Heaps

- O(n) algorithm for building a heap
- Change key (indexed priority queue)

(binary) heap is a common implementation of the priority queue ADT. A heap is a "complete binary tree", such that each node holds a key (and an object).

min heap - the key of a parent node is less than the keys of its children. Hence the root holds the smallest key.

Resources for today
- see my lecture 30 from COMP 250 (my lectures 28, 29 cover standard COMP 250 heaps)

"Queue" - add an object to end of the list, remove object from front.

"Priority Queue" - remove object with highest priority (defined by a "key")

[ A priority queue is an ADT. A heap is a common implementation of a priority queue. ]

Example:

keys (priorities) are numbers
We ignore associated objects.

<table>
<thead>
<tr>
<th>object</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>
Array-based implementation

Recall from COMP 250

```
upHeap(i) used by add(key)

downHeap(i) used by remove(key)
```

Both are $O(\log n)$.

buildHeap {
  for $i = 1$ to $n$
    upHeap($i$)
}

This algorithm takes $O(n \log n)$ time.

Intuitively obvious (?) since tree has height $\log n$ and most elements are near the leaves. But let's formalize this!

Consider a complete binary tree of height $h$, with all levels full.

Binary trees should not be drawn like this:

Rather, they should be drawn like this because level $i$ has $2^i$ nodes.

level $i$ has $2^i$ nodes: $2^1$, $2^2$, $2^3$, ..., $2^{i-1}$

number of nodes: $n = 2^{h+1} - 1$

height of tree: $h = \log(n+1) - 1$

buildHeap {
  for $i = 1$ to $n$
    upHeap($i$)
}

Worst case: each element "bubbles up" all the way to the root (element at depth $i$ is swapped $i$ times).

$\Rightarrow$ total # swaps = sum of depths
e.g. tree height \( h = 5 \)

**Sum of depths**

\[
\sum_{i=1}^{n} \log i \leq \log(n) \leq \sum_{i=0}^{n} 2^i \leq \sum_{i=0}^{\log n} x^i = \frac{x^{\log n + 1} - 1}{x - 1} \leq n 2^0 = n
\]

(suppose \( n = 2^{2+1} - 1 \))

Use a trick (calculus)

\[
\sum_{i=0}^{n} x^i = \frac{1}{1-x} x^i
\]

Check for yourself:

\[
\sum_{i=0}^{k} x^i = \frac{x^{k+1} - 1}{x - 1}
\]

... use quotient rule from calculus

... and substitute \( x = 2 \)

**Faster way to build a heap**

for \( i = n/2 \) down to 1

down Heap(i)

I will show this algorithm is \( O(n) \).

**Example (n=6)**

**Initial**

**Final**
for $i = \lfloor n/2 \rfloor$ down to 1
down Heap(n, i)

Claim: buildHeap takes time $O(n)$

Proof:

Suppose the heap has height $h$.
Downheaping a node at level $l$ requires at most $h-l$ swaps
($h-l$ is the height of the node.)

Summary:

worst case for total number of swaps
= sum of node heights

= \sum_{k=0}^{h} (h-k) \cdot 2^k

= \sum_{k=0}^{h} \frac{k}{2^k}

= \sum_{k=0}^{h} \frac{k}{2^k}

= n - \log(n+1)

For $i = 1$ to $n$
upHeap(i)

For $i = \lfloor n/2 \rfloor + 1$
downHeap(i)

$O(n \log n)$

$O(n)$

Heaps

- $O(n)$ algorithm for building a heap
- Change Priority, indexed priority queues

Each node is

- object (or name of object)

- child handled implicitly by array indexing

- left, right child
Problem: how to change the priority of some object?

The heaps we have seen (including Java PriorityQueue\(\langle E\rangle\)) do not support change priority (object, new priority).

Indeed, find(object) is \(O(n)\).

To change the priority of some object, we need to know where that object is in the heap.

\[
\begin{array}{c}
3, \text{`cat'} \\
8, \text{`pig'} \\
5, \text{`frog'} \\
10, \text{`elk'} \\
9, \text{`dog'} \\
7, \text{`mouse'} \\
\end{array}
\]

\(\text{e.g. ChangePriority}(4, \text{`dog'})?\)

What do we need to add to previous slide to have an indexed heap?
Assignment 1 posted today.

due Sunday Jan. 26

(in 10 days)

What if we want to change a priority?

\[ \text{increase } \Rightarrow \downarrow \text{Heap}(i) \]

\[ \text{decrease } \Rightarrow \uparrow \text{Heap}(i) \]