TREAP

Huarui Liu, Jingzhou Qiu, Zicheng He

What is a Treap?

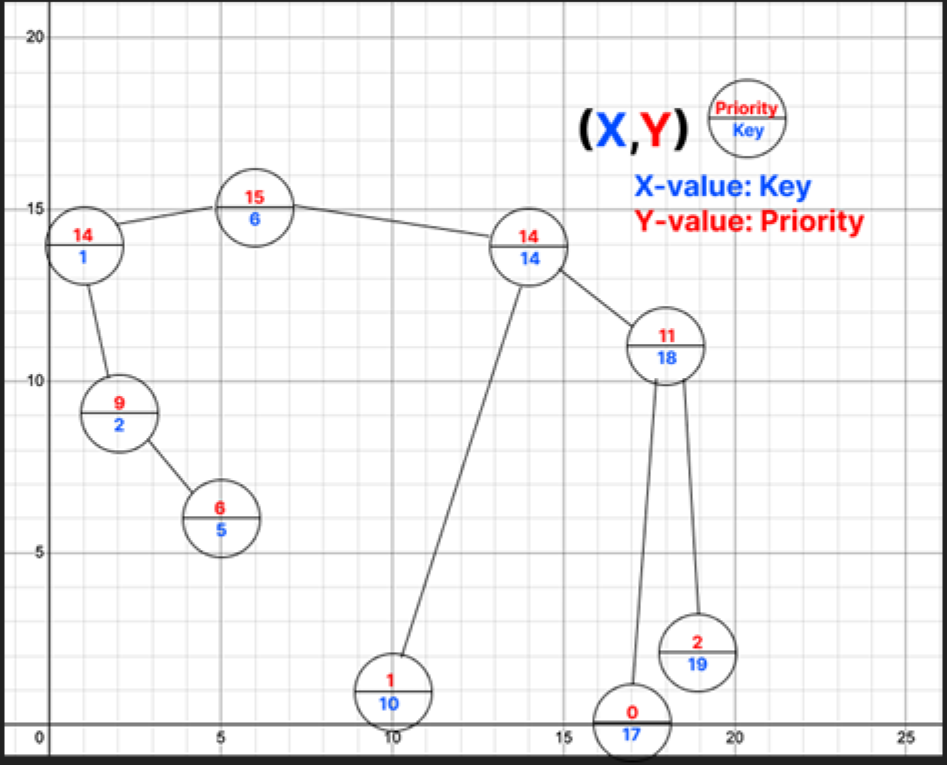
Keys in sorted order like a **BST**

Priorities follow the **heap property**

- Randomized BST Combines BST and heap properties
- A type of Cartesian Tree
- lookup, insertion, and removal in O(logN)

Additional Operations:

- Split: O(logN)
- Merge: O(logN)



Why Treap?

Self-Balancing via random priorities

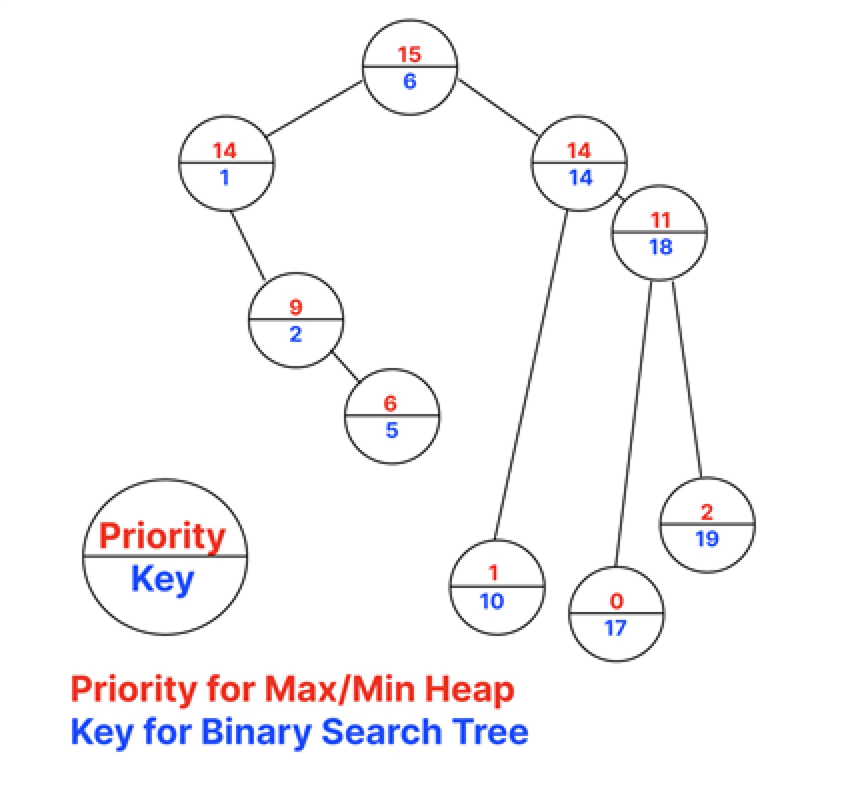
Simpler to implement than AVL or Red-Black trees

