

Indian Institute of Technology Madras
ZANZIBAR CAMPUS

Z5007 · PROGRAMMING AND DATA STRUCTURES

Binary Search Trees: Why, the Rule, Search and Insert

Week 7 · Thursday 19 November 2026 · Canning, Broder & Lafore Ch 8

Dr. Innocent Nyalala | MTech Semester 1 | IIT Madras Zanzibar

Today's Plan

IIT Madras Zanzibar

1. (0–8 min) The problem: fast search and fast insert together; where search trees are used
2. (8–16 min) The rule, its properties, and checking a tree
3. (16–28 min) Search and insert: step by step, with the code
4. (28–36 min) Cost and shape; minimum, maximum; range queries
5. (36–40 min) Practice, then Lab 5 this afternoon

Learning Outcomes

IIT Madras Zanzibar

By the end of this session, you will be able to:

Explain	why search trees exist and what they do better than lists, arrays and dictionaries
State	the BST rule and its properties, and test whether a tree obeys it
Search and insert	by hand and in code, following one path from the root
Predict	the shape and cost of a tree from its insertion order

The Problem: Fast Search and Fast Insert Together

IIT Madras Zanzibar

structure	search	insert	sorted output
unsorted list	O(n): scan everything	O(1): append	O(n log n): sort first
sorted array	O(log n): binary search	O(n): shift items	O(n): already sorted
linked list	O(n): follow the links	O(1) at the head	O(n log n): sort first
balanced search tree	O(log n)	O(log n)	O(n): inorder walk

? **Think:** A system holds 1,000,000 records and handles millions of mixed searches and inserts a day. Which row do you choose?

Answer: The tree. Lists make every search scan up to 1,000,000 items; the sorted array makes every insert shift up to 1,000,000; the tree does both in about 20 steps.

Where Search Trees Are Used

IIT Madras Zanzibar

1 **Ordered maps**

C++ map and set, Java TreeMap: balanced search trees keep keys sorted.

2 **Range queries**

All keys between 20 and 30, without scanning everything.

3 **Database indexes**

B-trees and B+ trees are many-child relatives of the search tree.

4 **Operating systems**

Linux keeps memory regions and runnable processes in red-black trees.

5 **Nearest key**

The closest key above or below a value: successor and predecessor.

