COMP9024 20T0

Week 4a Problem Set

Self-adjusting Search Trees

Data Structures and Algorithms

1. (Insertion at root)

   a. Consider an initially empty BST and the sequence of values

   ```
   1 2 3 4 5 6
   ```

   - Show the tree resulting from inserting these values "at leaf". What is its height?
   - Show the tree resulting from inserting these values "at root". What is its height?
   - Show the tree resulting from alternating between at-leaf-insertion and at-root-insertion. What is its height?

   b. Complete this week's Binary Search Tree ADT (`BST.h`, `BST.c`) from the lecture by an implementation of the function:

   ```
   Tree insertAtRoot(Tree t, Item it) { ... }
   ```

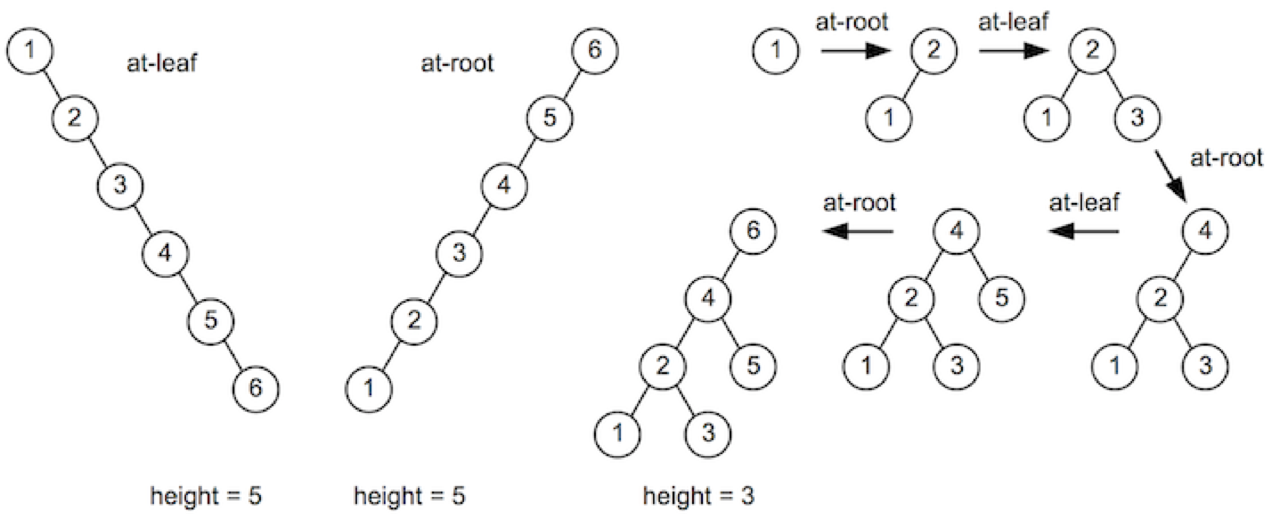
We have created a script that can automatically test your program. To run this test you can execute the `dryrun` *program that corresponds to this exercise. It expects to find the file named* `BST.c` *in the current directory with your implementation of the function* `insertAtRoot()`.

You can use dryrun as follows:

```
prompt$ 9024 dryrun BST
```

Answer:

a. At-leaf-insertion results in a "right-deep" tree while at-root insertion results in a "left-deep" tree. Both are fully degenerate trees of height 5. Alternating between the two styles of insertion results in a tree of height 3. Generally, if n ordered values are inserted into a BST in this way, then the resulting tree will be of height $\left\lfloor \frac{n}{2} \right\rfloor$.



