Heaps & Heapsort

Priority Queue

Collection of items with associated key values (ordered universe, e.g., integers).

Basic Ops:

Insert — add item to the PQ

DeleteMin — remove item with minimum key value from PQ

FindMin — return item with min key value. PQ is unchanged (assumes nonempty)

Empty — tests if PQ is empty

DecreaseKey — update the key value of an item in the PQ to a smaller value.

Min PQ

Max PQ: reverse order of keys
Max PQ supports:

- Insert
- DeleteMax
- FindMax
- Empty
- IncreaseKey

Same as min PQ
with comparisons reversed.

Implementation:

Array (binary heap)

Binary heap: full binary tree with items stored at nodes.

Binary tree as full as possible, all leaves are on bottom level or one up, and all bottom level leaves are as far left as possible.

Implement full binary tree as an array $A[1..n]$, where $n$ elements in the tree.
root: \(A[1]\)
index \(i\) into \(A\)
left(\(i\)) = \(2i\)
right(\(i\)) = \(2i + 1\)
parent(\(i\)) = \(\lfloor i/2 \rfloor \) (\(i \geq 2\))

Max heap (implements max PQ)
Min heap is similar

A FBT is in max heap order if for every \(i \geq 2\)

\[ A[i] \leq A[\text{parent}(i)] \]

FindMax(\(A\)) — returns \(A[1]\)
A has an attribute
heapSize(\(A\)) — # of elements in heap stored in A
Heap is \(\{A[1] \ldots \text{heapSize}(A)\}\)
Empty($A$) — true iff
\[ \text{heapSize}[A] = 0 \]

Insert($A$, $x$) — insert
item $x$ into $A$
(\text{let key}[$x$] be $x$’s
key value)

```
    40
   / \   \n45   40
 / \   / \   \n40  28 22 12
```

\[ x = 45 \quad \text{“cascading up”} \]

Time is $\Theta(\lg n)$

\text{depth of the tree}

DeleteMax($A$) — remove
root,
insert item
at index $A[\text{heapSize}[$A$]]$
into heap by “cascading
down”:

\[ \text{decement heapSize}[A] \]

IncreaseKey($A$, $i$, $k$)
// assume $k \geq \text{key}[$A[$i$]]
// $x$ is an item in $A$
\[ \text{key}[^A[i]] := k \]
cascade $A[i]$ up.

<table>
<thead>
<tr>
<th>Op</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>FindMax</td>
<td>(\Theta(1))</td>
</tr>
<tr>
<td>DeleteMax</td>
<td>(\Theta(\lg n))</td>
</tr>
<tr>
<td>Insert</td>
<td>(\Theta(\lg n))</td>
</tr>
<tr>
<td>Empty</td>
<td>(\Theta(1))</td>
</tr>
<tr>
<td>Increase Key</td>
<td>(\Theta(\lg n))</td>
</tr>
</tbody>
</table>

HeapSort

Input: A[1..n] of numbers
Output: A arranged in increasing order

1. Make A a max heap
   (rearrange elements into max heap order)
   MakeMax Heap

2. Do \(n\) times:
   \[
   \begin{align*}
   &x := \text{FindMax}(A) \\
   &\text{DeleteMax}(A) \\
   &\text{heapSize}[A] := \frac{n - 1}{n - 2} \\
   &\sum_{i=1}^{n} \lg i = \Theta(n \lg n)
   \end{align*}
   \]

Actual time

Part 1:

\[
\text{heapSize} [A] := 0
\]

For \(i := 1\) to \(n\) do

\[
\text{Insert}(A, A[i])
\]

\(O(n \lg n)\)

\(\Theta(n \lg n)\) time

Heapsort takes \(\Theta(n \lg n)\) time
Part 1 can be done in time $O(n)$.  

For $i := n$ downto 1 do bottom-up  

Make subtree rooted at index $i$ into a max heap