Can be modified to support segment tree operations and even more– all in O(logN)

- Reverse on the interval.
- Addition / painting on the interval.

Applications

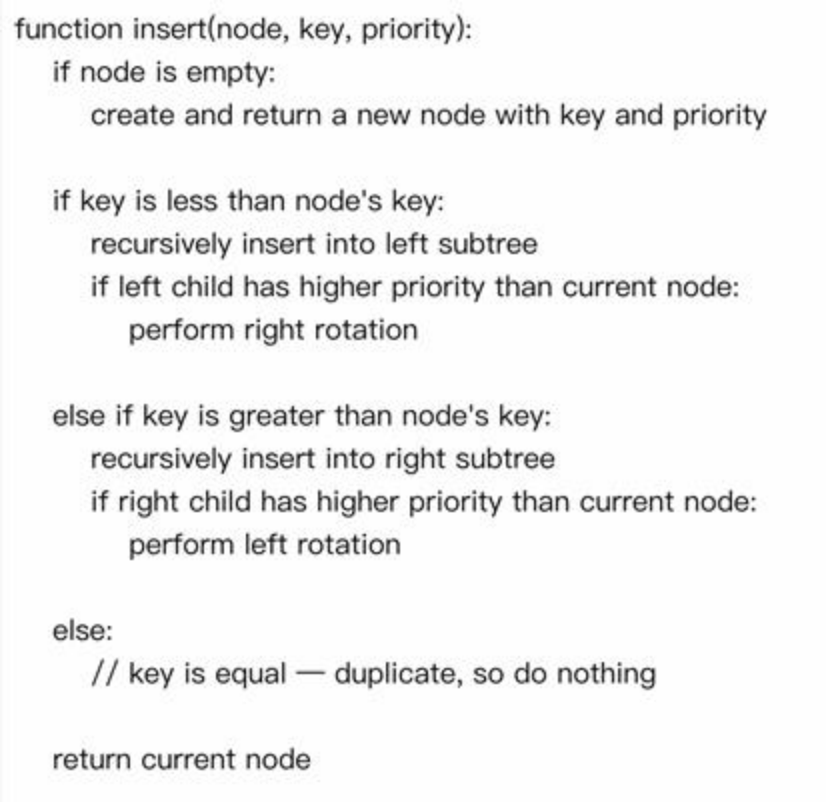
- Linux kernel page cache management
- General Purpose Allocator (GPA)



Insert (Min-heap)

- Pick a random priority/specify a priority
- Insert as inserting in BST
- Rotate until the heap order is maintained