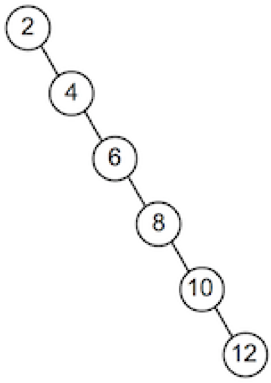
b.

```
Tree insertAtRoot(Tree t, Item it) {
   if (t == NULL) {
      t = newNode(it);
   } else if (it < data(t)) {
      left(t) = insertAtRoot(left(t), it);
      t = rotateRight(t);
   } else if (it > data(t)) {
      right(t) = insertAtRoot(right(t), it);
      t = rotateLeft(t);
```

```
    }
    return t;
}
```

2. (Rebalancing)

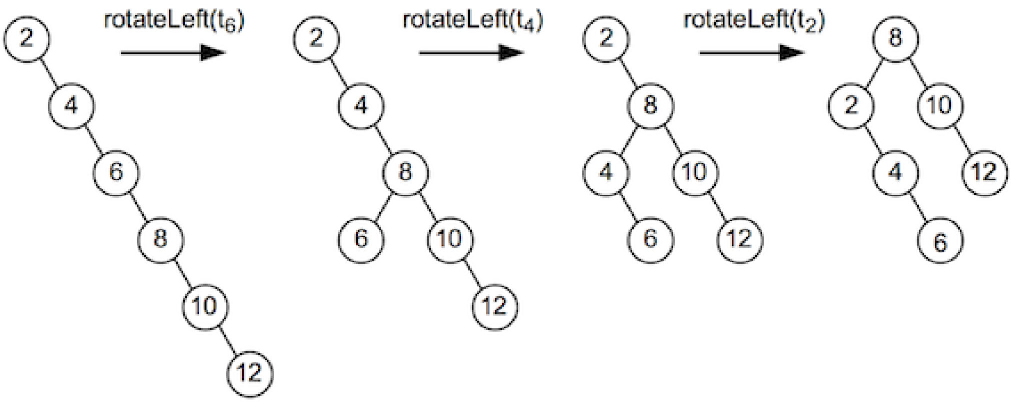
   Trace the execution of `rebalance(t)` on the following tree. Show the tree after each rotate operation.



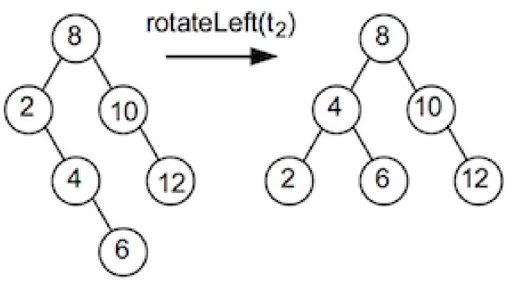
Answer:

In the answer below, any (sub-)tree $\mathtt{t}_\mathtt{n}$ is identified by its root node `n`, e.g. $\mathtt{t}_2$ for the original tree.

Rebalancing begins by calling $\mathtt{partition(t_2,3)}$ since the original tree has 6 nodes. The call to $\mathtt{partition(t_2,3)}$ leads to a series of recursive calls: $\mathtt{partition(t_4,2)}$, which calls $\mathtt{partition(t_6,1)}$, which in turn calls $\mathtt{partition(t_8,0)}$. The last call simply returns $\mathtt{t}_8$, and then the following rotations are performed to complete each recursive call:



Next, the new left subtree $\mathtt{t}_2$ gets balanced via $\mathtt{partition(t_2,1)}$, since this subtree has 3 nodes. Calling $\mathtt{partition(t_2,1)}$ leads to the recursive call $\mathtt{partition(t_4,0)}$. The latter returns $\mathtt{t}_4$, and then the following rotation is performed to complete the rebalancing of subtree $\mathtt{t}_2$:



The left and right subtrees of $\mathtt{t}_4$ have fewer than 3 nodes, hence will not be rebalanced further. Rebalancing continues with the right subtree $\mathtt{t}_{10}$. Since this tree also has fewer than 3 nodes, rebalancing is finished.

