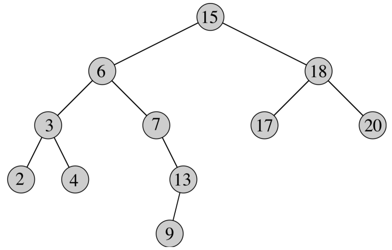


Binary Search Trees 1
Lecture 12

COSC 242 – Algorithms and Data Structures

Today’s outline

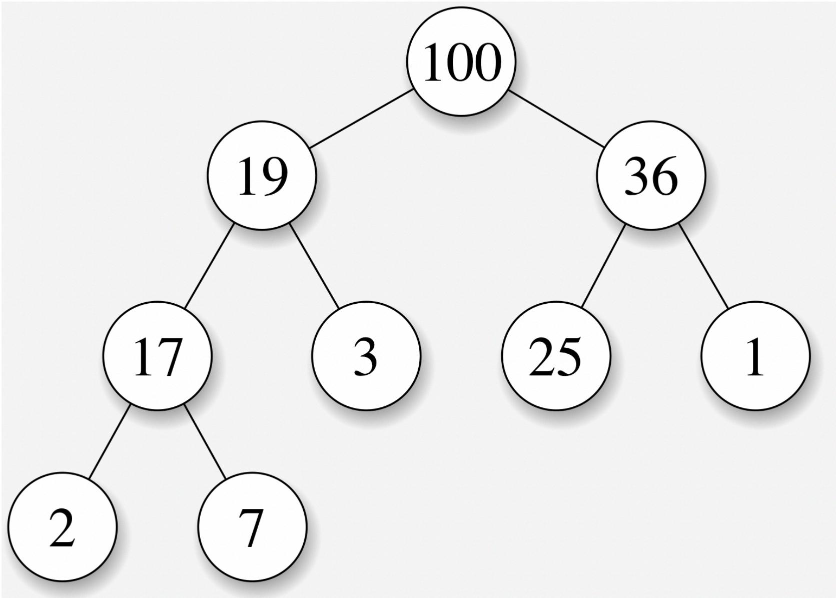
1. Introduction
2. BST Definition
3. Examples
4. Definitions
5. Properties and proof
6. C and Insertion

Today's outline

1. Introduction
2. BST Definition
3. Examples
4. Definitions
5. Properties and proof
6. C and Insertion

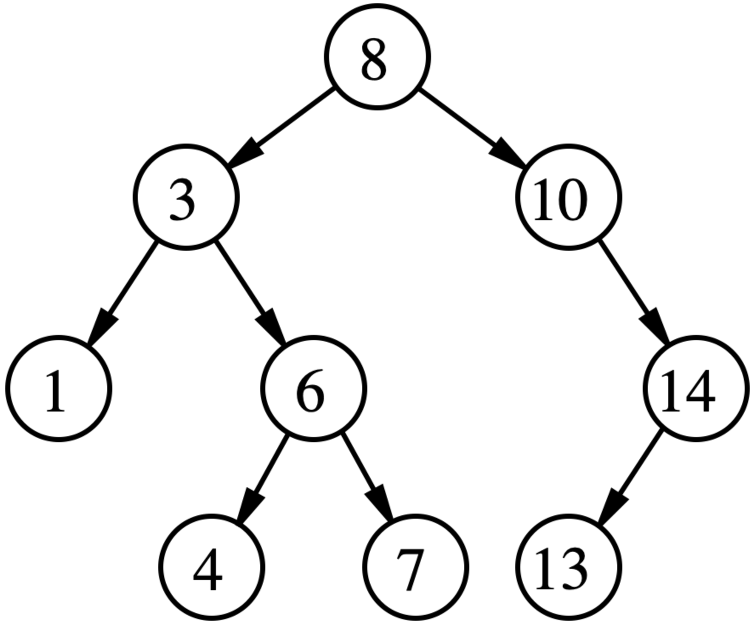
Binary Trees

In COSC241, you were introduced to a data structure called a heap where a tree node was usually bigger than all it's children. Most heaps used are binary trees (maximum of 2 children).



Binary search trees

A binary search tree (BST) is a binary tree such that all children to the left of a node are less than the node and all children to the right are greater than the node:



Hash Tables

Hash tables are convenient when:

- Maximum size of table is known beforehand
- Actual size does not fluctuate too much or spend too much time at a small fraction of the maximum size
- The emphasis is on insertion and retrieval.

BSTs or hash tables?

But are hash tables are not good when the app needs to:

- Perform many deletions.
- Perform traversals. For example, print out items in order of increasing key values.
- Use dynamic storage, as max tablesize is unknown, or size fluctuates a lot.

BSTs are good for very dynamic problems, or if traversals are needed.

But they come at a cost: insertion and deletion are O(log n) on average, and O(n) in the worst case.

Today's outline

1. Introduction
2. BST Definition
3. Examples
4. Definitions
5. Properties and proof
6. C and Insertion

BST definition

A binary tree T is either

- The empty tree, or
- A root node containing a key field and data fields, a left subtree T_L, and a right subtree T_R.

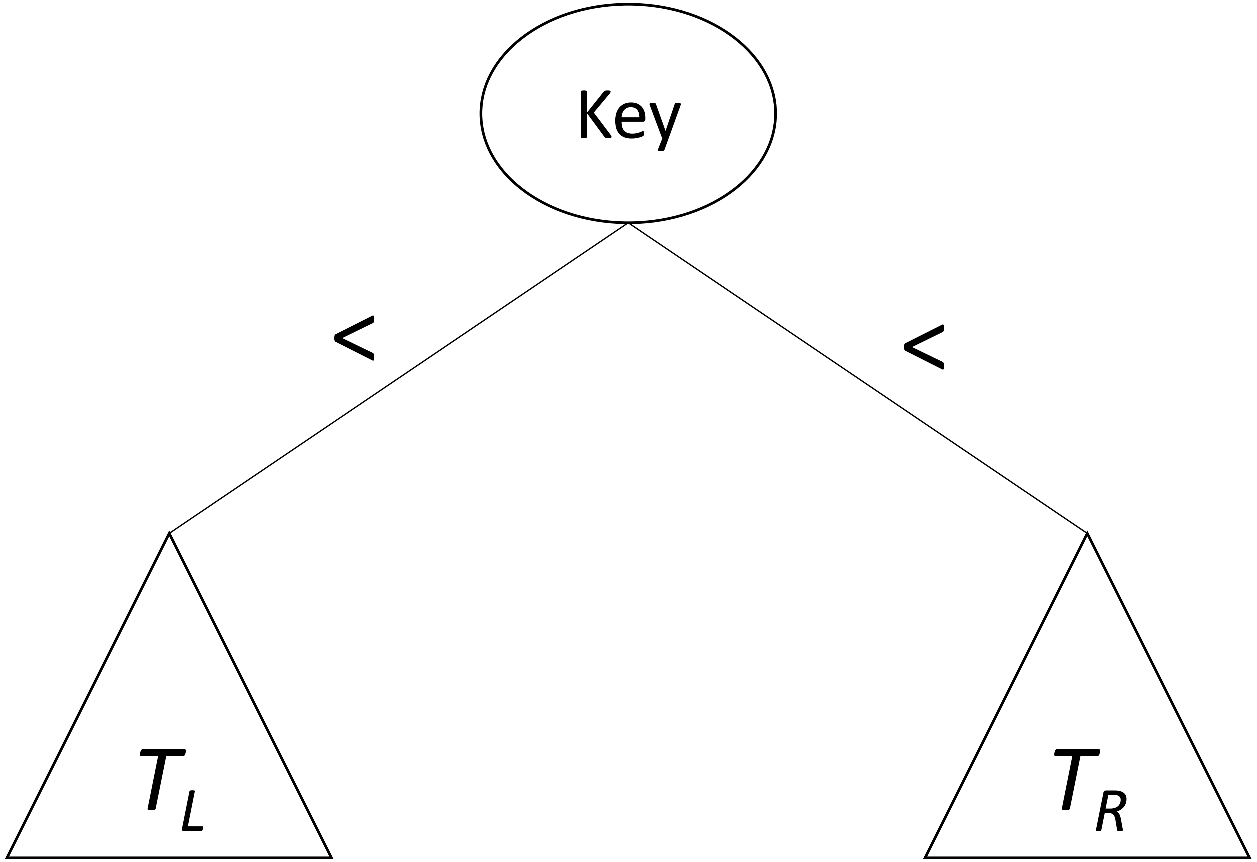
Leaves: Nodes with empty left and right subtrees.

