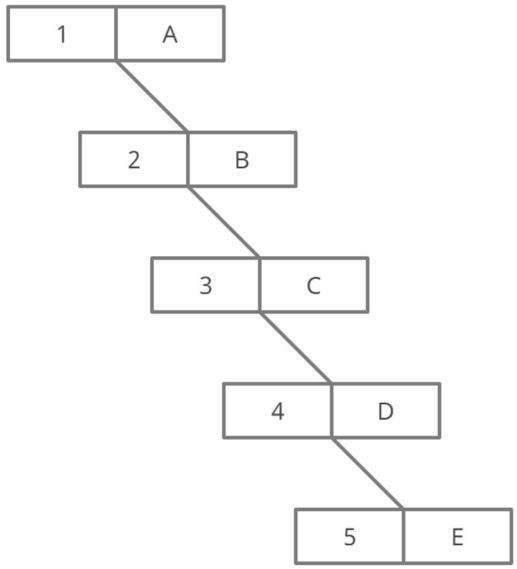
CSCI 220 -- Homework 2

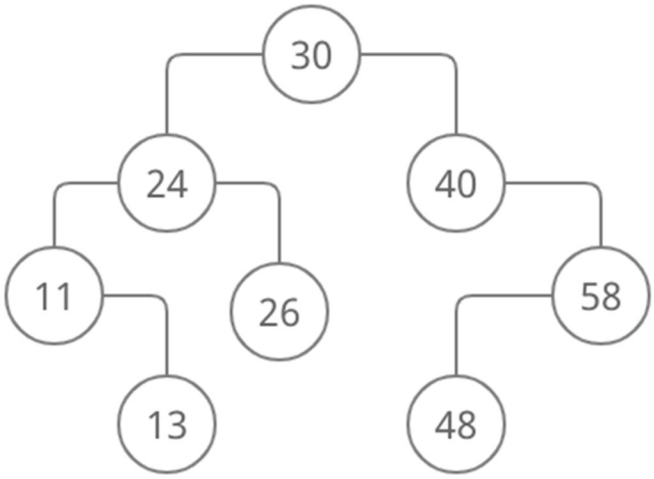
Nero Li

Chapter 10

R-10.1



R-10.3

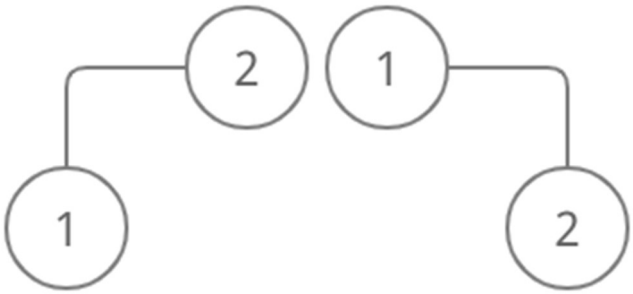


R-10.4

There are **5** different binary search trees can store the keys {1,2,3}

R-10.6

If we have a tree that store the keys {1,2}, here are two possible AVL trees that we can generate.

