http://www.cs.unh.edu/~ruml/cs758
Heaps
1. Finding the min
1. Finding the min
2. Finding the min with insertions
1. Finding the min
2. Finding the min with insertions
3. Finding the min with insertions and deletions
- asst 1
- asst 2
Invariant: parent comes before (or equal to) children
parent $i = \frac{(\text{child } i) - 1}{2}$

left child of $i = 2i + 1$

right child of $i = 2i + 2$

automatic balance!
1. insert at end
1. insert at end
2. re-establish invariant by
1. insert at end
2. re-establish invariant by pulling up if necessary
assume heap except at $i$, and $A[i]$ might be too small
so consider pulling $A[i]$ up

pullup($i$)
1. if $A[i]$ comes before $A[parent]$
2. exchange $A[i]$ with $A[parent]$
3. pullup($parent$)

invariant: initialization, maintenance, termination
Extract Min

1. remove first elt
2. copy last into first
1. remove first elt
2. copy last into first
3. re-establish invariant by
1. remove first elt
2. copy last into first
3. re-establish invariant by pushing down if necessary
Extract Min

1. remove first elt
2. copy last into first
3. re-establish invariant by pushing down if necessary

heapsort
assume heap except at $i$, and $A[i]$ might be too large
so consider pushing $A[i]$ down

pushdown($i$)

1. $\text{smallest} \leftarrow \text{index of smallest among } i \text{ and children}$
2. if $\text{smallest} \neq i$ then
3. exchange $A[i]$ with $A[\text{smallest}]$
4. pushdown($\text{smallest}$)

invariant: initialization, maintenance, termination
Correctness
What’s the space complexity?
What’s the time complexity?
More Heaps
Given array, how to form heap?
Given array, how to form heap?

Can we do better than $\Theta\left(\frac{n}{2} \lg \frac{n}{2}\right) = \Theta(n \lg n - n) = \Theta(n \lg n)$?
Given array, how to form heap?

Can we do better than $\Theta\left(\frac{n}{2} \lg \frac{n}{2}\right) = \Theta(n \lg n - n) = \Theta(n \lg n)$?

bottom up:

1. for $i$ from $\frac{\text{length}}{2} - 1$ to 0
2. pushdown($i$)

how long does this take?
The height of a node is (longest) distance to a leaf

\[ \sum_{h=0}^{\lg n} (O(h) \times \text{\#-nodes-with-height-h}) \]
**height** of a node is (longest) distance to a leaf

\[
\sum_{h=0}^{\lg n} (O(h) \times \#\text{-nodes-with-height-}h) = O(n) \sum_{h=0}^{\lg n} \frac{h}{2h+1}
\]

There are \(\frac{n}{2^{h+1}}\) nodes with height \(h\).
**height** of a node is (longest) distance to a leaf

\[
\sum_{h=0}^{\lg n} (O(h) \times \#-nodes-with-height-h)
\]

There are \(\frac{n}{2^{h+1}}\) nodes with height \(h\).

\[
\sum_{h=0}^{\lg n} O(h) \frac{n}{2^{h+1}} = O(n \sum_{h=0}^{\lg n} \frac{h}{2^{h+1}})
\]

\[
\sum_{h=0}^{\infty} \frac{h}{2^h} = 2
\]
(longest) distance to a leaf

\[
\sum_{h=0}^{\lg n} (O(h) \times \#-nodes-with-height-h)
\]

There are \(\frac{n}{2^{h+1}}\) nodes with height \(h\).

\[
\sum_{h=0}^{\lg n} O(h) \frac{n}{2^{h+1}} = O(n \sum_{h=0}^{\lg n} \frac{h}{2^{h+1}})
\]

\[
\sum_{h=0}^{\infty} \frac{h}{2^h} = 2
\]

\[
O(n \sum_{h=0}^{\lg n} \frac{h}{2^{h+1}}) = O(n \sum_{h=0}^{\infty} \frac{h}{2^h}) = O(n)
\]
resize by doubling!
resize by doubling!

‘amortized’ analysis: the ‘accounting method’

1. start half full, with zero credit
2. each insertion costs 3:
   (a) insert self now
   (b) eventually move self when full
   (c) eventually move an existing elt when full
3. when full, have credit for each item
4. now half full, with zero credit
‘amortized’ analysis: the ‘aggregate method’

Let \( c_i = i \) if \( i - 1 \) is a power of 2, 1 otherwise.

\[
\sum_{i=1}^{n} c_i \leq n + \sum_{j=0}^{\lg n} 2^j \\
< n + 2n \\
< 3n
\]
Problems

Heaps

More Heaps
- Creation
- Creation Time
- Sizing the Array
- Amortization 2

Problems
- EOLQs

1. Finding the min
2. Finding the min with insertions
3. Finding the min with insertions and deletions
4. Finding the $k$th largest
What’s still confusing?
What question didn’t you get to ask today?
What would you like to hear more about?

Please write down your most pressing question about algorithms and put it in the box on your way out.

Thanks!