface PriorityQueue<T extends Comparable<T>>{
    // Returns the smallest value in the queue if non-empty
    // Returns the smallest value in the priority queue
    public T extractMin();
    // Adds the specified item to the priority queue
    public void add(T data);
    // Returns the number of elements in the queue
    public int size();
    public boolean isEmpty();
}
```
Priority queue

two key methods:
- add
- extractMin (highest priority)

How can we do this?

See how many options you can come up with that have *different* runtimes for operations!

Option 1: unordered ArrayList

add: add to the end of the ArrayList

extractMin: search for the smallest, return and remove it

Worst case running times?

Option 1: unordered ArrayList

add: add to the end of the ArrayList

extractMin: search for the smallest, return and remove it

Worst case running times?
Option 1b: unordered LinkedList

add: add to the end of the linked list

extractMin: search for the smallest, return and remove it

Worst case running times?

Option 2: sorted order linked list

add: 

extractMin:

Worst case running times?

```java
public class SimpleLinkedListPriorityQueue<T extends Comparable<T>> implements PriorityQueue<T> {
    private LinkedList<T> pq = new LinkedList<T>();
    
    public void add(E e) {
    }
    
    public E extractMin(){
        if (pq.size() == 0) {
            throw new NoSuchElementException();
        }
        E min = pq.get(0);
        pq.removeFirst();
        return min;
    }
    
    public boolean isEmpty(){
        return pq.size() == 0;
    }
    
    public int size(){
        return pq.size();
    }
}
```
Option 2: sorted order linked list

- **add**: search for the correct location and insert
- **extractMin**: remove and return the first thing from the list

**Worst case running times?**

Option 3: heap

- **add/insert**: $O(\log n)$
- **extractMin**: $O(\log n)$

**Worst case running times?**
Priority queues summarized

Different scenarios may benefit from different implementations

Priority queue ≠ heap

<table>
<thead>
<tr>
<th></th>
<th>add</th>
<th>extractMin</th>
</tr>
</thead>
<tbody>
<tr>
<td>unordered linked list</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>sorted linked list</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>heap</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>