UNIT-I

IntroductiontoAlgorithmAnalysis: SpaceandTime ComplexityAnalysis,AsymptoticNotations
AVLTrees –Creation,Insertion,DeletionoperationsandApplications
B-Trees–Creation,Insertion,Deletionoperationsand Applications

AlgorithmAnalysis:

There may be number of algorithms available to solve a given problem.However our task is to choose the best algorithm. Then how would we decide the best algorithm for the given problem.For that, we conduct the performance analysis of algorithm which is also referred as algorithm analysis.

Wehavetwo waystoanalysethealgorithms,theyare

1. **PrioriAnalysis**
2. **PosteriorAnalysis**

SNo	PrioriAnalysis	PosteriorAnalysis
1	Itistheoreticaland absoluteanalysis	Itispracticaland relative analysis
2	Itdoesnotrequireanyresources.i.eitisindependent ofresourceslikecomputerand programminglanguage	Itrequiresresources like computer and programminglanguage
3	Itwillproduceanapproximateresults	Itwillproduceexactresults
4	Itis donewithout executingofalgorithm	It isdoneonlybytheexecutionof algorithm
5	Itusesasymptoticnotations to represent complexityofalgorithm	Itdoesnotuseanyasymptoticnotationsto representcomplexityofalgorithm

Analyzing an algorithm means determining the amount of resources such as time and space (memory) needed to execute it. In this analysis we can determine the efficiency of algorithms. The efficiency or complexityof an algorithm is stated in terms of time and space complexity. Hence there are two main measures for the efficiency or complexity of an algorithm. In other words the complexity of an algorithm is divided into two types and they are

1. **Space Complexity**
2. **TimeComplexity**

Space Complexity:

- Space Complexitycan be defined as amount of memory(or) space required byan algorithm to run.
- Tocomputethespacecomplexityweuse2factorsi. Constantii.Instancecharacteristics.
- ThespacerequirementdenotedbyS(p)canbegivenasS(p)=C+Sp

  WhereC- Constant, it denotesthe space taken for input and output.Sp –Amountofspacetaken by an instruction, variable and identifiers.

Time complexity:

- The time complexityof an algorithm is the amount of computing timerequired byan algorithmto run its completion.
- Thereare2types ofcomputingtime1. Compiletime2. Runtime
- Thetimecomplexitygenerallycomputed atruntime(or) executiontime.

The***timecomplexity***of analgorithmisbasically therunning timeofaprogramasafunction ofthe input size

When analyzing the time complexity of an algorithm, it's important to consider the worst-case, best-case, and average-case scenarios.

1. Worst-CaseTimeComplexity

The worst-case time complexity gives an upper bound on the time an algorithm can take, means that the algorithm will never take longer than this time.

2. Best-CaseTimeComplexity

Thebest-casetime complexityrepresentstheminimumtimeanalgorithmcantake. Itprovidesalower bound on the time complexity.

3. Average-CaseTimeComplexity

The average-case time complexity represents the expected time an algorithm will take over allpossible inputs. This case is often the most realistic measure of an algorithm's performance.

AsymptoticNotations

It is one of the methods used to estimate and represent an efficiency of algorithm using simple formula. Execution time of an algorithm depends on the instruction set, processor speed, disk I/O speed, and memory etc. Hence, we estimate the efficiency of an algorithm asymptotically. According to this method, the functional behaviour of an algorithm is represented by**f(n)**, where**n**is the input size.So the time and space complexitycan be expressed using a function f(n) where n is the input size for a given instance of the problem being solved.

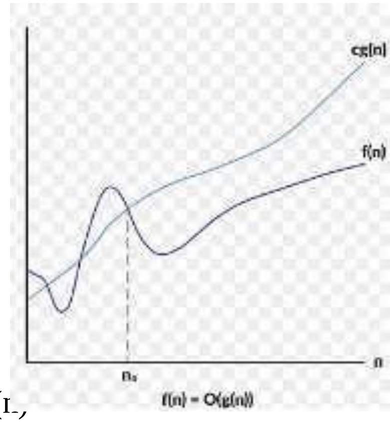
Differenttypesofasymptoticnotationsareusedtorepresentthecomplexityofanalgorithm Following asymptotic notations are used to calculate the running time complexity of an algorithm.