BST Property

A binary tree T has the BST-property if:

- nodes in T have a key field of ordinal type, so they can be ordered by <
- for each node N in T, N's key value is greater than all keys in its left subtree T_L and less than all keys in its right subtree T_R, and T_L and T_R are binary search trees.

BST



Today’s outline

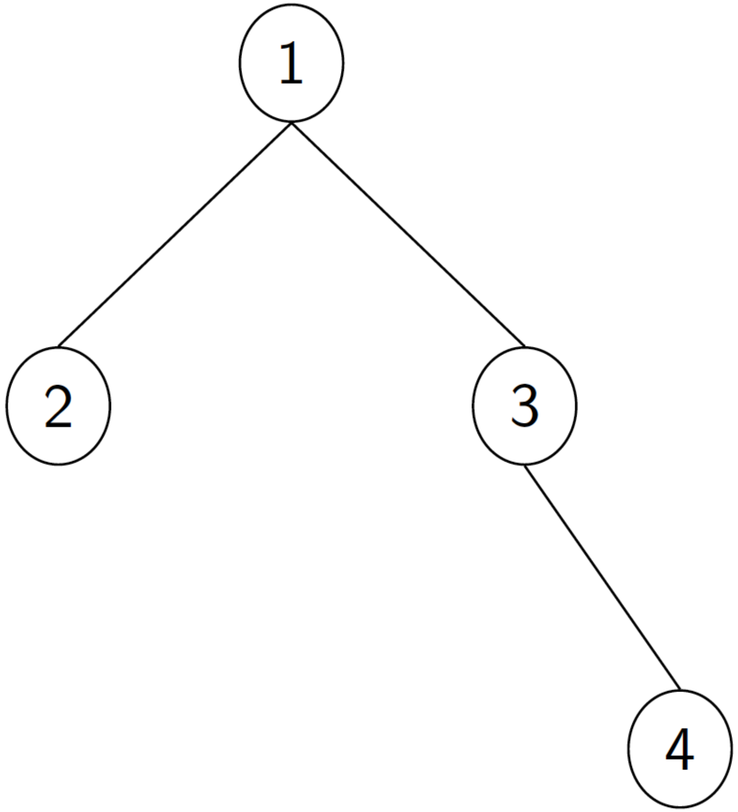
1. Introduction
2. BST Definition
3. Examples
4. Definitions
5. Properties and proof
6. C and Insertion

Example 1

Does this tree satisfy the BST property? If not, why not?

Answers

- Yes
- No

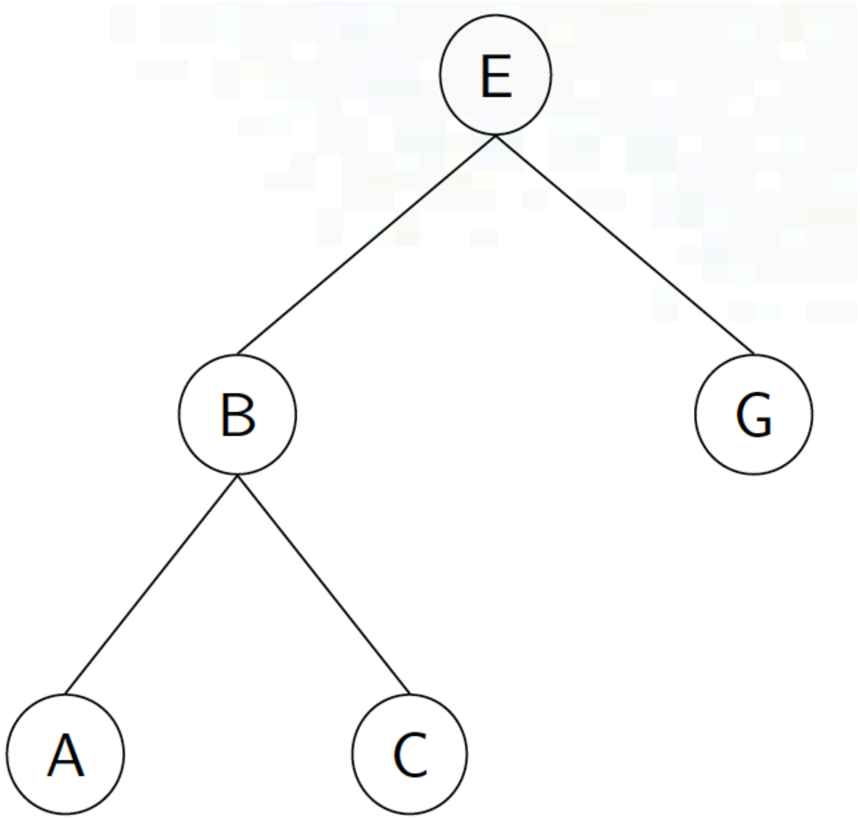


Example 2

Does this tree satisfy the BST property? If not, why not?