3. (Splay trees)

a. Show how a Splay tree would be constructed if the following values were inserted into an initially empty tree in the order given:

```
3 8 5 7 2 4
```

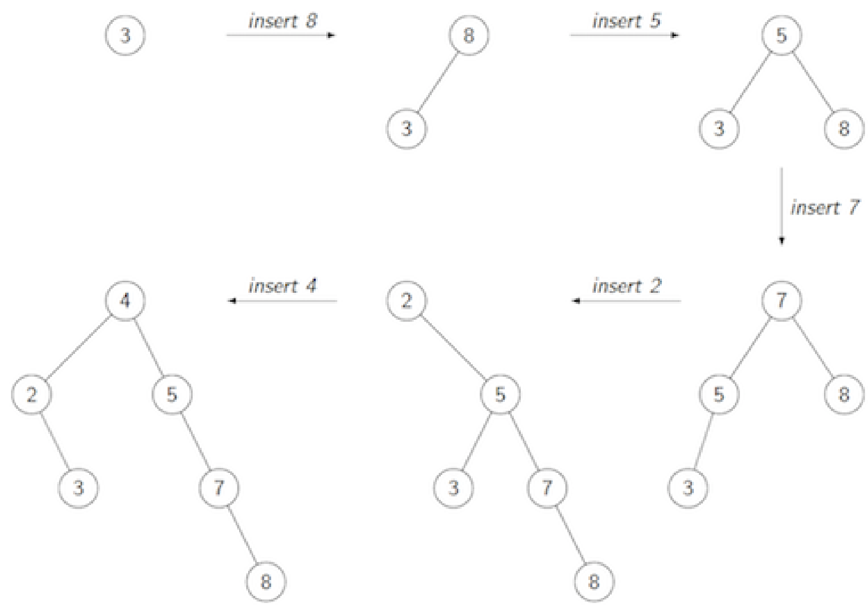
b. Let `t` be your answer to question a., and consider the following sequence of operations:

```
SearchSplay(t,3)
SearchSplay(t,5)
SearchSplay(t,6)
```

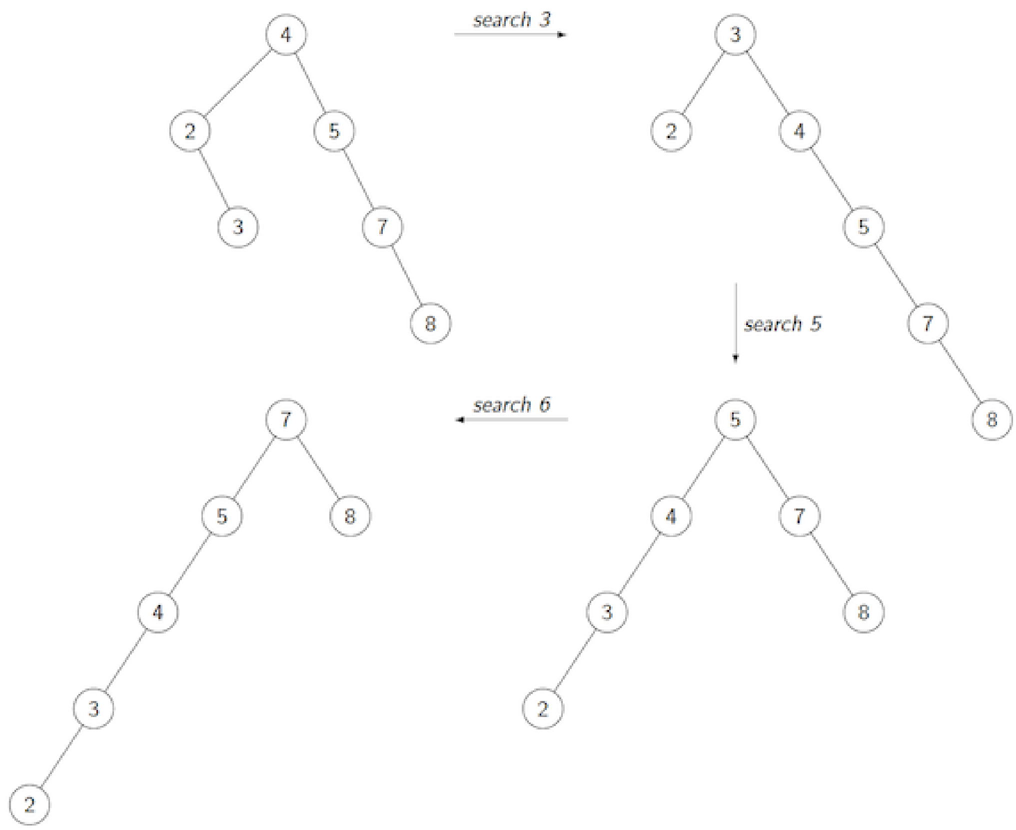
Show the tree after each operation.