- **O**−Big Oh
- **Ω**−Bigomega
- **θ**− Big theta
- **o**−LittleOh
- **ω**−Little omega

Big Oh(O):

If f(n) describes the running time of an algorithm, f(n) is O(g(n)) if there exist apositive constant c and $n_0$such that, $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$. It gives the worst-case complexity of an algorithm.

Example: If f(n)=3n+2thenprovethatf(n)=O(n)



Solution:Bythe definitionofBigOh,

f(n)=O(g(n))wheref(n)⩽c.g(n)foralln≥n_0 let

g(n)=n and c=4, then

0⩽3n+2⩽4n

When n=1, 0 ⩽ 5 ⩽ 4

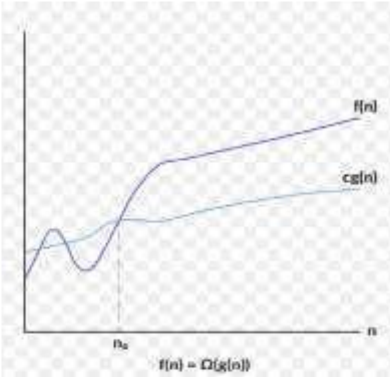
Whnen=2, 0⩽ 8⩽ 8

When n=3, 0⩽ 11⩽ 12

Therefore the givenfunction f(n) can satisfy all the cases by setting n_0= 2, g(n) = n and c=4.So we can say that f(n)= O(n)

Big Omega(Ω):

This notation is defined as, a function f is said to be Ω(g(n)), if there is a constant $c > 0$ and a natural number n_0 such that $c*g(n) \leq f(n)$ for all $n \geq n_0$. However, it provides the best case complexity of analgorithm.



Example:Iff(n)=3n+2thenprovethatf(n)=Ω(n)

Solution: Bythe definitionofBigΩ,**f(n) =Ω(g(n)**

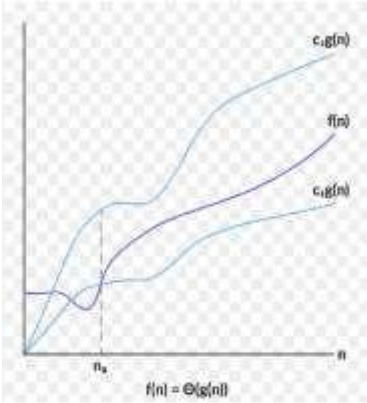
wherec*g(n) ≤ f(n) for all $n \geq n_0$ Let

c=3 and g(n)=n, then
3n+2≥3n
Whenn=1,5≥3
Whenn=2,8≥6

Therefore3n+2≥3ncanbesatisfiedfornwheren≥1,hencewecansaythatf(n)=Ω(n)

Big Theta(θ):

The function f is said to be**θ**(g(n)), if there are constants c1, c2 > 0 and a natural number n_0 suchthat c1* g(n) ≤ f(n) ≤ c2 * g(n) for all $n \geq n_0$.It is used for analyzing the**average-case** complexityof an algorithm.



Example: Iff(n)=3n+2thenprovethatf(n)=**θ**(n) **Solution**:

Bythe definition of Big**θ,f(n) =θ(n)**

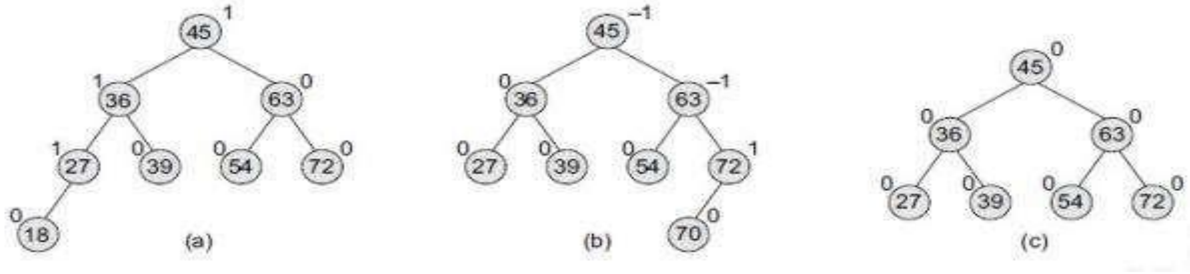
wherec1*g(n)≤f(n)≤c2*g(n)foralln≥n_0

Letc1=3,c2=4andg(n)=n,then 3n≤
3n+2 ≤ 4n
Whenn=1,3≤5≤4
When n = 2, 6 ≤ 8 ≤ 8
When n = 3, 9≤11≤ 12

Therefore3n≤3n+2≤4ncanbesatisfiedforalln,wheren≥2.Hencewesaythatf(n)=**θ**(n).