Answers

- Yes
- No

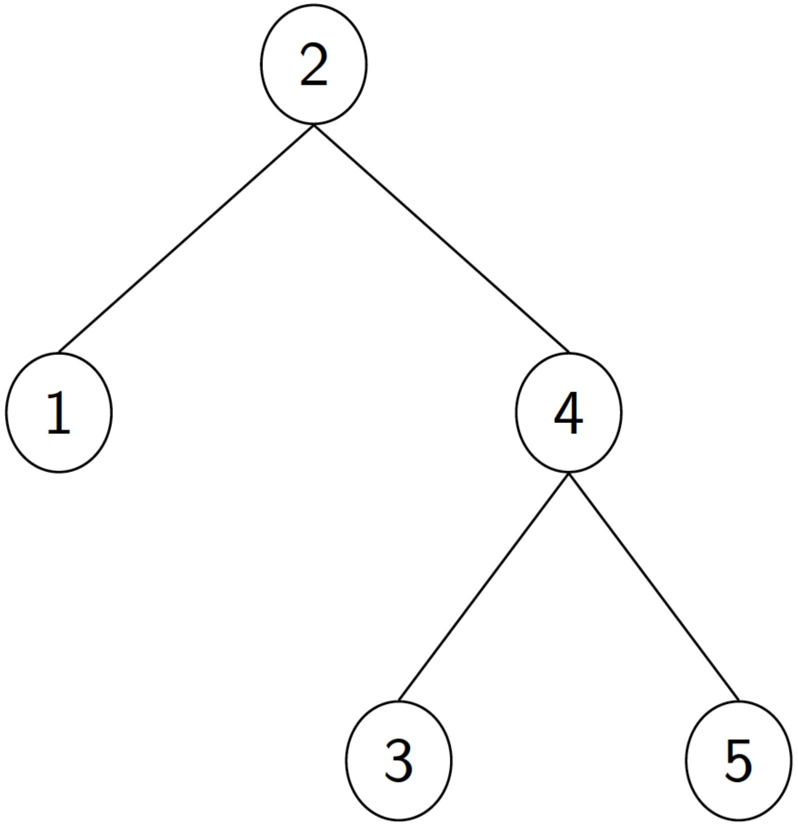


Example 3

Does this tree satisfy the BST property? If not, why not?

Answers

- Yes
- No

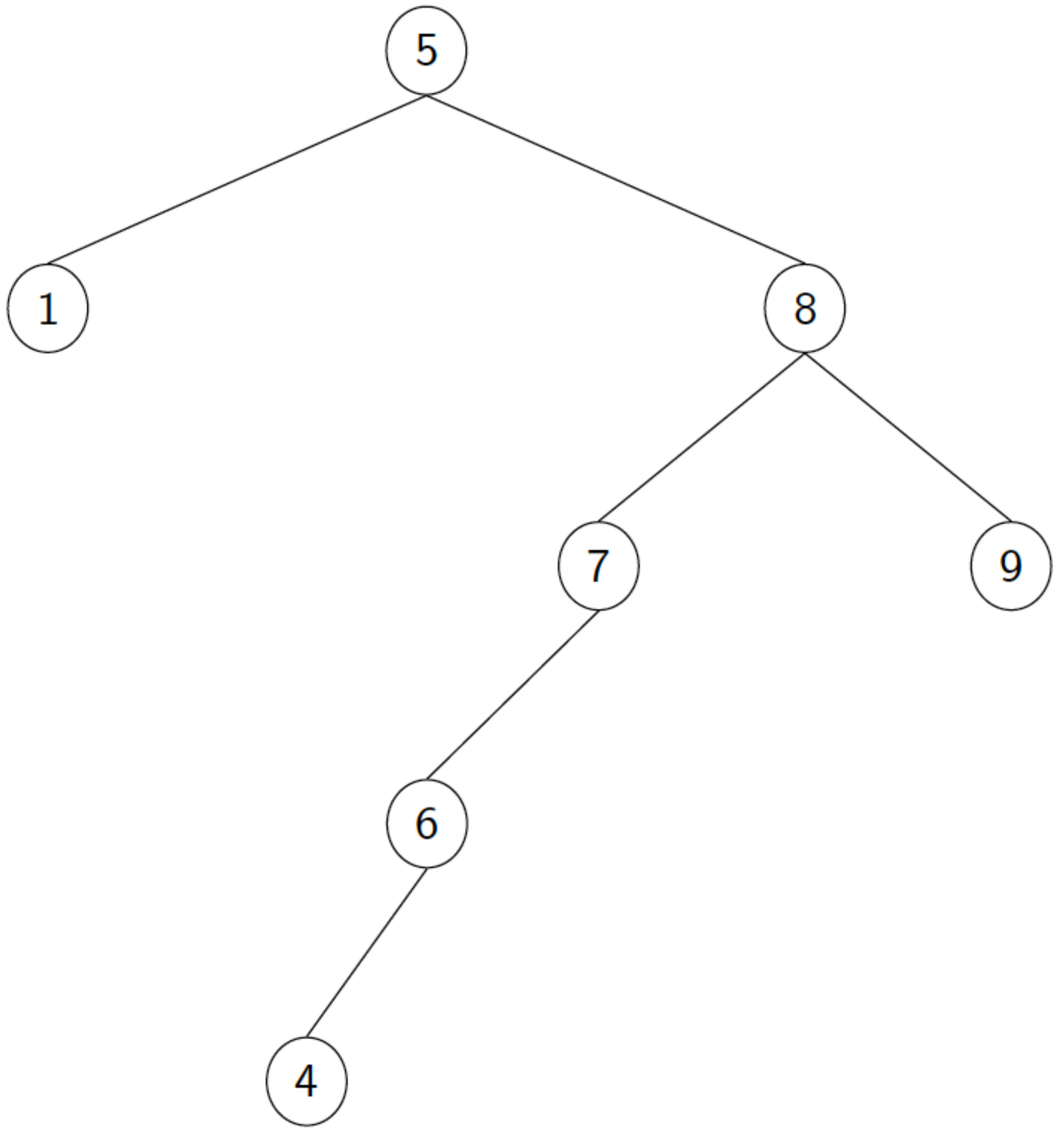


Example 4

Does this tree satisfy the BST property?
If not, why not?

Answers

- Yes
- No



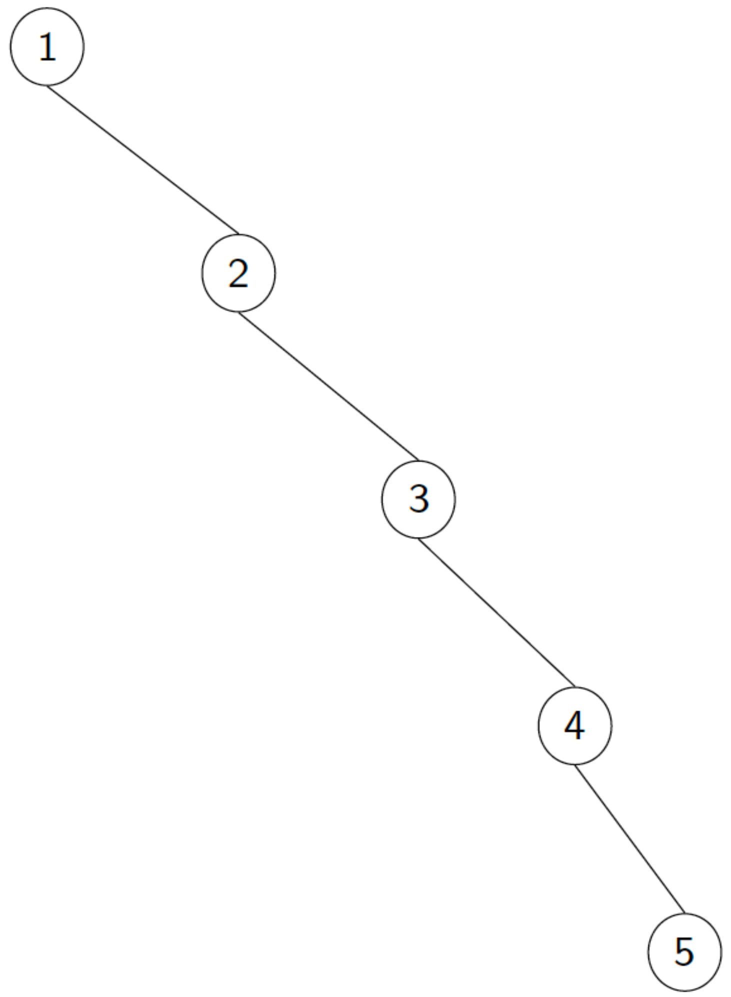
Example 5

Does this tree satisfy the BST property?

