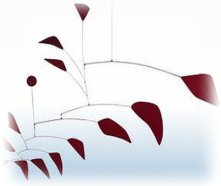


EECS 4101-5101
Advanced Data Structures

Shahin Kamali

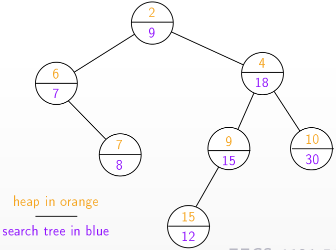
Topic 6: Randomized Data Structures (Treaps)

York University

Picture is from the cover of the textbook CLRS.

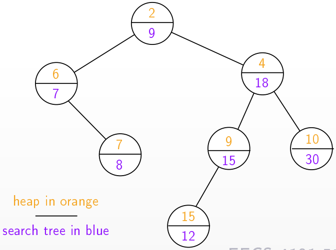
Treap Dictionary Data Structure

- A binary tree in which each node has a **key** and a **priority**
  - Keys have binary search tree property: each node's key is larger than its left and smaller than its right.
  - Priorities have heap property: each node priority is smaller than its parent (or could be larger to form a max-treap)



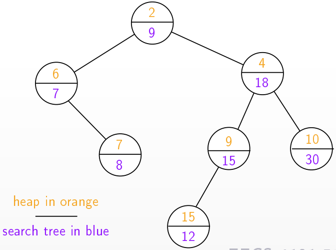
Treap Dictionary Data Structure

- A binary tree in which each node has a **key** and a **priority**
  - Keys have binary search tree property: each node's key is larger than its left and smaller than its right.
  - Priorities have heap property: each node priority is smaller than its parent (or could be larger to form a max-treap)
- Treaps are used for implementing dictionaries: keys are dictionary keys and priorities are chosen randomly!



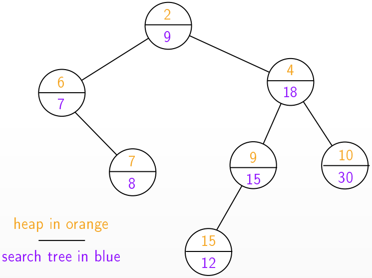
Treap Dictionary Data Structure

- A binary tree in which each node has a **key** and a **priority**
  - Keys have binary search tree property: each node's key is larger than its left and smaller than its right.
  - Priorities have heap property: each node priority is smaller than its parent (or could be larger to form a max-treap)
- Treaps are used for implementing dictionaries: keys are dictionary keys and priorities are chosen randomly!
  - *search*(k) is identical to seacrh in a BST $\rightarrow$ it takes $O(h)$, where h is the height of the tree



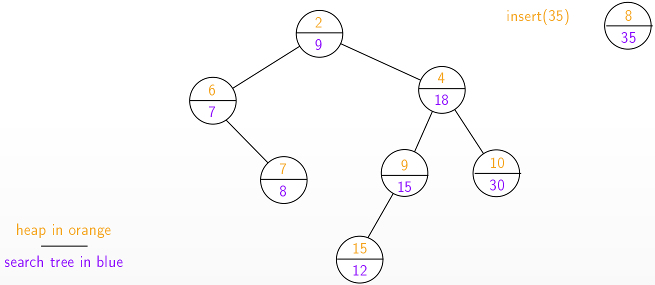
Treap Dictionary Data Structure

- Consider *insert*(k)



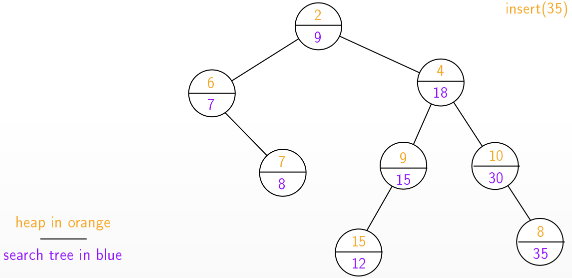
Treap Dictionary Data Structure

- Consider *insert*(k)
  - Choose a random priority (a random number generated say between $[1, n^2]$)



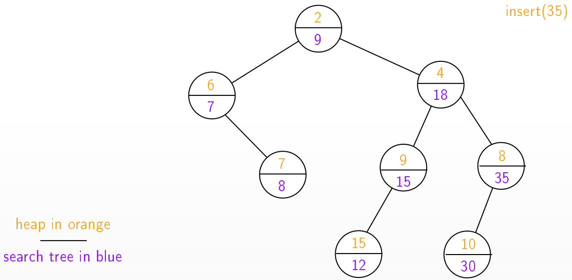
Treap Dictionary Data Structure

- Consider *insert*(k)
  - Choose a random priority (a random number generated say between $[1, n^2]$)
  - Insert as in normal BST



