Lecture 30
Heaps 3

- Sum of depths
- Sum of heights
- $O(n)$ algorithm for building a heap

Binary trees should not be drawn like this:

They should be drawn like this:

because level $k$ has $2^k$ nodes.

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

Level $k$ has $2^k$ nodes: $2^1, \cdots, 2^{k+1} - 1$

Number of nodes: $n = \sum_{i=1}^{h+1} - 1$

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

What is sum of depths of nodes?

Note: average depth $= \frac{1}{n} \cdot \text{(sum of depth)}$

What is sum of heights of nodes?

Note: average depth $= \frac{1}{n} \cdot \text{(sum of depth)}$

Sum of depths $= \sum_{i=1}^{n} \log i$

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

Inserting $n$ nodes into a binary tree.

Best case (for number of nodes traversed): sum of depths.
\[
\begin{align*}
\sum_{l=0}^{h} l \cdot 2^l &= 2 \sum_{l=0}^{h} l \cdot 2^{l-1}, \quad x=2 \\
&= 2 \sum_{l=0}^{h} l \cdot x^{l-1}, \quad x=2 \\
&= 2 \sum_{l=0}^{h} \frac{d}{dx} \frac{x^l}{8} \\
&= 2 \frac{d}{dx} \left( \frac{x^{h+1} - 1}{x - 1} \right) \\
\text{and substitute } x=2 \\
\sum \text{of depths} &= (h-1)2^{h+1} + 2 \\
&= (\log(n+1) - 2)(n+1) + 2 \\
\text{which is } O(n \log n) \text{ and } \Omega(n \log n).
\end{align*}
\]

Generalizing to complete \(k\)-ary trees (recall Assignment 3 tried)

number of \(= \sum_{l=0}^{h} k^l \)

sum of depths \(= \sum_{l=0}^{h} k^l \cdot l \)

Use same trick \(x = k \).

\(\Rightarrow\) sum of depths is \(O(n \log n)\) \(\Omega(n \log n)\)

Parent-child index relations

\[
\begin{align*}
\text{parent} &= \text{child} / 2 \\
\text{left child} &= 2 \cdot \text{parent} \\
\text{right child} &= 2 \cdot \text{parent} + 1
\end{align*}
\]

Claim: left child of \(i\) is \(2^i \cdot i\).

\[
\begin{align*}
\text{Proof: Let } i \text{ be the } m \text{th node in level } l. \\
\text{Then } i &= 2^l + m - 1. \\
\text{But the last node of level } l \text{ is } 2^{l+1} - 1. \\
\text{Thus, the children of nodes } 2^l, \ldots, i \text{ are} \\
2^{l+1}, \ldots, 2^{l+1} + 2m - 1. \\
\text{Thus the left child of } i \text{ is } 2^{l+1} + 2m - 2. \\
&= 2(2^l + m - 1) \\
&= 2^i \cdot i.
\end{align*}
\]
Last lecture: `buildHeap()`

for \( i = 1 \) to \( n \)

```
upHeap( i )
```

worst case: total number of swaps \( t(n) \)

\[ t(n) < n \log n \]

A second way to build a heap

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

```
downHeap(a, n, i)
```

**downHeap(a, n, k)?**

\( i = k \)

while \((2 \times i \leq n)\)

```
child = 2 \times i

// left child

if (child < n)
    if a[child+1] < a[child]

        child++
    
        if a[child] < a[i] {
            swap keys (a, i, child)
        }
```

Example

1. **Summary**

Most nodes are near level \( h \)

```
for \( i = 1 \) to \( n \)
upHeap(a, n, i)
```

\( O(n \log n) \)

for \( i = n \) to 1
downHeap(a, n; i)

\( O(n) \)

faster!