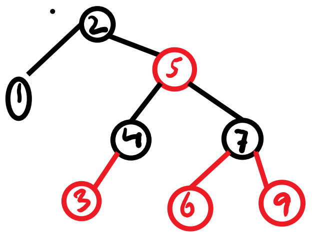


CS 201 Data Structure II (L2 / L5)

Meldable Heaps

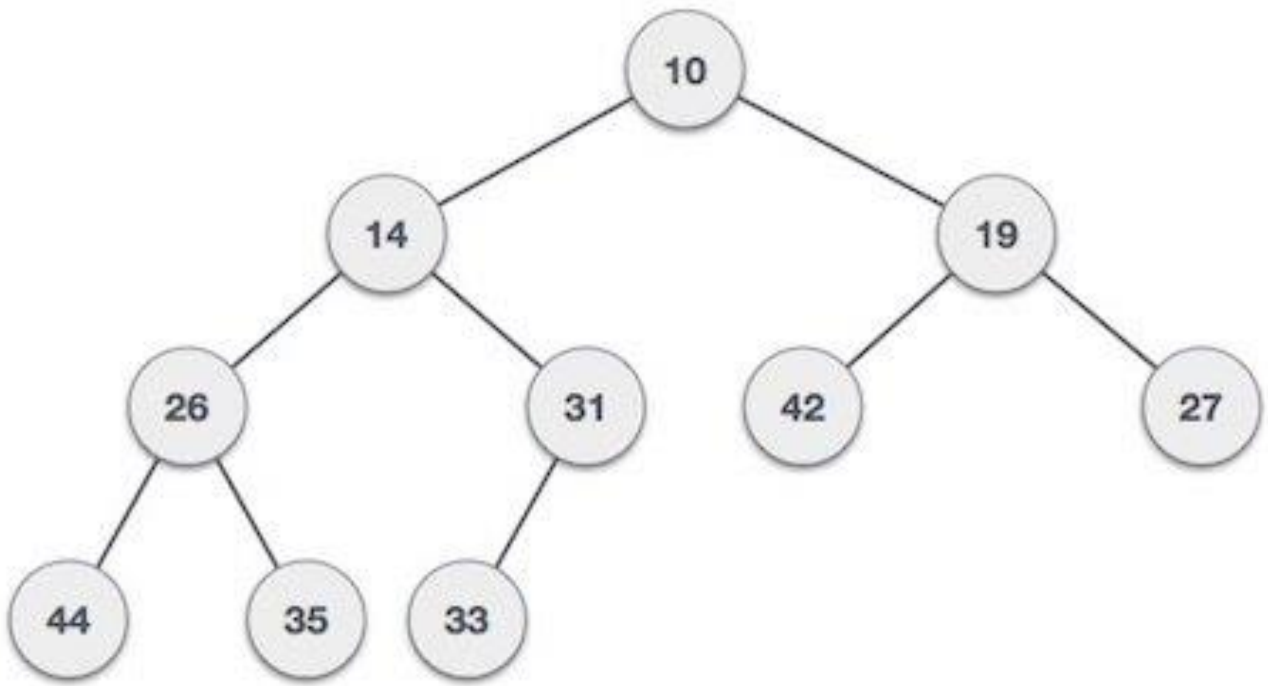
Chapter 10, Open Data Structures

Muhammad Qasim Pasta

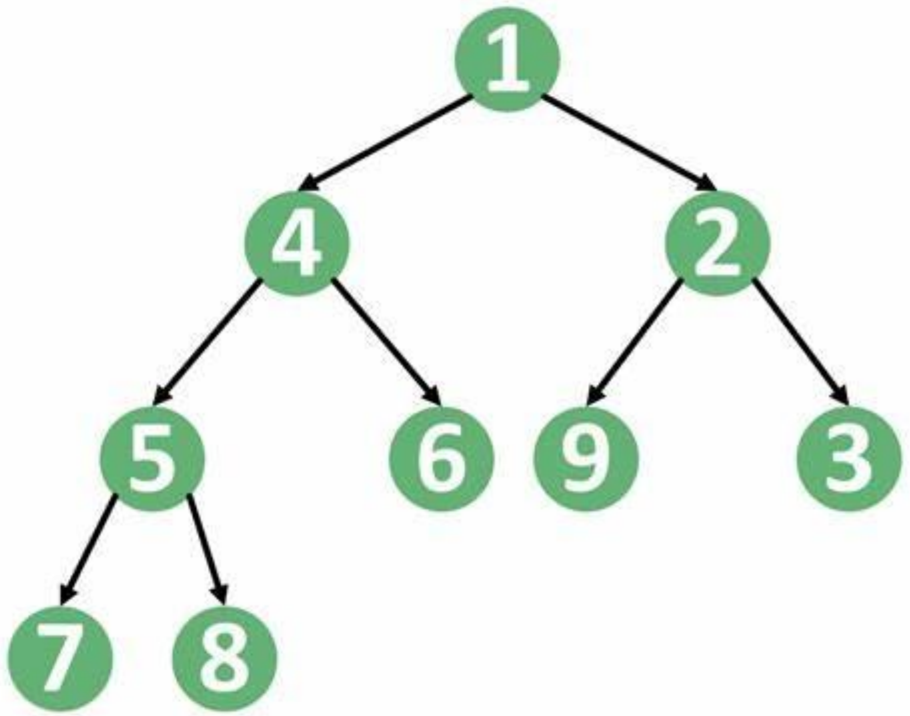
qasim.pasta@sse.habib.edu.pk

Slides are designed to be filled during the lectures. Some details are intentionally mentioned to be discussed