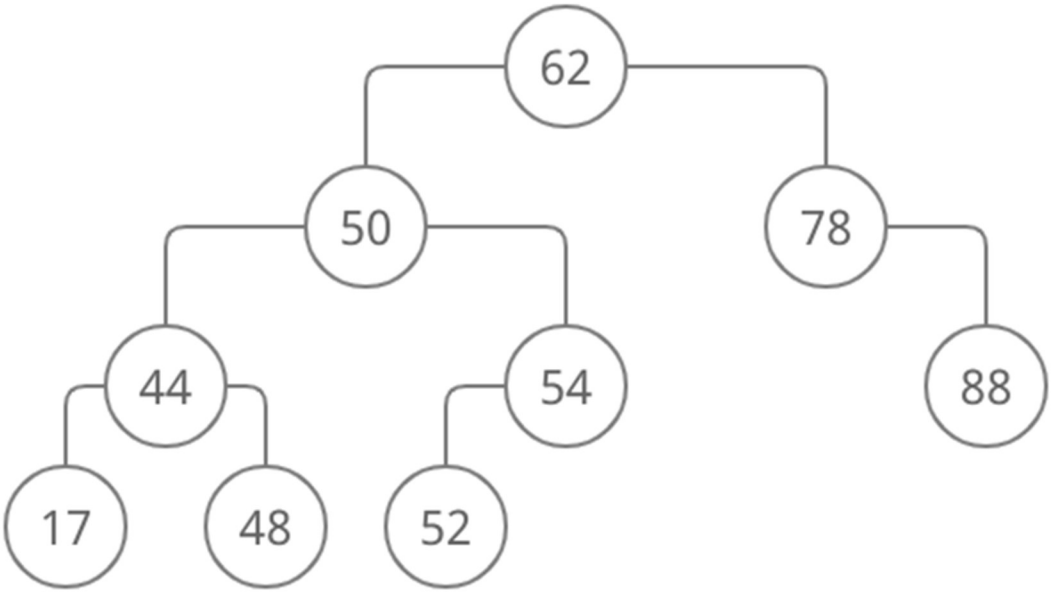


R-10.7

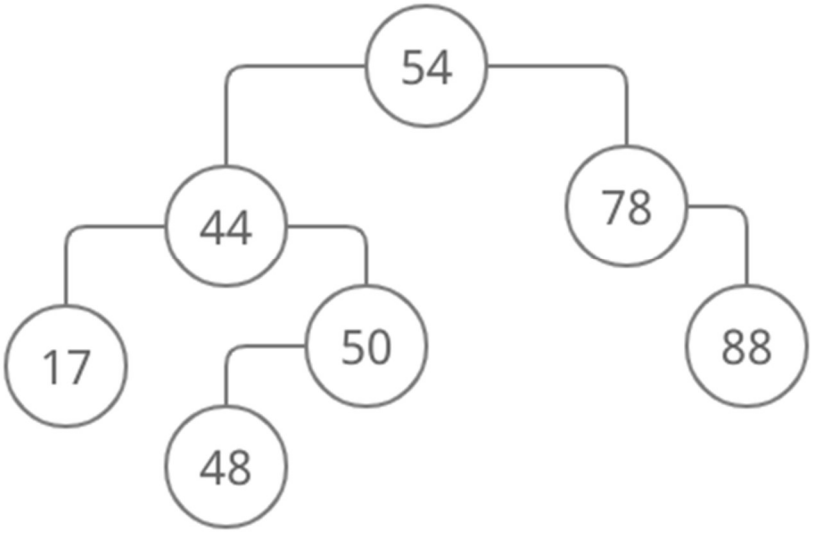
The rotations in Figures 10.9 is **double rotation.**

The rotations in Figures 10.11 is **single rotation.**

R-10.8



R-10.9



R-10.11

It can be called as (2,4) in a extreme situation since for (2,4) we always need more values in one space, but for the tree that we have saw, it worked as a general binary tree.

R-10.12

If in ascending order, we will have k1, k2, k3, and k4, k2 will be in v or root.

Since v` is a 2 node, k1 will get in as v's left child.

Since v`` is a 3 node, k3 and k4 will get in as v's right child.

R-10.19

a. `2log(100000)`

b. `log(100000)`

c. `2log(100000)`

d. `100000`

e. `100000`

C-10.1

Use an int value n to store the total entry that has been stored into the BST. To find that, we do the in-order traversal n/2 times. To make sure it will work for our class, we need to modify that our class can access the position value for the class. Or we can do other way to receive tree's position list. This position list we need to change the original pre order traversal to in-order traversal, so that we will know which one will have the rank n/2. We will use the second way to operate that since it is easier to work.

```
Algorithm findMid():
     Get PositionList pl from tree in class, this list should store all value
by in-order traversal.
     For i from 1 to n/2 - 1:
          pl.pop_front()
          return pl.front().value()
```

Chapter 8

R-8.6

(1, d) (3, j) (4, b) (5, a) (2, h) (6, c)

R-8.7

```
class AirTrafficControl
{
private:
    struct Plane
    {
        int time;
        string event;
    };

    class isLess
    {
    public:
        bool operator()(Plane p, Plane q) const
        {
            return p.time < q.time;
        }
    };

    ListPriorityQueue<Plane, isLess> pq;

public:
    void insert(int t, string s)
    {
        Plane newPlane;
        newPlane.time = t;
        newPlane.event = s;
        pq.insert(newPlane);
    }

    string extract()
```

```
    {
        string extractInfo;
        extractInfo = pq.min().event;
        pq.removeMin();
        return extractInfo;
    }
};
```

R-8.12

An entry with the largest key can be stored **at root or any other external node**.

