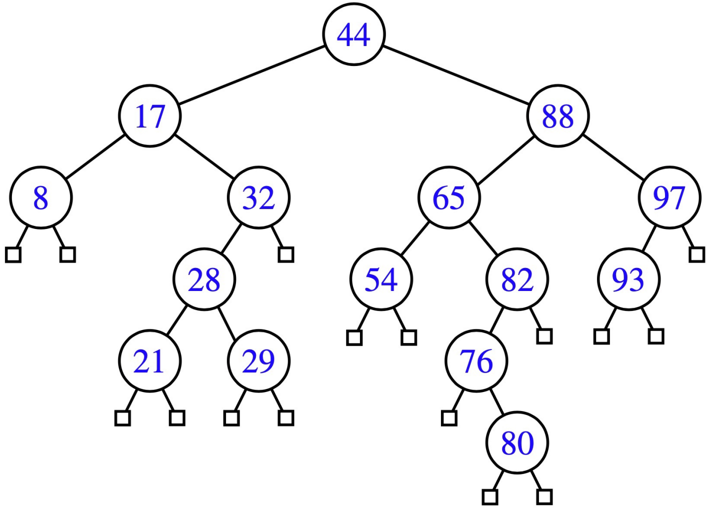
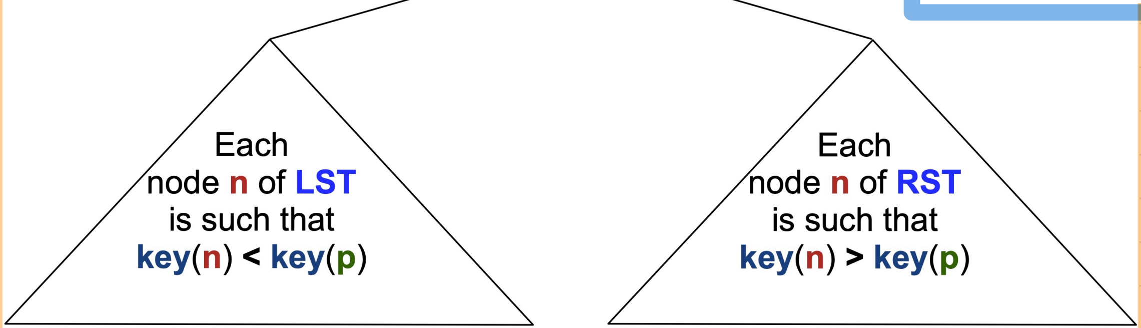
Binary Search Trees: Recursive Definition

- external node
- internal node
  - + LST
  - + RST



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



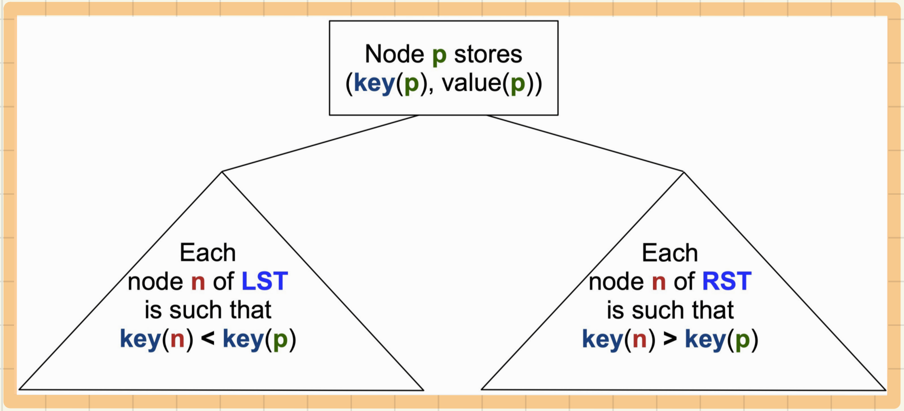
Each
node **n** of **LST**
is such that
key(**n**) < **key**(**p**)

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

Is a Singleton BT a BST?