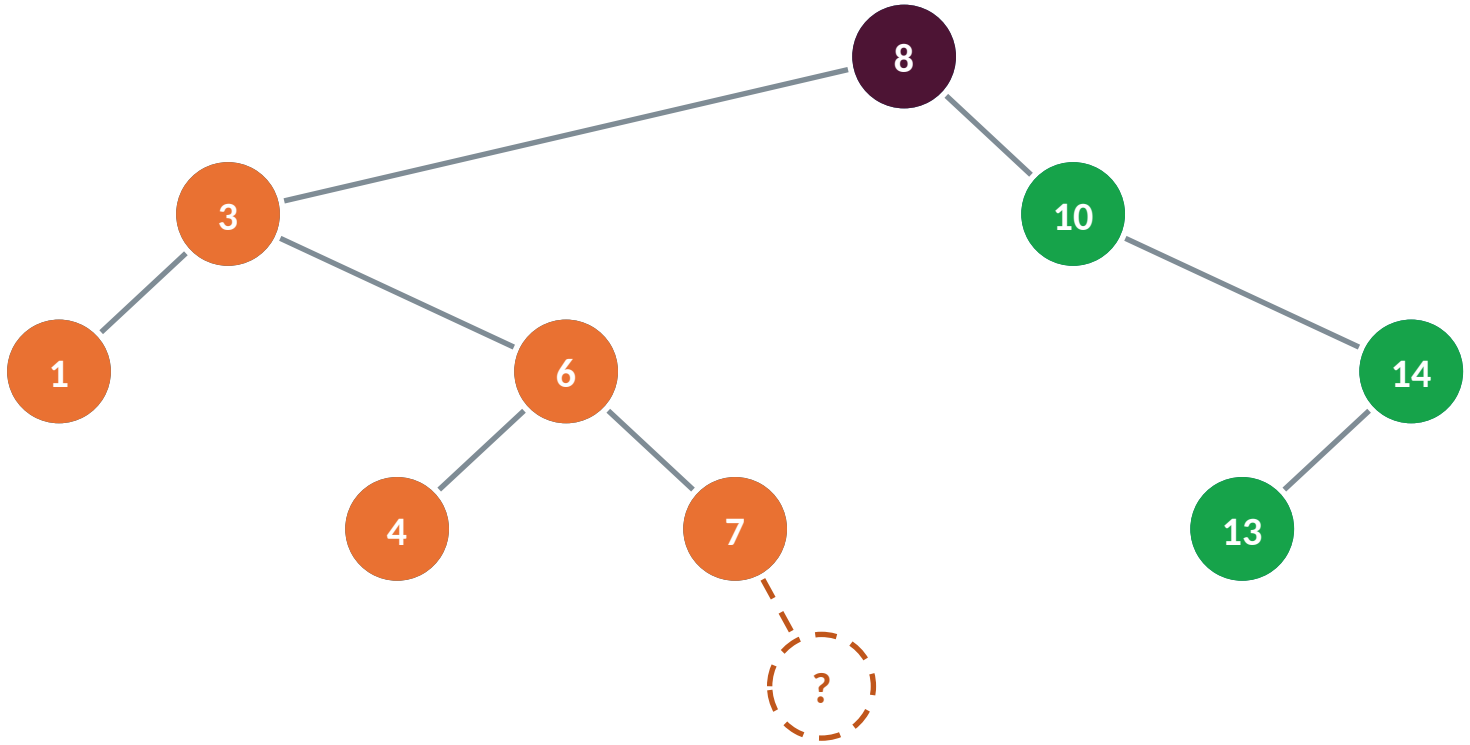
6 **Sorted output**

An inorder walk lists all keys in order, with no sorting step.

? **Think:** A Python dictionary finds a key in about constant time. Why would a search tree still be useful?

Answer: A dictionary keeps no order: no minimum, no next larger key, no sorted listing, no range 20 to 30 without checking every item. A search tree gives all of these quickly.

The Binary Search Tree Rule



left of 8: all smaller (3, 1, 6, 4, 7)

right of 8: all larger or equal (10, 14, 13)

the same rule holds at EVERY node, not only at the root

? **Think:** Which keys are allowed to sit in the dashed empty spot to the right of 7?

Answer: Larger than 7 (right of 7) and smaller than 8 (inside the left subtree of 8): the open range (7, 8), for example 7.5.

Properties of a Binary Search Tree

IIT Madras Zanzibar

1 **Inorder is sorted**

Left, node, right lists the keys in increasing order.

2 **Min and max**

The minimum is the leftmost node, the maximum the rightmost.

3 **One path per operation**

Search, insert and delete walk a single path: cost O(height).

4 **Shape depends on history**

The same keys in another order can give another tree.

5 **Height range**

About $\log_2 n$ up to $n - 1$, as for any binary tree.

