AN EFFICIENT ALGORITHM FOR THE AUTOMATIC CONSTRUCTION OF AVL TREE WITHOUT ROTATIONS BY SORTING TECHNIQUE

S.Muthusundari[1], R.M. Suresh[2]

ABSTRACT

In this paper we propose a new algorithm for AVL Tree with Elimination of rotations by using sorting Technique. The basic idea of this paper is easy to implement and understand, to increase the performance by elimination of rotations.

This can be achieved by sorted the given elements, then apply divide and conquer technique to split the given elements into subsets. Then construct the binary search tree individually and then merge the trees. In this paper we prove the effectiveness of the AVL tree performance is achieved by without rotations by using sorting Technique.

***Keywords** : divide & conquer, sort, merge, elimination, rotations*

I. INTRODUCTION

Balanced tree structures are efficient way of non linear data structures **for storing information.** We know basically, that binary search is the most efficient method than any other method [1]. This non linear data structure needs to search an item in the tree with average number of comparison O (log2n). Searching time in BST is normally dependent on the height of the tree [8]. The height of the tree is increased by performing an insertion operation in the BST. As height increases the searching time is also increases. In case of deletion in the BST may decrease the height and hence it requires less time of requirement. To make the Binary Search Tree as a height balanced tree Adelson, Velskii and Landis proposed the concept of AVL tree. It is well known to us, that every AVL tree is a BST (Binary Search Tree), where every BST is not an AVL tree [1]. AVL trees are binary search trees with a balanced condition [8]. The balance condition is "the height of the left subtree of any node differs from the height of the right subtree by 1". To make the BST, as a height balanced tree, it requires various rotations in case of insertion and deletion [9]. In order to do the insertion operation in the tree, we need the following rotations.

A. Different Rotations in Insertion Operation

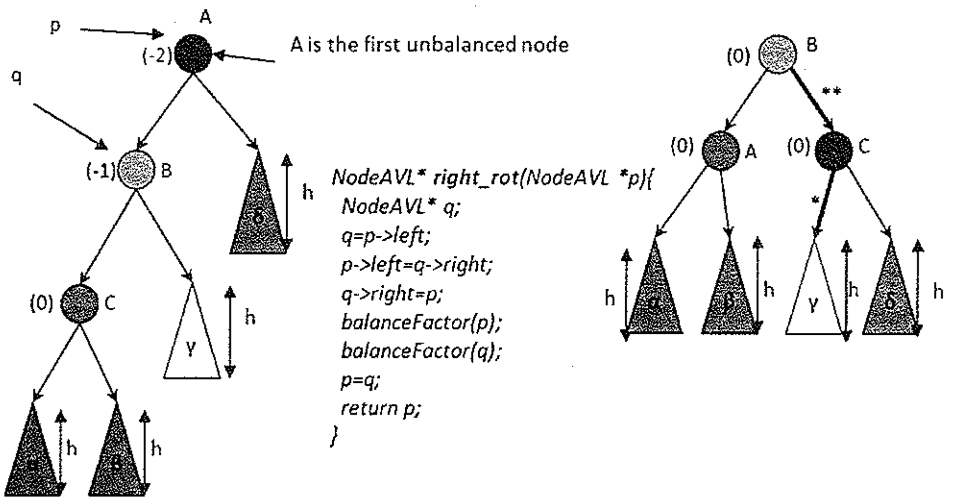
To insert the key elements in to the BST tree as a height balanced [8], we look into the following rotations [2].