AVL Trees:

- AVLtreeisaheightbalancedbinarysearchtreeinventedbyG.M.Adelson-VelskyandE.M. Landis in 1962. Hence this tree is named AVL in honour of its inventors.
- A binary tree is said to be balanced if difference between heights of left and right sub trees of every node is either-1, 0 or 1.
- ThestructureofanAVLtreeisthesameasthatofabinarysearchtreebutwithalittle difference. i.e BalanceFactor.
- Everynodehasabalancefactorassociatedwithit.Thebalancefactorofanode iscalculated by subtracting the height of its right sub-tree from the height of its left sub-tree.
- Balancefactor=Height(leftsub-tree)–Height(rightsub-tree)
- Thereforeabinarysearchtreeinwhicheverynodehasabalancefactorof –1,0,or1issaidto be height balanced tree(AVL).
- Anodewithanyotherbalancefactorisconsideredtobeunbalancedandrequiresrebalancing of the tree by performing concern operations.
- ExamplesofAVLtreesaregivenbelow



Operations:WecanperformthefollowingoperationonAVLtrees
1) InsertionandCreation
2) Deletion
3) Search
4) Rotation

1) **Insertion**: In an AVL tree, the insertion operation is similar to insertion operation in BST with**O(log n)** time complexity. In AVL Tree, a new node is always inserted as a leaf node. But after every insertion operation, we need to check with the Balance Factor condition. If the tree is balanced after insertion then go for next operation otherwise perform suitable rotation to make the tree Balanced. The following steps can be used for this insertion operation

Step1-Insertthenewelement intothe treeusingBinarySearch Treeinsertion logic.
Step2 -Afterinsertion, checkthe **BalanceFactor**ofeverynode.
Step 3 - If the **Balance Factor** of every node is **0 or 1 or -1** then go for next operation.
Step4-Ifthe**BalanceFactor**ofanynodeisotherthan**0or1or-1**thenthattreeissaidto be unbalanced. In this case, perform suitable **Rotation** to make it balanced and go fornextoperation.

2) Deletion: The deletion operation in AVL Tree is similar to deletion operation in BST. But after every deletion operation, we need to check with the Balance Factor condition. If the tree is balanced after deletion go for next operation otherwise perform suitable rotation to make the tree Balanced.
Thisdeletionoperation in AVLhas 3casesas follows:

1. Deletingleafnode
2. Deletingnodewith one child
3. Deletingnodewith two children

Deletinga leaf node: Wecan usethefollowingsteps to delete aleaf nodefromAVL.
Step1: find thenodeto delete bysearch function.
Step 2:deletethe nodebyusingfree function
Step3:check balance factor ofnodesand applyrotation operationsif required.

Deletinganodewithonechild:Wecanusethefollowingstepstodeleteanodewithone child.
Step1: Find thenodetodeletebysearch function
Step2:Ifit hasonechildthendeletethe nodeusingfreefunctionandreplaceitwithits child.
Step3: checkbalance factor ofnodesandapplyrotation operationsif required.