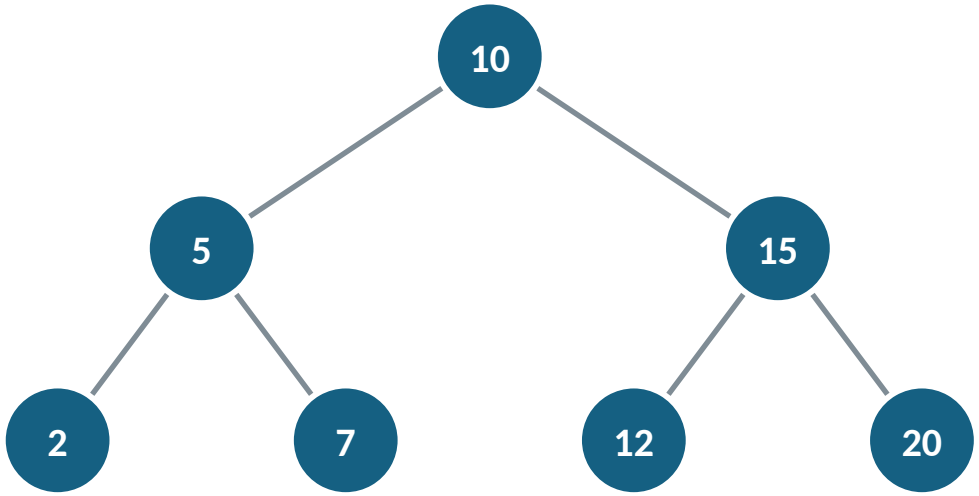
6 **Equal keys**

Our code sends an equal key to the right; or reject duplicates.

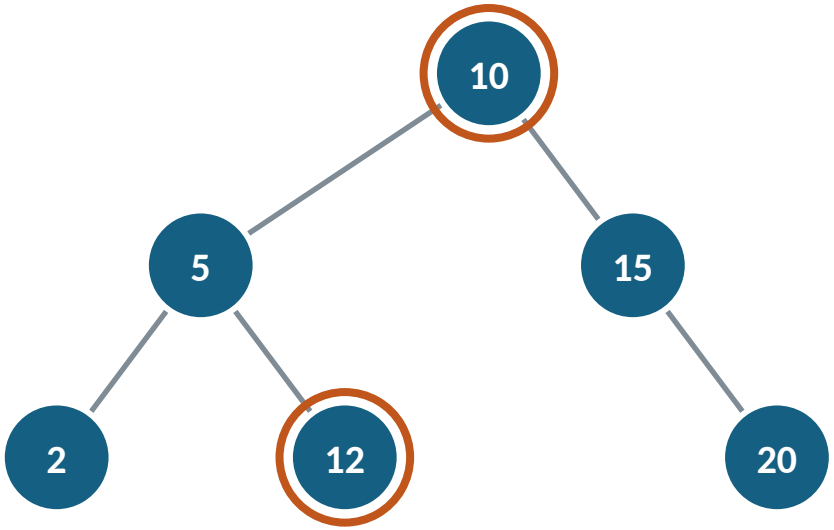
? **Think:** The keys 1, 2, 3 can be inserted in 6 different orders. How many different search trees result?

Answer: Five. 2 1 3 and 2 3 1 build the same balanced tree; 1 2 3 and 3 2 1 give the two chains; 1 3 2 and 3 1 2 give the two zigzags.

