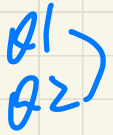
Lecture 21 - Monday, March 27

Announcements



- Assignment 3 due soon
- ProgTest2 guide & practice questions released
- Makeup Lecture for WrittenTest2
  + Expected to complete by: Exam Day

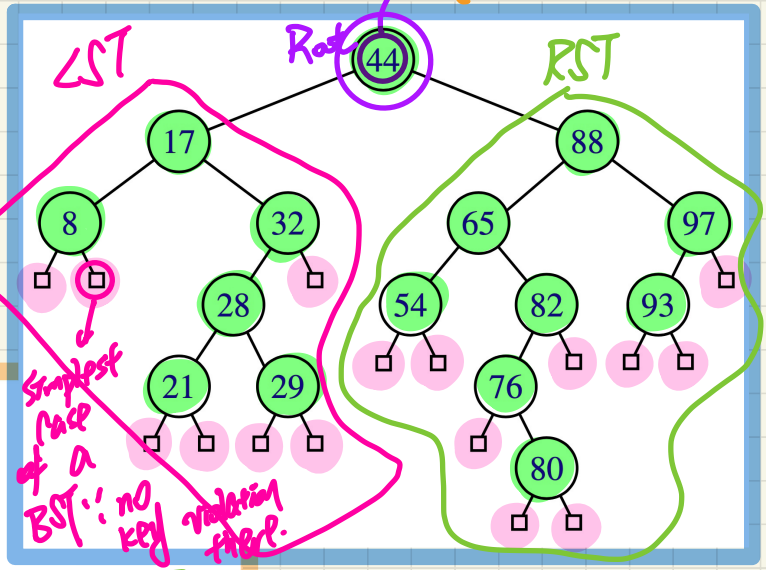
Lecture

Binary Search Tree (BST)

Definition and Property

Binary Search Trees: Recursive Definition

- external node
- internal node
  + LST
  + RST



searching.

data el.

entry

Node p stores
(key(p), value(p))

p

LST

$\forall n : n \in LST(p) \Rightarrow key(n) < key(p)$

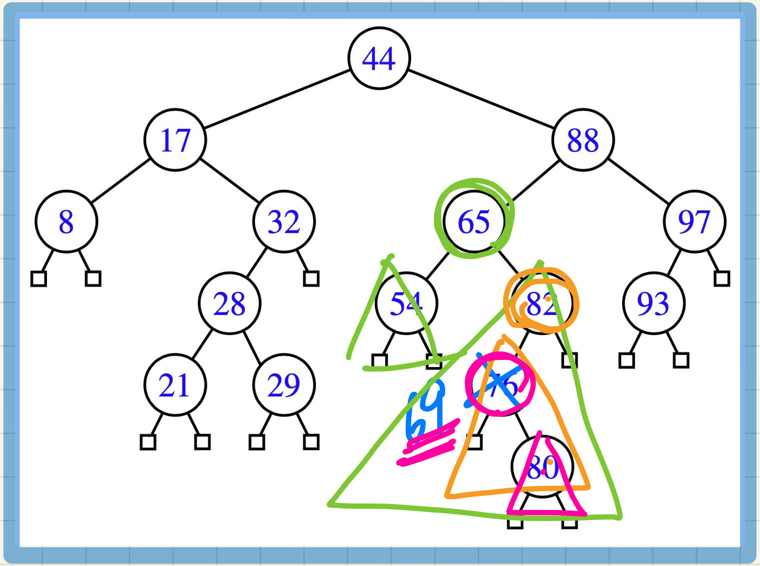
Each
node n of LST
is such that
key(n) < key(p)

RST

$\forall n : n \in RST(p) \Rightarrow key(n) > key(p)$

Each
node n of RST
is such that
key(n) > key(p)