Answer:

a. The following diagram shows how the tree grows:



b. The following diagram shows how the tree changes with each search operation:



4. (AVL trees)

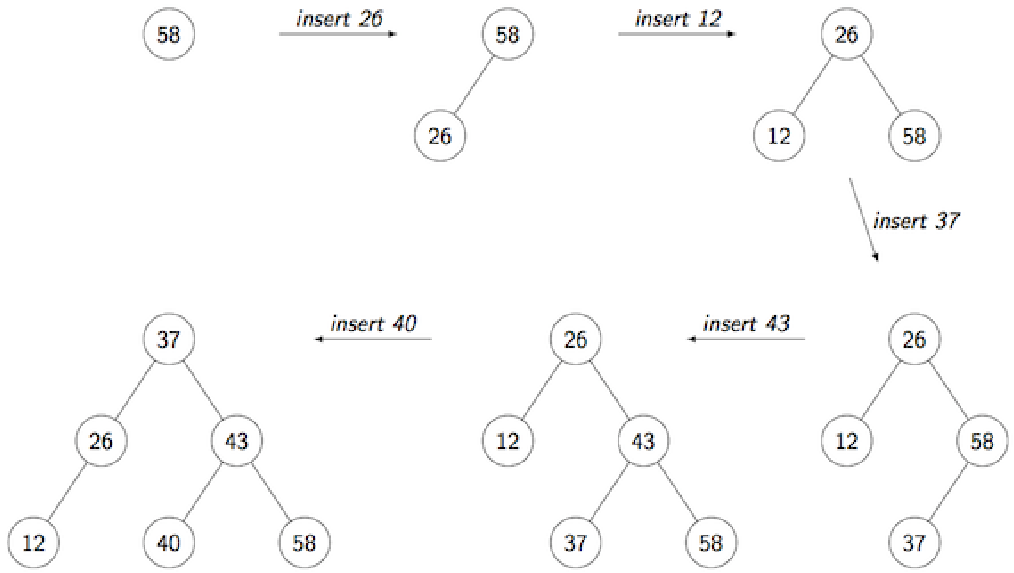
Note: You should answer the following question without the help of the `treeLab` program from the lecture.

Show how an AVL tree would be constructed if the following values were inserted into an initially empty tree in the order given:

```
58 26 12 37 43 40
```

Answer:

The following diagram shows how the tree grows:



Imbalances happen:

- when 12 is inserted, which triggers a right rotation at the root;
- when 43 is inserted, which triggers a left rotation at 37 followed by a right rotation at 58;
- when 40 is inserted, which triggers a right rotation at 43 followed by a left rotation at the root.