Is It a Search Tree? A Local Check Is Not Enough



Tree 1



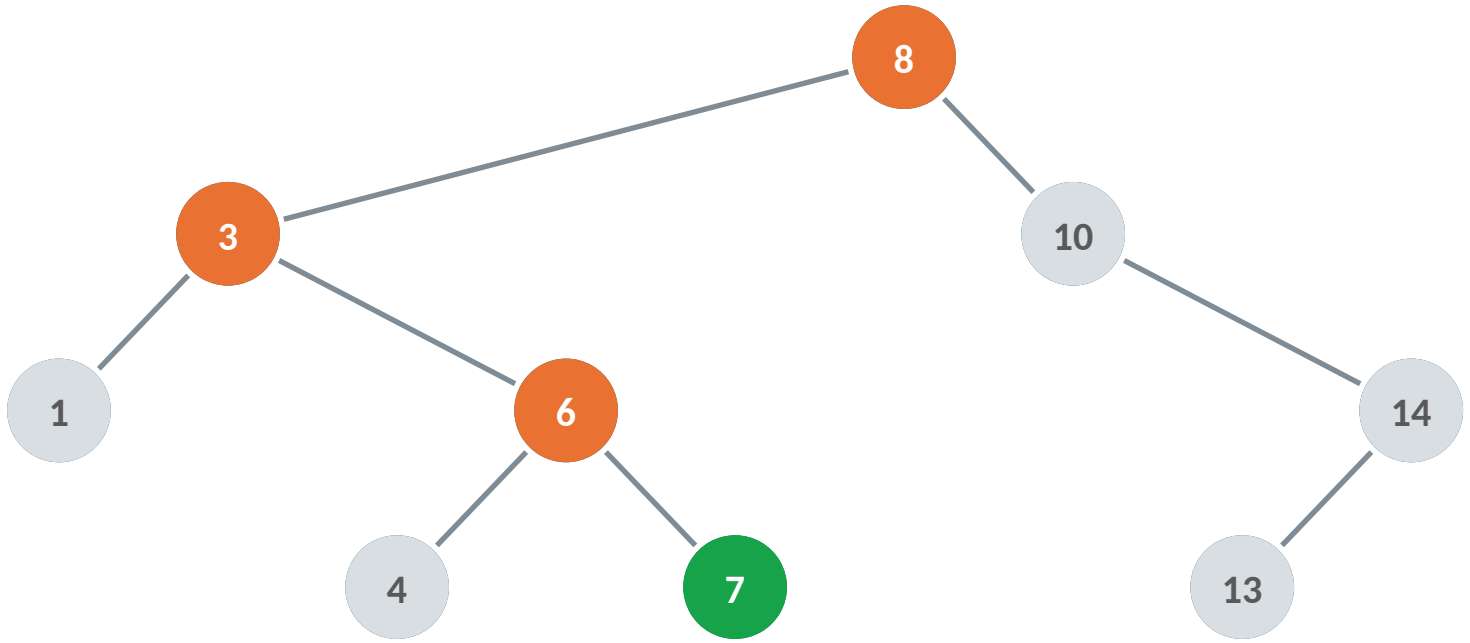
Tree 2

inorder: 2 5 12 10 15 20 (12 then 10: decrease)

? **Think:** Which tree is a binary search tree? Find the exact key that breaks the rule in the other. How could one traversal test this?

Answer: Tree 1 is. In Tree 2, 12 sits left of 10 but is larger. Test: the inorder must be strictly increasing; Tree 2 gives 2 5 12 10 15 20.

Search: Follow One Path, Rule Out a Subtree at Every Step



search for 7

at 8: 7 < 8: go left, rule out the right subtree

at 3: 7 > 3: go right, rule out the left subtree

at 6: 7 > 6: go right, rule out the left subtree

at 7: 7 = 7: found, 4 comparisons

? **Think:** How many comparisons does it take to find 13, and how many nodes does the very first comparison rule out?

Answer: Four comparisons (8, 10, 14, 13). At the root 13 > 8, which rules out the whole left subtree: 5 of the 9 nodes.

The search Code, Line by Line

IIT Madras Zanzibar

```
1  def search(node, key):
2      if node is None: return None
3      if key == node.value: return node
4      if key < node.value:
5          return search(node.left, key)
6      return search(node.right, key)
```

Line by line

Line 2 empty spot: the key is not here, return None

Line 3 equal: found, return this node

Lines 4–5 smaller: only the LEFT subtree can hold it

Line 6 larger: only the RIGHT subtree can hold it

? **Think:** Rewrite search as a loop, with no recursion. (Same move as factorial in Week 5.)

Answer: while node is not None and node.value != key: node = node.left if key < node.value else node.right; return node.

