BINARY TREES 2 : Guided Exercises

1. Develop the following iterative algorithms for traversing binary search trees using a stack:
- Inorder
- Reverse Inorder
- Preorder
- Reverse Preorder
- Postorder
- Reverse Postorder

2. Develop the following iterative algorithms for traversing binary search trees using the Parent operation (without a stack):
- Inorder
- Reverse Inorder
- Preorder
- Reverse Preorder
- Postorder
- Reverse Postorder

3. Develop the following iterative algorithms for navigating binary search trees using a stack:
- Next Inorder of a given node
- Previous Inorder of a given node
- Next Preorder of a given node
- Previous Preorder of a given node
- Next Postorder of a given node
- Previous Postorder of a given node

4. Develop the following algorithms for navigating binary search trees using the Parent operation (without a stack):
- Next Inorder of a given node
- Previous Inorder of a given node
- Next Preorder of a given node
- Previous Preorder of a given node
- Next Postorder of a given node
- Previous Postorder of a given node

5. Build a Right-threaded binary search tree, search for a given element, insert an element into such a tree, and delete an element from this tree.

6. Develop the following algorithms for navigating right-threaded binary search trees:
- Next Inorder of a given node
- Previous Inorder of a given node
- Next Preorder of a given node

- Previous Preorder of a given node
- Next Postorder of a given node
- Previous Postorder of a given node

7. Develop the following algorithms for traversing right-threaded binary search trees:
- Inorder
- Reverse Inorder
- Preorder
- Reverse Preorder
- Postorder
- Reverse Postorder

8. Build a left-threaded binary search tree. This involves replacing every left child with a Nil value with its inorder predecessor. Search for a given element, insert an element into such a tree, and delete an element from this tree.

9. Develop the following algorithms for navigating left-threaded binary search trees:
- Next Inorder of a given node
- Previous Inorder of a given node
- Next Preorder of a given node
- Previous Preorder of a given node
- Next Postorder of a given node
- Previous Postorder of a given node

10. Develop the following algorithms for traversing left-threaded binary search trees:
- Inorder
- Reverse Inorder
- Preorder
- Reverse Preorder
- Postorder
- Reverse Postorder

11. Build a Doubly threaded binary search tree. It is both a left-threaded binary tree and a right-threaded binary tree. Search for a given element, insert an element into such a tree, and delete an element from this tree.

12. Develop the following algorithms for navigating doubly threaded binary search trees:
- Next Inorder of a given node
- Previous Inorder of a given node
- Next Preorder of a given node
- Previous Preorder of a given node
- Next Postorder of a given node
- Previous Postorder of a given node

13. Develop the following algorithms for traversing doubly threaded binary search trees:
- Inorder
- Reverse Inorder
- Preorder
- Reverse Preorder
- Postorder
- Reverse Postorder

14. Provide an algorithm that traverses a binary tree in breadth-first order. This order is defined as follows:
- The root is visited first,
- Then the children of the root from left to right,
- Next, the children of the children of the root from left to right, and so on.

15. Provide an algorithm that provides the next node at the same level as a given node using a queue, stack, or the parent operation.

16. Provide an algorithm that provides the next node at the same level as a given node using a stack.

17. Provide an algorithm that provides the next node at the same level as a given node using the parent operation.

18. Provide a recursive algorithm that traverses the leaves of a binary tree from left to right using a stack.

19. Provide an iterative algorithm that traverses the leaves of a binary tree from left to right using a stack.

20. Provide an iterative algorithm that traverses the leaves of a binary tree from left to right using the parent operation.

21. Provide an iterative algorithm that traverses the leaves of a right-threaded binary tree from left to right without using a stack and the parent operation.