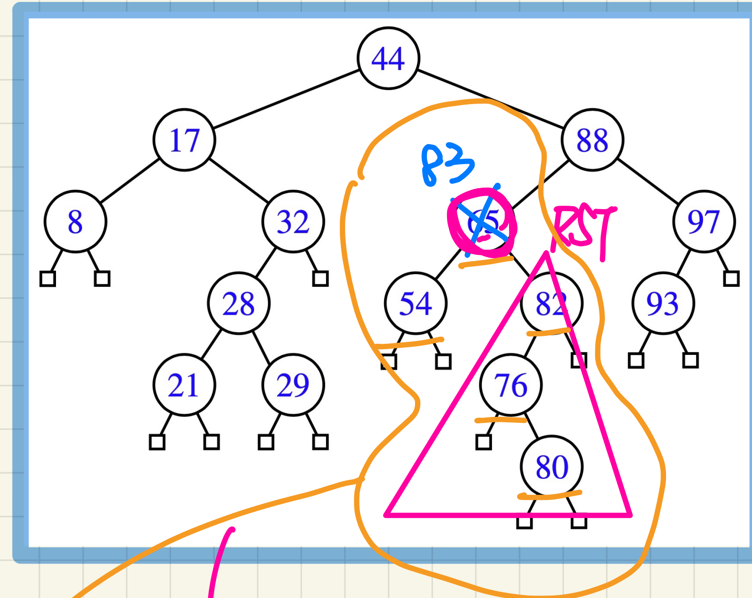
Q1.



✓ still a BST.

Tot: 54 83 76 82 80
not sorted

Q2.



not a BST
∵ 82 in 83's RST is not greater than it.

Binary Search Trees: Sorting Property

- BST: Non-Linear Structure
- In-Order Traversal

In-order traversal : 8 17 21 28 29 32 44 54 65 76 80 82 88 93 97

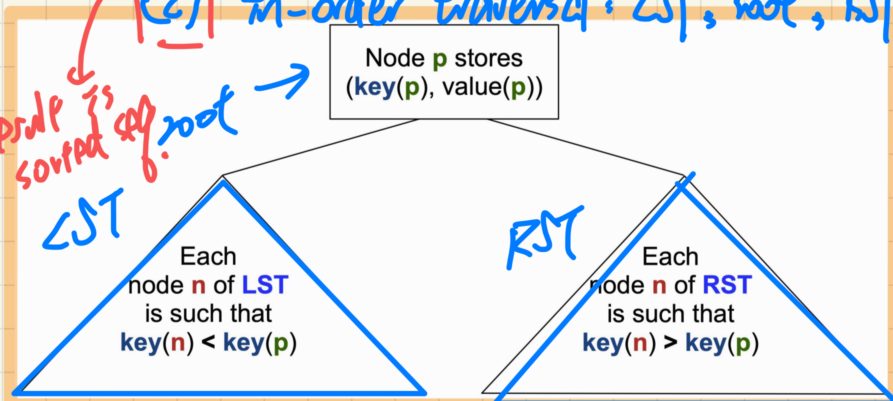
(In-order traversal of 44's LST: 8 17 21 28 29 32; I.o.t. of 44's RST: 54 65 76 80 82 88 93 97)

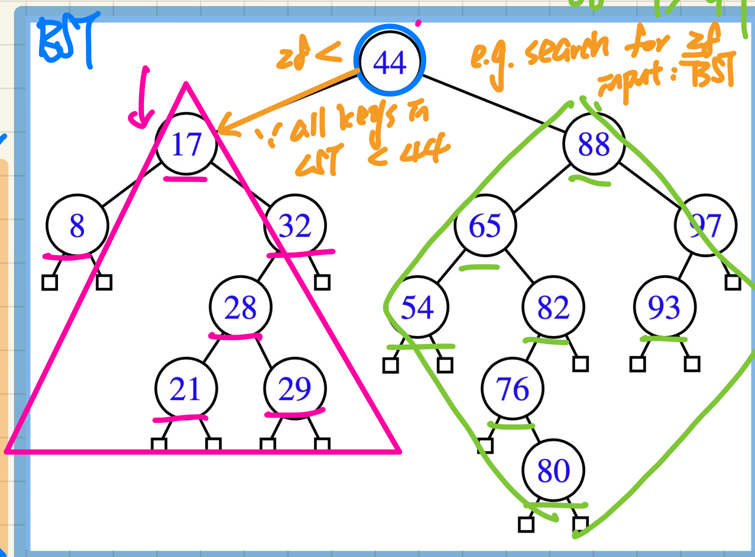
Given a T. that's a BST:

(1) search prop. satisfied

(2) in-order traversal : LST, root, RST

→ result is a sorted seq.





