# A03G48Q25 : Converting a Binary tree into a binary search tree

Harsh Kochar (IIT2018049), Madhu(IIT2018068)

*IV Semester B.Tech, Department of Studies in Information Technology,*

*Indian Institute of Information Technology Allahabad, India*

***Abstract: In this paper we have devised an algorithm to convert a binary tree to Binary Search Tree by maintaining its original structure. We did this by loading data from out binary tree in an InOrder fashion to an array and then Loading the orignal data back to the same struture.***

## I. INTRODUCTION

A binary tree is a special type of tree in which every node or vertex has either no child node or one child node or two child nodes.

Binary Search Tree is a node-based binary tree data structure which has the following properties: The left subtree of a node contains only nodes with keys lesser than the node's key. The right subtree of a node contains only nodes with keys greater than the node's key.

The algorithm uses the following simple steps to solve this problem. It creates the inorder traversal array of a given binary tree. Then it sorts the created inorder array.Now it again traverses given binary tree in inorder fashion and simultaneously it traverses sorted inorder array. At each node visit of binary tree, the value of that node is changed to corresponding element in the inorder array. The value of the first node visited during inorder traversal is changed to value of element at 0th index in sorted inorder array, value of second node visited during inorder traversal is changed to value of element at 1st index and so on.The overall time complexity of this method is **O(nlogn).** The algorithm works is described in detail in part II.

We have assumed that a binary tree is already formed in our algorithm.

## II. ALGORITHM DESCRIPTION

This algorithm works in mainly 2 stages.
If n is the size of sequence. Where R is the root of Binary tree.

Stage 1: We inorder traverse R by itterative manner, to create an array A which has all the data. We traverse through the nodes and push every node we look to the stack, and traverse to the left until we reach null.
The we pop the last element, view it, push it's right component (if exist) and continue.
Here, we have considered all the numbers are diffrent. In the code, we got numbers from a random function so, it might have repeated numbers, we choose to ignore them. By descarding them from A.

Stage 2: Now we run a loop through the given array and if the value is smaller than current node, it will go to left, if it's bigger than right. When it reaches a point where there is nothing it can relate to (null place) it will crearte a new node and put it's value there to make a BST **B.**

## III. ALGORITHM AND ANALYSIS

**Main**
Input: Balanced Binary tree **R**.
Output: BST **B**

```
node← root(R)                                   Step 1
Array   A                                       Step 2
n←size                                          Step 3
s ← empty stack                                 Step 4
while (not s.isEmpty() or node ≠ null)          Step 5
   if (node ≠ null)                             Step 6
     s.push(node)                               Step 7
     node ← node.left                           Step 8
   else
     node ← s.pop()                             Step 9
     A.push(node.value)                         Step 10
     node ← node.right                          Step 11
i←0                                             Step 12
root←newNode(A[i])                              Step 13
i←1                                             Step 14
for i<n, i:=i+1                                 Step 15
   node←root                                    Step 16
   while (node ≠ null)                          Step 17
      if A[i] < node.value                      Step 18
         node←node.left                         Step 19
      else
         node←node.right                        Step 20
   node←newNode(A[i])                           Step 21
End for
B←root
```

**Explaination:**
Step 1,2,3,4: have a frequency of 1, and will take 1 unit of time.

Step 5: The loop has a frequency of O(n), as it runs for n times.
Step 6-14: have a frequency of the loop in step(5), and will take 1 unit of time each run.
Step 15: the for loop runs through n sized array. And each time will take 2 unit of time.
Step 16,21: Will take 1 unit of time.
Step 17: this loop will run log(n) times for best case and for a sorted array will take n time
Step 18-20: Single stepped process.

This results in overall complexity to be:
T $\alpha \; n + log(n) * n$
T best $\alpha \; n + log(n) * n$
T worst $\alpha \; n + n^2$
Which gives average time complexity of O(nlog(n)).
And Space complexity of O(n), as n is the size of extra space (maximum) was used.