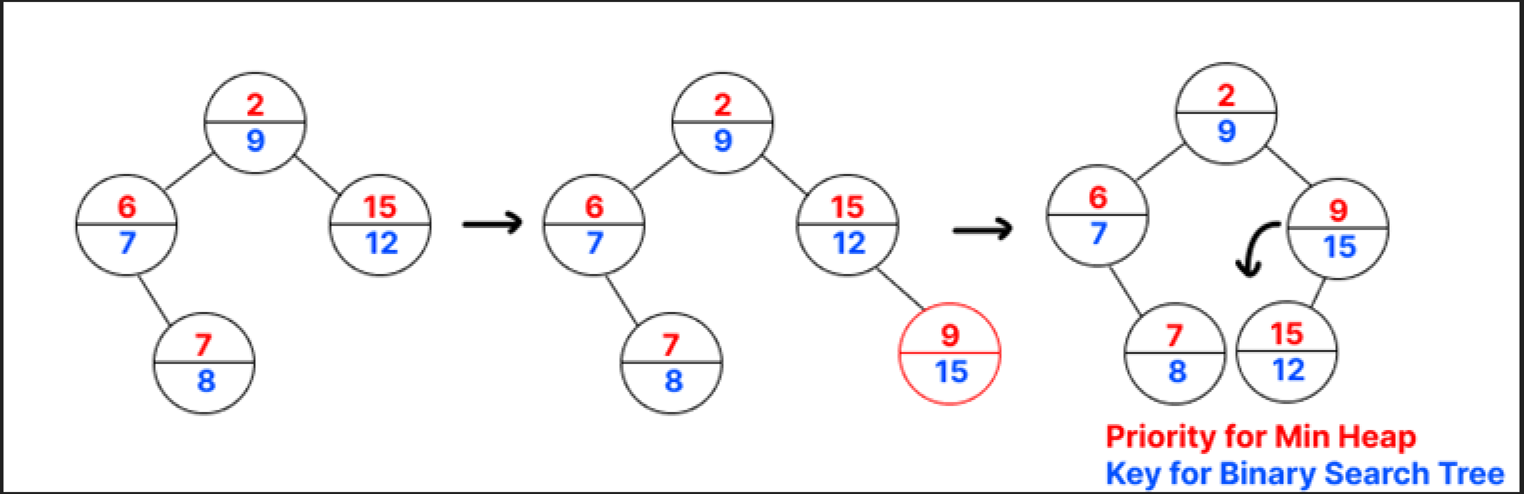
Runtime: O(logN)



```
function insert(node, key, priority):
    if node is empty:
        create and return a new node with key and priority

    if key is less than node's key:
        recursively insert into left subtree
        if left child has higher priority than current node:
            perform right rotation

    else if key is greater than node's key:
        recursively insert into right subtree
        if right child has higher priority than current node:
            perform left rotation

    else:
        // key is equal — duplicate, so do nothing

    return current node
```

Insert-Example

Insert(15) ->Random priority=9

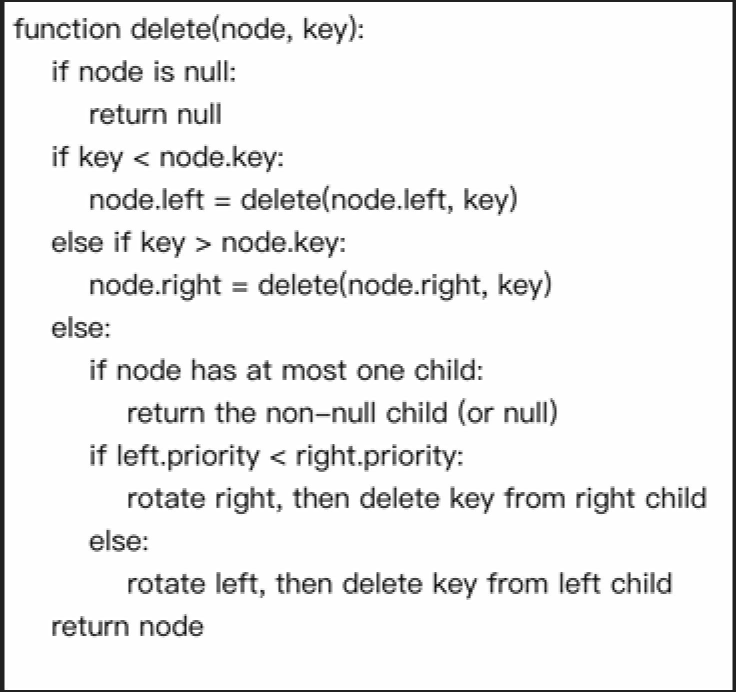


Priority for Min Heap
Key for Binary Search Tree

Delete(Min-heap)

- Find the node by key (BST-style).
- If the node has 0 or 1 child:
  - return non-null child or null
- If the node has 2 children:
  - Rotate the child with the smaller priority up
  - Recurse on the same key to delete it

