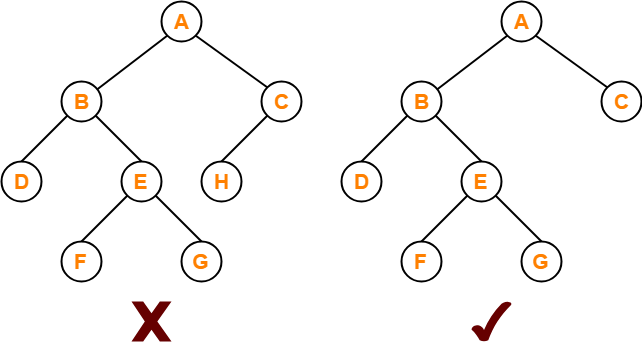
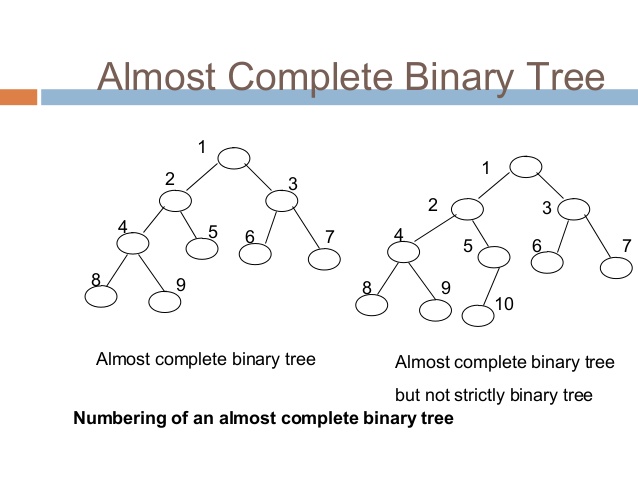


Strictly Binary Tree

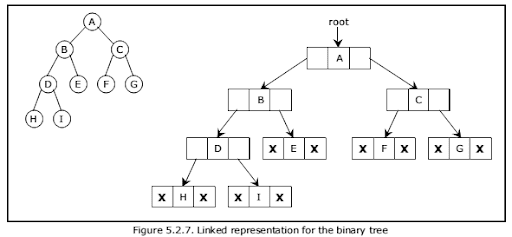


**Complete / Full Binary Tree with height/depth 4**

Perfect Binary Tree Specific Level Order Traversal - GeeksforGeeks



Linked List Representation of tree



8000

2000 A 3000

1000

root

3500 B 4000

5000 C 6000

2000

3000

7000 D 8000

X E X

X F X

X G X

3500

4000

5000

6000

X H X

X I X

7000

Data Type

struct node

{

struct node \*left;

int info;

struct node \*right;

};

struct node \* root = NULL;