1. Single Rotation
   - Left rotation
   - Right rotation
2. Double rotation
   - Left right rotation
   - Right left rotation

[1]Research Scholar, Sathyabama University, Chennai,India,
E-mail : nellailath@yahoo.co.in

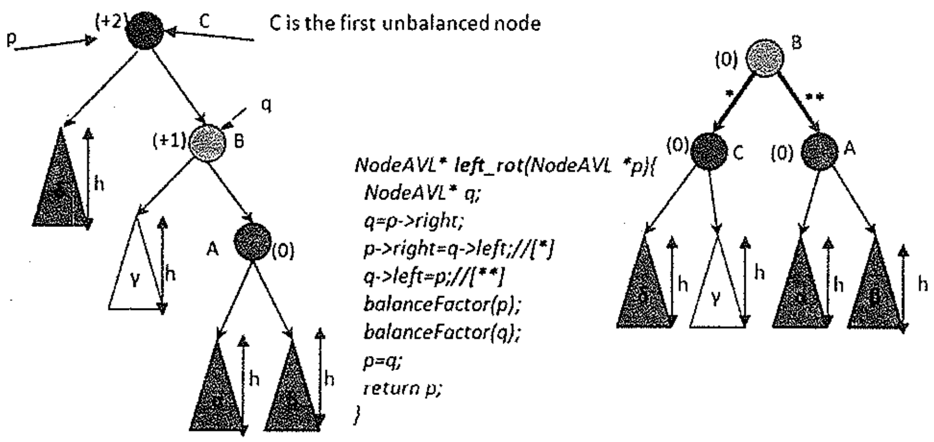
[2] Principal, Sri Muthukumaran Institute of Technology, Chennai,India,
E-mail : rmsuresh@hotmail.com

Case 1: Single right rotate



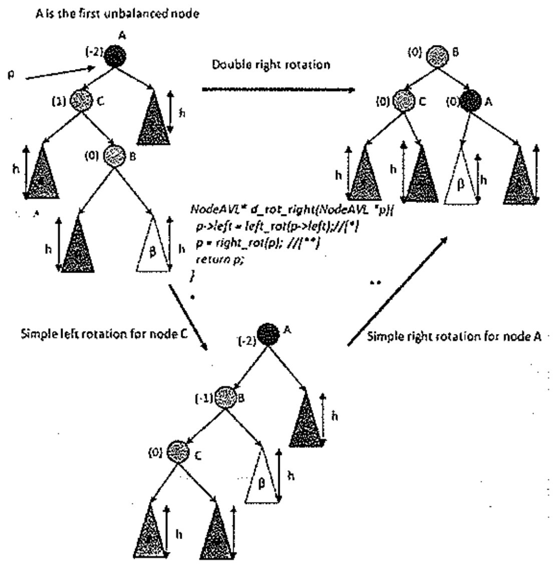
```
NodeAVL* right_rot(NodeAVL *p){
  NodeAVL* q;
  q=p->left;
  p->left=q->right;
  q->right=p;
  balanceFactor(p);
  balanceFactor(q);
  p=q;
  return p;
}
```

Case 2: Single left rotate (14)