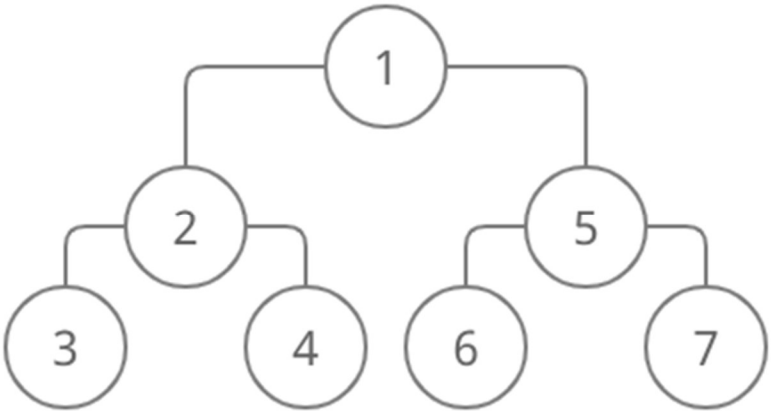
R-8.15

Tree T is a heap.

Since each level will have the same number, the root node will remain the smallest number which is one, and all node's child will not be bigger than their parents, or in other words, they will all have one lower value than their parents. Hence, it is a heap.

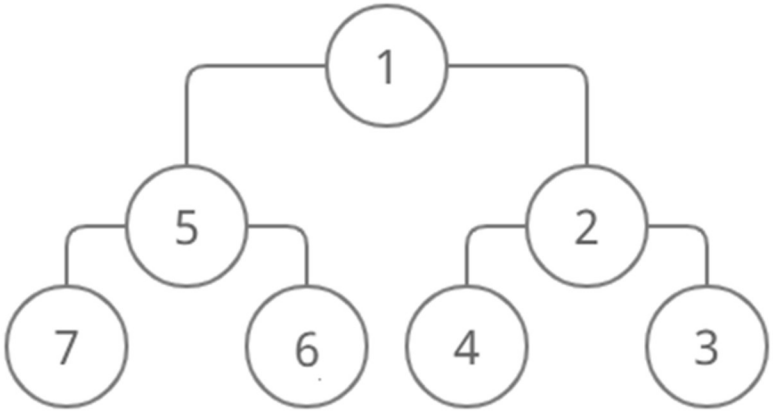
R-8.17

For pre-order, here is an example tree and its traversal is 1,2,3,4,5,6,7.

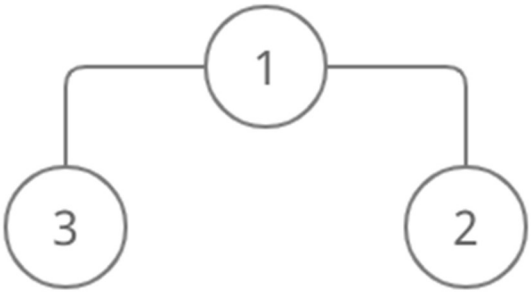


I cannot find a way to create a Heap that will yields the elements of T in sorted order since if I want to do that, the right child and left child will have one bigger than parent and one smaller than parent, which violate the definition for Heap tree.

For post-order, here is an example tree and its traversal is 7,6,5,4,3,2,1.



R-8.21



C-8.4

```
template <typename E>
class Stack
{
private:
    struct Stuff
    {
        int key;
        E value;
    };

    class isLess
    {
    public:
        bool operator()(Stuff p, Stuff q) const
        {
            return p.key < q.key;
```

```
        }
    };

    ListPriorityQueue<Stuff, isLess> pq;
    int k;
public:
    void push(E n)
    {
        Stuff newStuff;
        newStuff.key = k--;
        newStuff.value = n;
        pq.insert(newStuff);
    }

    E pop()
    {
        E elem;
        elem = pq.min().value;
        pq.removeMin();
        ++k;
        return elem;
    }
};
```

C-8.14

```
Algorithm func(heap T, int k):
     List L
     While T.min() <= 7:
          Report T.min()
          L.push_front(T.min())
          T.removeMin()
     While !L.empty():
          T.insert(L.back())
          L.push_back()
```