A +B \*C pre: +A\*BC

In: A+B \*C

Post: A B C \* +

preorder: ABC

inorder: BAC

postorder: BCA

preorder-ABC, inorder-CBA, post-CBA

preorder: ABC

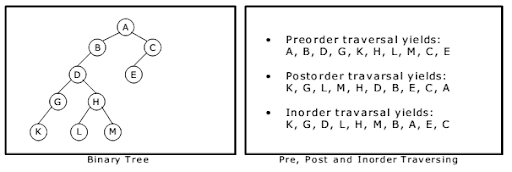
inorder: BCA

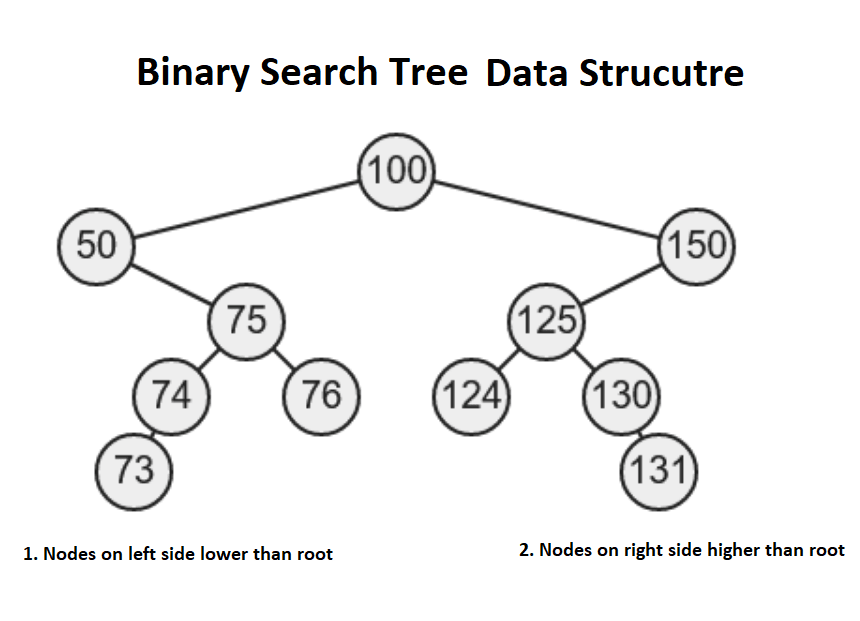
postorder: CBA

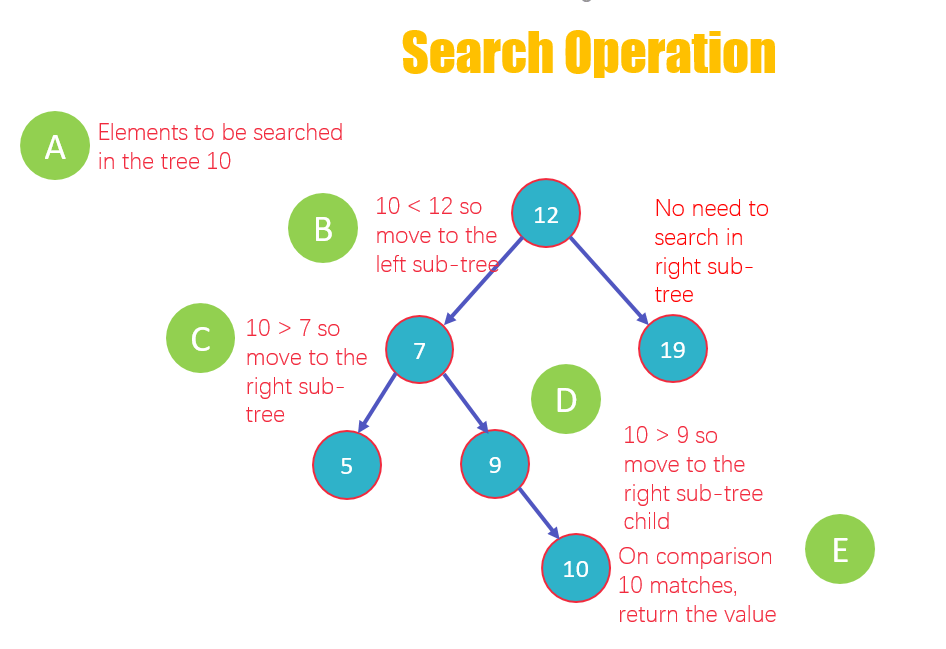
pre: ABC pre-ABC

in: ABC in- ACB

post: CBA post-CBA







Construct a Binary Search Tree (BST) for the following sequence of numbers-

50, 70, 60, 20, 90, 10, 40, 100

When elements are given in a sequence,

* Always consider the first element as the root node.
* Consider the given elements and insert them in the BST one by one.

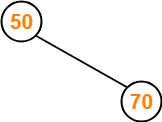
The binary search tree will be constructed as explained below-

**Insert 50-**

https://www.gatevidyalay.com/wp-content/uploads/2018/07/Binary-Search-Tree-Construction-Step-01.png

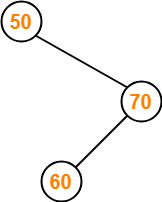
**Insert 70-**

* As 70 > 50, so insert 70 to the right of 50.



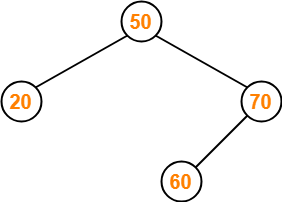
**Insert 60-**

* As 60 > 50, so insert 60 to the right of 50.
* As 60 < 70, so insert 60 to the left of 70.



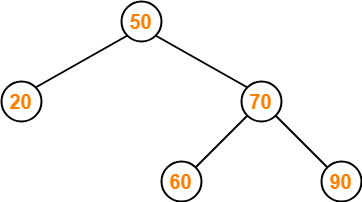
**Insert 20-**

* As 20 < 50, so insert 20 to the left of 50.



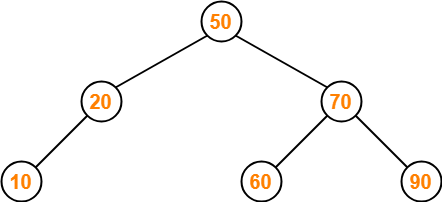
**Insert 90-**

* As 90 > 50, so insert 90 to the right of 50.
* As 90 > 70, so insert 90 to the right of 70.



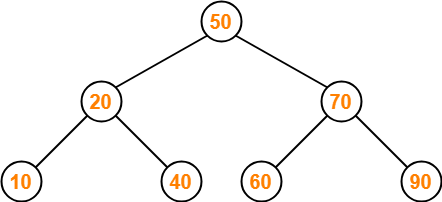
**Insert 10-**

* As 10 < 50, so insert 10 to the left of 50.
* As 10 < 20, so insert 10 to the left of 20.