Insert: Search, Then Add a Leaf Where It Fails

IIT Madras Zanzibar

insert 8: empty tree: becomes the root

insert 3: left

insert 10: right

insert 1: left, left

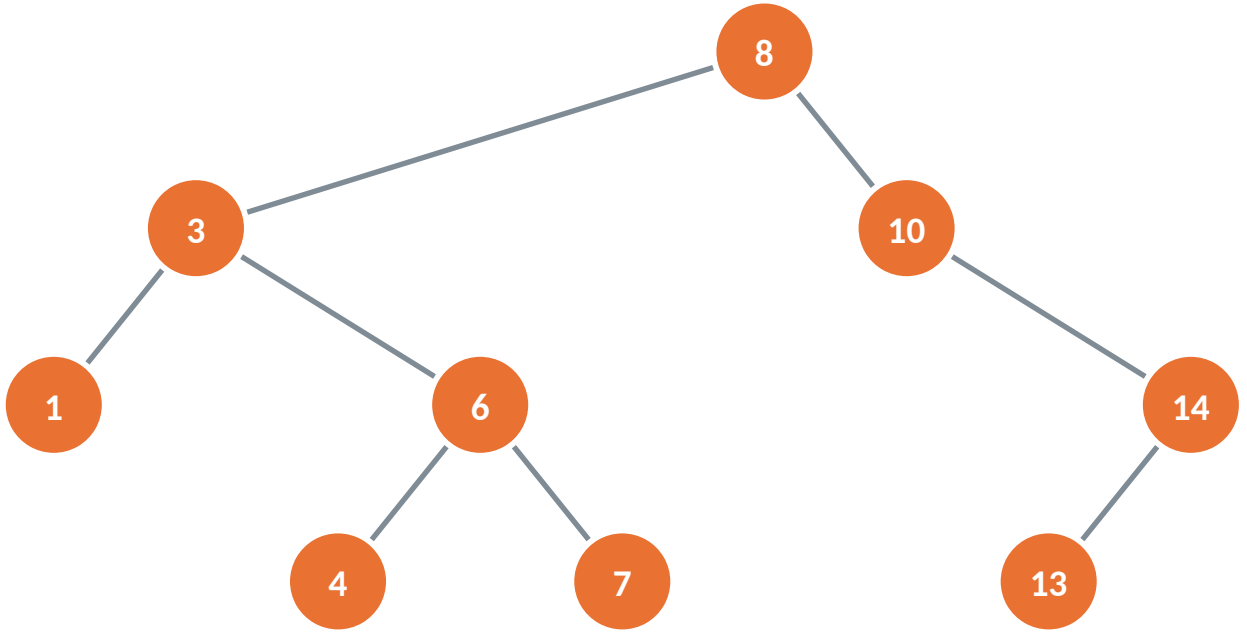
insert 6: left, right

insert 14: right, right

insert 4: left, right, left

insert 7: left, right, right

insert 13: right, right, left



? **Think:** Insert 5 into this tree. Which node becomes its parent, and does the height change?

Answer: 5 goes 8 left, 3 right, 6 left, 4 right: the right child of 4, on level 4. The height grows from 3 to 4.

The insert Code, Line by Line

IIT Madras Zanzibar

```
1  def insert(node, key):
2      if node is None: return Node(key)
3      if key < node.value:
4          node.left = insert(node.left, key)
5      else:
6          node.right = insert(node.right, key)
7      return node
```

Line by line

Line 2 empty spot found: return a new leaf

Line 3–4 smaller: the key belongs in the left subtree

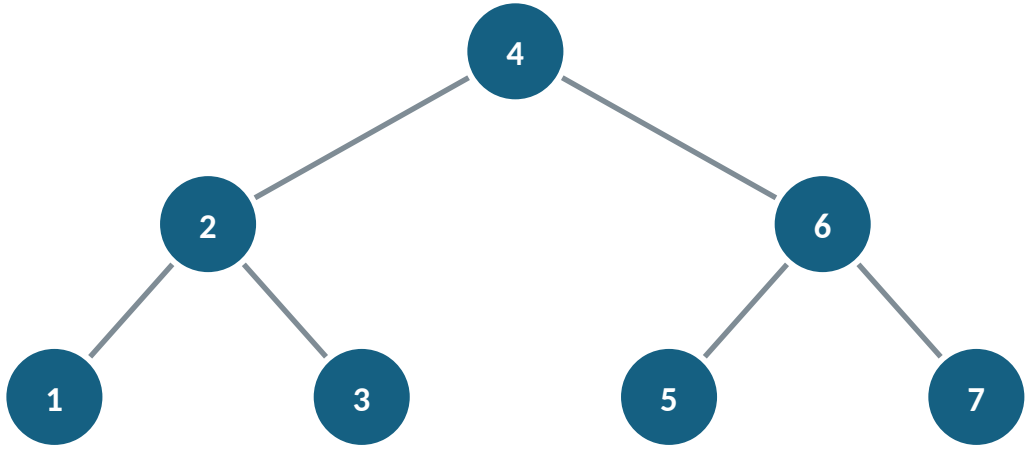
Line 5–6 otherwise: in the right subtree