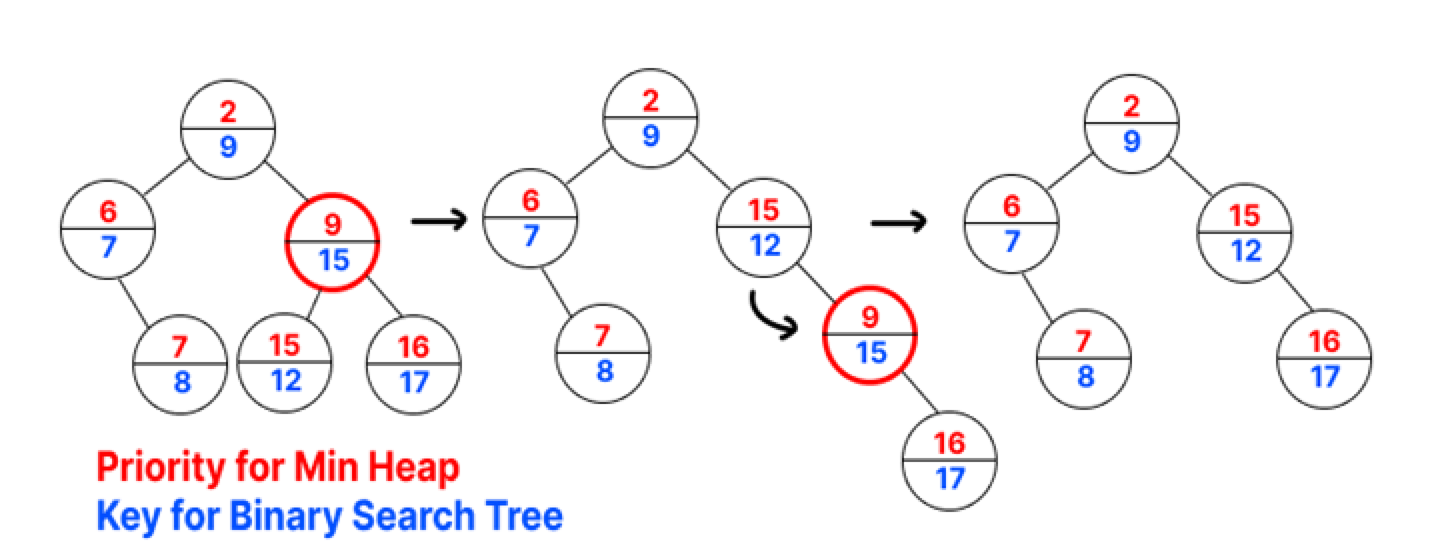
Runtime: O(logN)



```
function delete(node, key):
    if node is null:
        return null
    if key < node.key:
        node.left = delete(node.left, key)
    else if key > node.key:
        node.right = delete(node.right, key)
    else:
        if node has at most one child:
            return the non–null child (or null)
        if left.priority < right.priority:
            rotate right, then delete key from right child
        else:
            rotate left, then delete key from left child
    return node
```

Delete-Example

Delete (15)



Build

Builds a tree from a list of values.

Heapify ensures the parent node has the highest/lowest priority by recursively swapping with the larger/smaller-priority child

Case 1: Input Keys Are Sorted -> Build in O(N) time

Select the middle element to construct BST

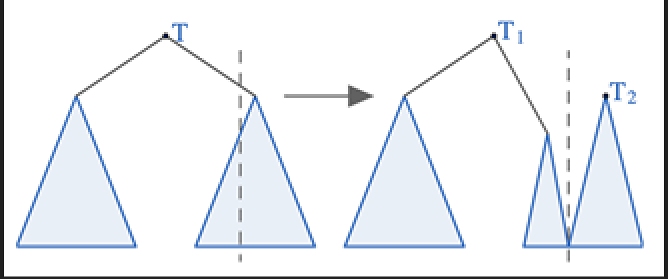
Use heapify to ensure the heap property based on priorities

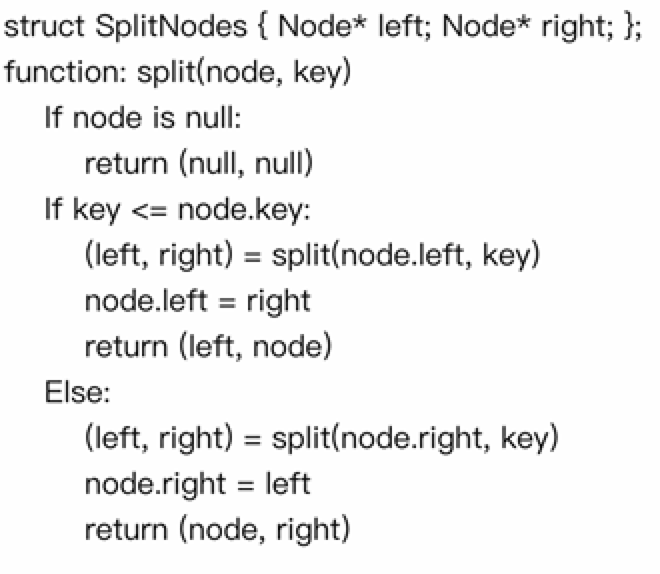
Case 2: Input Keys Are NOT Sorted -> O(N log N) time