5. (2-3-4 trees)

Show how a 2-3-4 tree would be constructed if the following values were inserted into an initially empty tree in the order given:

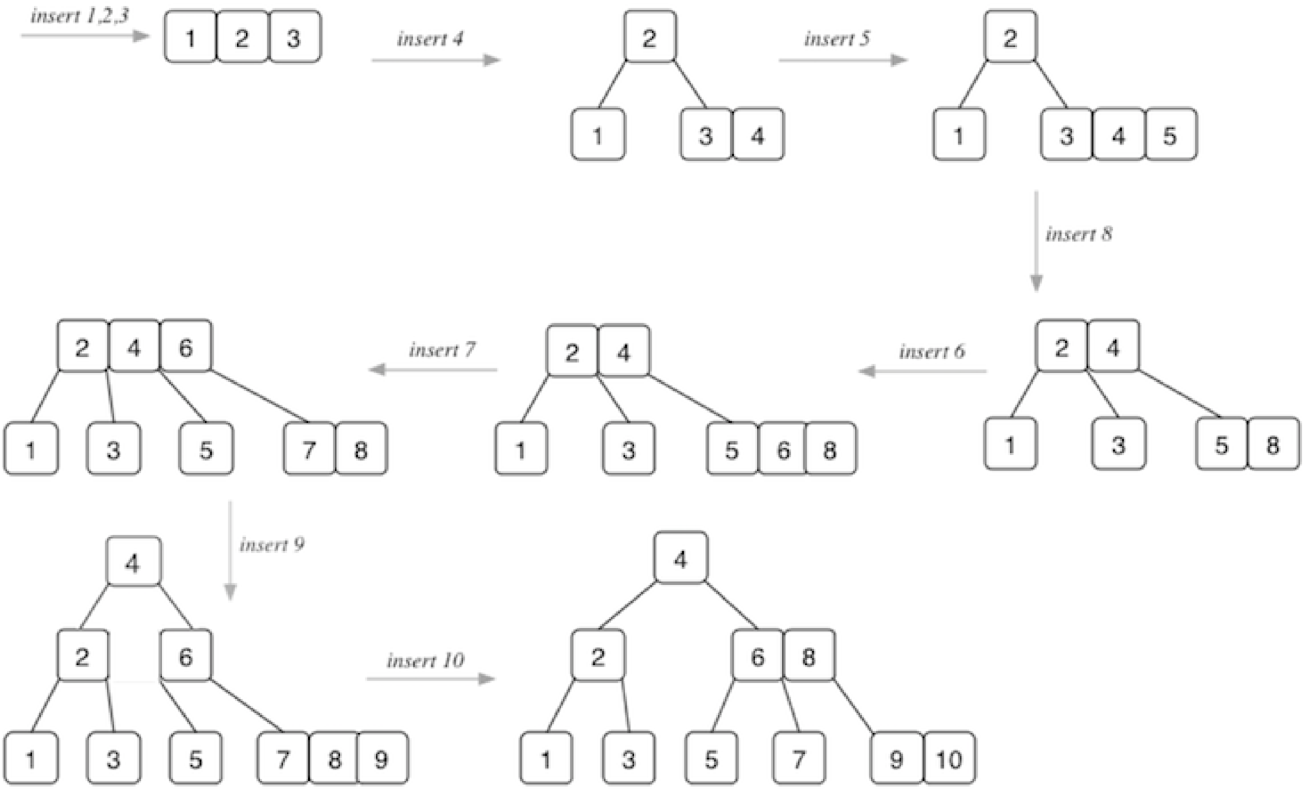
```
1 2 3 4 5 8 6 7 9 10
```

Once you have built the tree, count the number of comparisons needed to search for each of the following values in the tree:

```
1  7  9  13
```

Answer:

The following diagram shows how the tree grows:



Search costs (for tree after insertion of 10):

- search(1): cmp(4),cmp(2),cmp(1) ⇒ cost = 3
- search(7): cmp(4),cmp(6),cmp(8),cmp(7) ⇒ cost = 4
- search(9): cmp(4),cmp(6),cmp(8),cmp(9) ⇒ cost = 4
- search(13): cmp(4),cmp(6),cmp(8),cmp(9),cmp(10) ⇒ cost = 5