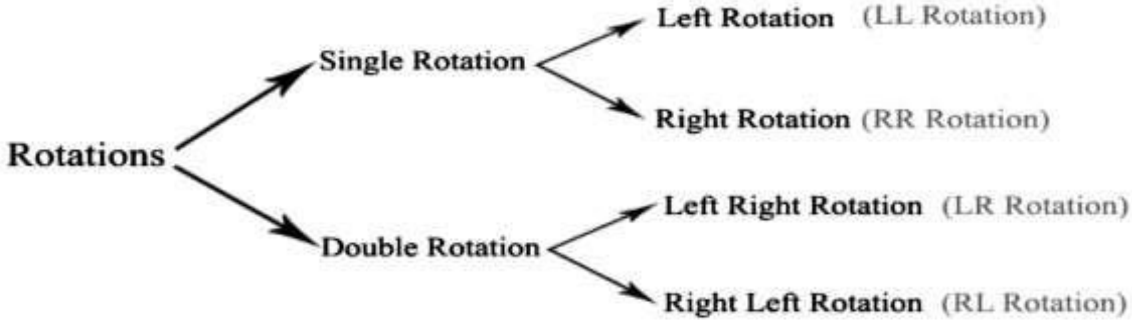
Deleting a node with two children: We can use the following steps to delete a node with two children.
Step 1: Find thenodetobedeletebysearch function.
Step2:Ifithastwochildren,thenfindthelargestnodeinleftsubtreeorsmallestnodein right sub tree.
Step3: Swapthe deletingnodewith nodefoundin step 2
Step4:check balance factor ofnodesand applyrotation operationsif required.

3) Search: In an AVL tree, the search operation is performed with **O(log n)** time complexity. The search operation in the AVL tree is similar to the search operation in a Binary search tree.The following algorithm is used for this operation

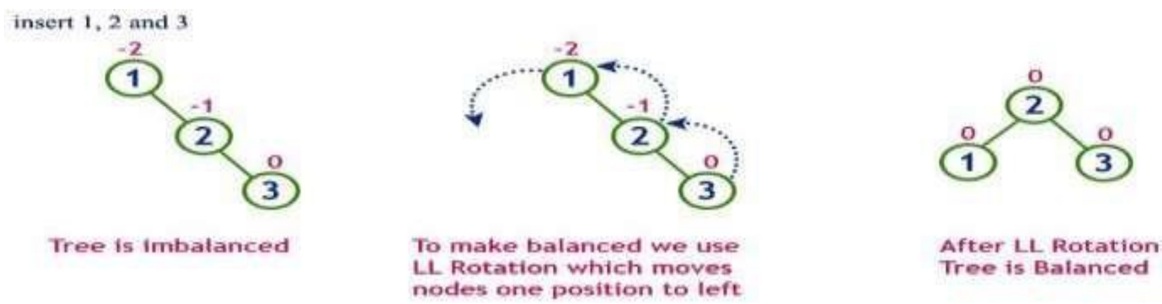
Step1-Readthesearchelementfromtheuser.
Step2-Comparethe searchelement withthe valueofrootnode inthe tree.
Step3-Ifbotharematched,thendisplay"Givennodeisfound!!!"andterminatethe function
Step4-Ifbotharenotmatched,thencheckwhethersearchelementissmalleror larger than that node value.
Step5- Ifsearchelementissmaller,thencontinuethesearchprocessinleftsubtree.
Step6- Ifsearchelementislarger,thencontinuethesearchprocessinright subtree.
Step7-Repeatthesameuntilwefindtheexactelementoruntilthesearchelementis compared with the leaf node.
Step8-Ifwereachtothenodehavingthevalueequaltothesearchvalue,then display "Element is found" and terminate the function.
Step 9 - If we reach to the leaf node and if it is also not matched with the search element,thendisplay"Elementisnotfound"andterminatethefunction.

4) Rotations:

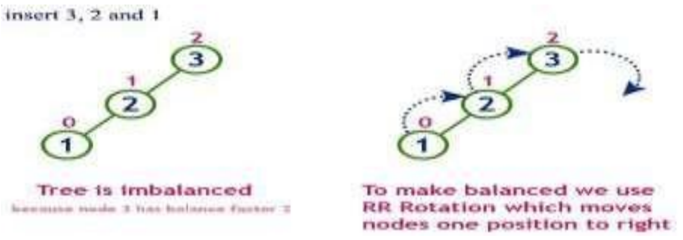
- InsertionsandDeletionscanbedoneinAVLinthesamewayasinBST.
- Afterinsertionordeletionweneedtocheckthebalancingfactorofeverynodeinthetree.
- Ifeverynodesatisfiesthebalancingfactorthenweconcludetheoperation,Otherwise wemustmake it balanced.
- WecanuseRotationoperationtomakethetreebalanced.
- Hence,rotationistheprocessofmovingnodeseithertoleftortorighttomakethetreebalanced There are 4 rotations and they are classified into two types.



