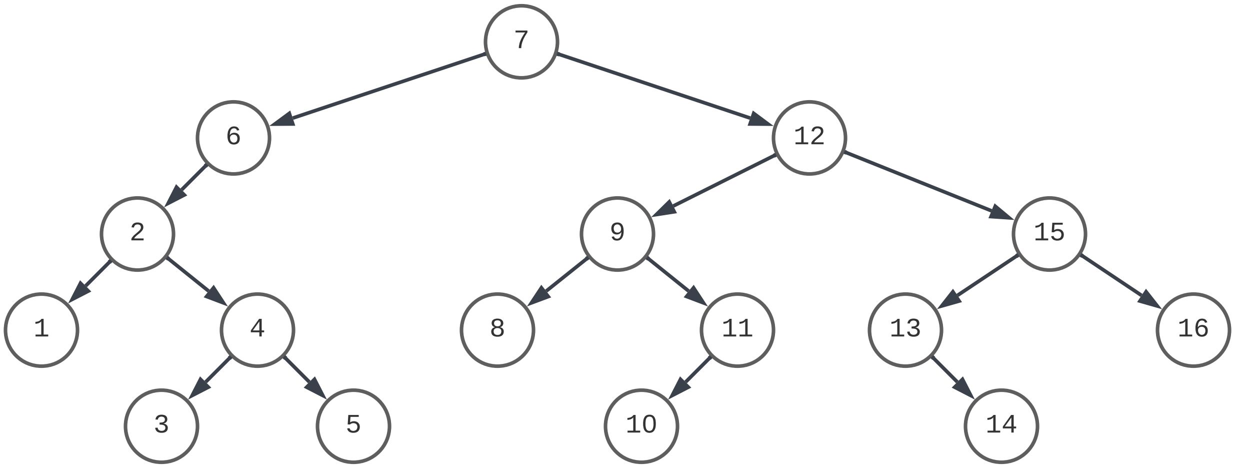
Problem 1. Consider the following binary search tree.



(a) In the space below, draw the tree resulting from calling the method `remove(6)` followed by `remove(12)`.

(b) Suppose the binary tree from the previous page is represented by the following (partial) `BinaryTree` class:

```
public class BinaryTree<E> {
    Node<E> root;

    ...

    public void printPreOrder() {
        printPreOrder(root);
    }

    private void printPreOrder(Node<E> nd) {
        if (nd == null) return;
        System.out.print(nd.value + " ");
        printPreOrder(nd.left);
        printPreOrder(nd.right);
    }

    protected class Node<E> {
        Node<E> left;
        Node<E> right;
        E value;
    }
}
```

The method `printPreOrder()` prints the contents of a tree in pre-order traversal. In the space below, write the output of executing `printPreOrder(root)` on the (original) tree on the previous page.

Problem 2.

(a) In the space below, draw a tree representation of the binary heap resulting from the numbers 1 through 13 being added in the following order: $8, 4, 3, 7, 2, 9, 11, 5, 10, 12, 1, 6, 13$.

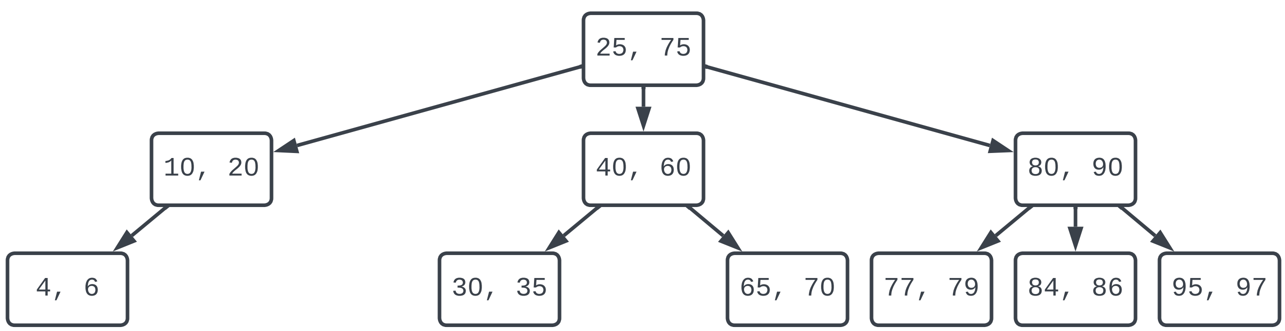
(b) In the space below, write the representation of your binary heap from part (a) as an array.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Problem 3. A *ternary search tree* (TST) is a tree in which each node `v` stores two comparable values: `v.small` and `v.large` satisfying `v.small < v.large`. Each node additionally has three children: `left`, `mid`, and `right`. The values stored in a TST satisfy the following condition. For every node `v` and descendant `w` of `v`, exactly one of the following conditions holds:

1. `w.large < v.small`
2. `v.small < w.small` and `w.large < v.large`
3. `v.large < w.small`

If condition 1 holds, then `w` is a descendant of `v.left`, if condition 2 holds, then `w` is a descendant of `v.mid`, and if condition 3 holds, then `w` is a descendeant of `v.right`. For example, the following figure depicts a valid ternary search tree.



We can represent a `TernarySearchTree` class storing `int` values in Java as follows

```
public class TernarySearchTree {
    private Node root;

    ...

    private class Node {
         int small;
         int large;
         Node left;
         Node mid;
         Node right;
    }
}
```

(a) In the space below, write a method `boolean find(int x)` for the `TernarySearchTree` class that returns `true` if `x` is a value stored in the tree, and `false` otherwise. The running time of your method should be $O(h)$ where h is the height of the tree (i.e., h is the length of the longest path from the tree's root to any leaf).

(b) In the space below, write a method `void printContents()` for the `TernarySearchTree` class that prints the contents (i.e., all values stored in the tree) in sorted order. The running time of your method should be $O(n)$ where n is the number of elements stored in the tree.

Problem 4. Consider the following partial binary tree implementation:

```
public class BinaryTree<E> {
    Node<E> root;

    public int size() {...}

    protected class Node<E> {
        Node<E> leftChild;
        Node<E> rightChild;
        E value;

        int numDescendants() {...}
    }
}
```

(a) In the space below, write *a recursive method* `numDescendants()` for the `Node<E>` class such that calling `nd.numDescendants()` on a node `nd` will return the number of descendants of `nd`.

(b) Write a method `size` for the `BinaryTree<E>` class that returns the total number of nodes in the tree. (Hint: use part (a).)

(c) Use big O notation to describe the running time of your `size` method from part (b) on a `BinaryTree` with n nodes.