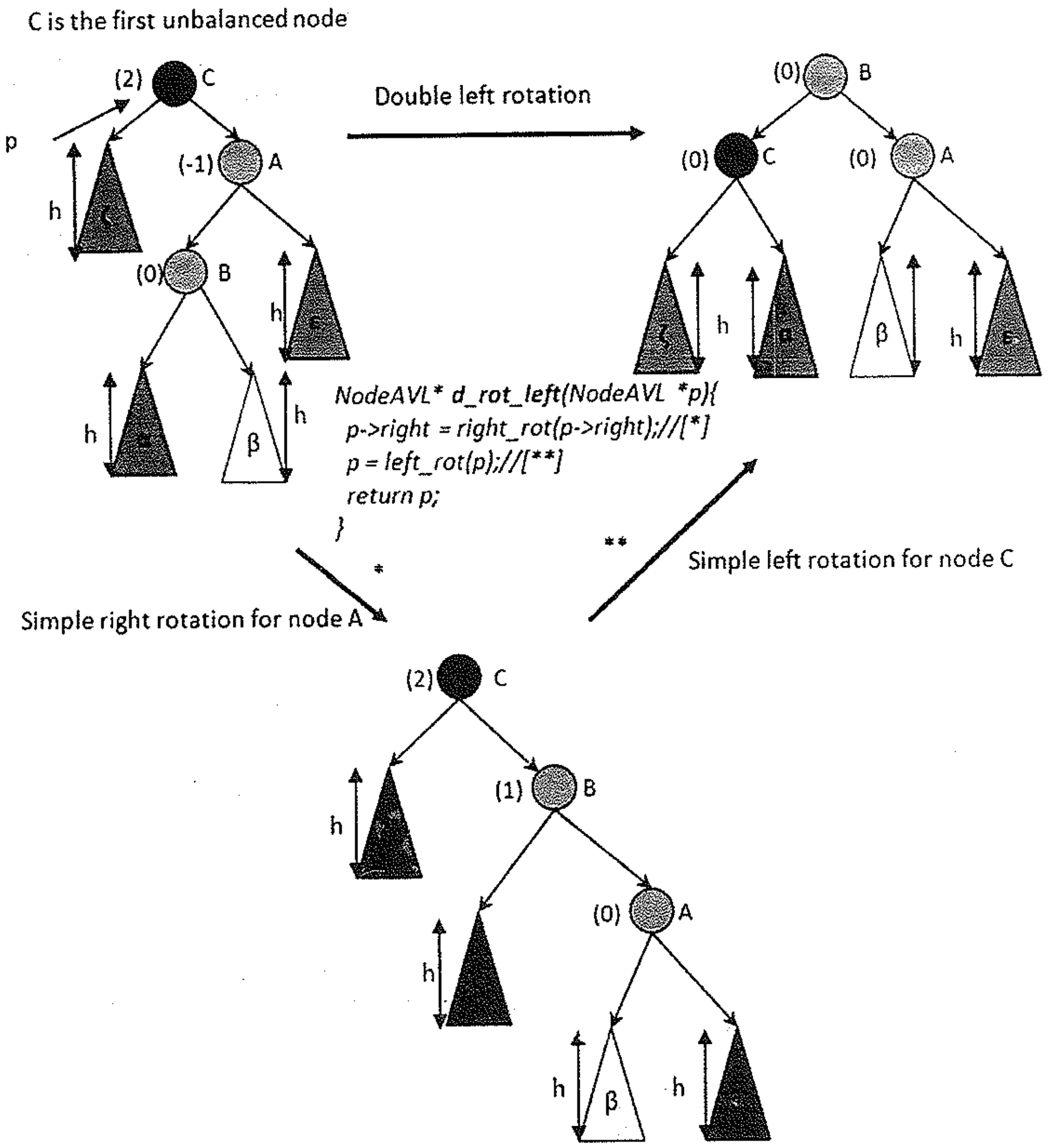
```
NodeAVL* left_rot(NodeAVL *p){
  NodeAVL* q;
  q=p->right;
  p->right=q->left;//[*]
  q->left=p;//[**]
  balanceFactor(p);
  balanceFactor(q);
  p=q;
  return p;
}
```

Case 3: Double right rotate


A is the first unbalanced node
A
(-2)
p
(1) C
(0) B
β
h
Double right rotation
(0) B
(0) C
(0) A
Simple left rotation for node C
Simple right rotation for node A
(-2) A
(-1) B
(0) C
*
**

```
NodeAVL* d_rot_right(NodeAVL *p){
  p->left = left_rot(p->left);//[*]
  p = right_rot(p); //[**]
  return p;
}
```

Case 4: Double left rotate


C is the first unbalanced node
(2) C
p
h
(-1) A
(0) B
h
h
h
β
Double left rotation
(0) B
(0) C
(0) A
h
β
h
*
**
Simple right rotation for node A
Simple left rotation for node C
(2) C
h
(1) B
h
(0) A
h
β
h

```
NodeAVL* d_rot_left(NodeAVL *p){
 p->right = right_rot(p->right);//[*]
 p = left_rot(p);//[**]
 return p;
}
```

II. Related Work

Rajeev R. Kumar Tripathi [5] proposed a new model of balancing of the AVL tree using the concept of virtual node. This virtual is a divinatory node which is inserted into the inorder traversal of the BST and by doing the inorder traversal (left, root, right) they made a BST. Ultimately this virtual node is deleted to get an avl tree.