If not, why not?

Answers

- Yes
- No

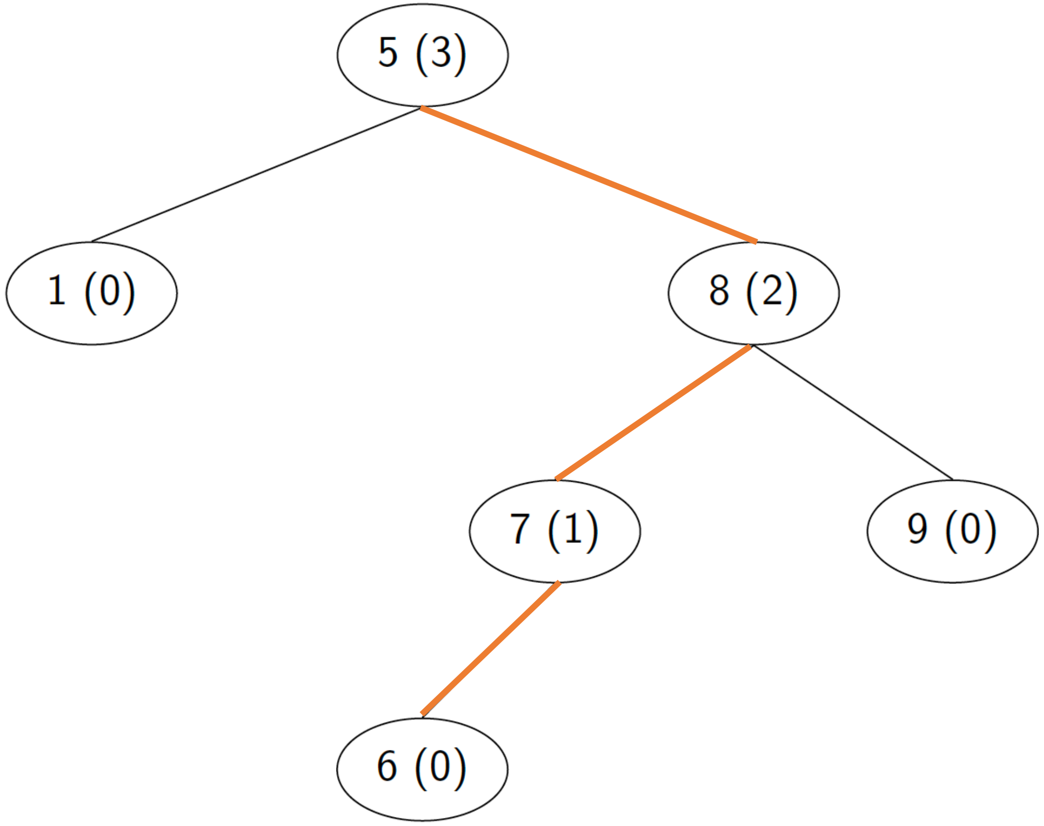


Today’s outline

1. Introduction
2. BST Definition
3. Examples
4. Definitions
5. Properties and proof
6. C and Insertion

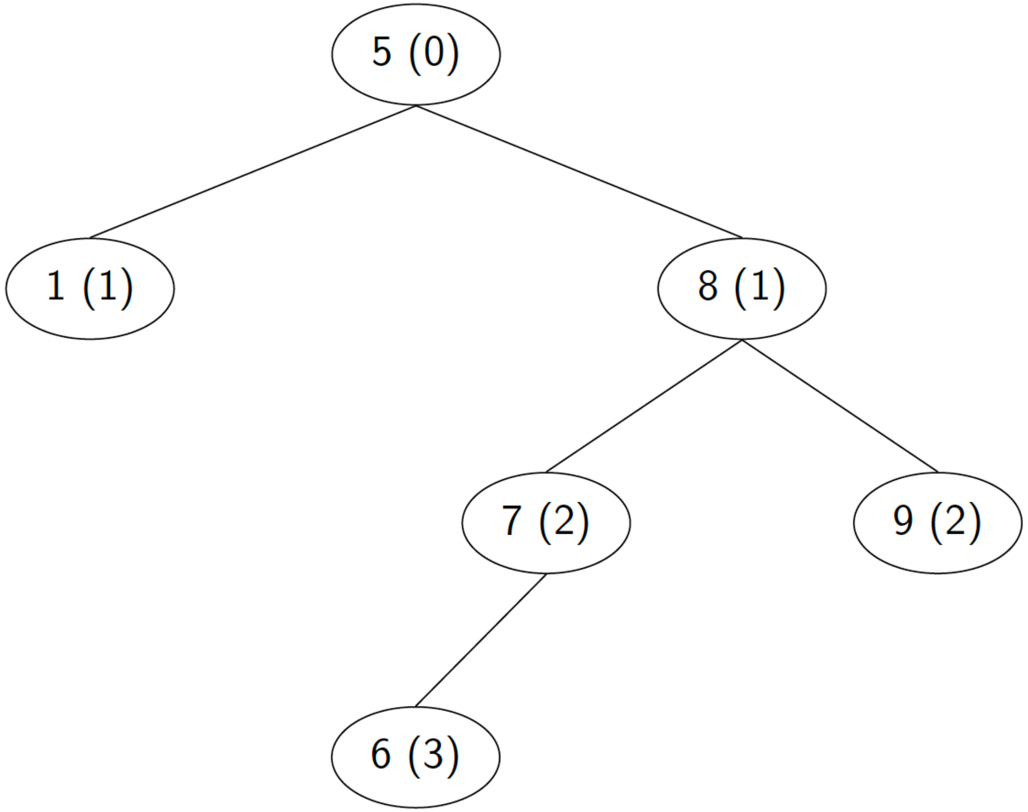
Definition: Height of a BST

The height of a BST is the number of edges from the root to the deepest leaf.