Binary Search Trees: Sorting Property

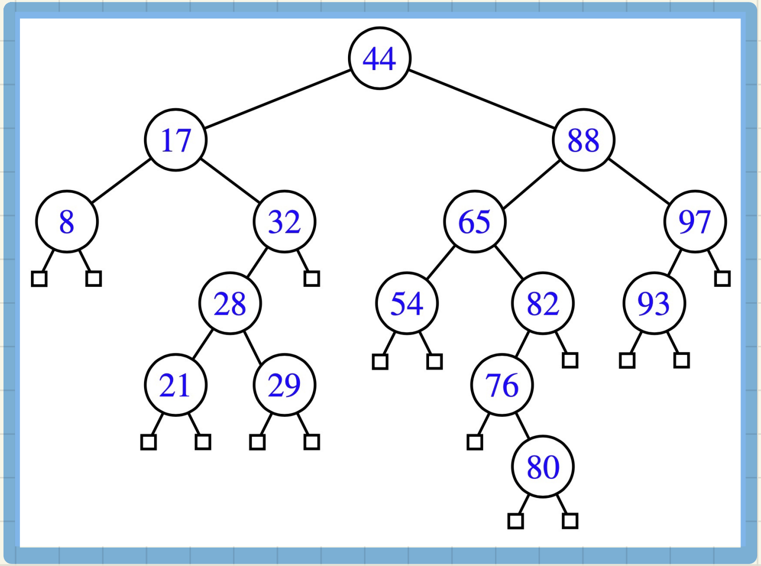
- BST: Non-Linear Structure
- In-Order Traversal



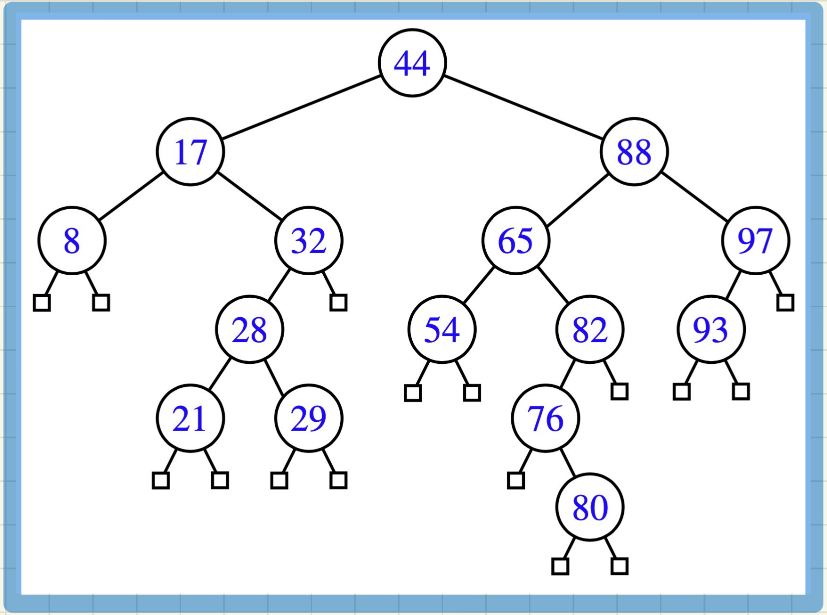
Node p stores (key(p), value(p))