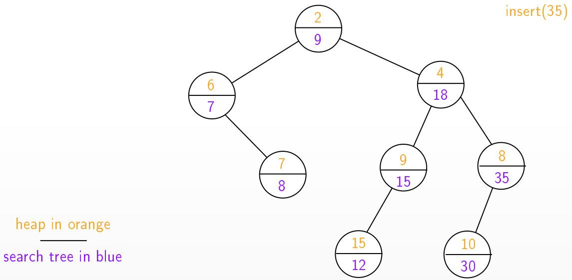
Treap Dictionary Data Structure

- Consider $insert(k)$
  - Choose a random priority (a random number generated say between $[1, n^2]$)
  - Insert as in normal BST
  - Rotate up until heap order is restored (maintaining BST property while rotating)



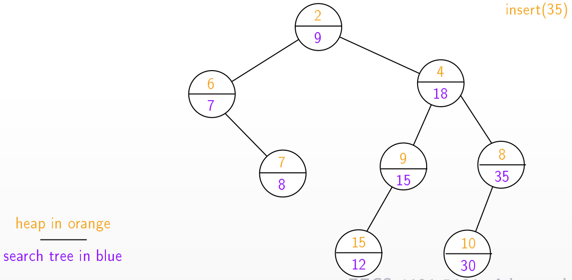
Treap Dictionary Data Structure

- Consider *insert*(k)
  - Choose a random priority (a random number generated say between $[1, n^2]$)
  - Insert as in normal BST
  - Rotate up until heap order is restored (maintaining BST property while rotating)
- How long does *insert*(k) take?



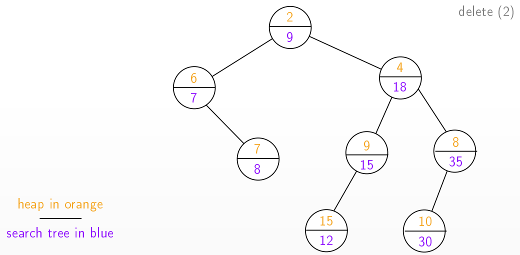
Treap Dictionary Data Structure

- Consider $insert(k)$
  - Choose a random priority (a random number generated say between $[1, n^2]$)
  - Insert as in normal BST
  - Rotate up until heap order is restored (maintaining BST property while rotating)
- How long does $insert(k)$ take? $O(h)$, where h is the height of the tree!



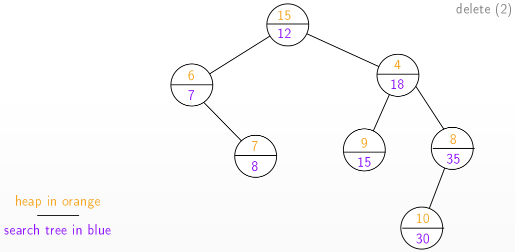
Treap Dictionary Data Structure

- Consider *delete*(k)
  - Search for k and delete its node as in regular BSTs
    - Just remove the node if a leaf
    - Replace the node with its child if only one child
    - Replace the ndoe with its successor or predecessor if two children
  - Rotate down, switching with the child with smaller priority, until heap order is restored (maintaining BST property while rotating)



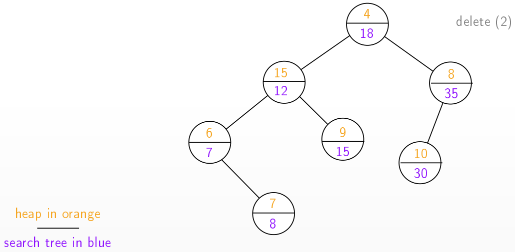
Treap Dictionary Data Structure

- Consider *delete*(k)
  - Search for k and delete its node as in regular BSTs
    - Just remove the node if a leaf
    - Replace the node with its child if only one child
    - Replace the ndoe with its successor or predecessor if two children
  - Rotate down, switching with the child with smaller priority, until heap order is restored (maintaining BST property while rotating)



