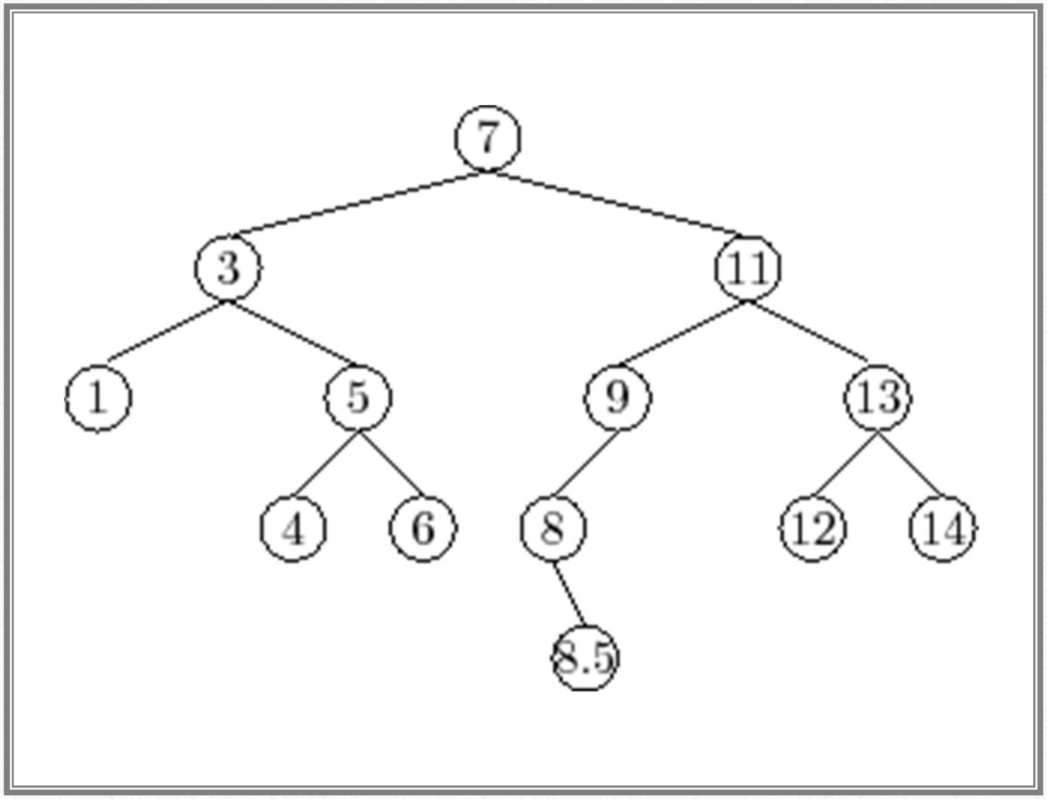
Data Structures

Exercise: Search Trees



Unbalanced Binary Search Tree

Time Complexity:

- member, insert, delete:

O (height).

- Height : Worst case: Θ (size).

12/12 1

Analyze the time complexity of the following code, expressed in n:

```
for (int i = 0; i <n; i ++) {
        t.insert (i); }
```

Use the course's uniform cost model, and make the following assumptions:

- That n is a non-negative integer, and that the type of int can represent all integers.
- That t is an unbalanced binary search tree that initially is plot.
- That the common order for integer (... <-1 <0 <1 <2 <...) used when depositing in the search tree.

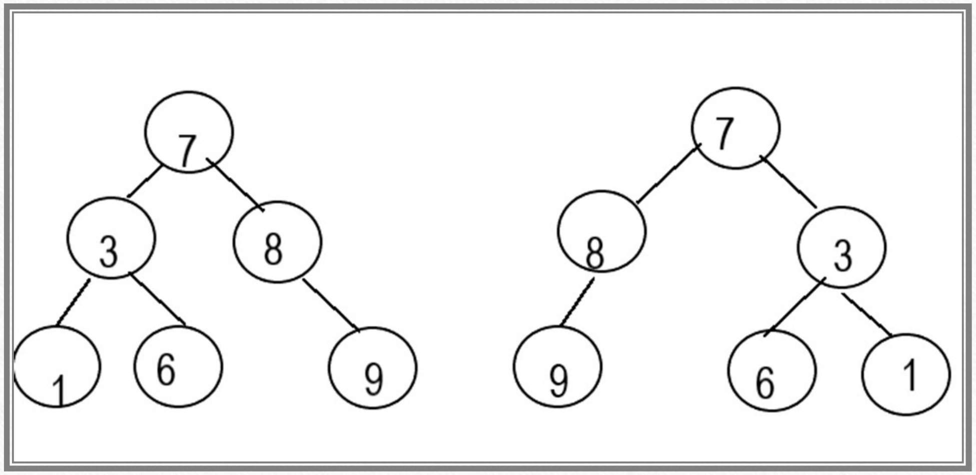
Unnecessary precise analysis can be rejected; Please use Θ-notation.

