Problem Set 4

Name: Vivek Verma

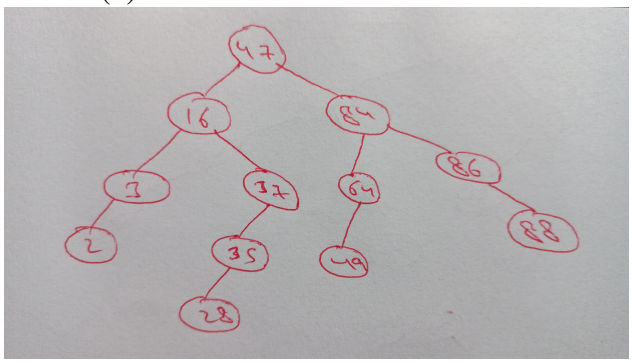
Collaborators: None

Problem 4-1.

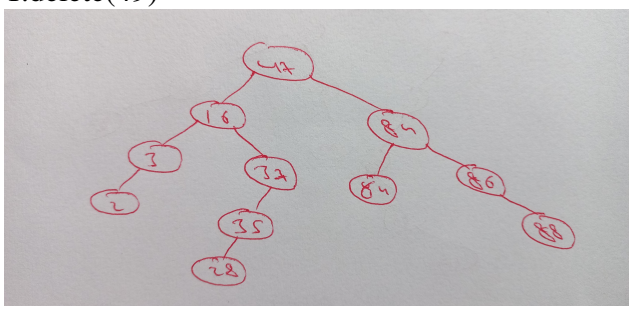
(a) Keys of nodes that are not height-balanced: 16, 37.
$skew(16) = height(37) - height(3) = 2 - 0 = 2$
$skew(37) = height(Nothing) - height(35) = -1 - 1 = -2$

(b)

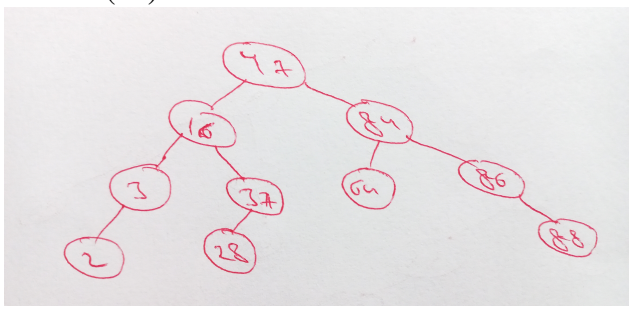
- T.insert(2)



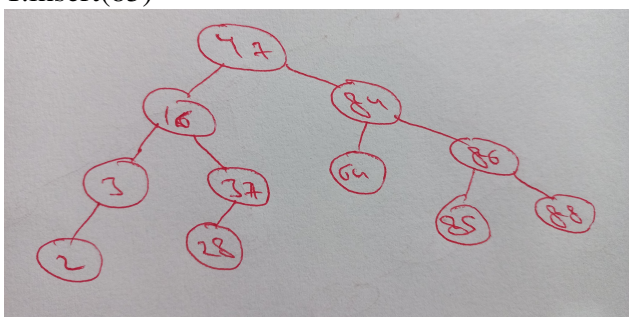
- T.delete(49)



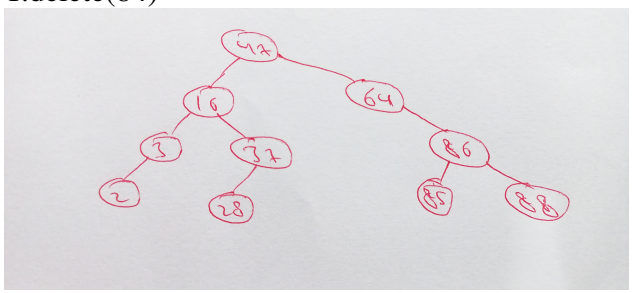
- T.delete(35)



- T.insert(85)