Lecture

Binary Search Tree (BST)

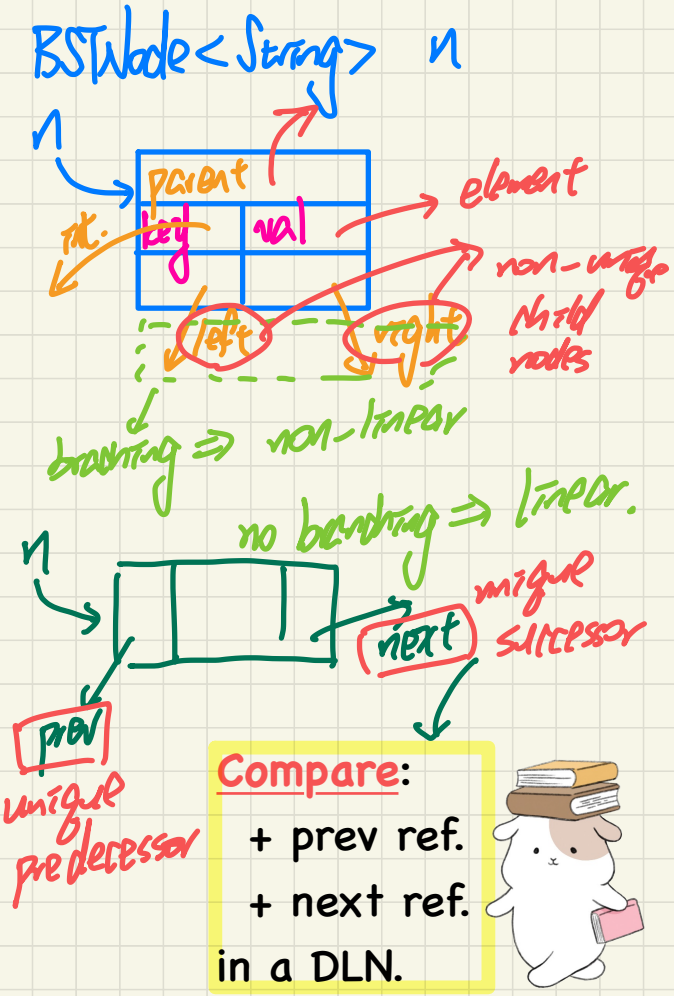
Implementing a Generic BST in Java
Tree Construction and Traversal

Generic, Binary Tree Nodes

```
public class BSTNode<E> {
  private int key; /* key */
  private E value; /* value */
  private BSTNode<E> parent; /* unique parent node */
  private BSTNode<E> left; /* left child node */
  private BSTNode<E> right; /* right child node */

  public BSTNode() { ... }
  public BSTNode(int key, E value) { ... }

  public boolean isExternal() {
    return this.getLeft() == null && this.getRight() == null;
  }
  public boolean isInternal() {
    return !this.isExternal();
  }
  public int getKey() { ... }
  public void setKey(int key) { ... }
  public E getValue() { ... }
  public void setValue(E value) { ... }
  public BSTNode<E> getParent() { ... }
  public void setParent(BSTNode<E> parent) { ... }
  public BSTNode<E> getLeft() { ... }
  public void setLeft(BSTNode<E> left) { ... }
  public BSTNode<E> getRight() { ... }
  public void setRight(BSTNode<E> right) { ... }
}
```



Generic, Binary Tree Nodes - Traversal

```java
import java.util.ArrayList;
public class BSTUtilities<E> {
  public ArrayList<BSTNode<E>> inOrderTraversal(BSTNode<E> root) {
    ArrayList<BSTNode<E>> result = null;
    if(root.isInternal()) {
      result = new ArrayList<>();

      if(root.getLeft().isInternal) {
        result.addAll(inOrderTraversal(root.getLeft()));
      }

      result.add(root);

      if(root.getRight().isInternal) {
        result.addAll(inOrderTraversal(root.getRight()));
      }
    }
    return result;
  }
}
```

