****

**2-3 TREES**

**GROUP PROJECT**

**PROJECT REPORT**

**DATA STRUCTURES CS-201**

**SECTION-D**

**FAST NUCES**

**SUBMITTED TO**

**MA’AM ANAM QURESHI**

**&**

**SIR. FAIZAN YOUSUF**

**BY**

**FAREEHA SATTAR SHAIKH (16K-3934)**

**TABAN SHAUKAT (16K-3937)**

**HAFIZ.MUHAMMAD HUZAIFAH PUNJANI (16K-3924)**

# TABLE OF CONTENTS

[INTODUCTION: 3](#_Toc500538183)

[TIME COMPLXITY ANAYLSIS: 3](#_Toc500538184)

[INSERTION: 3](#_Toc500538185)

[SEARCHING: 3](#_Toc500538186)

[TRAVERSAL: 3](#_Toc500538187)

[DELETION: 4](#_Toc500538188)

[REDISTRIBUTION: 4](#_Toc500538189)

[MERGING: 4](#_Toc500538190)

[REDISTRIBUTION: 4](#_Toc500538191)

[MERGING: 5](#_Toc500538192)

[COMPARATIVE ANALYSIS WITH DIFFERENT DATA STRUCTURES: 6](#_Toc500538193)

[AVL VS 2 3 TREE: 6](#_Toc500538194)

[RED BLACK TREE VS 2 3 TREE: 6](#_Toc500538195)

[BST VS 2 3 TREE: 6](#_Toc500538196)

[B-TREE AND 2 3 TREE: 6](#_Toc500538197)

[GRAPH OF TIME IN NANO SECONDS FOR INSERTION IN BST, AVL, 2-3 TREE and RBT: 6](#_Toc500538198)

[REAL LIFE APPLICATIONS OF 2 3 TREES: 7](#_Toc500538199)

[PROS AND CONS: 7](#_Toc500538200)

[CONCLUSION: 7](#_Toc500538201)

# INTODUCTION:

A 2-3 tree is a type of a tree data structure whose one node can have maximum 2 keys and minimum 1 key. The operation of insertion only occurs at leaf nodes. The node having one key are called 2 node and node having two keys are called 3 nodes. B tree of maximum degree 3 also makes a 2 3 tree. The path from any 2 or 3 node in its sub tree to leaf node must be same.

# TIME COMPLXITY ANAYLSIS:

## INSERTION:

Insertion in 2 3 trees always occurs at leaf node. So, to initiate a tree we first make a root node which is also a leaf node and insert and element in it. After that, when we are about to insert second element we have to make sure that the elements must be arranged in ascending order. So, with respect to ascending order we insert second element. Now when third element comes as per the rules of 2 3 trees we cant insert 3 elements in a node so splitting will occur and the middle element among them will go to the parent and other two will again settle with respect to ascending order and will become children of that parent. This is the method of insertion.

Insertion in 2 3 trees take O(log n) time complexity. This is because it corresponds to height, the height of 2 3 tree is always O(log n) because it is a self balancing tree , it always balance its height by splitting the node whenever the node reaches its maximum limit. So, the splitting occurs toward the parent and height remain O(log n). Where n is the number of nodes.

## SEARCHING:

Searching in 2 3 trees is just like the search operation of binary search tree. First we will check if the root is null so if it is we will return root. Else if root is not null first we will recursively check the left sub tree and in left sub tree we will traverse through every key of a node in order to reach our required element. Then after left sub tree the same process will occur for right sub tree and that’s how our searching will complete.

The time taken in search is proportional to the height of the tree. The number of leaves is always greater than N/2 (i.e., more than half the nodes in the tree are leaves). So the time for lookup is also O(log M), where M is the number of key values stored in the tree.

## TRAVERSAL:

Traversal can be of Pre Order, Post Order, In Order and Level Order. In Pre Order first root node will be visited then left sub tree and then right sub tree. In Post Order first left sub tree, right sub tree and then root will be visited. In In Order first left sub tree then root and then right sub tree will be visited. This visit includes the traversing of every data element in a node.

Traversal and searching are quite similar so it also takes O (log M) where M is the number of keys stored in the tree.