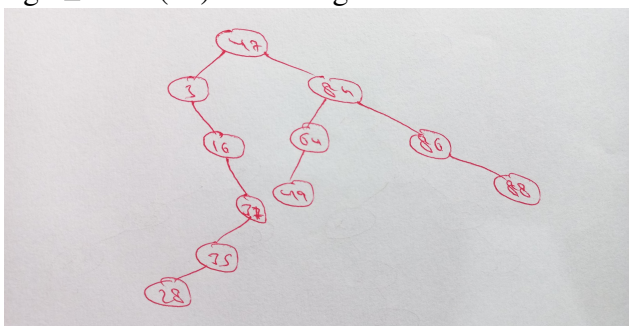


- T.delete(84)

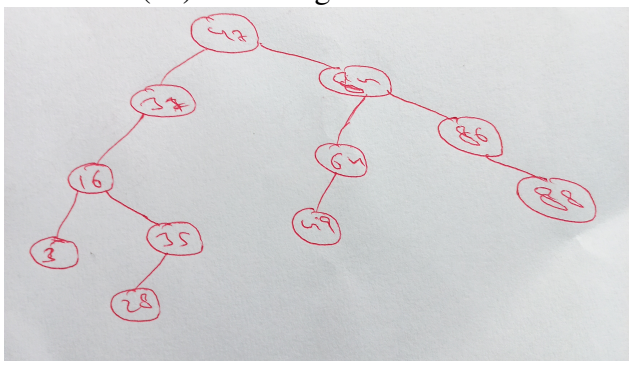


(c)

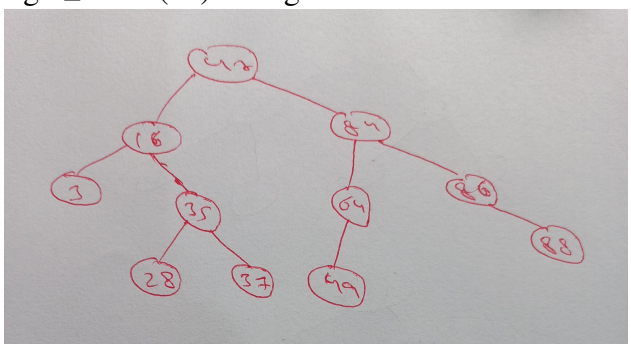
- right_ rotate(16) : Not height-balanced.



- left_ rotate(16) : Not height-balanced.



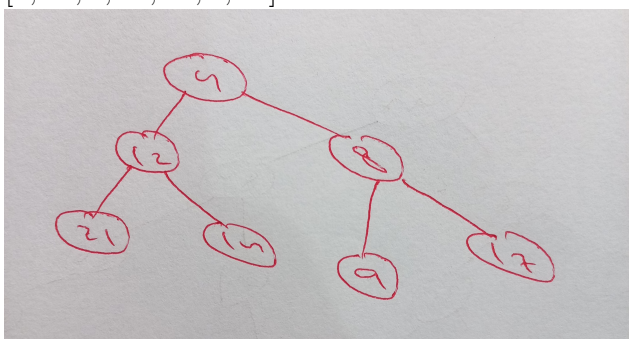
- right_ rotate(37) : Height-balanced.



- left_ rotate(37) : Not possible

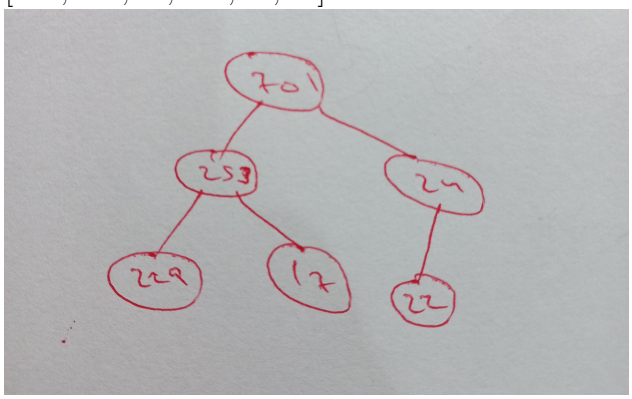
Problem 4-2.