in the class. These slides should not be used as reading.

How to merge two heaps?



h_1



h_2

Melding two heaps:

merge (h1, h2)

IF h1 is nothing THEN return h2

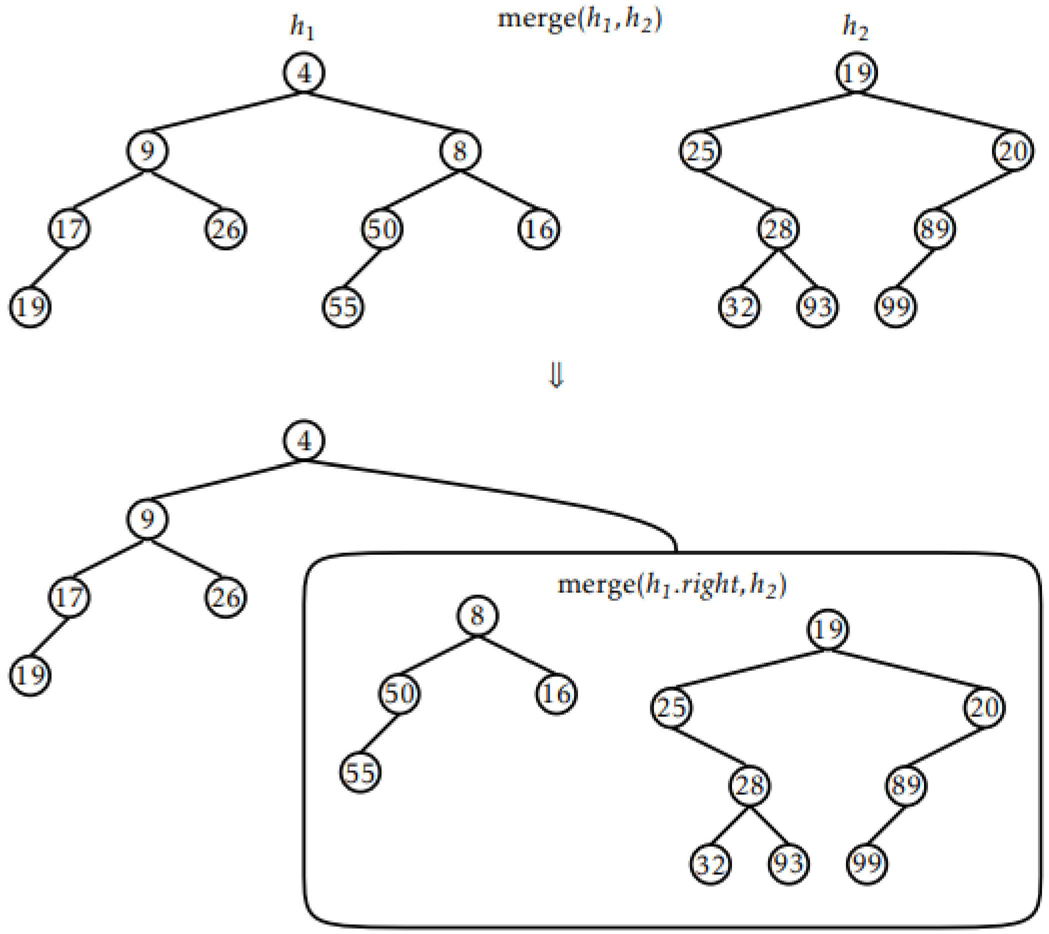
IF h2 is nothing THEN return h1

IF h2.x < h1.x THEN (h1, h2) = (h2, h1)

h1.right = merge(h1.right, h2)

h1.right.parent = h1

RETURN h1



Without biasness!