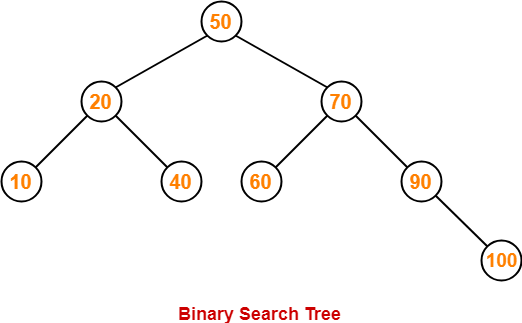
**Insert 40-**

* As 40 < 50, so insert 40 to the left of 50.
* As 40 > 20, so insert 40 to the right of 20.



**Insert 100-**

* As 100 > 50, so insert 100 to the right of 50.
* As 100 > 70, so insert 100 to the right of 70.
* As 100 > 90, so insert 100 to the right of 90.



Create a BST by inserting following info

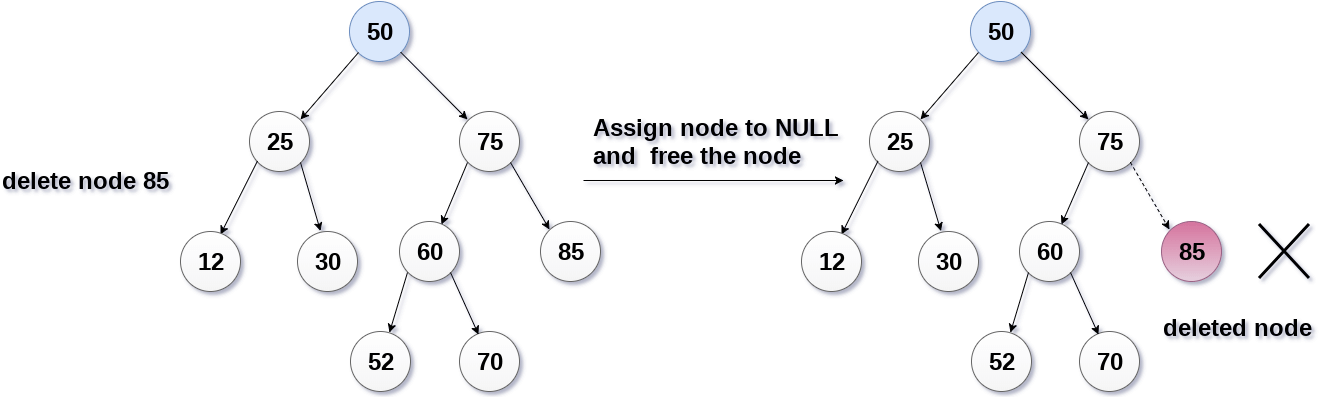
13, 3, 4, 12, 14, 10, 5, 1, 8, 2, 7, 9, 11, 6, 18

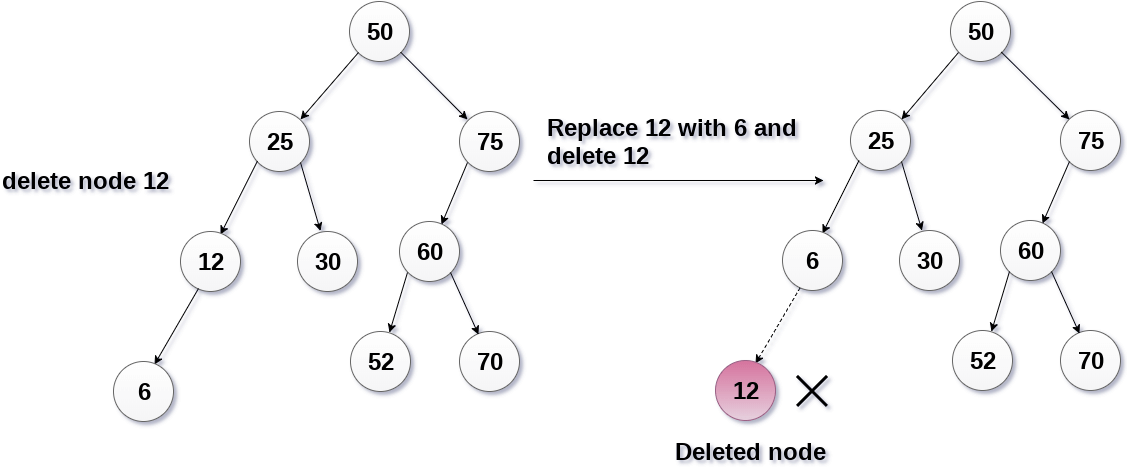
\begin{figure}
\centerline{\psfig{figure=figures/Fbstexample.ps}}
\end{figure}