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

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



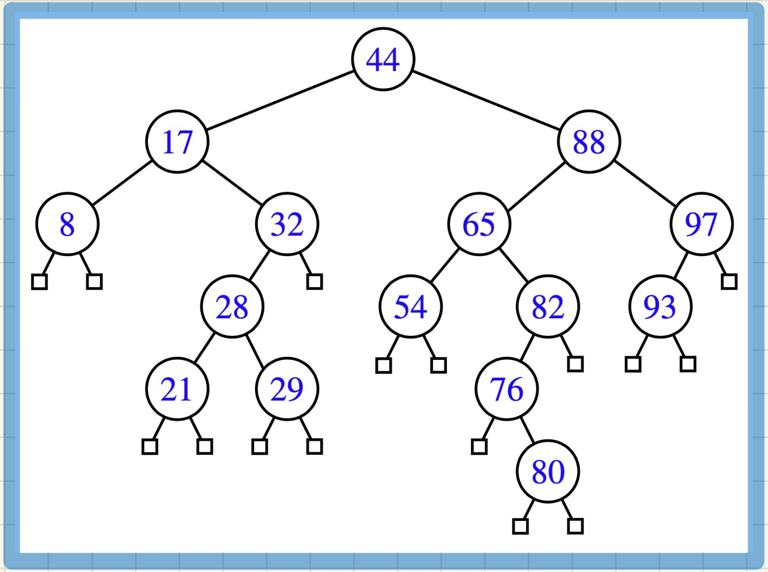
Building Sorted Seq. from In-Order Traversal on BST



Exercise: Checking the Search Property (1)

Remember: For a BT to be a BST, the Search Property should hold **recursively** on the **root** of each **subtree**.

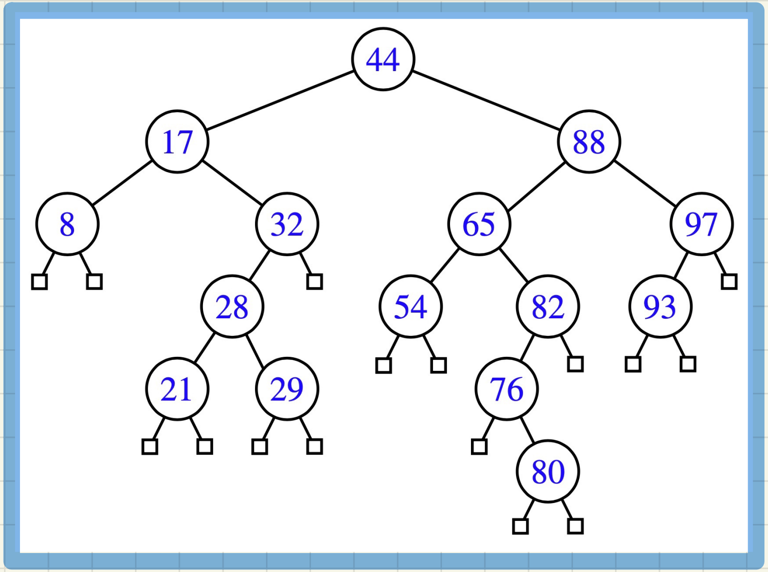
In-Order: <8, 17, 21, 28, 29, 32, 44, 54, 65, 76, 80, 82, 88, 93, 97>



Exercise: Checking the Search Property (2)