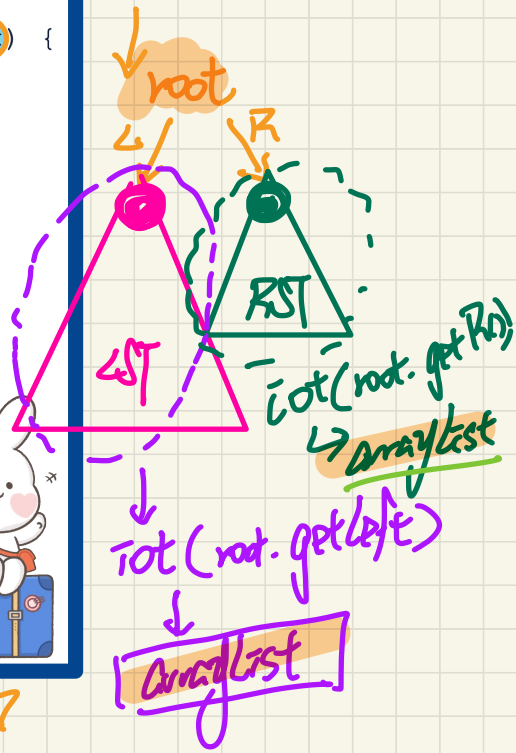
pre-order t.: B2, B1, B3

B1

B2

B3

tot result: ArrayList root ArrayList



Exercise 1. BSTNode<E>[]

2. SLLNode<BSTNode<E>>

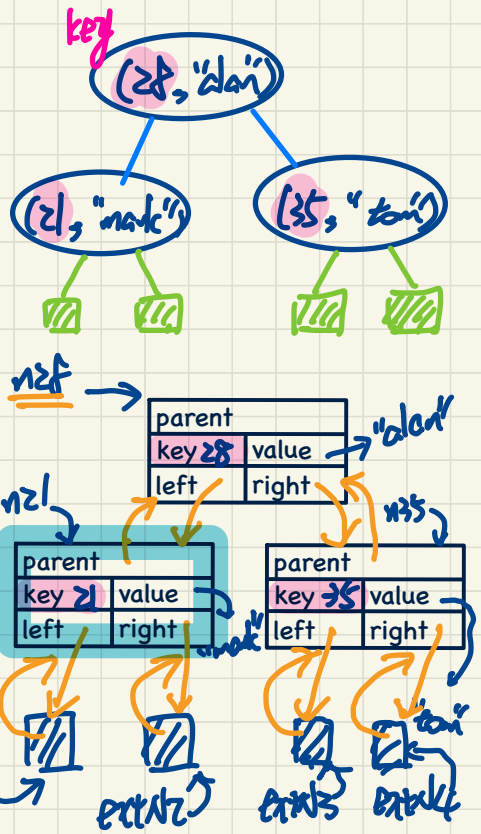
pre-order?

post-order?

Tracing: Constructing and Traversing a BST

```java
@Test
public void test_binary_search_trees_construction() {
  BSTNode<String> n28 = new BSTNode<>(28, "alan");
  BSTNode<String> n21 = new BSTNode<>(21, "mark");
  BSTNode<String> n35 = new BSTNode<>(35, "tom");
  BSTNode<String> extN1 = new BSTNode<>();
  BSTNode<String> extN2 = new BSTNode<>();
  BSTNode<String> extN3 = new BSTNode<>();
  BSTNode<String> extN4 = new BSTNode<>();
  n28.setLeft(n21); n21.setParent(n28);
  n28.setRight(n35); n35.setParent(n28);
  n21.setLeft(extN1); extN1.setParent(n21);
  n21.setRight(extN2); extN2.setParent(n21);
  n35.setLeft(extN3); extN3.setParent(n35);
  n35.setRight(extN4); extN4.setParent(n35);
  BSTUtilities<String> u = new BSTUtilities<>();
  ArrayList<BSTNode<String>> inOrderList = u.inOrderTraversal(n28);
  assertTrue(inOrderList.size() == 3);
  assertEquals(21, inOrderList.get(0).getKey());
  assertEquals("mark", inOrderList.get(0).getValue());
  assertEquals(28, inOrderList.get(1).getKey());
  assertEquals("alan", inOrderList.get(1).getValue());
  assertEquals(35, inOrderList.get(2).getKey());
  assertEquals("tom", inOrderList.get(2).getValue());
}
```

sorting property.



inorder traversing - n21

- n28.getLeft()

- n28.getL().getR().getR()

extN1.getP()