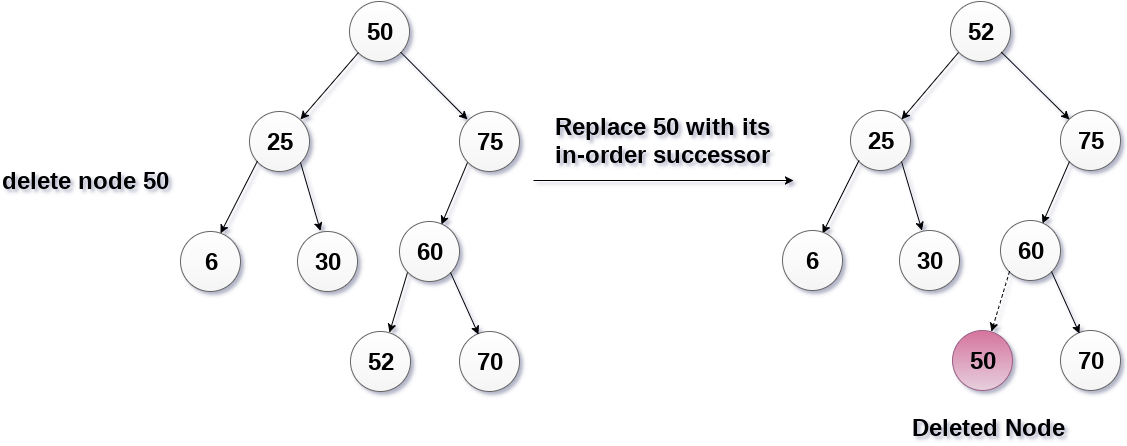
Deletion from the bst

\begin{figure}\centerline{\psfig{figure=figures/Fdelbst.ps}}
\end{figure}

Ex-Deletion







struct node\* minValue(struct node\* r)

{

struct node\* min = r;

if (min == NULL)

{

printf("BST is empty");

return min;

}

/\* loop down to find the leftmost leaf \*/

while (min->left != NULL)

{

min = min->left;

}

return(min);

}

main : struct node \* min = minValue(root); printf(“ Minmum value= %d”, min->info);

struct node\* maxValue(struct node\* r)

{

struct node\* max = r;

if (max == NULL)

{

printf("BST is empty");

return max;

}

/\* loop down to find the leftmost leaf \*/

while (max->right != NULL)

{

max = max->right;

}

return(max);

}

//recursive function

struct node\* rminValue(struct node\* r) main=>min= rmin(1000)

{ 4000

if (r == NULL ) rmin(1000->left=2000)

return NULL;

else if (r->left == NULL) 4000

return r; rmin(2000->left=4000)

else

return rminValue(r->left);

}

struct node\* rmaxValue(struct node\* r)

{

if (r == NULL)

return r;

else if ( r->right == NULL)

return r;

else

return rmaxValue(r->right);

}

struct node\* search(struct node\* r, int val)

{

struct node \*p;

p=r;

while( p!=NULL && p->info!=val)

{

if(p->info > val)

p=p->left;

else if(p->info < val)

p=p->right;

}

return p;

}

Main 🡺 struct node \* t = search(root,29); if(t ==NULL) printf(“Node is present”);

struct node\* rsearch(struct node\* r, int key)

{

if (r == NULL || r->info == key)

return r;

// Key is greater than root's key

if (r->info < key)

return search(r->right, key);

// Key is smaller than root's key

else if (r->info > key )

return search(r->left, key);

}

struct node\* insert(struct node\* r, int key)

{

struct node \* new, \*par,\*t;

new = (struct node \*) malloc(sizeof(struct node));

new->info = key;

new->left = NULL;

new->right = NULL;

if( r == NULL )

r = new;

else

{

t = r;

par = NULL;

while (t != NULL)

{

par = t;

if (key < t->info)

t = t->left;

else

t = t->right;

}

if ( key < par -> info)

par->left = new;

else

par->right = new;

}

return r;

}

Main🡺 root = insert(root,10);

struct node\* rinsert (struct node \*r, int ele)

{

struct node \*new;

if(r= =NULL)

{

new =(struct node\*)malloc(sizeof(struct node));

new -> info = ele;

new ->left=NULL;

new ->right=NULL;

r = new;

}

else if(ele < r->info)

root

1000

r->left=rinsert(r->left,ele);

else if(ele > r->info)

r->right=rinsert(r->right,ele);

return r;

}

root = NULL

main => root = rinsert(root,10) => rinsert(NULL,10)

root = 1000

main =>root = rinsert(root,20) = rinsert(1000,20)

root

1000

1000

2000

=> 1000->right = rinsert(1000->right,20) =>

1000->right = rinsert(NULL,20) => 1000->right = 2000

root = 1000

main =>root = rinsert(root,15) = rinsert(1000,15)

=> r->right = rinsert(r->right,15) =>

1000->right = rinsert(2000,15)

=> 2000->left = rinsert(2000->left=NULL,15)