1) SingleLeftRotaion(LL Rotation):
InLLrotation,everynodemovesonepositiontoleftfromcurrentposition.TounderstandLL Rotation, consider the following insertion operation in AVL tree. Insert 1, 2, 3

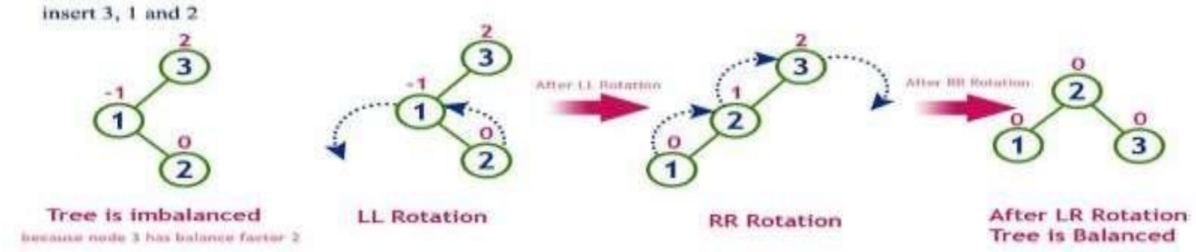


2) SingleRightRotation(RRRotation)
In RR Rotation, every node moves one position to right from the current position. To understand RR Rotation, let us consider the following insertion operation in AVL Tree. Insert 3, 2, 1



3) LeftRightRotation(LRRotation)

The LR Rotation is a sequence of single left rotation followed bya single right rotation. In LR Rotation, at first, every node moves one position to the left and one position to right from the current position. To understand LR Rotation, let us consider the following insertion operation in AVL Tree. Insert 3, 1, 2



4) RightLeftRotation(RL Rotation)

The RL Rotation is sequence of single right rotation followed by single left rotation. In RL Rotation, at first every node moves one position to right and one position to left from the current position. To understand RL Rotation, let us consider the following insertion operation in AVL Tree. Insert 1, 3, 2

AdvantagesofAVL:

- The height of the AVL tree is always balanced which means the height never growsbeyond log N, where N is the total number of nodes in the tree.
- ItgivesbettersearchtimecomplexitywhencomparedtosimpleBinarySearch trees.
- AVLtrees haveself-balancing capabilities.

Dis-advantagesof AVL:

- AVLtreescanbedifficulttoimplement.
- AVLtreeshavehighconstantfactorsforsomeoperations.

ApplicationsorUses:

- AVLtreesaremostlyused forin-memorysorts ofsetsand dictionaries.
- AVLtreesarealsousedextensivelyindatabaseapplicationsinwhichinsertionsanddeletions are fewer but there are frequent lookups for data required.
- Itisusedinapplicationsthatrequireimprovedsearchingapartfromthedatabaseapplications.

B Tree:

Introduction:

- Insearchtreeslikebinarysearchtree,AVLTree,Red-Blacktree,etc.,everynodecontains only one value (key) and a maximum of two children.
- But there is a special type of search tree called B-Tree in which a node contains more than one value (key) and more than two children.
- B-Tree was developed in the year 1972 by**Bayer and McCreight**with the name***HeightBalanced m-way Search Tree***. Later it was named as B-Tree.
- It'sabroader versionofthebinarysearch tree.
- OneofthemainadvantagesoftheBtreeisitscapacitytostorealargenumberofkeys inside a single node and huge key values while keeping the tree's height low.

Definition:B-Treeisaself-balancedsearchtreeinwhicheverynodecontainsmultiplekeys and has more than two children".

- EveryB-Treehasanorder.Thenumberofkeysinanodeandnumberofchildrenfora node depends on the order of B-Tree.

B-TreeofOrdermhasthefollowingproperties...

- All**leafnodes**must be**atsamelevel**.
- Allnodesexceptrootmusthaveatleast**[m/2]-1**keysandmaximumof **m-1** keys.
- Allnonleafnodesexceptroot(i.e.allinternalnodes)musthaveatleast**m/2**children.
- If theroot nodeis anonleaf node,thenit musthave **atleast2** children.
- Anon leafnodewith **n-1** keys musthave **n**numberof children.
- Allthe**key valuesin a node**mustbein **Ascending Order**.