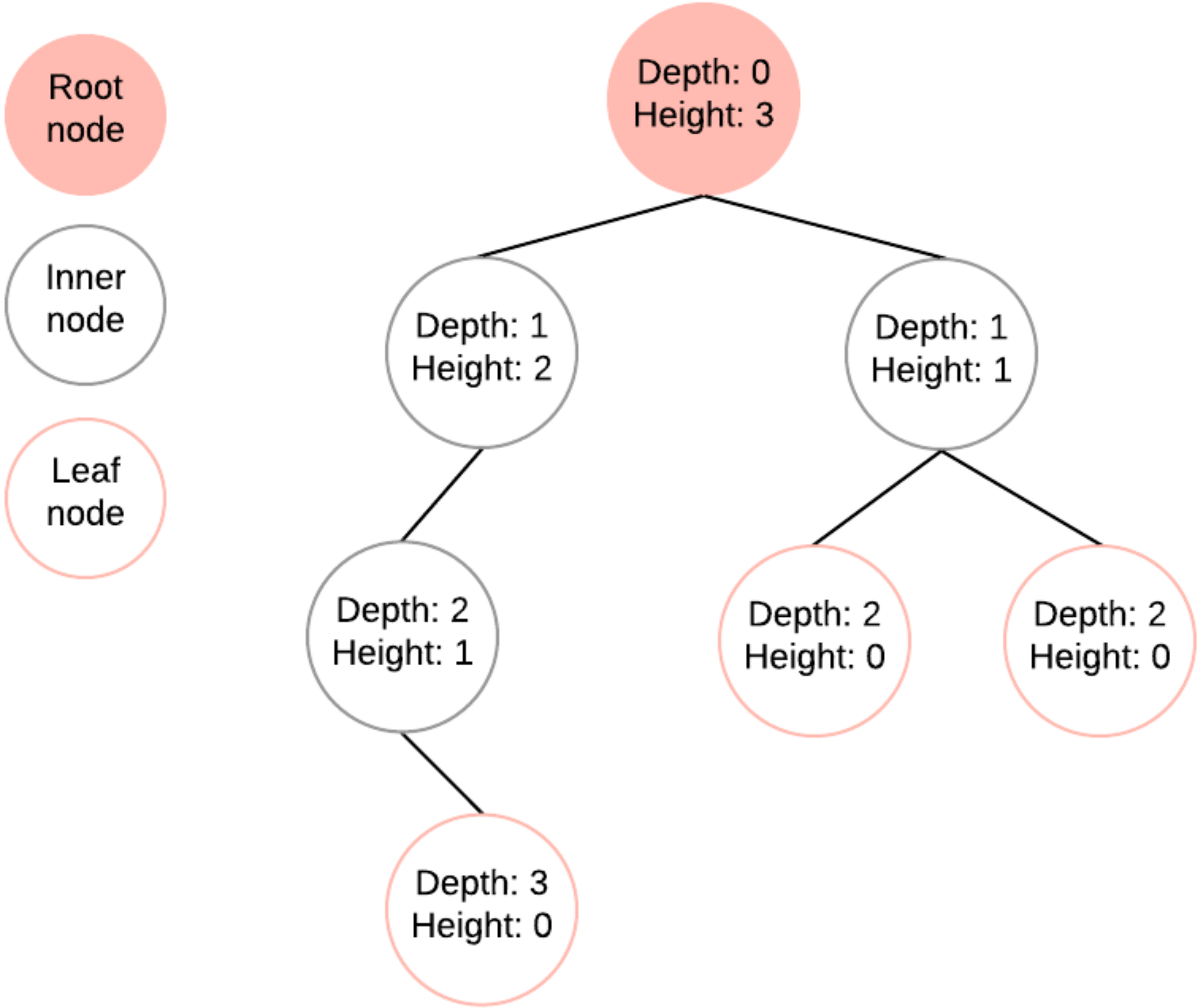


Definition: Depth of a node

The depth of a node is the number of edges from that node to the root of the tree.

