**LABORATORY ASSIGNMENT 5**

**NAME: AKSHAT SRIVASTAV**

**REGISTRATION ID: 19BCE0811**

**Data Structure and algorithms :**

**Q.) Implement the ‘Binary Tree to Binary Search Tree Conversion Algorithm’ in C/C++/C#/JAVA/Python/MATLAB/R Programming language.**

[NOTE: You already implemented Binary Tree and Binary Search Tree Program in LAB Assignment 2 and 4]

**C Code:**

1. #include <stdio.h>
2. #include <stdlib.h>
4. //Represent a node of binary tree
5. **struct** node{
6. **int** data;
7. **struct** node \*left;
8. **struct** node \*right;
9. };
11. //Represent the root of binary tree
12. **struct** node \*root = NULL;
14. **int** treeArray[100];
15. **int** ind = 0;
17. //createNode() will create a new node
18. **struct** node\* createNode(**int** data){
19. //Create a new node
20. **struct** node \*newNode = (**struct** node\*)malloc(**sizeof**(**struct** node));
21. //Assign data to newNode, set left and right children to NULL
22. newNode->data= data;
23. newNode->left = NULL;
24. newNode->right = NULL;
26. **return** newNode;
27. }
29. //calculateSize() will calculate size of tree
30. **int** calculateSize(**struct** node \*node)
31. {
32. **int** size = 0;
33. **if** (node == NULL)
34. **return** 0;
35. **else** {
36. size = calculateSize (node->left) + calculateSize (node->right) + 1;
37. **return** size;
38. }
39. }
41. //convertBTtoArray() will convert the given binary tree to its corresponding array representation
42. **void** convertBTtoArray(**struct** node \*node) {
43. //Check whether tree is empty
44. **if**(root == NULL){
45. printf("Tree is empty\n");
46. **return**;
47. }
48. **else** {
49. **if**(node->left != NULL)
50. convertBTtoArray(node->left);
51. //Adds nodes of binary tree to treeArray
52. treeArray[ind] = node->data;
53. ind++;
54. **if**(node->right!= NULL)
55. convertBTtoArray(node->right);
56. }
57. }
59. //createBST() will convert array to binary search tree
60. **struct** node\* createBST(**int** start, **int** end) {
62. //It will avoid overflow
63. **if** (start > end) {
64. **return** NULL;
65. }
67. //Variable will store middle element of array and make it root of binary search tree
68. **int** mid = (start + end) / 2;
69. **struct** node \*temp = createNode(treeArray[mid]);
71. //Construct left subtree
72. temp->left = createBST(start, mid - 1);
74. //Construct right subtree
75. temp->right = createBST(mid + 1, end);
77. **return** temp;
78. }
80. //convertBTBST() will convert a binary tree to binary search tree
81. **struct** node\* convertBTBST(**struct** node \*node) {
83. //Variable treeSize will hold size of tree
84. **int** treeSize = calculateSize(node);
86. //Converts binary tree to array
87. convertBTtoArray(node);
89. //Sort treeArray
90. **int** compare (**const** **void** \* a, **const** **void** \* b) {
91. **return** ( \*(**int**\*)a - \*(**int**\*)b );
92. }
93. qsort(treeArray, treeSize, **sizeof**(**int**), compare);
95. //Converts array to binary search tree
96. **struct** node \*d = createBST(0, treeSize - 1);
97. **return** d;
98. }
100. //inorder() will perform inorder traversal on binary search tree
101. **void** inorderTraversal(**struct** node \*node) {
103. //Check whether tree is empty
104. **if**(root == NULL){
105. printf("Tree is empty\n");
106. **return**;
107. }
108. **else** {
110. **if**(node->left!= NULL)
111. inorderTraversal(node->left);
112. printf("%d ", node->data);
113. **if**(node->right!= NULL)
114. inorderTraversal(node->right);
115. }
116. }
118. **int** main()
119. {
120. //Add nodes to the binary tree
121. root = createNode(1);
122. root->left = createNode(2);
123. root->right = createNode(3);
124. root->left->left = createNode(4);
125. root->left->right = createNode(5);
126. root->right->left = createNode(6);
127. root->right->right = createNode(7);
129. //Display given binary tree
130. printf("Inorder representation of binary tree: \n");
131. inorderTraversal(root);
133. //Converts binary tree to corresponding binary search tree
134. **struct** node \*bst = convertBTBST(root);
136. //Display corresponding binary search tree
137. printf("\nInorder representation of resulting binary search tree: \n");
138. inorderTraversal(bst);
140. **return** 0;
141. }

**Output:**

Inorder representation of binary tree:

4 2 5 1 6 3 7

Inorder representation of resulting binary search tree:

1 2 3 4 5 6 7