=>2000->left = 3000

1000

root

1000

2000

3000

struct node\* deleteNode(struct node\* r, int key)

{

struct node \* temp;

if (r == NULL)

return r;

else if (key < r->info)

r->left = deleteNode(r->left, key);

else if (key > r->info)

r->right = deleteNode(r->right, key);

// This is the node to be deleted

else

{

// node with only one child or no child

If (r->left == NULL)

{

temp = r;

r = r->right;

free(temp);

return r;

}

else if (root->right == NULL)

{

temp = r;

r = r->left;

free(temp);

return r;

}

else

{

// node with two children: Get the inorder successor (smallest

// in the right subtree)

temp = min(r->right);

// Copy the inorder successor's content to this node

r->info = temp->info;

// Delete the inorder successor

r->right = deleteNode(root->right, temp->key);

}

}

return r;

}

void preorder(node \*r)

{

if(r!=NULL)

{

printf(" %d",r->info);

preorder(r->left);

preorder(r->right);

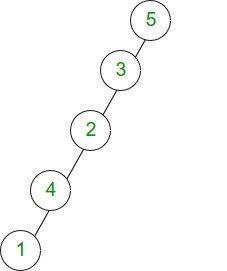
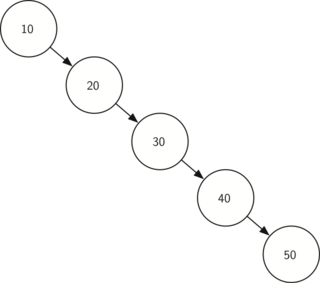
}