Kim S. Larsen [1] introduced the AVL trees with relaxed balance. The aim of this paper was improving the runtime performance by allowing a greater degree of concurrency.

This is obtained by uncoupling updating from rebalancing. An additional beneath is that rebalancing can be controlled separately. They made a new collection of rebalancing operations which allows for a significantly greater degree of concurrency than the original proposal.

Nicholas j. de.Lillo et.al [6] presented the implementation of AVL rotations in java. They had presented the implementation in java a number of rotation methods that convert original BST into an AVL tree.

Hussain Abu-Dalbouh1 and Norita Md Norwawi2 et. al [7] have proposed a new clustering algorithm by using AVL tree. This paper was about Bidirectional agglomerative hierarchical clustering, to create a hierarchy bottom-up, by repeatedly merging the closest pair of data-items into one cluster. The result is a derived in to AVL tree.

III. Proposed Methodology

In this section we address the efficiency issues regarding the AVL tree performance by using a Sorting technique, is composed of the following steps.

1. Sort the given elements
2. Apply divide & conquer technique
3. Construct the binary search tree
4. Merge the two subsets

A. Algorithm

```
Step 1: Get the n elements
Step 2: Sort the elements in ascending order by Bubble sort method.
Step 3: Apply divide and conquer technique to split the given n elements into two subsets.
Step 4: Check the no of elements in the subset. If the subset element is 3( n/2 = 3), then construct the binary search tree by placing the second element as root node and first element as left child and third element as right child.
Step 5: if the subset is 4 elements( n/2 = 4) then, construct the binary search tree by placing the second element as root node and place the first node as left child to the root, and the third element as right child to the root and the fourth element as right child to the right child.
Step 6: if the subset element n/2 > 4, then again apply divide and conquer technique to split the list into further subset. Then Construct the binary search tree for the subset based on the step 4 and step 5.
Step 5: While merging the first subset needs the pointer assignment of the right most node to become as a root node, and the root node which brings as a left node to the new root node. Then merge the second subset as it is in the first subset.
Step 6: Then merge the two subsets into single set
Step 7: The merged set will show the output of the AVL tree.
```

IV. ILLUSTRATION WITH EXAMPLE

Case : 1 Subset is only four elements

Step 1: Get the n elements.

5 8 4 3 11 7 15 9

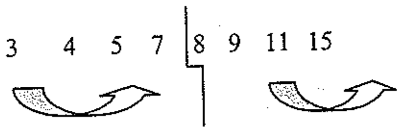
Step 2 : sort the given elements in to ascending order by Bubble sort method.

3 4 5 7 8 9 11 15

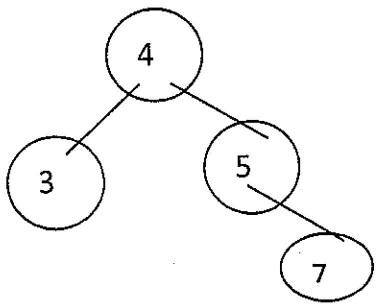
Step 3: Apply divide and conquer technique until three or four in each subset