Line 7 return this node so the parent keeps its arrow

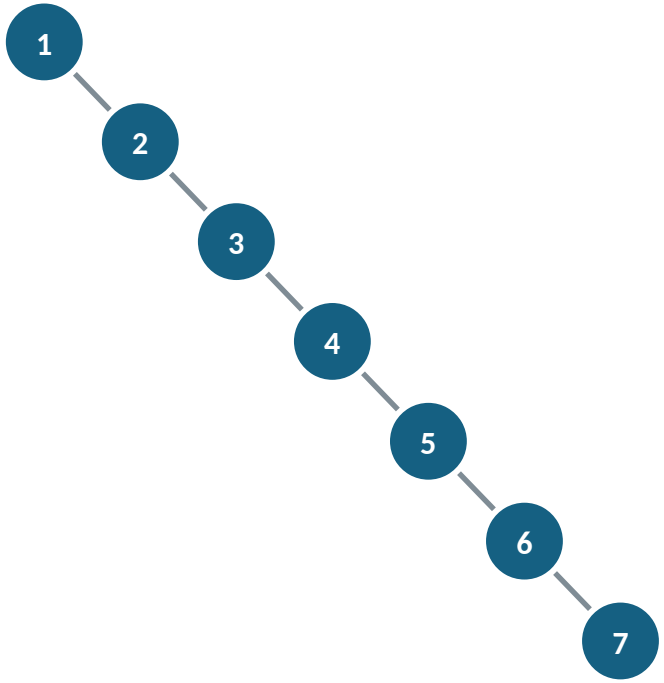
? **Think:** What does this code do when the key is already in the tree? How would you reject duplicates?

Answer: A duplicate is not smaller, so it goes right. To reject duplicates add: if key == node.value: return node (before the other tests).

Same Keys, Different Order, Different Shape



insert 4, 2, 6, 1, 3, 5, 7
height 2: a search needs at most 3 comparisons



insert 1, 2, 3, 4, 5, 6, 7
height 6: a chain, a search needs up to 7

? **Think:** Find a different insertion order that also builds the perfect tree on the left. What condition must every such order satisfy?

Answer: Any order where each parent comes before its children, e.g. 4 6 2 5 7 1 3. There are 80 such orders out of 5,040. Sorted order is the worst of them all.

What a Search Tree Costs: the Height

IIT Madras Zanzibar

tree of n keys	height	search or insert
balanced	about $\log_2 n$	n = 1,000,000: about 20 steps
random insertion order	about 2 to 3 × $\log_2 n$	usually fine: about 40 to 60 steps
sorted insertion order	n − 1 (a chain)	up to n steps: as slow as a list

? **Think:** Why is a random insertion order usually fine for a search tree, while sorted input is dangerous?

Answer: A random first key splits the rest into comparable halves, again and again (like a random pivot). Sorted input always goes right and builds a chain.