}

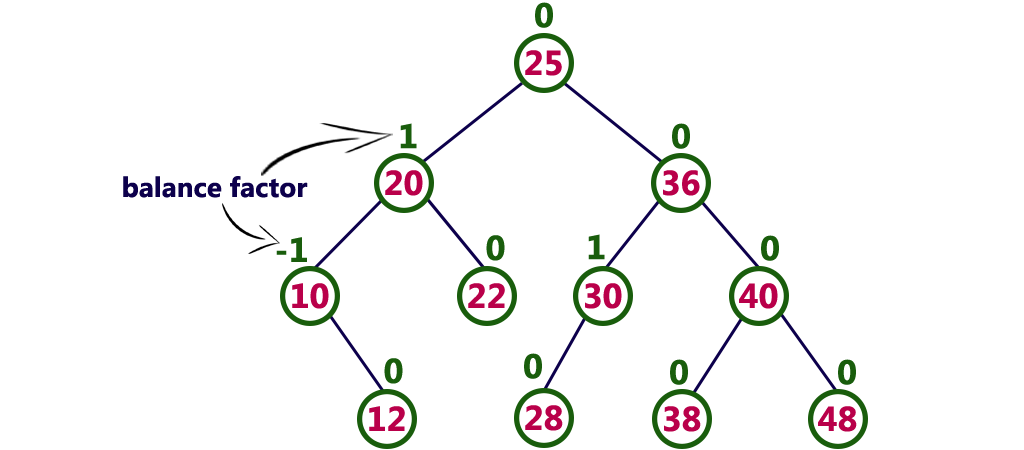
**Disadvantage of BST**

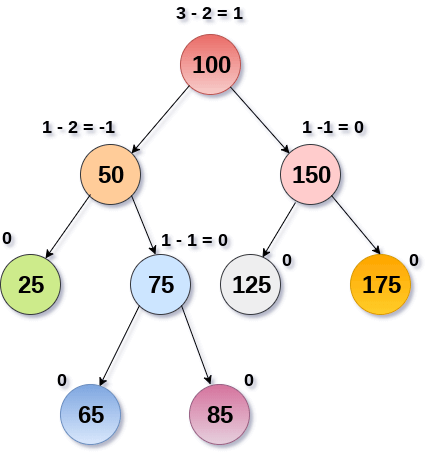
**Skewed BST**

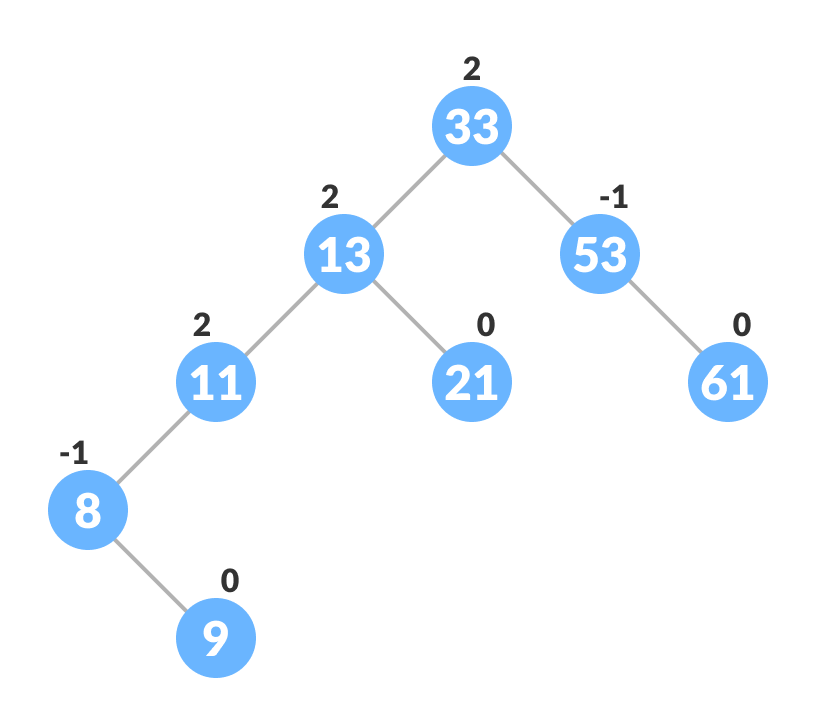
1. Left skewed bst
2. Right skewed bst

 [](https://www.google.com/url?sa=i&url=https://stackoverflow.com/questions/28315718/skewed-trees-relation-to-binary-search-tree&psig=AOvVaw3Vi0aXD3OrpyegUoZz0uCp&ust=1586930275743000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCKiyoK7U5ugCFQAAAAAdAAAAABAH)