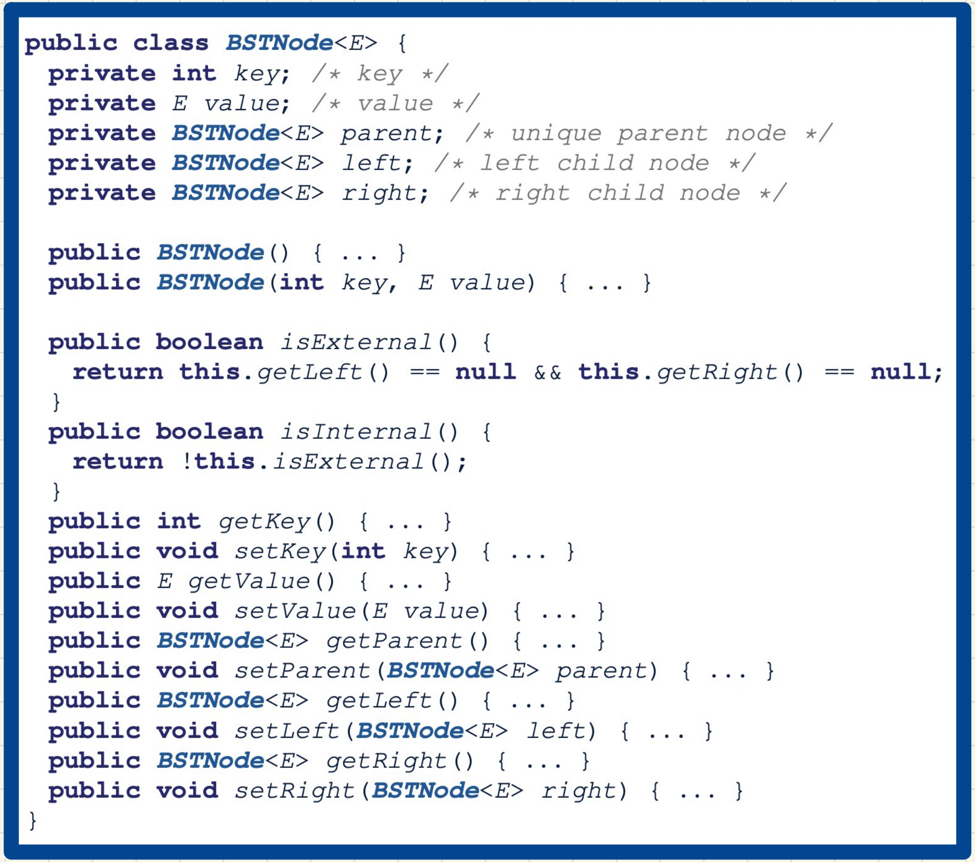
Remember: For a BT to be a BST, the Search Property should hold **recursively** on the **root** of each **subtree**.

In-Order: <8, 17, 21, 28, 29, 32, 44, 54, 65, 76, 80, 82, 88, 93, 97>



Visual Summary: In-Order Traversal on BST

Generic, Binary Tree Nodes



```java
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) { ... }
}
```

Compare:

+ prev ref.

+ next ref.

in a DLN.