(a) $[4, 12, 8, 21, 14, 9, 17]$



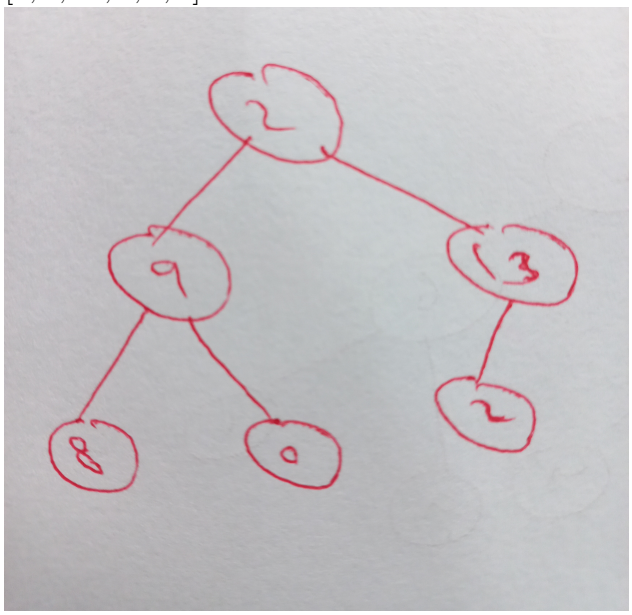
Min-heap.

(b) $[701, 253, 24, 229, 17, 22]$



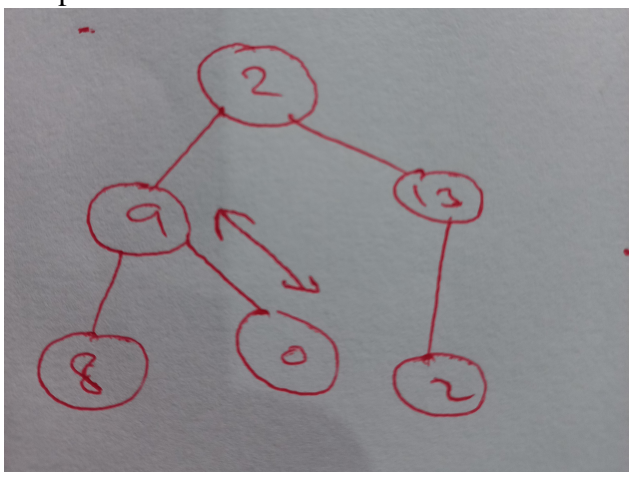
Max-heap.

(c) $[2, 9, 13, 8, 0, 2]$

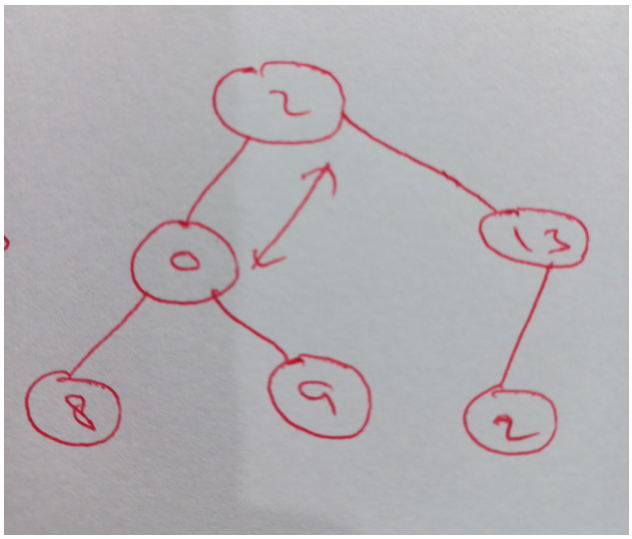


Neither.
3 swaps need to be performed to convert to min-heap:

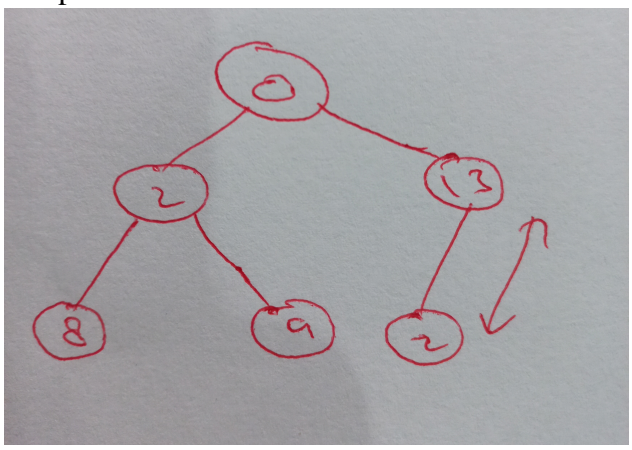
- Swap 1:



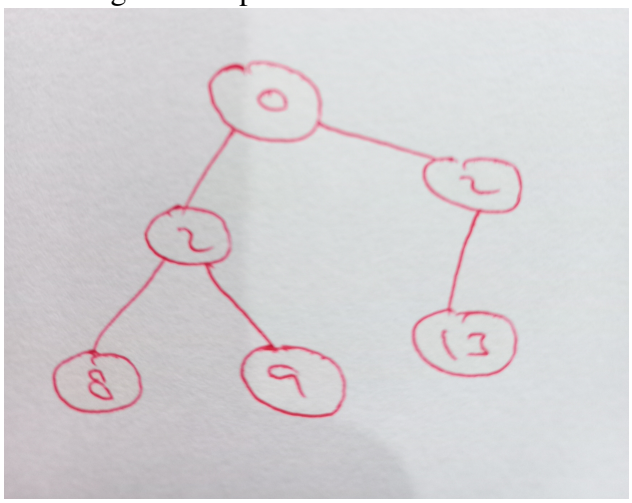
- Swap 2:



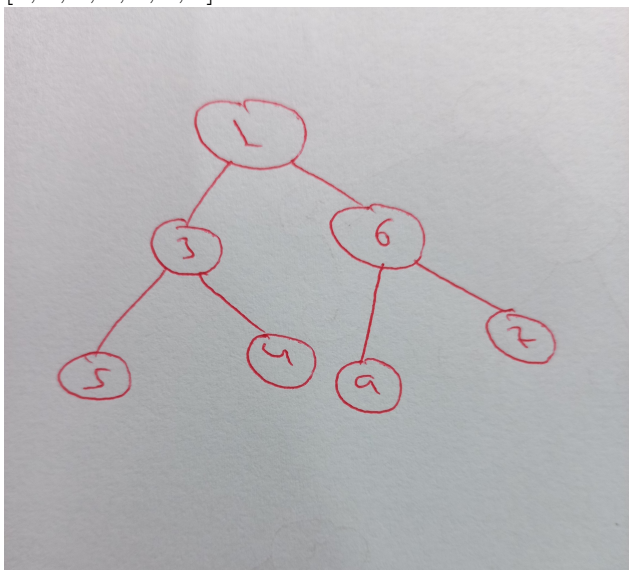
- Swap 3:



Resulting min-heap:



(d) $[1, 3, 6, 5, 4, 9, 7]$



Problem 4-3.

(a) Build a max-heap keyed on the score of garden. This takes $O(|A|)$ time.
Call $delete\_max()$ function on the max-heap k times.
Each call to $delete\_max()$ takes $log\,|A|$ time, Hence k calls requires $klog\,|A|$ time.