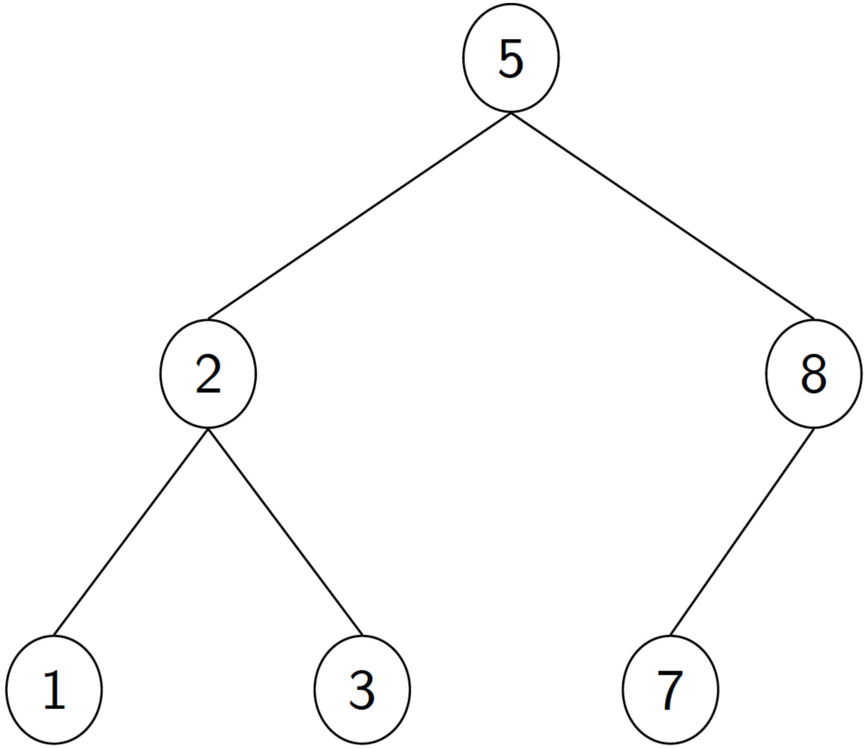


Height vs Depth



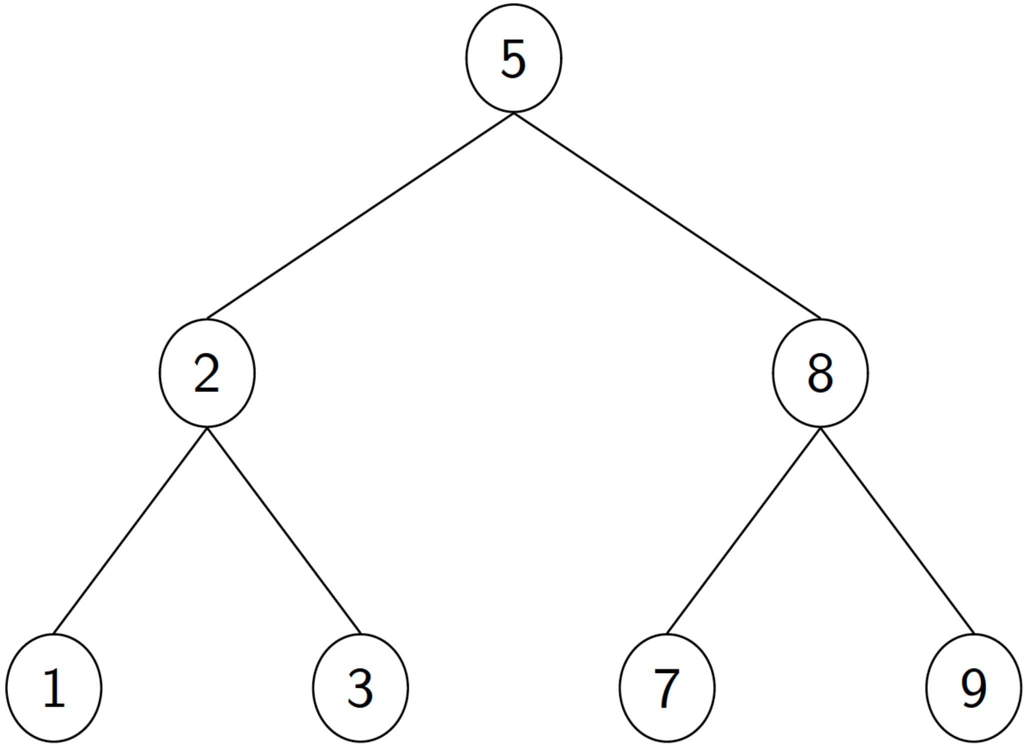
Definition: Complete BST

*A **complete** BST is a BST where each level, except possibly the last, is completely filled, and all nodes are as <u>far left as possible</u>.*



Definition: Fully complete BST

*A **fully complete** BST, sometimes called a **perfect** BST, is a BST where each level is completely filled.*



Today's outline

1. Introduction
2. BST Definition
3. Examples
4. Definitions
5. Properties and proof
6. C and Insertion

Proof

Theorem: The number of nodes in a fully complete binary tree of height h is $n = 2^{h+1} - 1$.

Lets prove this.

<u>Base case</u>: h = 0. Then, $2^{0+1} – 1 = 2 - 1 = 1$

Proof

Activity Time.

Assumption:

Inductive step:

Corollary

By corollary, the height of a **complete** tree of n nodes is $\lfloor \log_2 n \rfloor$.

Again, the number of nodes increases as a power of 2.

n = $2^{h+1} - 1$. Rearranging to solve for h, we get:

$$
\begin{aligned}
2^{\mathrm{h}+1} - 1 &= \mathrm{n} \\
2^{\mathrm{h}+1} &= \mathrm{n} + 1 \\
\mathrm{h} + 1 &= \log_2(n+1) \\
\mathrm{h} &= \lceil \log_2(n+1) \ \text{-}\ 1 \rceil \\
\mathrm{h} &= \lfloor \log_2 n \rfloor
\end{aligned}
$$

For a complete tree (not perfect tree), we need to take the ceiling, or floor.

Today's outline

1. Introduction
2. BST Definition
3. Examples
4. Definitions
5. Properties and proof
6. C and Insertion

C data structure

```
/* should live in bst.h */
typedef struct bst_node *BST;

/* should live in bst.c */
struct bst_node {
    KeyType key;
    BST left;
    BST right;
};
```

`KeyType` can be any comparable type (e.g. int, float, char, char *, etc).

Inserting


```
function BST_Insert(BST T, KeyType key)
    if T == NIL then
        return new bst_node(key)
    else
        if key < T -> key then
            T->left = BST_Insert(T->left, key)
        else
            T->right = BST_Insert(T->right, key)
        end if
        return T
    end if
end function
```

Suggested reading

Chapter 12 is all about binary search trees.

We're looking at things in a different order to the textbook. Insertion into a BST is in section 12.3, but the textbook uses an iterative version of insertion.

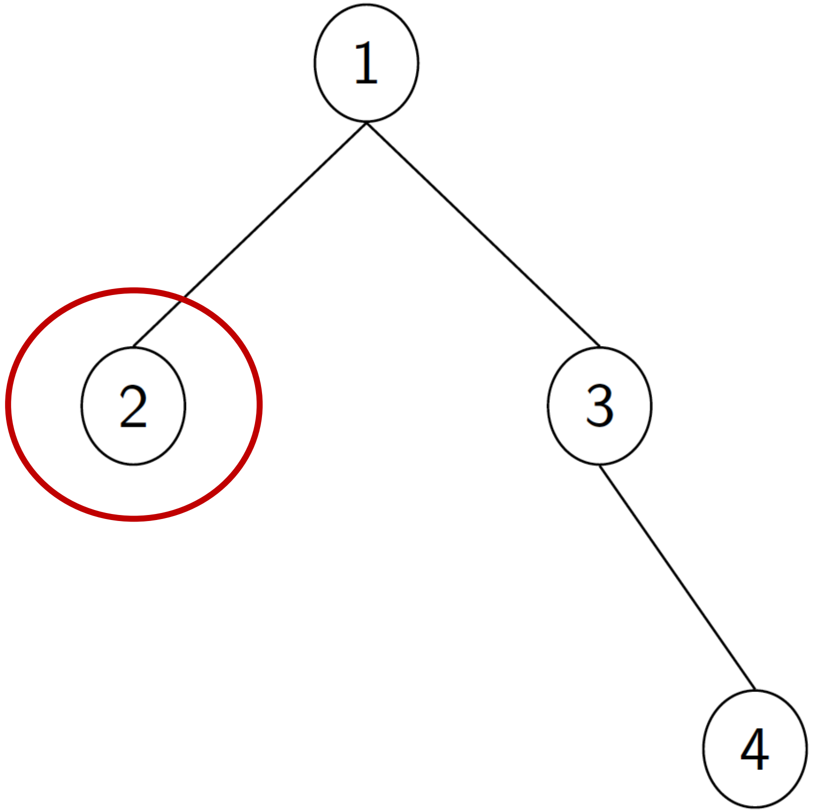
Solutions

Example 1

Does this tree satisfy the BST property? If not, why not?

Answers

- Yes
- No



Reason

Left subtree key $\nless$ parent node key

Example 2

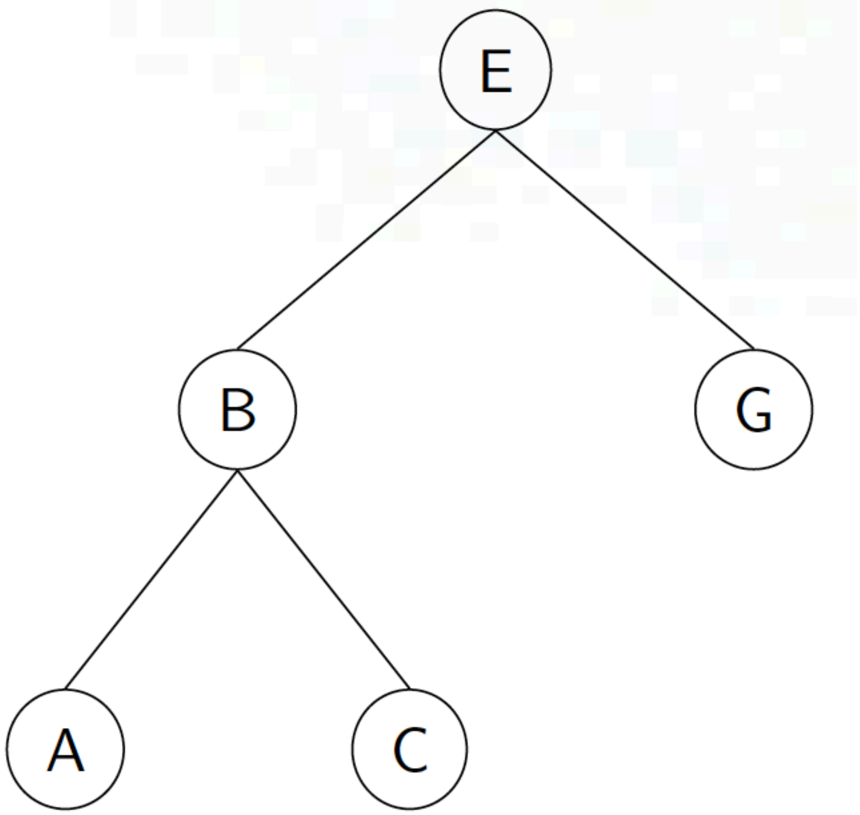
Does this tree satisfy the BST property? If not, why not?

