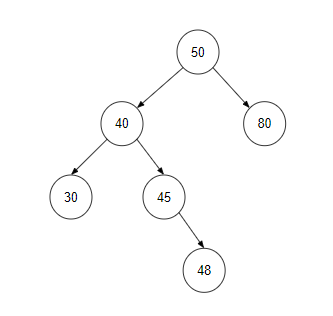
**General instruction:** This assignment includes two parts, written and programming. Please write/type your answers neatly so they can be readable. For the written part, please show the steps of your work. Please submit this assignment before 11:59 PM - October 23, 2023.

**Submission:** Please generate a single zip file containing a pdf contains your written part and all your \*.java files needed for this assignment. Name your file “A3\_YourFirstName\_ YourLastName,zip”. Submit your zip file electronically to Canvas.

**Special office hour**for this assignment will be on Friday, October 20, from 9 am to 10 am in Wickenden Building 517a and via Zoom (Meeting ID: 302 880 316), Please feel free to ask questions via email: [yxs1432@case.edu.](about:blank)

**Written Problems (50 pts)**

1. **AVL Trees (16 pts)**
2. True or false? (If false, explain why the statement is false.) **(4 pts)**
   1. If T is an AVL tree, then its left and right subtrees are also AVL trees. (2 pts)
   2. AVL trees and BSTs both have the same worst-case O(n) for deleting a node. (2 pts)
3. Given the following Tree: **(7 pts)**



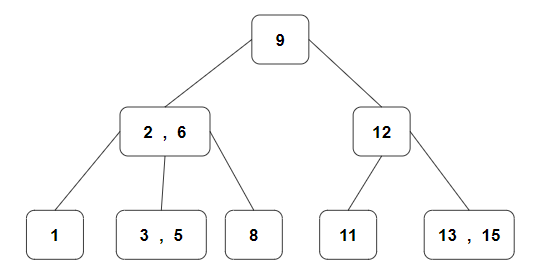
* 1. Compute the balance factor of the root. (1 pts)
  2. Compute the balance factor of the node 40. (1 pts)
  3. Rotation the tree to get a balanced AVL Tree. (5 pts)

1. Insert 12, 4, 1, 3, 7, 8, 10, 9, 2, 11, 6, 5 to an empty AVL Tree in the given order. Draw the final tree after all insertions. **(5 pts)**
2. **B Trees (21 pts)**
3. True or false? (If false, explain why the statement is false.) **(4 pts)**
   1. In an B-Tree, every node except root must contain at least 2t -1 keys (‘t’ refers to the minimum degree of the tree) and all nodes (including root) may contain at most t - 1 keys. (2 pts)
   2. All leaves in a B-Tree are at the same depth and make search operations efficient. (2 pts)
4. Given the following sequence of keys:

{4, 9, 32, 12, 24, 30, 51, 29, 69, 31, 90, 70, 75, 79, 80, 85, 91} **(10 pts)**

* 1. For a B-tree of order 5, if the root node is an internal node, what is the minimum and maximum of children that all nodes other than the root can have? (2 pts)
  2. For a B-tree of order 5, if the root node is an internal node, what is the minimum and maximum of keys that all nodes other than the root can have? (2 pts)
  3. Create a B-tree of order 5 from the given sequence of keys. (6 pts)

1. Given the following B-tree: **(7 pts)**



* 1. Insert key: 4, draw the new B-tree. (5 pts)
  2. Compute the time complexity of the insert operation. (2 pts)

1. **Heaps (13 pts)**
2. True or false? (If false, explain why the statement is false.) **(4 pts)**
   1. Every node’s value in the heap should be greater than or equal to the value of its children (considering max-heap). (2 pts)
   2. The worst-case time complexity for a heap sort is O(log n). (2 pts)
3. Consider the following sequence of numbers: **(9 pts)**

[10, 7, 2, 5, 1]

* 1. Insert the above sequence of numbers into a max heap. (3 pts)
  2. Insert element 16 at the end of the heap, showing the result after “heapify”. (3 pts)
  3. Delete element 10 from the heap [10, 7, 2, 5, 1], showing the result after “heapify”. (3 pts)

**Programming Exercise - AVL Tree Implementation in Java (50 pts)**

You will implement an AVL Tree with some operations through Java in this exercise. Please download the file A3\_AVLTree.zip. (Pseudocode is not accepted)

Some basic functions and hints are given, Please finish the coding base on the following requirements

You need to develop the class as efficiently as possible. You are allowed to use the provided sample code, all necessary classes are defined in the given file, (please do not change the name of the given classes) or you could also start from scratch.

**Grading Criterion:** Implementation (30 pts); Test (10 pts); Design and style (commands and Proper encapsulation/information hiding) (10 pts)

Note: TAs will feed other examples to your code as input, so please make sure your program works for any input before your submission.

1. Create a class named AVLTree with the following operations:
   1. private void preOrder(AVLTreeNode<T> tree)

// Preorder traversal "AVL tree", print the result.

* 1. private void inOrder(AVLTreeNode<T> tree)

// In-order traversal "AVL tree", print the result.

* 1. private void postOrder(AVLTreeNode<T> tree)

// Post-order traversal "AVL tree", print the result.

* 1. private AVLTreeNode<T> minimum(AVLTreeNode<T> tree)

// Finds min node：return the smallest node of the AVL tree with "tree" as the root.

* 1. private AVLTreeNode<T> maximum(AVLTreeNode<T> tree)

// Finds max node: return the largest node of the AVL tree with "tree" as the root

* 1. private AVLTreeNode<T> insert(AVLTreeNode<T> tree, T key)

// Insert an element into the tree, then return the root node

* 1. private AVLTreeNode<T> remove(AVLTreeNode<T> tree, AVLTreeNode<T> z)

// Delete the node (z), then return the root node.

1. Test the operations by running AVLTreeTest, which will display:

* Input Sequence
* PreOrder Traversal Result
* InOrder Traversal Result
* PostOrder Traversal Result
* The height of the tree
* The smallest node of the tree
* The largest node of the tree
* The details of the tree
* Delete one node
* Height of the tree after deleted one node
* InOrder Traversal result after deleted one node
* The details of the tree again
* Add one node
* The PreOrder Traversal result after added one node

Here is a sample output: (Next page)