$\therefore$ Total time complexity is $O(|A| + klog\,|A|)$

(b) Initialize an empty linked list L.
Start at the root of the max-heap A. If root is greater than x, add the root to L, otherwise, return.

Recursively call function on $root.left$ and $root.right$.
There will be total $\theta(n_x)$ recursive calls and adding an element to linked list costs $O(1)$ time.

$\therefore$ Total time complexity is $O(n_x)$.

Problem 4-4.

Data structures:

- **Max-heap** (M) of solar farms stored as tuples $(s_i, a_i) \in S$ keyed by available capacity (a_i).
- **Hash table** (H_1) to map buildings to solar farm i.e b_j to s_i.
- One more **Hash table** (H_2) to map solar farms (s_i) to Set of buildings connected to it. With each s_i the pointer to node corresponding to it is also stored.

Operations:

- $initialize(S)$
  Initializing a Max-heap with solar $S = ((s_0, c_0), ..., (s_{n-1}, c_{n-1}))$ requires $O(n)$ time.
  Building empty H_1 and H_2 requires $O(1)$ time.
  $\therefore$ Total time to initialize database requires $O(n)$ worst-case time.
- $power\_on(b_j, d_j)$
  Call $delete\_max()$ on M, to get the solar farm with maximum available capacity.
  If the farm's available capacity minus building's demand is less than zero, then insert the farm back into M and return *"No such solar farm exists."*
  Otherwise, add the building's name and solar farm's ID $< key = b_j, value = s_i >$ to H_1. subtract the building's demand d_j from the farm's available capacity a_i and insert the tuple (s_i, a_i) back into M. Get the pointer at which the tuple is stored in max-heap. Update the pointer of s_i in H_2 and add the building b_j to Set corresponding to s_i in H_2.
  Deleting the max. element and inserting an element in max-heap both requires $O(log(n))$ time.
  Updating an entry in hash-table H_2 requires $O(1)$ time.
  Adding an element to set corresponding to s_i also requires $O(1)$ time.
  $\therefore$ Total time complexity is $O(log(n))$.
- $power\_off(b_j)$
  Remove building b_j from H_1 and Get the value of s_i corresponding to it. $O(1)$ time.
  Remove b_j from the Set corresponding to s_i in H_2 in $O(1)$ time . Get the pointer to s_i's location in max-heap.
  Find s_i in max-heap and increase it's available capacity by d_j in $O(1)$ time. Max-heapify $O(log(n))$ to maintain the invariant.
  $\therefore$ Total time complexity is $O(log(n))$ time.
- $customers(s_i)$
  Get the set corresponding to s_i from H_2 in $O(1)$ time.
  Iterate over the set and output all the buildings' names in $O(k)$ time, where k is the size of the set.
  $\therefore$ Total time complexity is $O(k)$.

Problem 4-5.

Data structure:
* A Sequence AVL tree can be used to store the transformation matrices $\mathcal{M} = (M_0, ..., M_{n-1})$.
Each $node$ can be augmented with two entries:

- m_l : Matrix multiplication of subtree rooted at $node.left$.
- m_r : Matrix multiplication of subtree rooted at $node.right$.

Operations:

- $initialize(\mathcal{M})$
  Building a Sequence AVL tree requires $O(n)$ time. And augmentation entries can be added to each node in $O(1)$ time.
  $\therefore$ Total time complexity is $O(n)$.
- $update\_joint(k, M)$
  Find the node at index k in $O(log(n))$ time.
  Update the matrix stored at the node in $O(1)$ time.
  Update the augmentations of the parents repeatedly until the root of the tree in $O(log(n))$ time.
  $\therefore$ Total time complexity is $O(log(n))$.
- $full\_transformation()$
  Get the matrix m stored at root node of the tree in $O(1)$ time.
  Compute $m_l \; x \; m \; x \; m_r$ in $O(1)$ time and return.
  Total time complexity is $O(1)$.

Problem 4-6.

(a)

(b)

(c)

(d) Submit your implementation to `alg.mit.edu`.