P9

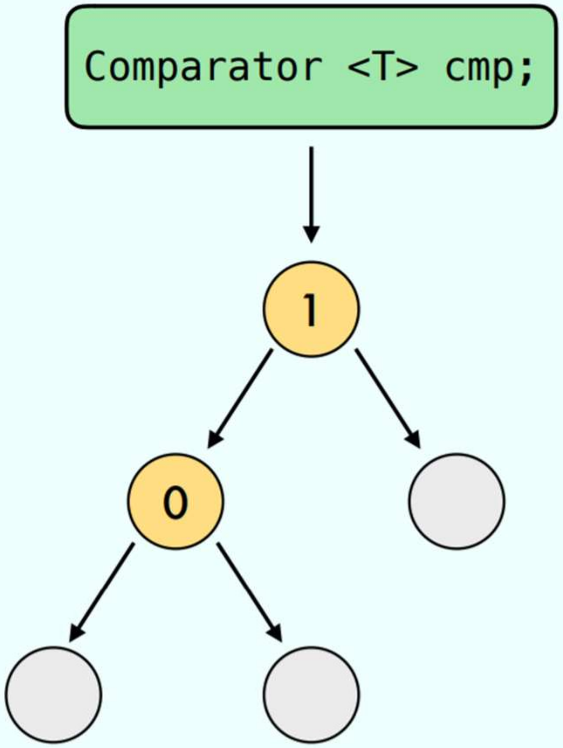
- Implement an operation which reverses a binary search tree. Explain why it is correct. Analyse its time complexity.
- Modify the binary tree data structure so that reversal can be implemented in constant time. The asymptotic worst-case time complexities of other operations should not change.

Reverse a BST

- Swap Left and Right Trees



Reversible Binary Search Tree



```
Comparator <T> cmp;
```

AVL Trees

- Binary search tree.
- Invariant (for each node):

  The height of the left and right tree trees differs maximum of 1.
- Height: Θ (log n).
- Because the height is Θ (log n), it takes all operations O (log n).

P10

- Implement a procedure which checks if a binary tree is an AVL tree (i.e. a search tree satisfying the AVL tree balancing invariant). The procedure's worst-case big-*O* time complexity should be as low as possible (assuming that tree elements can be compared in constant time).

13/04 1

Analyze the time complexity of the following code, expressed in n:

```
for (int i = 0; i <n; i ++) {
        for (int j = 0; j <i; j ++) {
                 t.insert (n);
          }
}
```

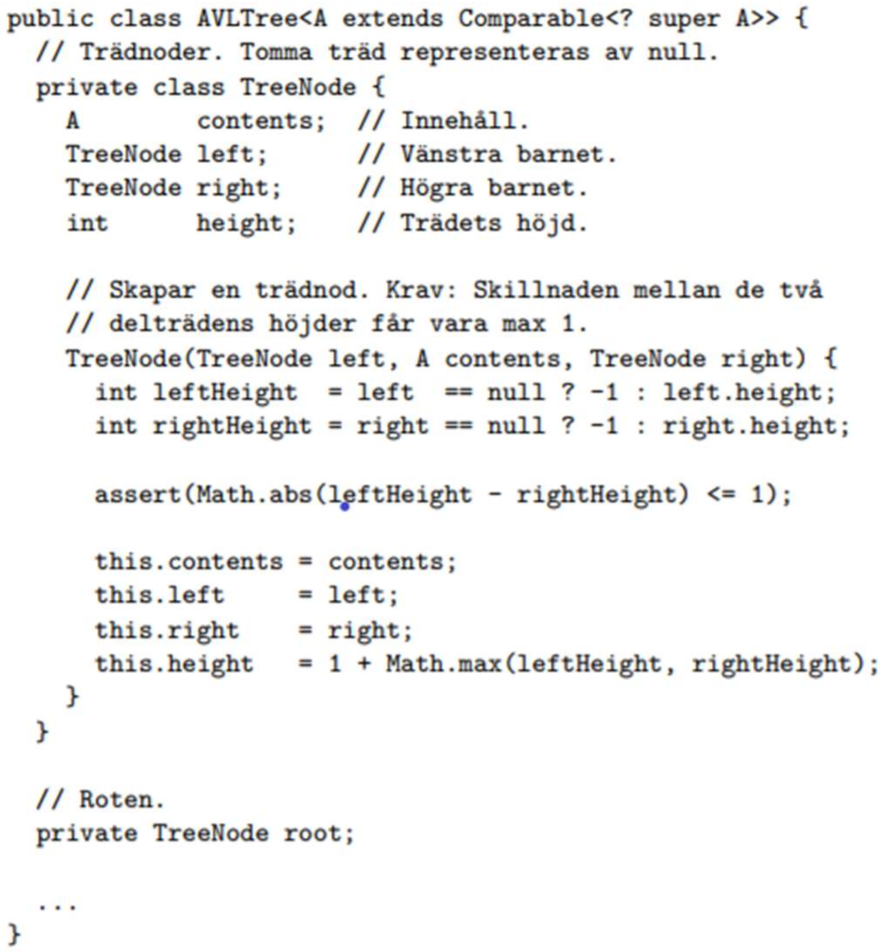
Use the course's uniform cost model, and make the following assumptions:

- That n is a non-negative integer, and that the type of int can represent all integers.
- That t is an AVL tree that is initially empty.
- That the common order for integer (... <-1 <0 <1 <2 <...) used when depositing in the tree.
- **If the same element is inserted twice in the tree, it will be written previous occurrence over.**

Unnecessary precise analysis can be rejected; Please use Θ-notation

12/08 3

Describe an algorithm that converts a sorted array into one AVL tree.



```
public class AVLTree<A extends Comparable<? super A>> {
  // Trädnoder. Tomma träd representeras av null.
  private class TreeNode {
    A        contents;  // Innehåll.
    TreeNode left;      // Vänstra barnet.
    TreeNode right;     // Högra barnet.
    int      height;    // Trädets höjd.

    // Skapar en trädnod. Krav: Skillnaden mellan de två
    // delträdens höjder får vara max 1.
    TreeNode(TreeNode left, A contents, TreeNode right) {
      int leftHeight  = left  == null ? -1 : left.height;
      int rightHeight = right == null ? -1 : right.height;

      assert(Math.abs(leftHeight - rightHeight) <= 1);

      this.contents = contents;
      this.left     = left;
      this.right    = right;
      this.height   = 1 + Math.max(leftHeight, rightHeight);
    }
  }

  // Roten.
  private TreeNode root;

  ...
}
```

12/12 4

The task is to construct a data structure for an image ADT with following operations:

new map () Constructs an empty image.

insert (k, v) Adds the pair (k, v) to the image.

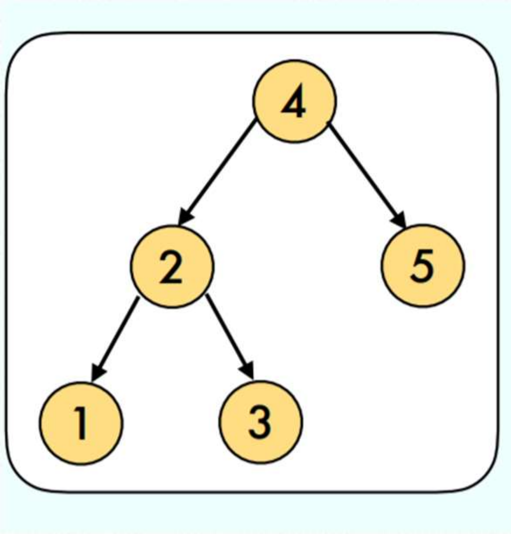
member (k) Determines if there are any pairs (k, v) in the image.

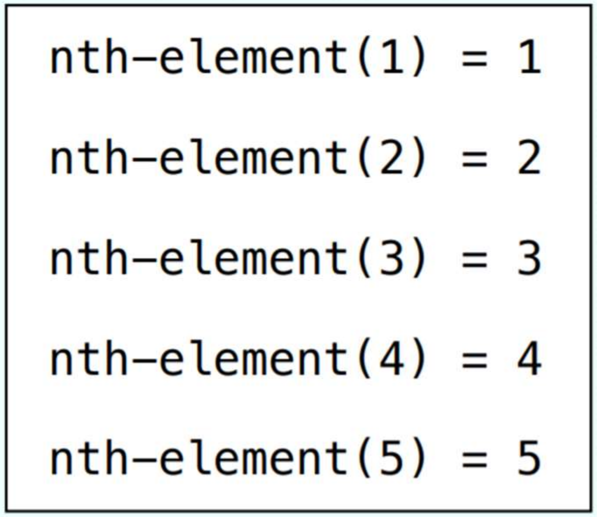
nth-smallest (i) Can only be run if i is a positive integer and image contains at least i elements.

You can assume that all keys and values are integers.

Time Complexity: new: $O(1)$, insert, member, nth-smallest: $O(\log n)$.

Find the n-th element in an AVL





```
nth-element(1) = 1

nth-element(2) = 2

nth-element(3) = 3

nth-element(4) = 4

nth-element(5) = 5
```

References

- Data Structures 2016 – Exercise Slides by Marco Vaseena
- Open Data Structures