Answers

- Yes
- No

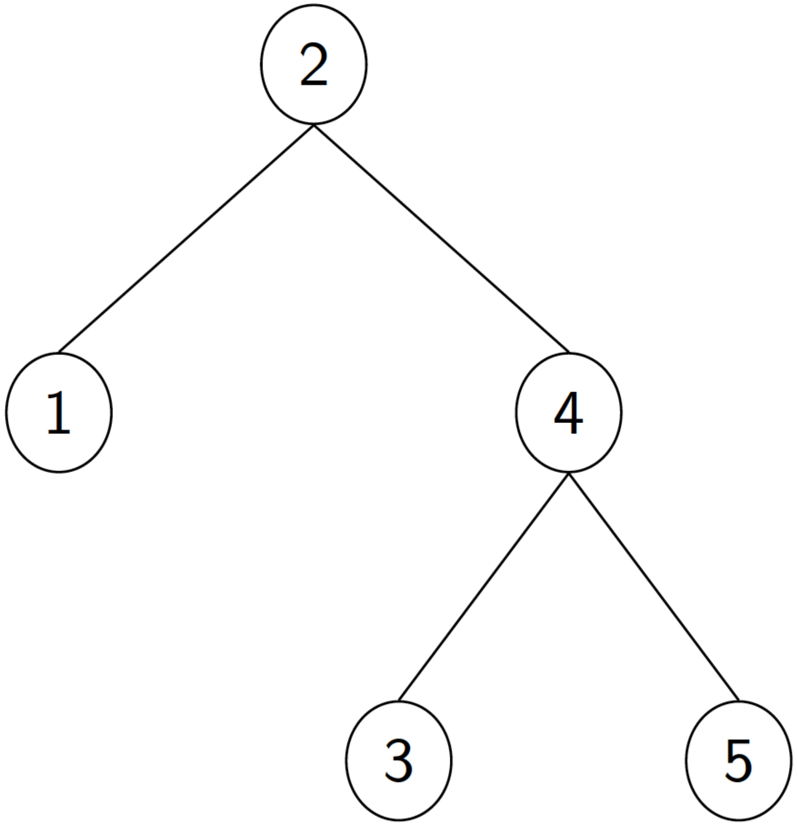
Reason

Ordinal data with all $T_L < T_R$



Example 3

Does this tree satisfy the BST property? If not, why not?



Answers

- Yes
- No

Reason

Ordinal data with all $T_L < T_R$

Example 4

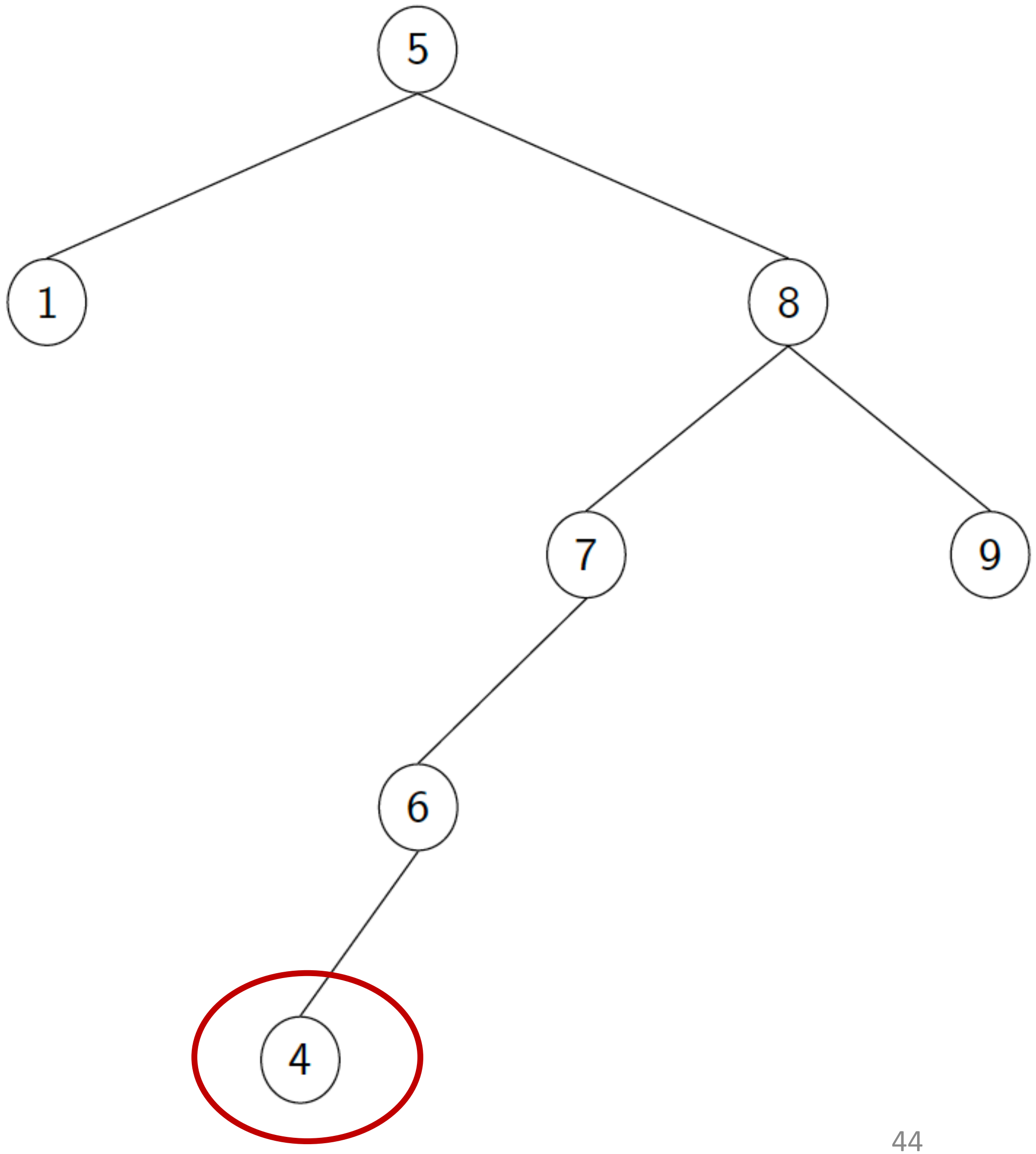
Does this tree satisfy the BST property?
If not, why not?