III (a). Apriori Analysis

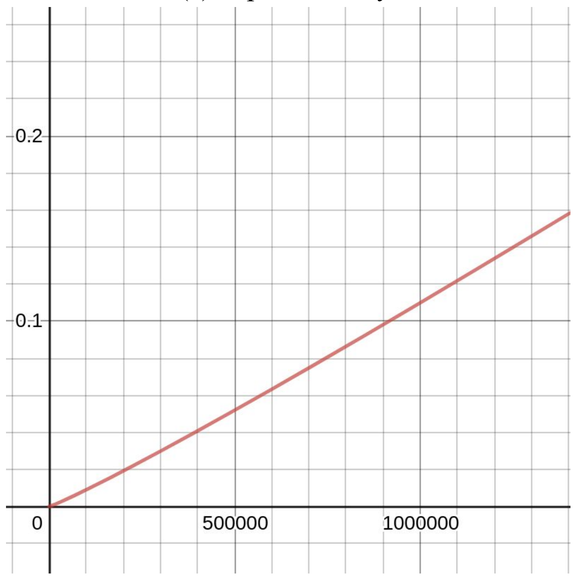


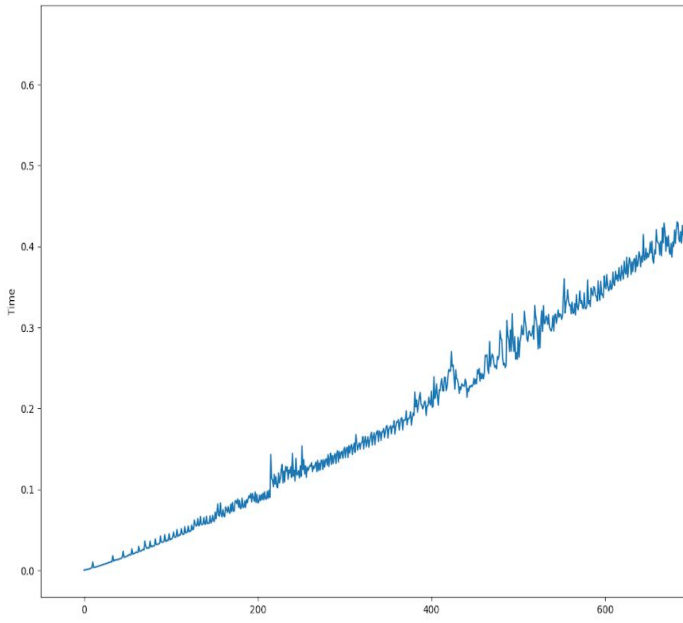
Figure1:Time complexity graph for apriori analysis.

Table1: time complexity for Apriori analysis

| | Time | FreqBest | Frequency |
|---|---|---|---|
| Step 1 | 1 | 1 | 1 |

| | | | |
|---|---|---|---|
| Step 2 | 1 | 1 | 1 |
| Step 3 | 1 | 1 | 1 |
| Step 4 | 1 | 1 | 1 |
| Step 5 | 1 | n | n |
| Step 6 | 1 | 1 | 1 |
| Step7 | 1 | 1 | 1 |
| Step 8 | 1 | 1 | 1 |
| Step 9 | 1 | 1 | 1 |
| Step 10 | 1 | 1 | 1 |
| Step 11 | 1 | 1 | 1 |
| Step12 | 1 | 1 | 1 |
| Step13 | 1 | 1 | 1 |
| Step14 | 1 | 1 | 1 |
| Step15 | 2 | n | n |
| Step16 | 1 | 1 | 1 |
| Step 17 | 1 | $log(n)$ | $n$ |
| Step18 | 1 | 1 | 1 |
| Step19 | 1 | 1 | 1 |
| Step20 | 1 | 1 | 1 |
| Step21 | 1 | 1 | 1 |

## IV. EXPERIMENTAL ANALYSIS AND PROFILING.



In the above graph, the input 'n' is considered along x-axis and time along y-axis. The graph represents Time v/s n. The above graph shows that the APosteriori analysis is consistent with Apriori analysis.

## V. CONCLUSION

From this paper we can conclude that algorithmic best and average is O(nlog(n)), worst is O(n^2) and space complexity is O(n)

## VI. REFERENCES

[1]https://www.ideserve.co.in/learn/convert-binary-tree-to-binary-search-tree
[2]https://en.wikipedia.org/wiki/Tree_traversal#In-order_(LNR)
[3]https://en.wikipedia.org/wiki/Binary_search_tree#Insertion
[4]https://www.tutorialspoint.com/data_structures_algorithms/array_data_structure.htm