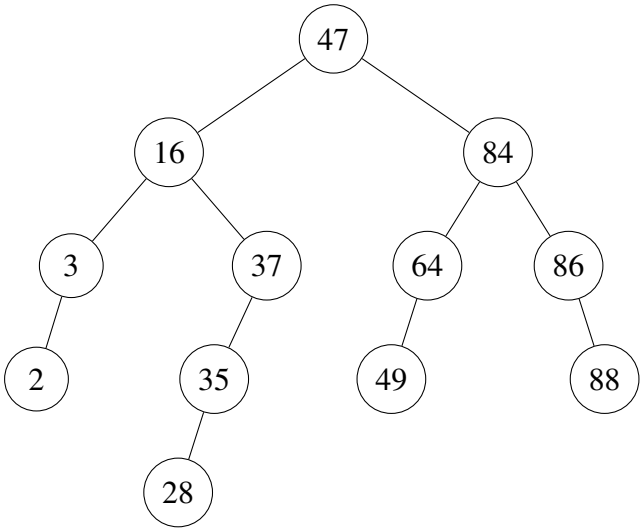
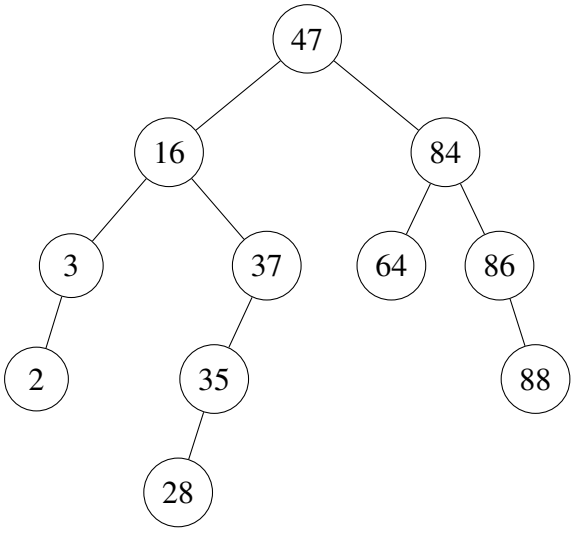
Name: Asem Ashraf

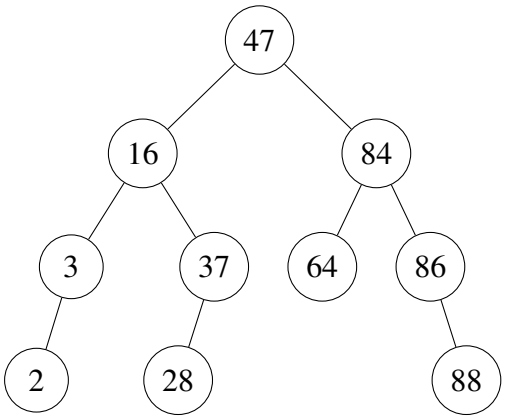
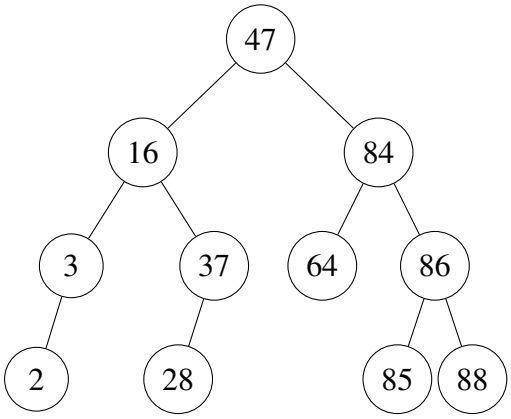
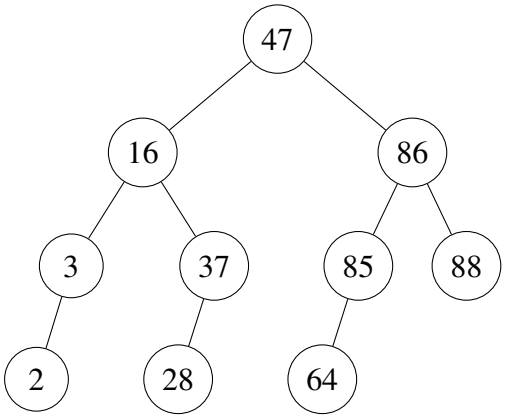
Collaborators: None

Problem 4-1.

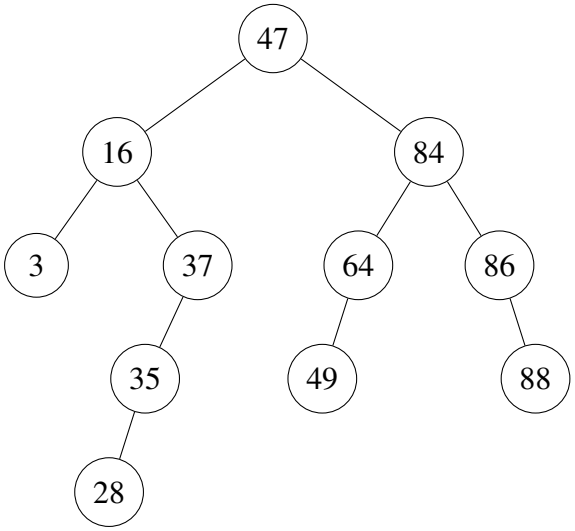
(a)
- 47 skew -1
- 16 skew 2
- 37 skew -1
- 64 skew -1
- 86 skew 1

(b)
-


-


- 
47
16
84
3
37
64
86
2
28
88
- 
47
16
84
3
37
64
86
2
28
85
88
- 
47
16
86
3
37
85
88
2
28
64

(c)



Problem 4-2.

(a)

(b)

(c)

(d)

Problem 4-3.

(a)

(b)

Problem 4-4. Firstly, we will use a priority queue that is implemented using the max heap data structure, so we can have the build time be O(1). The priority queue will be build based on the available capacity of a solar plant. Every time we delete max out of the priority queue, we would be deleting the solar farm with the maximum available capacity.

Secondly, we will have to build a set AVL tree keyed on the building names, and has at each node the demand of the building and a pointer to the solar farm powering it in the priority queue.

In the priority queue, each solar farm will be pointing to 2 values. The available capacity and a list with all the building that it is powering.

In this database we will use multiple AVL trees. The main AVL tree is the one keyed on the building's unique ID. The other AVL tree is keyed by the available capacity of the solar farm.

Problem 4-5.

The data structure in this problem will be a sequence AVL binary tree. At each there will be the matrix of that specific joint in the matrix along with the matrix multiplication of the left node's matrix with self matrix then with right node's matrix in this order. If this invariant is kept along the tree at each change in any matrix, the resultant matrix at the node(not the matrix of the node) would be the full transform.

Intializing the sequence AVL tree would take O(n) to put each matrix in its node and calculating its resultant matrix. This is quite vague but this can be done in a specific way, from the bottom up to be done in O(n) time. Based on the number of n, the leaves of the tree could be specified and then building the tree from its leaves up.

To update a joint, first we would have to find the node in O(logn), then update its matrix. Then we do 2 matrix multiplication of the the said matrices in O(1). Then go along the ancestors of that nodes updating the resultant matrix at each node in O(logn). Which evaluates to total run time of O(logn).

If we kept all these invariant, then we will find the full transformation in the root resultant matrix.

Problem 4-6.

(a)

(b)

(c)

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