Answers

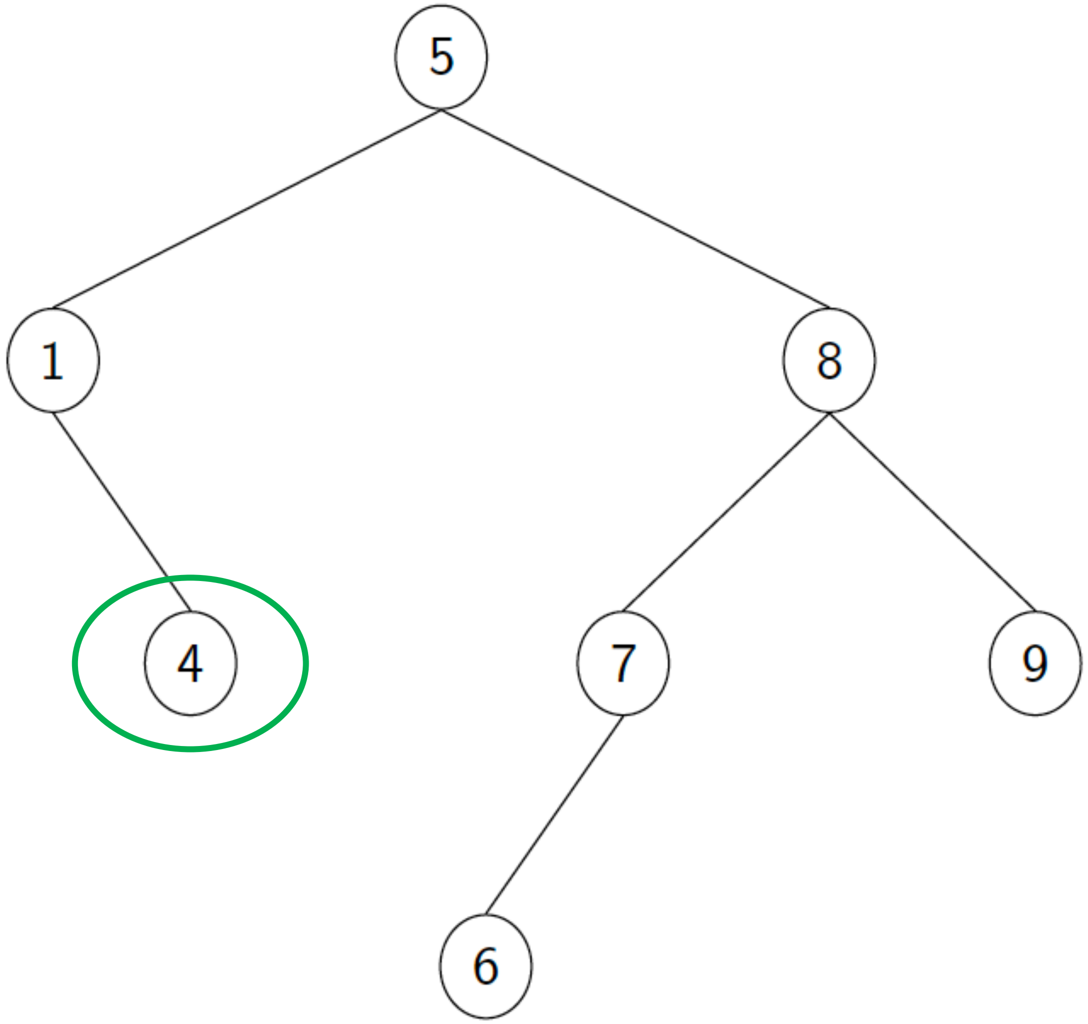
- Yes
- No

Reason

Right subtree key 4 $\ngtr$ root node key



Example 4 (corrected)



Example 5

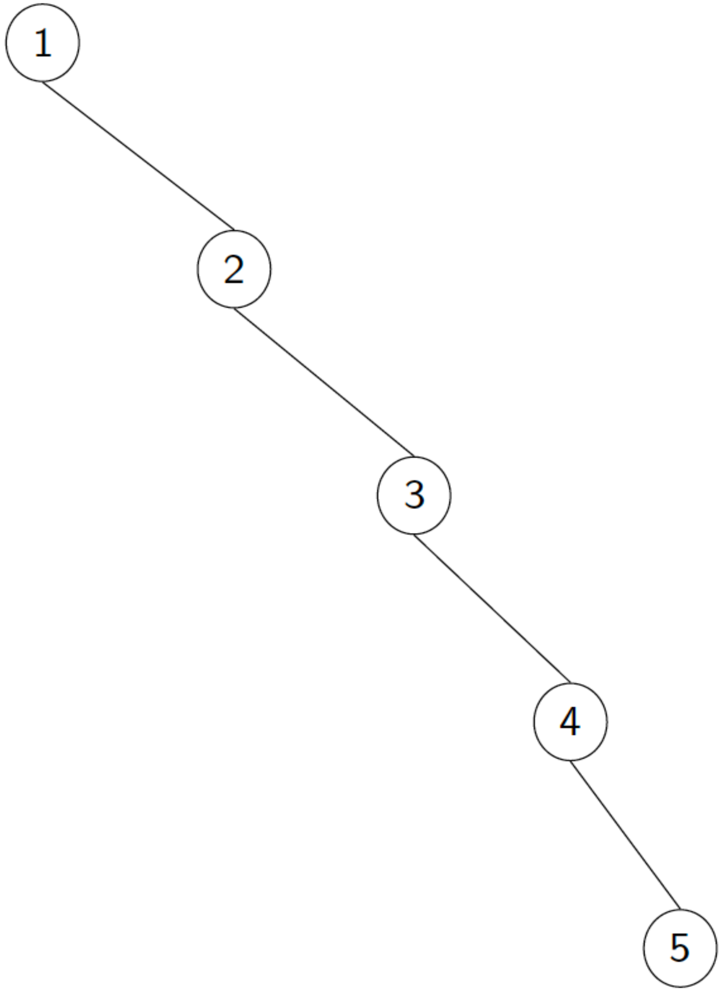
Does this tree satisfy the BST property?

If not, why not?

Answers

- Yes
- No

Valid, but what do you notice about it?



Proof

Assumption: $2^{h+1} - 1$ is true for all perfect trees.

Inductive step: Prove formula holds for a tree of height h + 1. That is,

$n = 2^{h+2} - 1$.

Notice that in a binary tree, the number of extra nodes increases as a power of 2, at each height level increasing by 2^{h+1}. Therefore:

$$
\begin{aligned}
2^{h+2} - 1 \\
&= 2^{h+1} + 2^{h+1} - 1 \\
&= 2 \cdot 2^{h+1} - 1 \\
&= 2^{h+1+1} - 1 \\
&= 2^{h+2} - 1 \qquad \blacksquare
\end{aligned}
$$