Minimum, Maximum and the Next Key

IIT Madras Zanzibar

```
 1  def minimum(node):
 2      while node.left is not None:
 3          node = node.left
 4      return node
 5  def maximum(node):
 6      while node.right is not None:
 7          node = node.right
 8      return node
 9  def inorder(node):
10      if node is None: return
11      inorder(node.left)
12      print(node.value)
13      inorder(node.right)
```

Line by line

Line 1–4 minimum: go left until you cannot (1 in our tree)

Line 5–8 maximum: go right until you cannot (14)

Line 9–13 inorder walk prints the keys sorted

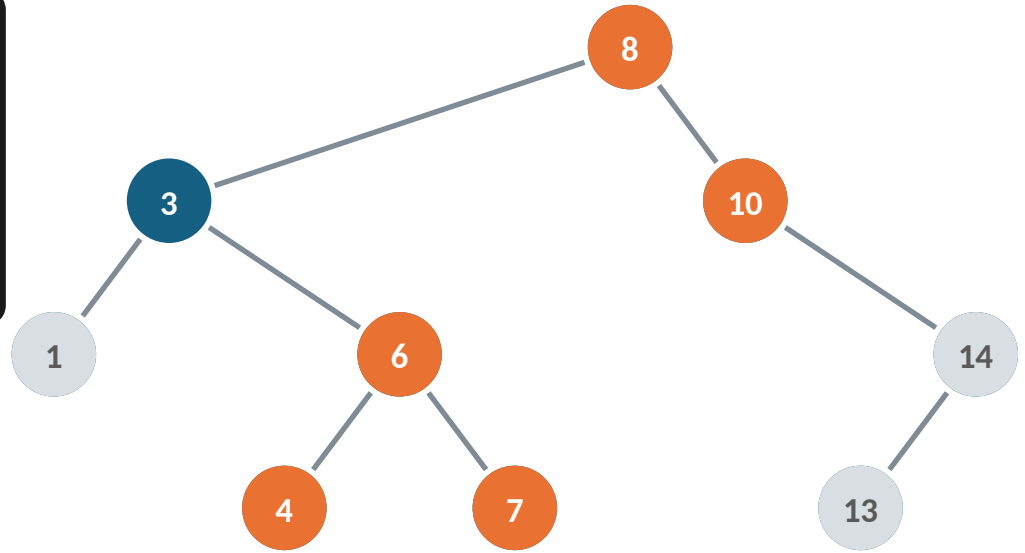
? **Think:** How would you find the second smallest key in a search tree?

Answer: It is the successor of the minimum: the leftmost node of the minimum's right subtree if it has one, otherwise the minimum's parent.

A Query a Dictionary Cannot Do: Range Search

IIT Madras Zanzibar

```
def in_range(node, lo, hi):
    if node is None: return
    if lo < node.value: in_range(node.left, lo, hi)
    if lo <= node.value <= hi: print(node.value)
    if node.value < hi: in_range(node.right, lo, hi)
```



range 4 to 10, step by step

1. 8, 3, 6, 4: at 3, lo > 3 so 3's left subtree (1) is skipped. Print 4
2. back at 6: in range, print 6
3. go right to 7: print 7
4. back at 8: in range, print 8
5. go right to 10: print 10; 10 is not below hi, so 13 and 14 are skipped
6. never visited (grey): 1, 13, 14. Only 6 of 9 nodes visited

output

4	6	7	8	10

? **Think:** For the range 11 to 13, which part of the tree is skipped entirely, and what is printed?

Answer: The whole left subtree of 8 (3, 1, 6, 4, 7) is skipped because 11 > 8. Visited: 8, 10, 14, 13. Only 13 is printed.

In-Class Exercise (6 minutes, in pairs)