3 4 5 7 | 8 9 11 15

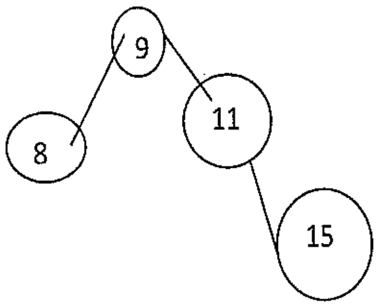
First subset second subset



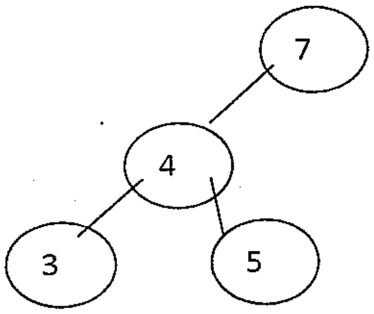
Step 4: Construct the BST for the first subset



Step 5: Construct the BST for the second subset

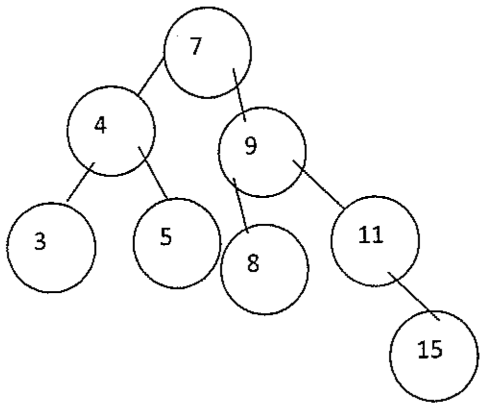


Step 6: Merge the two subsets the first subset needs the pointer assignment.



Step 6 & step 7 : And join the second subset in the first subset. Now the result shows the AVL TREE.

AVLTREE



Case : 2 Subset is more than four elements

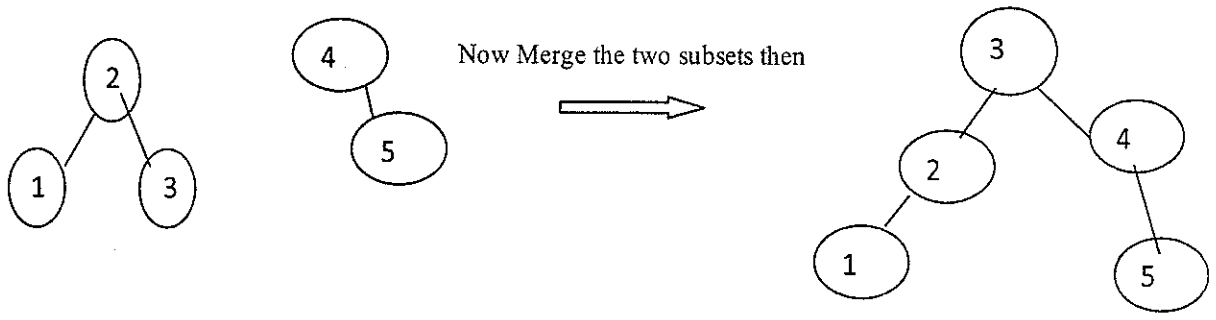
If the subset is more than 4 elements then divide into again two subsets

1 2 3 4 5 6 | 7 8 9 17

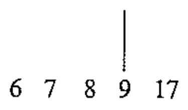
Then divide in to further subset until it becomes 3 or 4 in each subset.

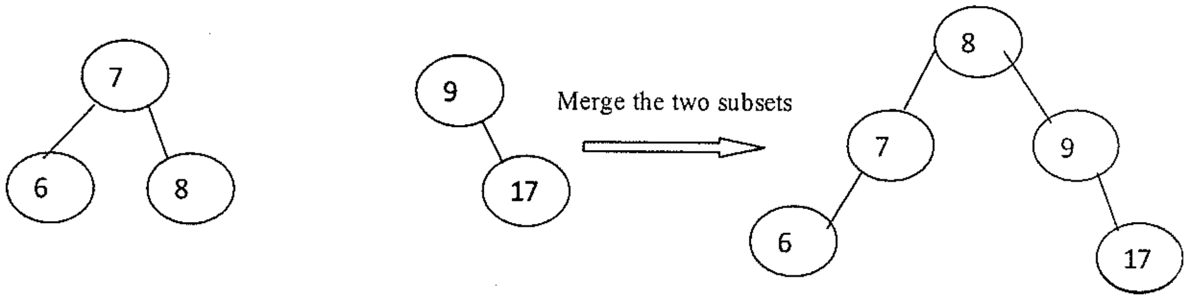
1 2 3 | 4 5 6 | 7 8 | 9 17

Then construct the binary search tree as per the step 4 and step 5 in the algorithm.