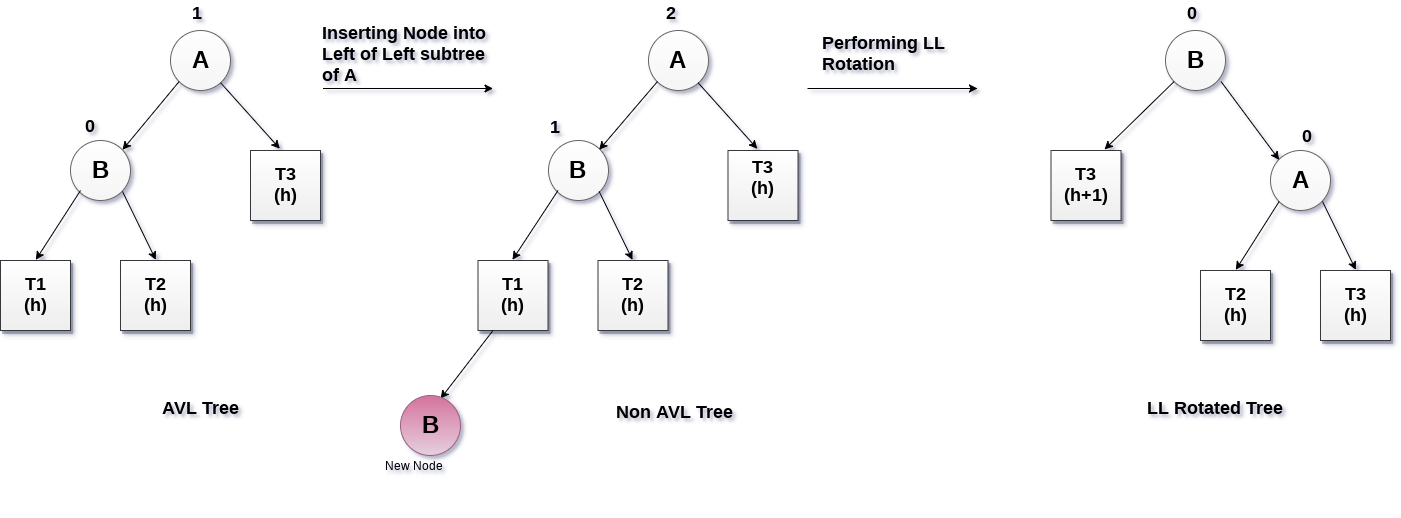
**Balance Factor Example**

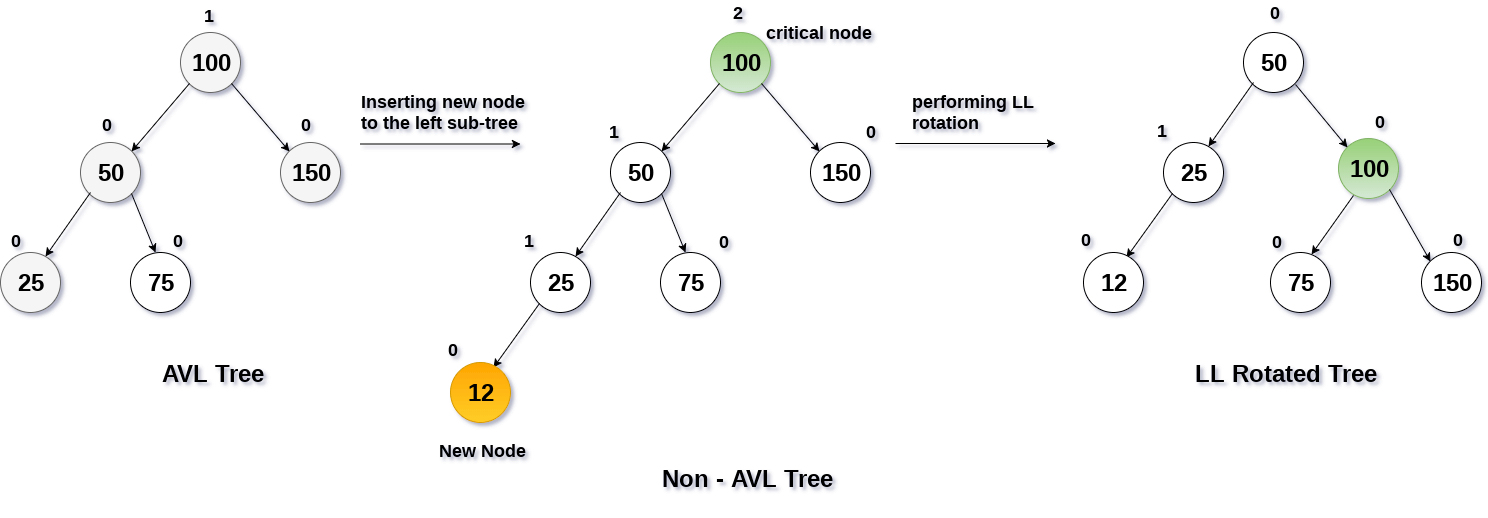