IIT Madras Zanzibar

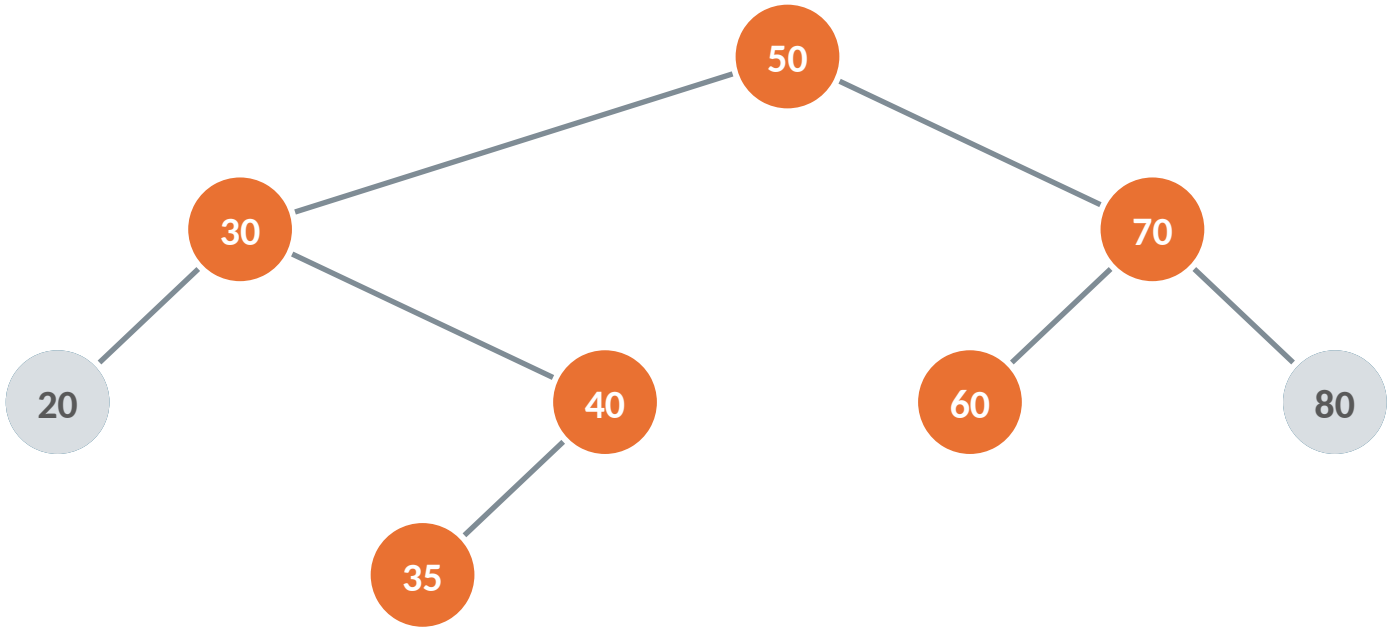
IN-CLASS TASK: build, search, range, insert

Insert these keys, in this order, into an empty search tree:

50	30	70	20	40	60	80	35

1. Draw the tree. (Paper.)
2. List the nodes visited when searching for 35, and when searching for 65.
3. Print all keys in the range 35 to 60 with in_range. Which nodes are visited?
4. Now insert 55, then 45. What is the height afterwards?

Solution: the Tree, the Paths, the Range and the Height



search 35: 50, 30, 40, 35 (4 steps)

search 65: 50, 70, 60, empty (not found)

range 35 to 60: visit 50, 30, 40, 35, 70, 60; print 35 40 50 60; skip 20 and 80

insert 55: left of 60; insert 45: right of 40; height stays 3

Check: the inorder output of a search tree is always sorted: 20 30 35 40 50 60 70 80. If yours is not, a key is in the wrong place.

Summary and Exam Checkpoints

IIT Madras Zanzibar

Why

fast search and insert together, plus order: min, max, ranges, sorted output

The rule

left subtree smaller, right subtree larger, at EVERY node

Operations

search and insert follow one path: O(height); insert adds a leaf where the search fails

Shape

depends on insertion order: sorted input gives a chain

Next

Today, lab: Lab 5: counting and radix sort, binary trees, search trees

Fri 20 Nov: delete from a search tree; rotations and balanced trees

References and Further Reading

[1] Canning, J., Broder, A. & Lafore, R. (2022). Data Structures & Algorithms in Python. Addison-Wesley. Chapter 8.

[2] Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2022). Introduction to Algorithms, 4th ed. MIT Press. Chapter 12.

[3] Skiena, S. S. (2020). The Algorithm Design Manual, 3rd ed. Springer. Section 3.4 (binary search trees).