## DELETION:

Deletion basically includes two cases:

1. Locate node n, which contains item I

2. If node n is not a leaf swap I with in order successor, deletion always begins at a leaf

3. If leaf node n contains another item, just delete item I

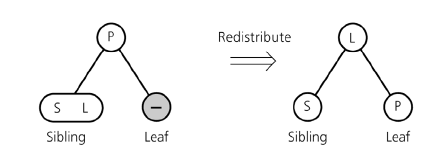
else

try to redistribute nodes from siblings

if not possible, merge node.

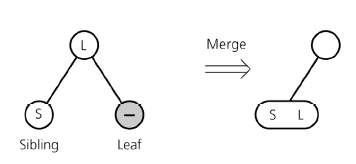
### REDISTRIBUTION:

1. A sibling has 2 items: Redistribute item between siblings and parent.



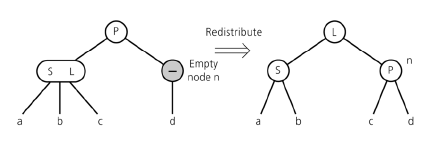
### MERGING:

1. No sibling has 2 items: Merge node, move item from parent to sibling



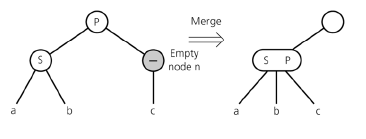
### REDISTRIBUTION:

1. Internal node n has no item left: Redistribute.



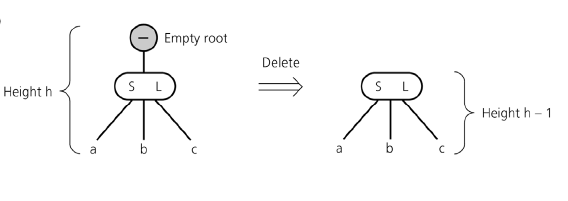
### MERGING:

1. Redistribution not possible: Merge Node, move item from parent to sibling and adopt child of n.



If n's parent ends up without item, apply process recursively

1. If merging process reaches the root, and root is without item then delete the root.



It takes O (log M) time complexity where M is the number of keys stored in the tree.

# COMPARATIVE ANALYSIS WITH DIFFERENT DATA STRUCTURES:

## AVL VS 2 3 TREE:

Although, the time complexities of the operations of AVL and 2 3 trees are same but the difference come in the height. AVL tree is not really a shallow tree and it requires to keep its balance factor balanced where as 2 3 trees are shallow and it is more easier to access the elements in 2 3 trees plus we don’t have to keep check any balance factor.

## RED BLACK TREE VS 2 3 TREE:

If we are talking about larger number of inputs then 2 3 trees are better because of better space complexity. Secondly, in red black trees we have to check imbalance after every insertion and if there is some imbalance we have to perform rotations.

## BST VS 2 3 TREE:

Worst case time complexity of BST (binary search tree) operations is O (n) where as 2 3 trees have O (log n) so, automatically it concludes that 2 3 tree is better than BST.

## B-TREE AND 2 3 TREE:

We can say that 2 3 tree is a specialized type of B tree. B tree can be constructed using two methods, which is by minimum and maximum order. B tree of maximum order 3 is 2 3 tree, but B tree can of more than 3 order. Well, the time complexity of B tree and 2 3 tree is same which is O (log N).

# GRAPH OF TIME IN NANO SECONDS FOR INSERTION IN BST, AVL, 2-3 TREE and RBT:

# REAL LIFE APPLICATIONS OF 2 3 TREES:

They can be used where faster search is required. It can be used for small management systems too. For example, for storing minimal data like Billing IDs’ with respective customer names in a departmental store.

# PROS AND CONS:

The limitation of 2 3 trees is that at most it’s one node can contain 2 data elements. Secondly, it’s applications are very rare because usually for large data management systems B trees are used. But, it is faster than AVL and BST in worst case scenarios which is a plus point for it.

# CONCLUSION:

In the end it can concluded that 2 3 trees are good to use in some special situations, where they work efficiently but there are other data structures too which are good to use. It has lesser practical applications.