if (value < par->data) {
    save_node = ___________; // Save the node to delete.
    par->left = delnode->left; // Readjust the parent pointer.
    free(delnode); // Free the memory for the deleted node.
}

// Deletes the node if it's a right child.
else {
    save_node = ___________; // Save the node to delete.
    par->right = delnode->left; // Readjust the parent pointer.
    free(delnode); // Free the memory for the deleted node.
}

return root; // Return the root of the tree after the deletion.

// Case 3: the node to be deleted only has a right child.
if (hasOnlyRightChild(delnode)) {

    // Node to delete is the root node.
    if (par == NULL) {
        save_node = delnode->right;
        free(delnode);
        return save_node;
    }

    // Delete's the node if it is a left child.
    if (value < par->data) {
        save_node = ___________
        par->left = delnode->right;
        free(delnode);
    }

    // Delete's the node if it is a right child.
    else {
        save_node = ___________
        par->right = delnode->right;
        free(delnode);
    }

    return root;
}

// Case 4: The deleted node has 2 children, find the replacement node
new_del_node = maxVal(del->left); save_val = max(del_node->data);

delete(root, ____________); // Now, delete the proper value.

// Restore the data to the original node to be deleted.
delnode->data = ___________

return root;