CS 180 - Heaps

Announcements:

- HW due Wednesday
- Next program will come out over the break sometime (due 1 week from Wed.)
- Ritter is closed this weekend (open again Monday) (no office hours Monday)
Every node has 0 or 2 children.
Array Based implementation:

Root is #1

For any node v with number n, left child gets number 2*n and right child gets 2*n+1
Priority Queue:

Supports the following operations:

- `insert(e)`: adds element `e` to the data structure

- `removeMax()`: removes the maximum element

- `maxItem()`: returns a reference to the maximum item in the priority queue

Also: size, empty, etc.
Implementing a priority queue with a heap:

A binary tree where:
- For every node \( v \) (other than the root), the key stored at \( v \) is less than or equal to the key stored at \( v \)'s parent.
- The tree is complete - levels 0 to \( h-1 \) have all possible nodes, and all internal nodes in level \( h-1 \) are on the left.
Picture - Max heap

level 0

level 1

level 2

level 3
So: insert

insert(17)
insert(50)
insert(14)
insert(14)
Remove Max:

\[ \text{removeMax}(\ast) \]
Running times:

Insert and removeMax will both run in time $O(h)$, where $h$ is the height of the tree.

How big can $h$ be?

How many nodes are on level $i$?

$2^i$ nodes on level $i$

Total # nodes: $n = \sum_{i=0}^{h} 2^i = 2^0 + 2^1 + 2^2 + \ldots + 2^h$

$\Rightarrow h = O(\log_2 n)$
### Running Times

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>size, empty</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>insert</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>removeMax</td>
<td>$O((\log n)^2)$</td>
</tr>
<tr>
<td>max Item</td>
<td>$\alpha(n)$</td>
</tr>
</tbody>
</table>
Now to code it:

```cpp
template <typename ItemType>
class Heap {
private:
    ItemType* _data;
    int _size;
    int _capacity;

public:
    Heap() : _data(new ItemType[1]), _size(0), _capacity(1) {};
```
void insert(const ItemType & val)
if (size == _capacity)
    
    _capacity = 2 * _capacity;
    ItemType* newdata = new ItemType[_capacity];
    for (int i = 0; i < size; i++)
        newdata[i] = data[i];
    delete data;
    data = newdata;
    
data[Size] = val;
    size++;

int current = size - 1;
int parent = (current + 1) / 2;

bubble it up