Generic, Binary Tree Nodes - Traversal

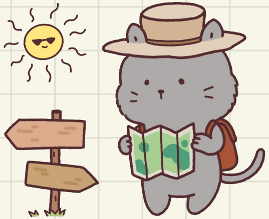
```java
import java.util.ArrayList;
public class BSTUtilities<E> {
  public ArrayList<BSTNode<E>> inOrderTraversal(BSTNode<E> root) {
    ArrayList<BSTNode<E>> result = null;
    if(root.isInternal()) {
      result = new ArrayList<>();

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

      result.add(root);

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

Tracing: Constructing and Traversing a BST

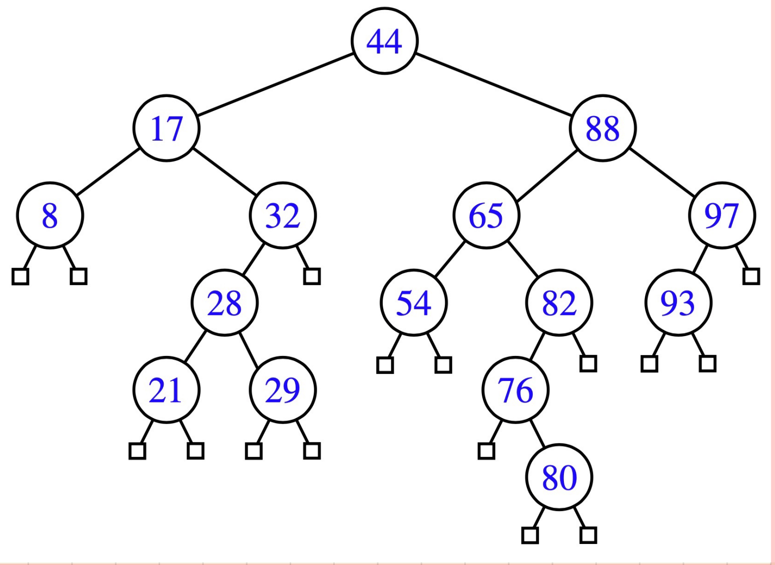
```
@Test
public void test_binary_search_trees_construction() {
  BSTNode<String> n28 = new BSTNode<>(28, "alan");
  BSTNode<String> n21 = new BSTNode<>(21, "mark");
  BSTNode<String> n35 = new BSTNode<>(35, "tom");
  BSTNode<String> extN1 = new BSTNode<>();
  BSTNode<String> extN2 = new BSTNode<>();
  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());
}
```

parent	
key	value
left	right

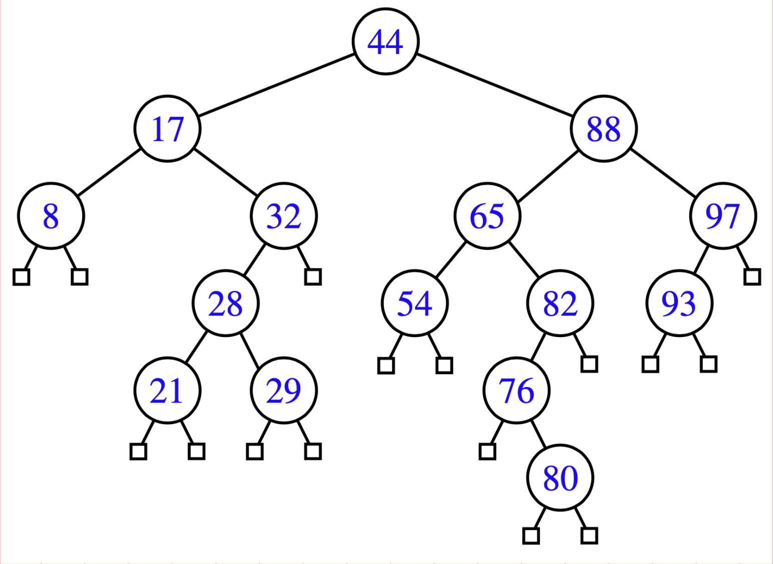
BST Operation: Searching a Key



Search key 65

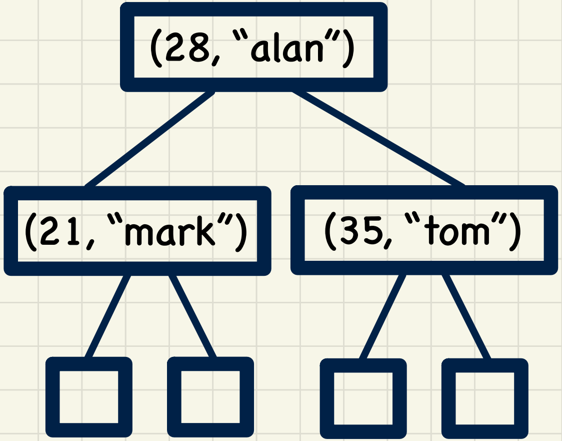


Search key 68



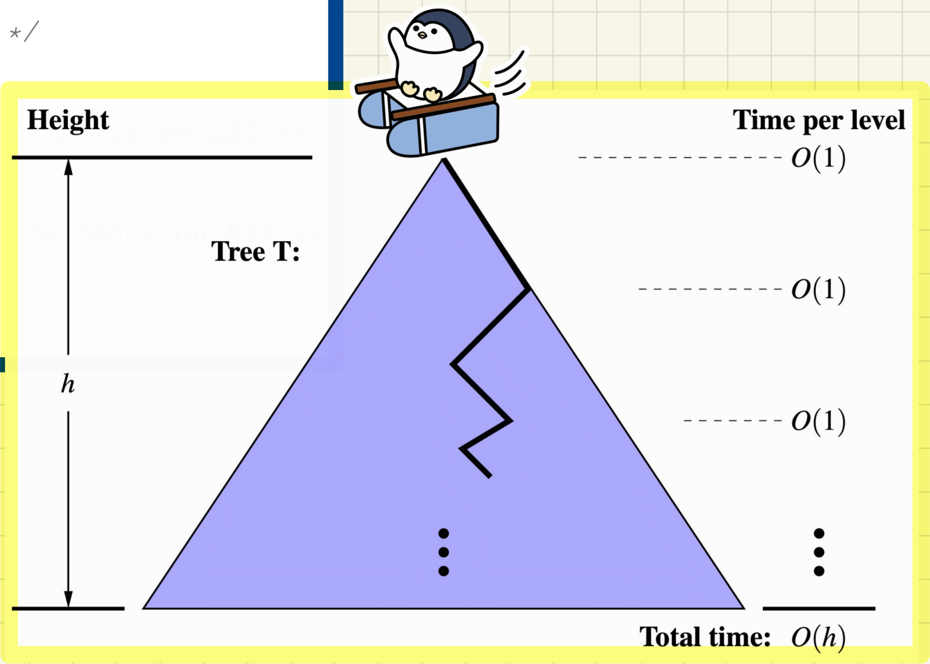
Tracing: Searching through a BST