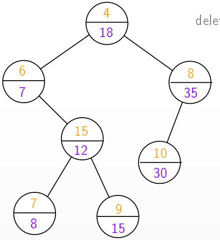
Treap Dictionary Data Structure

- Consider *delete*(k)
  - Search for k and delete its node as in regular BSTs
    - Just remove the node if a leaf
    - Replace the node with its child if only one child
    - Replace the ndoe with its successor or predecessor if two children
  - Rotate down, switching with the child with smaller priority, until heap order is restored (maintaining BST property while rotating)



Treap Dictionary Data Structure

- Consider *delete*(k)
  - Search for k and delete its node as in regular BSTs
    - Just remove the node if a leaf
    - Replace the node with its child if only one child
    - Replace the ndoe with its successor or predecessor if two children
  - Rotate down, switching with the child with smaller priority, until heap order is restored (maintaining BST property while rotating)



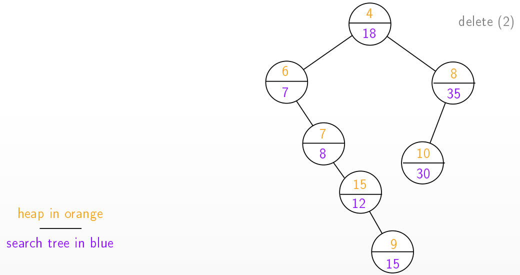
delete (2)

heap in orange

search tree in blue

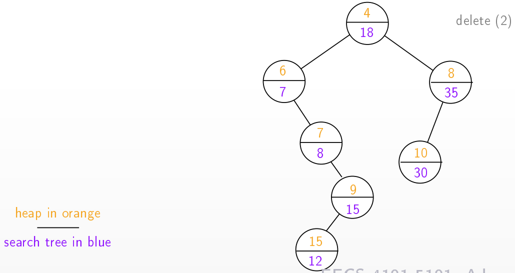
Treap Dictionary Data Structure

- Consider *delete*(k)
  - Search for k and delete its node as in regular BSTs
    - Just remove the node if a leaf
    - Replace the node with its child if only one child
    - Replace the ndoe with its successor or predecessor if two children
  - Rotate down, switching with the child with smaller priority, until heap order is restored (maintaining BST property while rotating)



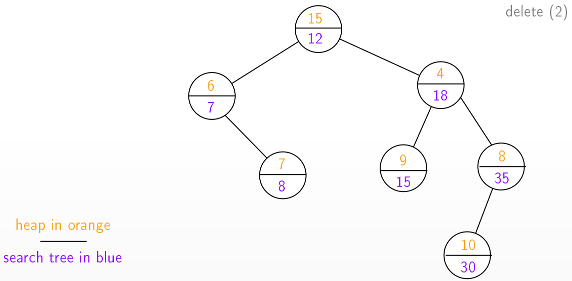
Treap Dictionary Data Structure

- Consider *delete*(k)
  - Search for k and delete its node as in regular BSTs
    - Just remove the node if a leaf
    - Replace the node with its child if only one child
    - Replace the ndoe with its successor or predecessor if two children
  - Rotate down, switching with the child with smaller priority, until heap order is restored (maintaining BST property while rotating)
- How long does *delete*(k) take? $O(h)$, where h is the height of the tree!



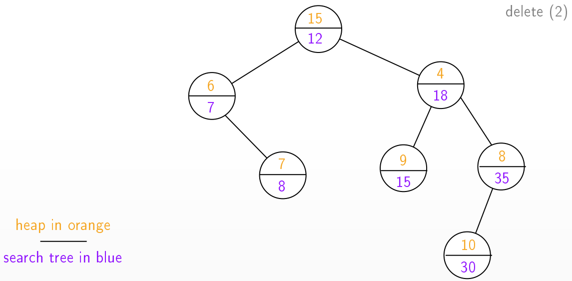
Treap Dictionary Data Structure

- Given that priorities are randomized, all elements have the same chance of being the root (i.e., have the smallest priority).



Treap Dictionary Data Structure

- Given that priorities are randomized, all elements have the same chance of being the root (i.e., have the smallest priority).
- The height of a treap is **expected** to be $O(\log n)$.



Treap Dictionary Data Structure

- Given that priorities are randomized, all elements have the same chance of being the root (i.e., have the smallest priority).
- The height of a treap is **expected** to be $O(\log n)$.
  - **Search, insert, and delete take $O(\log n)$ expected time in treaps.**
  - Treaps are very simple to implement, little overhead – less than AVL trees