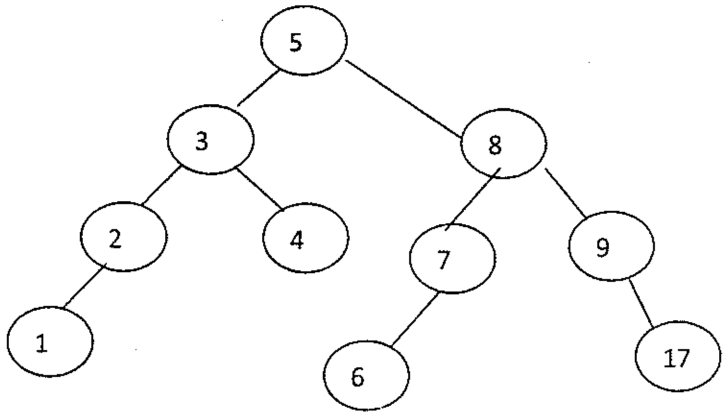


Similarly for the another subset



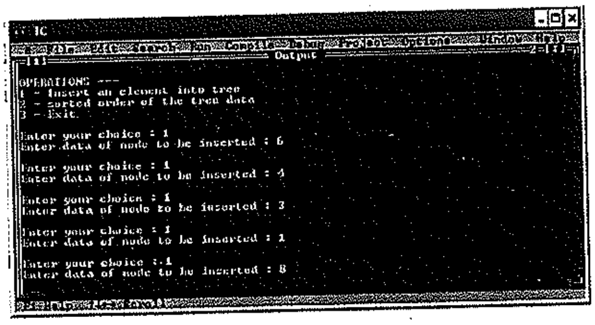


Conquer the two subsets, now it will give the Avl tree with no rotations



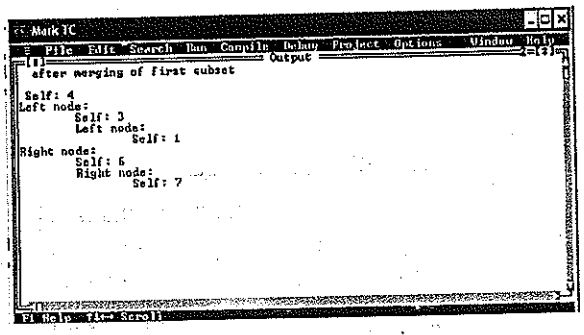
V. Results Discussions

We have the measured the performance of the proposed algorithm, by the implementation in C language. The result is given in the following figure. In this algorithm, the avl tree will be automatically constructed without by any single rotations. Normally we used to perform the avl tree by four cases of rotations. The input we have given, for the sample data : 6 4 3 1 8 7 14 10 13 . The following figures shows the result of the implementation of the algorithm.