```
@Test
public void test_binary_search_trees_search() {
  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<>();
  /* search existing keys */
  assertTrue(n28 == u.search(n28, 28));
  assertTrue(n21 == u.search(n28, 21));
  assertTrue(n35 == u.search(n28, 35));
  /* search non-existing keys */
  assertTrue(extN1 == u.search(n28, 17)); /* *17* < 21 */
  assertTrue(extN2 == u.search(n28, 23)); /* 21 < *23* < 28 */
  assertTrue(extN3 == u.search(n28, 33)); /* 28 < *33* < 35 */
  assertTrue(extN4 == u.search(n28, 38)); /* 35 < *38* */
}
```

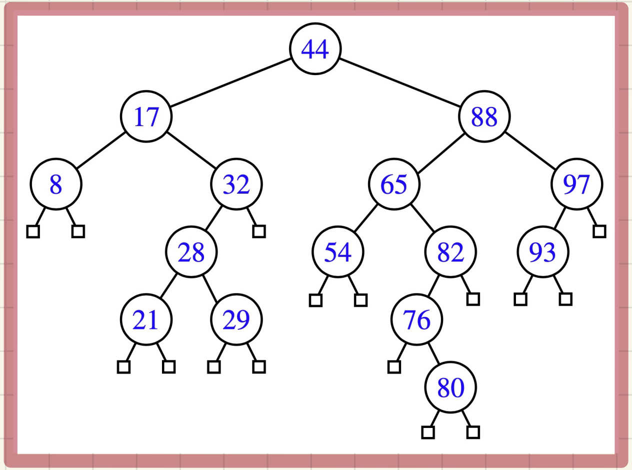


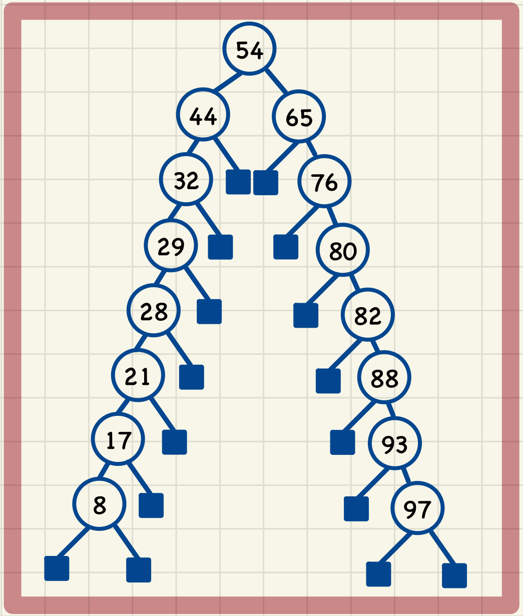
Running Time: Search on a BST