N insert calls

Split

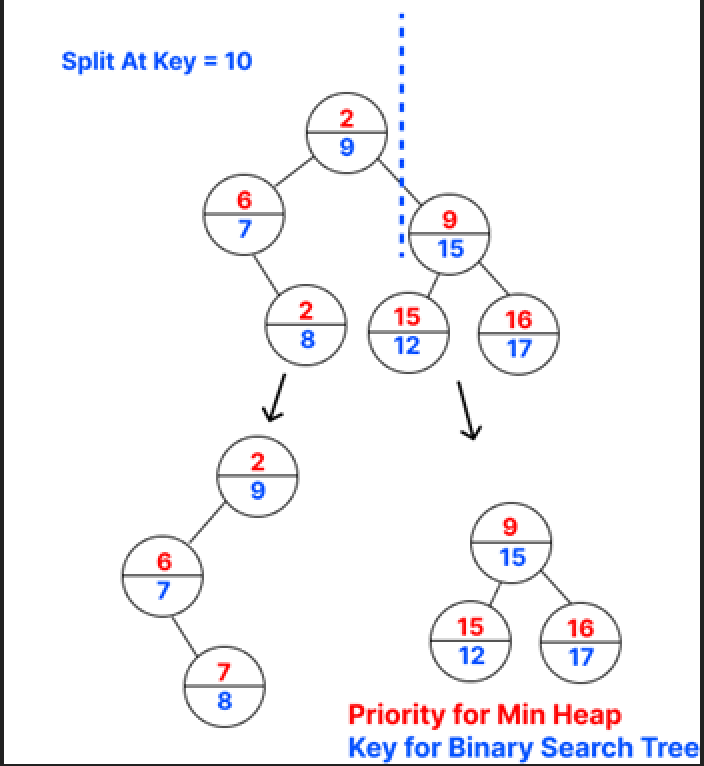
- Decide which subtree the root node would belong to (left or right)
- Recursively call split on one of its children
- Create the final result by reusing the recursive split call
- Runtime: O(logN)





```
struct SplitNodes { Node* left; Node* right; };
function: split(node, key)
    If node is null:
        return (null, null)
    If key <= node.key:
        (left, right) = split(node.left, key)
        node.left = right
        return (left, node)
    Else:
        (left, right) = split(node.right, key)
        node.right = left
        return (node, right)
```

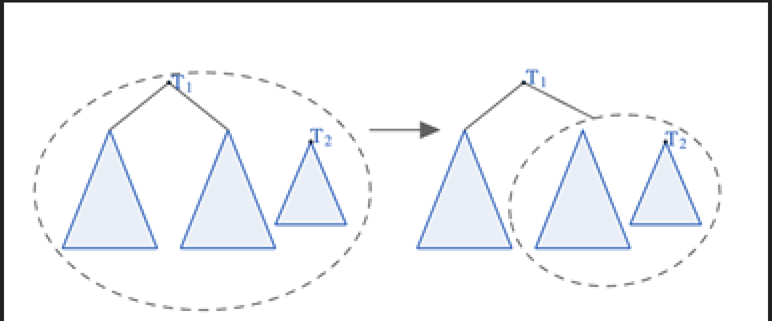
Split-Example

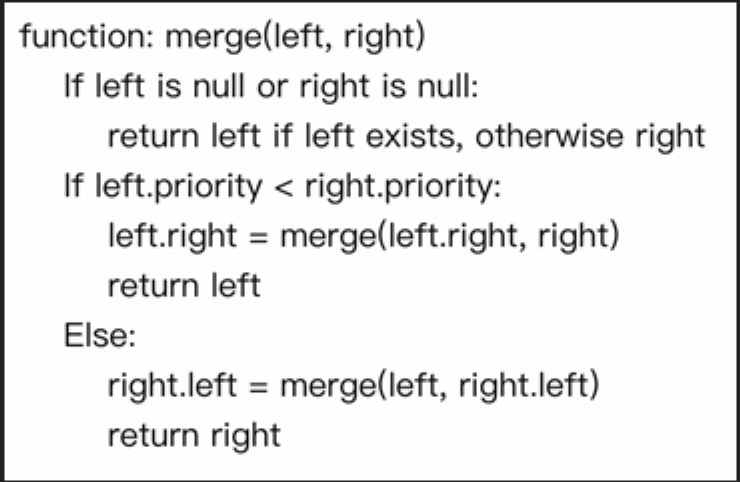


Merge(Min-heap)

- Merges two treaps (left and right) assuming all keys in left are less than those in right.
- Chooses the root with larger/smaller priority to maintain the heap property