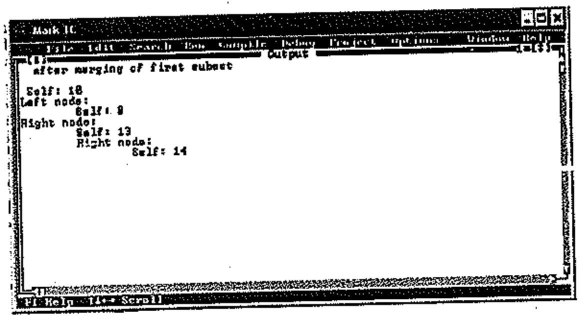
```
OPERATIONS ---
1 - Insert an element into tree
2 - sorted order of the tree data
3 - Exit.

Enter your choice : 1
Enter data of node to be inserted : 6

Enter your choice : 1
Enter data of node to be inserted : 4

Enter your choice : 1
Enter data of node to be inserted : 3

Enter your choice : 1
Enter data of node to be inserted : 1

Enter your choice : 1
Enter data of node to be inserted : 8
```

Figure : 1. Get the n elements and the sorted list



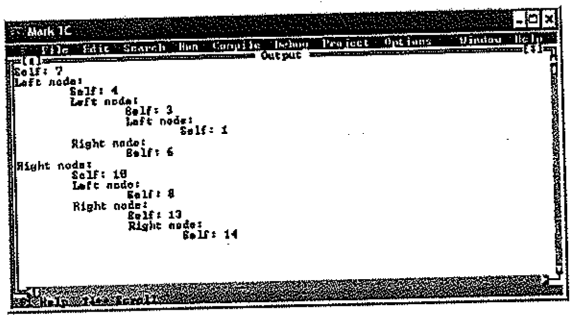
```
after merging of first subset

Self: 4
Left node:
        Self: 3
        Left node:
                Self: 1
Right node:
        Self: 6
        Right node:
                Self: 7
```

Figure : 2. Merge & Construct the BST for the first subset



```
after merging of first subset

Self: 10
Left node:
        Self: 8
Right node:
        Self: 13
        Right node:
                Self: 14
```

Figure : 3. Construct the BST for the second subset



```
Self: 7
Left node:
        Self: 4
        Left node:
                Self: 3
                Left node:
                        Self: 1
        Right node:
                Self: 6
Right node:
        Self: 10
        Left node:
                Self: 8
        Right node:
                Self: 13
                Right node:
                        Self: 14
```

Figure : 4. Automatic construction of AVL Tree without any Rotations

VI. Analysis

We have compared the performance of the proposed algorithm with the existing construction of AVL algorithm. Normally for the given input, it takes minimum of one rotation and maximum of n number of rotations based on the data we have considered for the input.. But in the proposed algorithm, if n input data is given, there is no rotation is required, and it constructs automatically AVL tree form the algorithm. Hence the performance is 100% achieved by without rotations. While we take up the rotations [15] , it will lead to confuse to perform whether single or double rotations [12] to be applied. Now this proposed algorithm helps to construct the AVL tree automatically by without rotations. Hence the efficiency is derived.

A. PERFORMANCE COMPARISON

The proposed algorithm compares with the existing method of AVL tree for the given random data, is given in the following table 1. The table 1 shows the comparisons of the minimum number of rotations are required to construct the AVL tree by the existing method [14] and the proposed algorithm for the random data set.

Table 1. Comparisons of Minimum no. of rotations in AVL Tree

Data	Minimum number of Rotations	
	Existing Method	Proposed algorithm
8	4	0
15	8	0
20	10	0
25	13	0
50	25	0
60	34	0

From the above table 1. The results were compared and analyzed, minimum we need the n/2 rotations for the given random set of data for the construction of AVL tree. But in the proposed algorithm, even though we have given 100 input of random data also, there is no rotation is required. Based on the sorted data and with the divide and conquer technique the algorithm is achieved 100% efficiency for the construction of AVL tree without using any rotaions.

VII. CONCLUSIONS

In this paper, we have presented a new algorithm for constructing the AVL tree without any rotations. For the n given input data, there is 0 rotations. For the worst case input data of 8 elements, we need 5 rotations to construct AVL tree. Normally if n elements are given, then minimum n/2 rotation is required by the normal method. In our proposed algorithm, the size may be immatter. If we are giving the input size is 100 also, there is 0 rotations. Without any rotations, based on the sorting concept, and with divide and conquer technique, the individual subset constructs the Binary search tree, and merging the subsets with one pointer adjustment it constructs automatically the AVL tree without any rotations. Hence the performance is achieved by the proposed algorithm. The efficiency is improved by 100 % in the proposed algorithm.