```
public BSTNode<E> search(BSTNode<E> p, int k) {
  BSTNode<E> result = null;
  if(p.isExternal()) {
    result = p; /* unsuccessful search */
  }
  else if(p.getKey() == k) {
    result = p; /* successful search */
  }
  else if(k < p.getKey()) {
    result = search(p.getLeft(), k); /* recur on LST */
  }
  else if(k > p.getKey()) {
    result = search(p.getRight(), k); /* recur on RST */
  }
  return result;
}
```



Binary Search: Non-Linear vs. Linear Structures

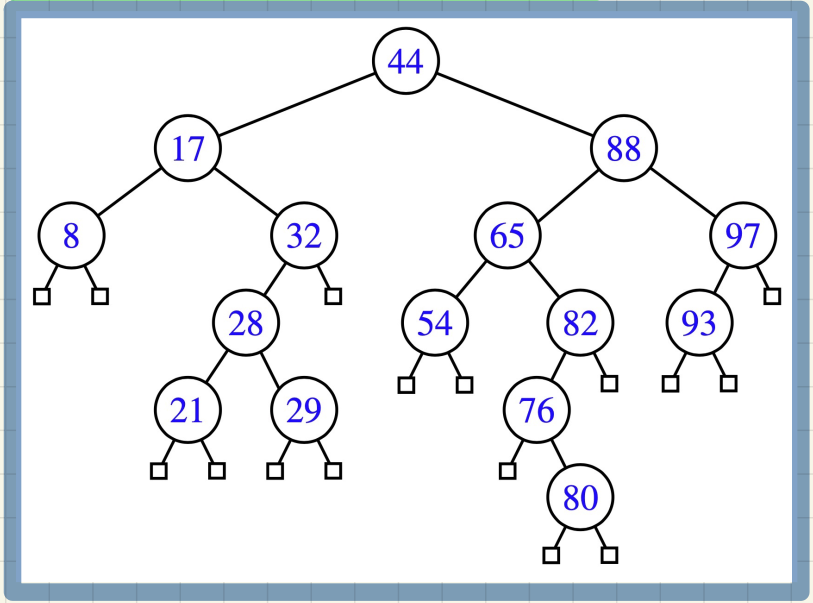




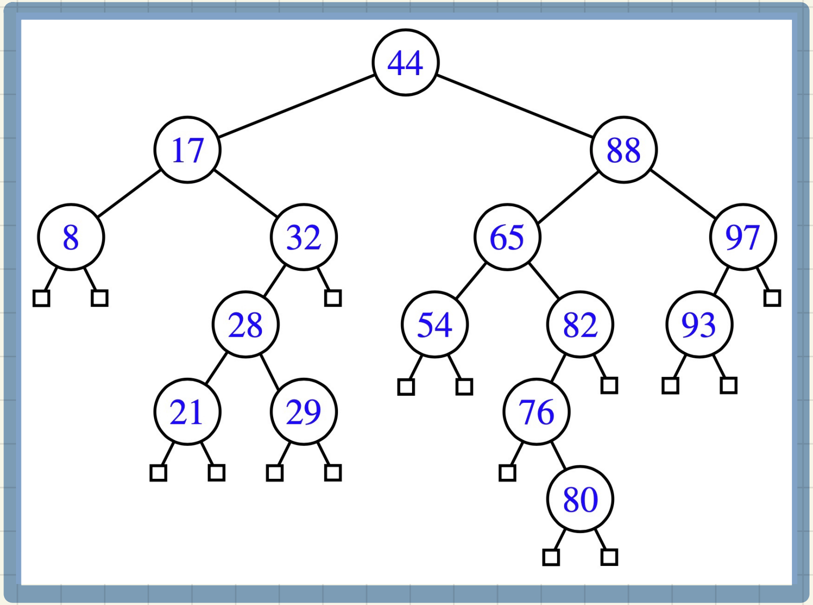
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
8	17	21	28	29	32	44	54	65	76	80	82	88	93	97

Visualizing BST Operation: Insertion

Insert Entry (28, “suyeon”)



Insert Entry (68, “yuna”)


44
17
88
8
32
65
97
28
54
82
93
21
29
76
80