OperationsonaB-Tree: Thefollowingoperations canbeperformedonaB-Tree

1. **Search**
2. **Insertion**
3. **Deletion**

1. SearchOperation inB-Tree

The search operation in B-Tree is similar to the search operation in Binary Search Tree.But,in a Binary search tree, the search process starts from the root node and we make a 2-way decision every time. However in B-Tree we make an n-way decision every time where 'n' isthetotalnumberofchildrenthenodehas.InaB-Tree,thissearchoperationisperformed with **O(log n)** time complexity. The search operation is performed as follows...

Step1-Readthesearchelementfromtheuser.
Step2 -Comparethesearch element with firstkeyvalueof rootnode in thetree.
Step 3 -If both are matched, then display "Given node is found!!!" and terminate thefunction
Step 4 -If both are not matched, then check whether search element is smaller or larger than that key value.
Step5- Ifsearchelementissmaller,thencontinuethesearchprocessinleftsubtree.
Step 6 -If search element is larger, then compare the search element with next key value in the same node and repeatesteps 3, 4, 5 and 6 until we findthe exact match oruntil thesearch element is compared with last key value in the leaf node.

Step 7 -If the last keyvalue in the leaf node is also not matched then display "Element is not found" and terminate the function.

2. InsertionOperationinB-Tree

In a B-Tree, a new element must be added only at the leaf node. That means, the new keyValue is always attached to the leaf node only. The insertion operation is performed as follows...

Step1-Checkwhethertreeis Empty.

Step2- Iftreeis **Empty**,thencreateanewnodewithnewkeyvalueandinsertitintothe tree as a root node.

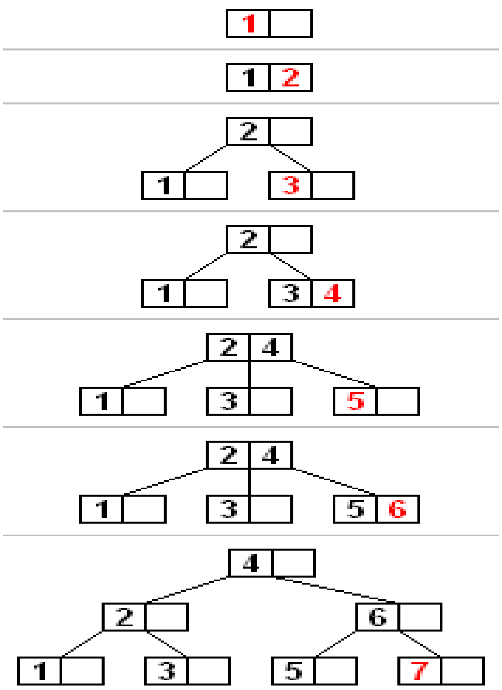
Step 3 -If tree is**Not Empty**, then find the suitable leaf node to which the new key value is added using Binary Search Tree logic.

Step4- Ifthatleafnodehasemptyposition,addthenewkeyvaluetothatleafnodein ascending order of key value within the node.

Step5-Ifthatleafnodeisalreadyfull, **split**thatleafnodebysendingmiddlevaluetoits parent node. Repeat the same until the sending value is fixed into a node.

Step6-Ifthespiltingisperformedatrootnodethenthemiddlevaluebecomesnewroot node for the tree and the height of the tree is increased by one.

For example: The following is Construction ofa**B-Tree of Order 3** by inserting numbers from 1 to 7.



3. DeletionoperationinBTree

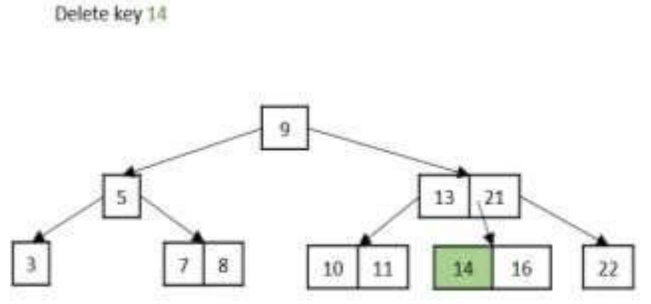
ThedeletionoperationinaBtreeisslightlydifferentfromthedeletionoperationofaBinary SearchTree.Duringthedeletion,weneedtoensurethatthenumberofkeysinthenodeafter deletion satisfy the minimum number of keys that a node can hold.So merge process takes place if required, just like split in insertion operation.