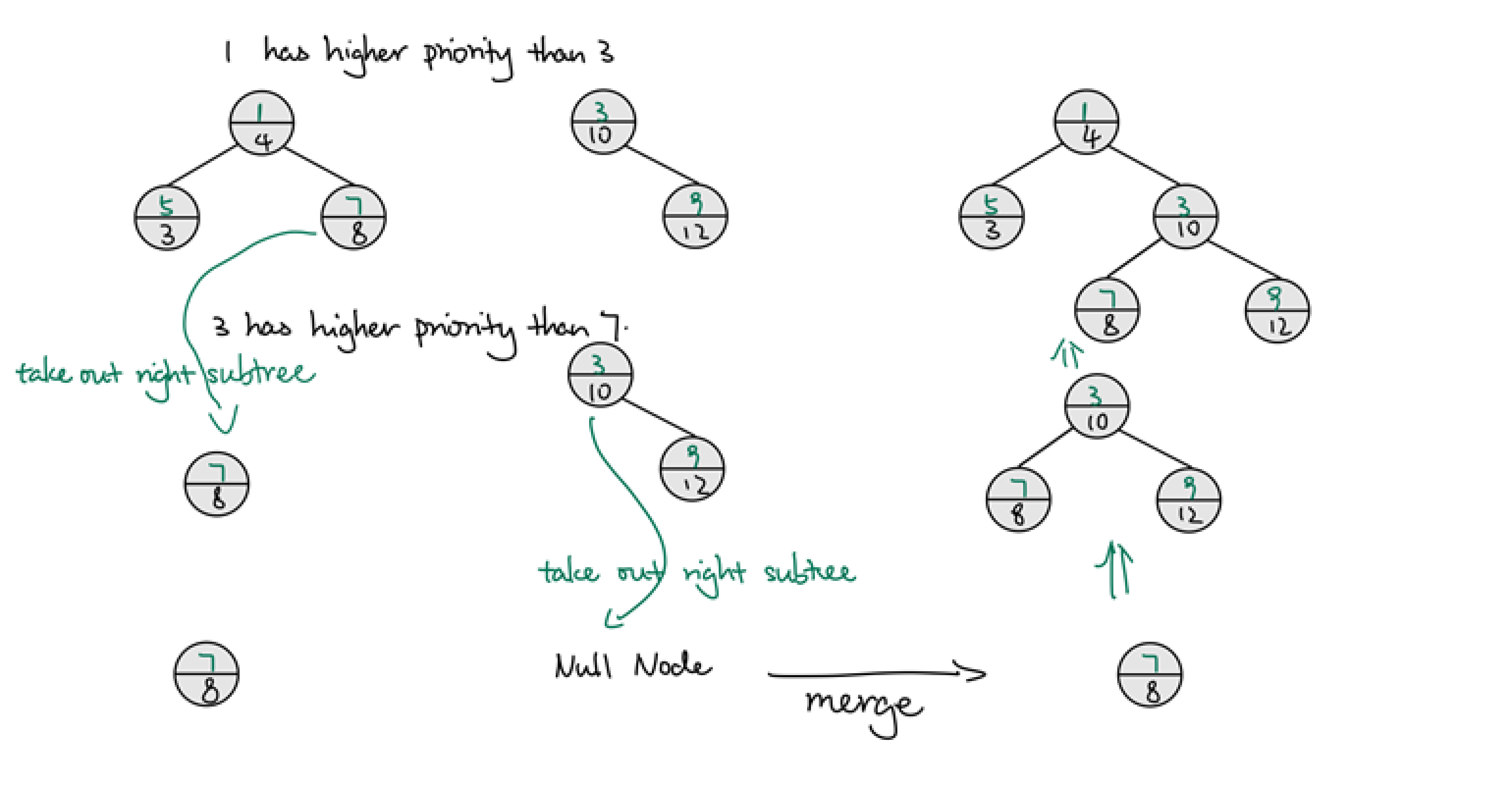
Runtime: O(logN)





```
function: merge(left, right)
    If left is null or right is null:
        return left if left exists, otherwise right
    If left.priority < right.priority:
        left.right = merge(left.right, right)
        return left
    Else:
        right.left = merge(left, right.left)
        return right
```

Merge-Example



Reference

https://cp-algorithms.com/data_structures/treap.html

https://www.youtube.com/watch?v=6x0UIIBLRsc

https://courses.cs.washington.edu/courses/cse326/00wi/handouts/lecture19/sld017.htm