merge(h_1,h_2)

    if $h_1 = nil$ **then return** h_2

    if $h_2 = nil$ **then return** h_1

    if $h_2.x < h_1.x$ **then** $(h_1, h2) \leftarrow (h_2, h_1)$

    if random_bit() **then**

        $h_1.left \leftarrow \mathrm{merge}(h_1.left, h_2)$

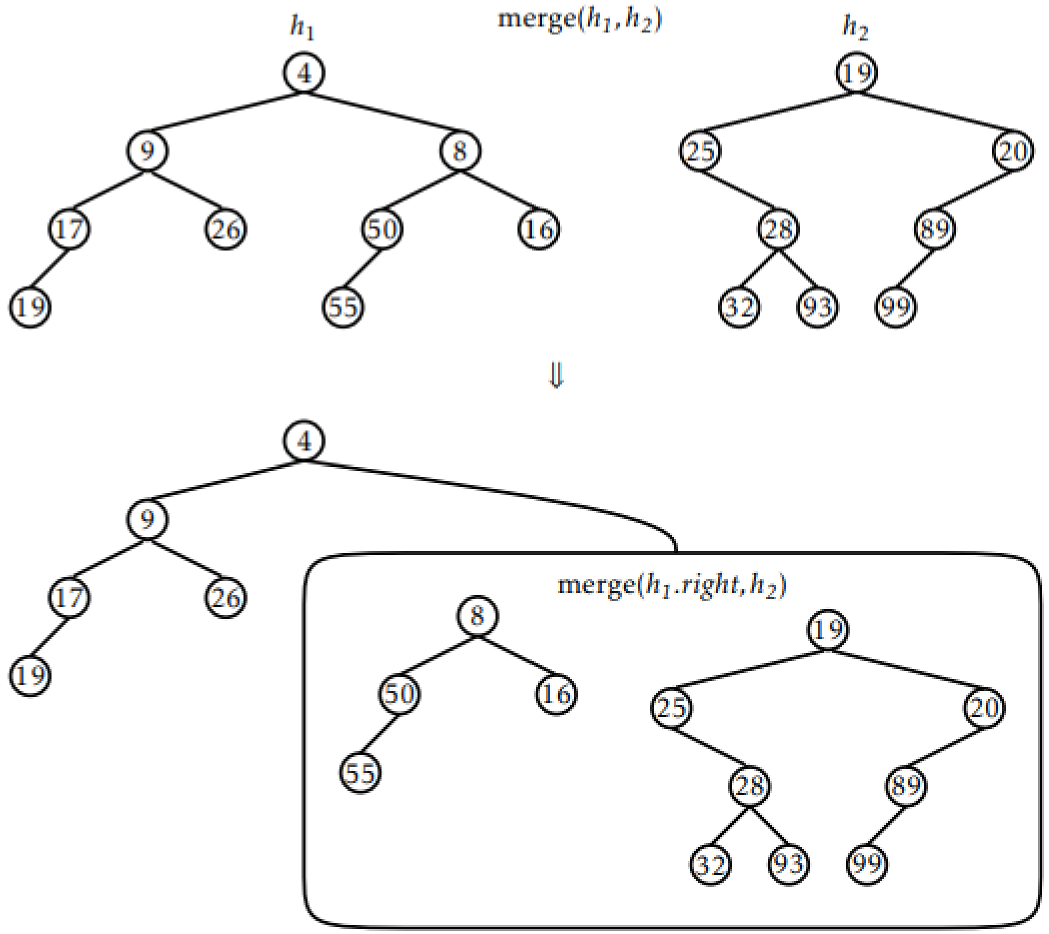
        $h_1.left.parent \leftarrow h1$

    else

        $h_1.right \leftarrow \mathrm{merge}(h_1.right, h_2)$

        $h_1.right.parent \leftarrow h1$

    return h_1



Add/Remove :

Add a new element:

- Merge the new element with existing heap

$$
\begin{array}{l}
\text{add}(x) \\
\quad u \leftarrow \text{new\_node}(x) \\
\quad r \leftarrow \text{merge}(u, r) \\
\quad r.parent \leftarrow nil \\
\quad n \leftarrow n + 1 \\
\quad \textbf{return } true
\end{array}
$$

Remove an element:

- Remove the root node by merging left and right sub trees

$$
\begin{array}{l}
\text{remove}() \\
\quad x \leftarrow r.x \\
\quad r \leftarrow \text{merge}(r.left, r.right) \\
\quad \textbf{if } r \neq nil \textbf{ then } r.parent \leftarrow nil \\
\quad n \leftarrow n - 1 \\
\quad \textbf{return } x
\end{array}
$$

Other Operations:

- Decrease Key: change the value of an element
  - How?
- Delete an element:
  - How?

Analysis

Theorem 10.2. *A MeldableHeap implements the (priority) Queue interface. A MeldableHeap supports the operations* add(x) *and* remove() *in* $O(\log n)$ *expected time per operation.*

Lemma 10.1. *The expected length of a random walk in a binary tree with n nodes is at most* $\log(n+1)$.

$$
\begin{aligned}
\mathrm{E}[W] &= 1 + \frac{1}{2}\log(n_1+1) + \frac{1}{2}\log(n_2+1) \\
&\le 1 + \log((n-1)/2+1) \\
&= 1 + \log((n+1)/2) \\
&= \log(n+1) \ .
\end{aligned}
$$