REFERENCES

[1]. Kim S. Larsen, *"AVL Trees With Relaxed Balance"*, November 1992

[2]. G. M. Adel'son-Vel'ski,[3] and E. M. Landis, *"An Algorithm for the Organisation of Information"*, *Dokl. Akad. Nauk SSSR*, 146:263-266, 1962. In Russian. English translation in *Soviet Math. Dokl.*, 3:1259-1263, 1962.

[3] J. F. Boyar and K. S. Larsen. E±cient Rebalancing of Chromatic Search Trees. In O. Nurmi and E. Ukkonen, editors, *LNCS 621: Algorithm Theory - SWAT'92*, pages 151-164. Springer-Verlag, 1992.

[4] L. J. Guibas and R. Sedgewick. A Dichromatic Framework for Balanced Trees. In *IEEE FOCS*, pages 8-21, 1978.

[5] Rajeev R Kumar Tripathi, *"Balancing of AVL Tree Using Virtual Node"*, International Journal of Computer Applications (0975 –8887) Volume 7– No.14 , October 2010.

[6] Nicholus et al., " *Implementations of AVL Rotations in Java*".

[7] Hussain Abu Dalbouh and Norita Md Norwawi F aculty of Science & Technology University Sains Islam Malaysia (USIM) Bandar Baru Nil ai, 71800 Nilai Negeri Sembilan, "*Bidirectional Agglomerative Hierarchical Clustering using AVL Tree Algorithm*", IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 5, No 1, September 2011 ISSN (Online): 1694-0814.

[8] D. E. Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming*. Addison-Wesley, 1968.

[9] Kicardo Baeza et.al, " *Improved bounds for the expected behavior of AVL Trees*", Journal BIT 32(1992), 297 – 315.

[10] Bayer, R., 1972, Symmetric Binary B-Trees: Data Structure and Maintenance Algorithms. Acta Informatica, 1, pp. 290–306.

[11] Chang, H., and Iyengar S.S., July 1984, Efficient Algorithms To Globally Balance a Binary Search Tree, Communication of the ACM 27, 8, pp. 695-702.

[12] Day, A. C., 1976, Balancing a Binary Tree, Computer Journal, XIX, pp. 360-361.

[13] Martin, W.A., and Ness, D.N., Feb 1972, Optimal Binary Trees Grown with a Sorting Algorithm . Communication of the ACM 15, 2, pp. 88-93

[14] Sleator, D.D., and Tarjon R. E., July 1985, Self-Adjusting Binary Search Trees. Journal of The ACM, 32(3), pp. 652-686.

[15] Ahmed S. Zaki, "*A comparative Study of 2 – 3 trees and AVL Trees*", International Journal of Computer & Information Science", 1983, vol 12, No 1.

AUTHOR'S BIOGRAPHY

S. Muthusundari received her M.Sc .,M.Phil and M.E., Degrees in Madurai Kamaraj University, Mother Teresa University and Sathyabama University respectively. She is pursuing her Ph.D at Sathyabama University. She has got 18 years of teaching Experience in Computer Science. She has presented papers in 4 National Conferences and 2 International Conferences. She has published 2 papers in the International Journals. Her area of interest includes Data Structures, Design of Algorithms and Cryptography.

Dr R. M. Suresh has completed B.E., M.Tech in CSE, Ph.D CSE in the year 2000. He had received the best engineering college teacher Award from ISTE 2009 both National and State. He has got 23 years hands of experience. He is currently working as a Principal in Sri Muthukumaran Institute of Technology(SMIT). His area of interest includes Web mining, Fuzzy logic, Data mining & image Processing.