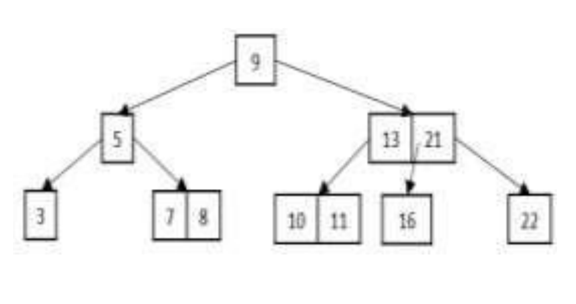
Thedeletion ofakeyfrom a Btree can takeplacein two cases,as follows–

1. Deletion of keyfrom leafnode
2. Deletion of keyfrom non-leaf node

Case1–Ifthekeytobedeletedisinaleafnodeandthedeletiondoesnotviolatethe minimum key property, just delete the node.

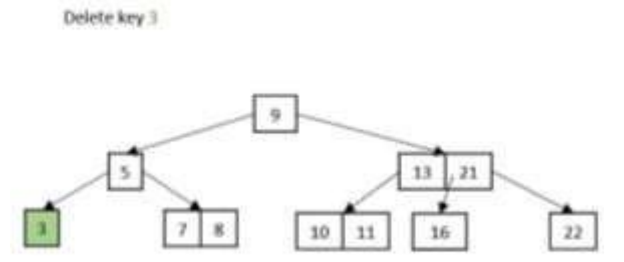
Beforedeletion ofkey14	Afterdeletion ofkey14

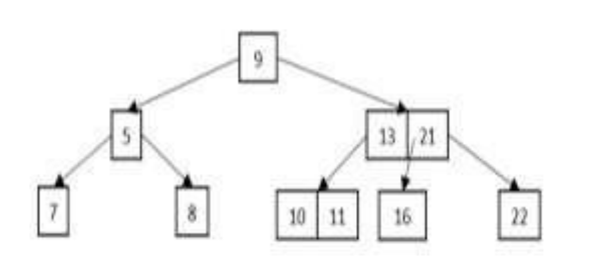




Case 2: If the key to be deleted is in a leaf node but the deletion violates the minimum key property, then borrow a key from either its left sibling or right sibling. In case if both siblings have exact minimum number of keys, merge the node in either of them.

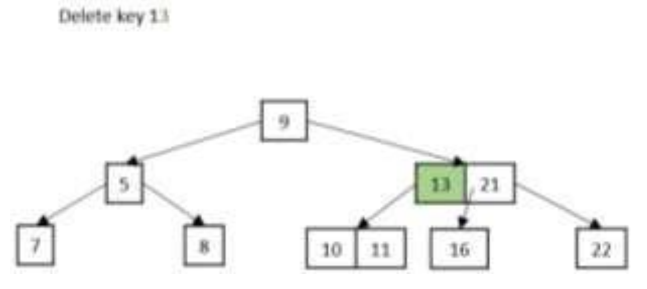
Beforedeletion of key3	**Afterdeletion ofkey3**

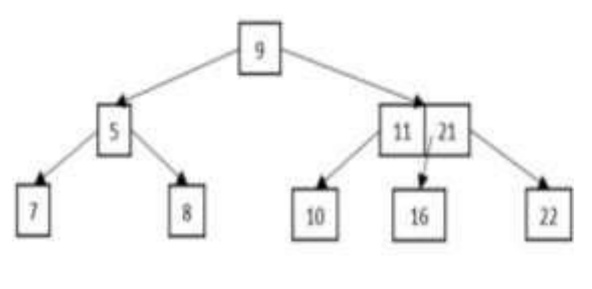




Case 3– If the key to be deleted is in an internal node, it is replaced by a key in either left child or right child based on which child has more keys. But if both child nodes haveminimum number of keys, then they are merged together.

Beforedeletion of 13	**Afterdeletionof 13**





Case 4– If the key to be deleted is in an internal node violating the minimum keys property, and both its children and sibling have minimum number of keys, then merge its sibling withits parent.

Beforedeletion of5	**Afterdeletionof5**