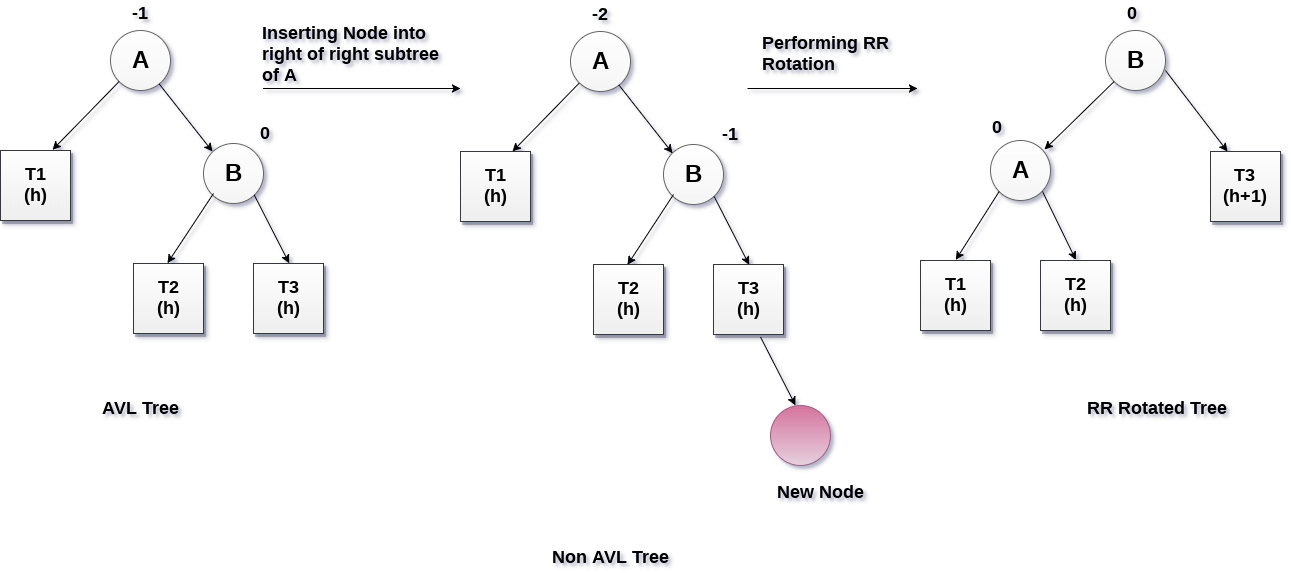


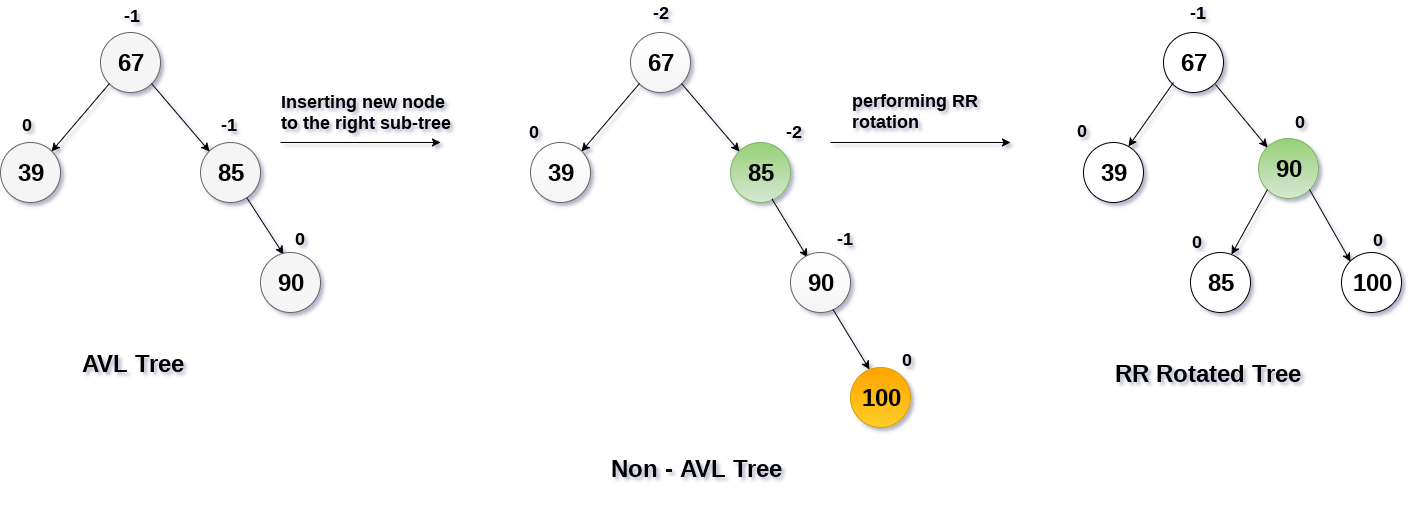
LL Rotation



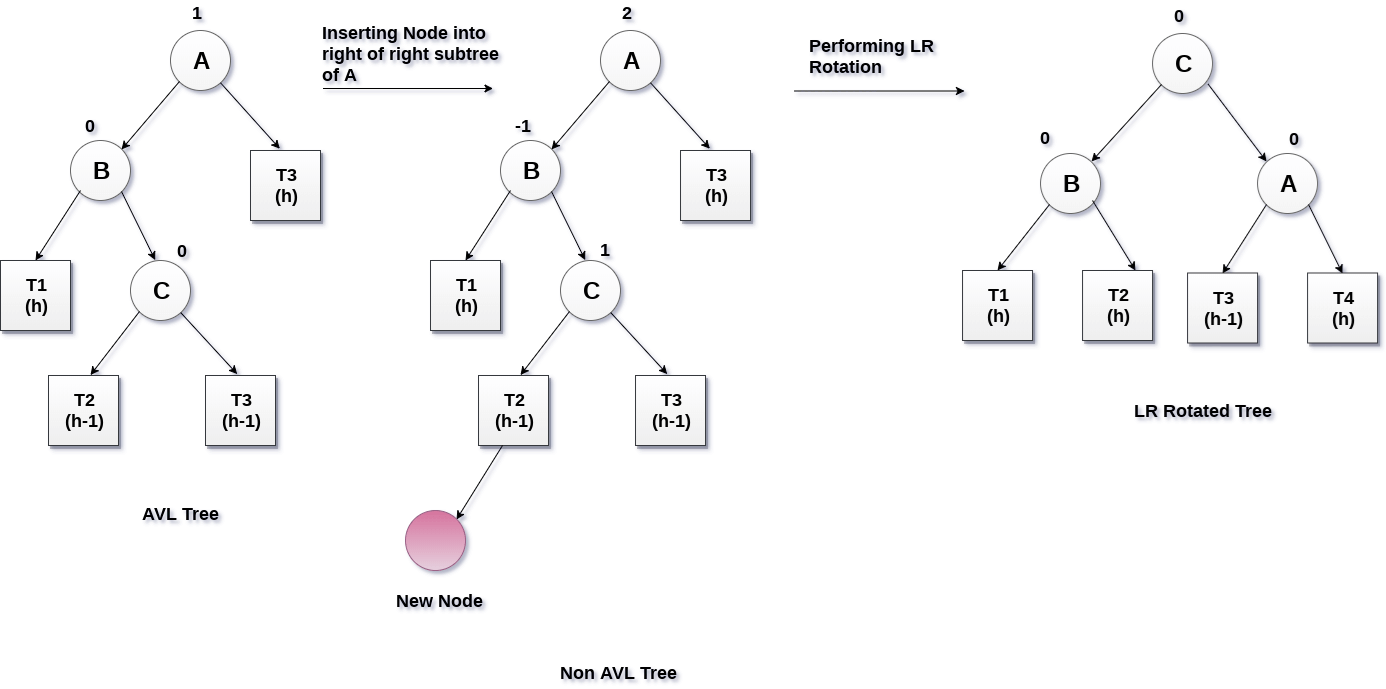


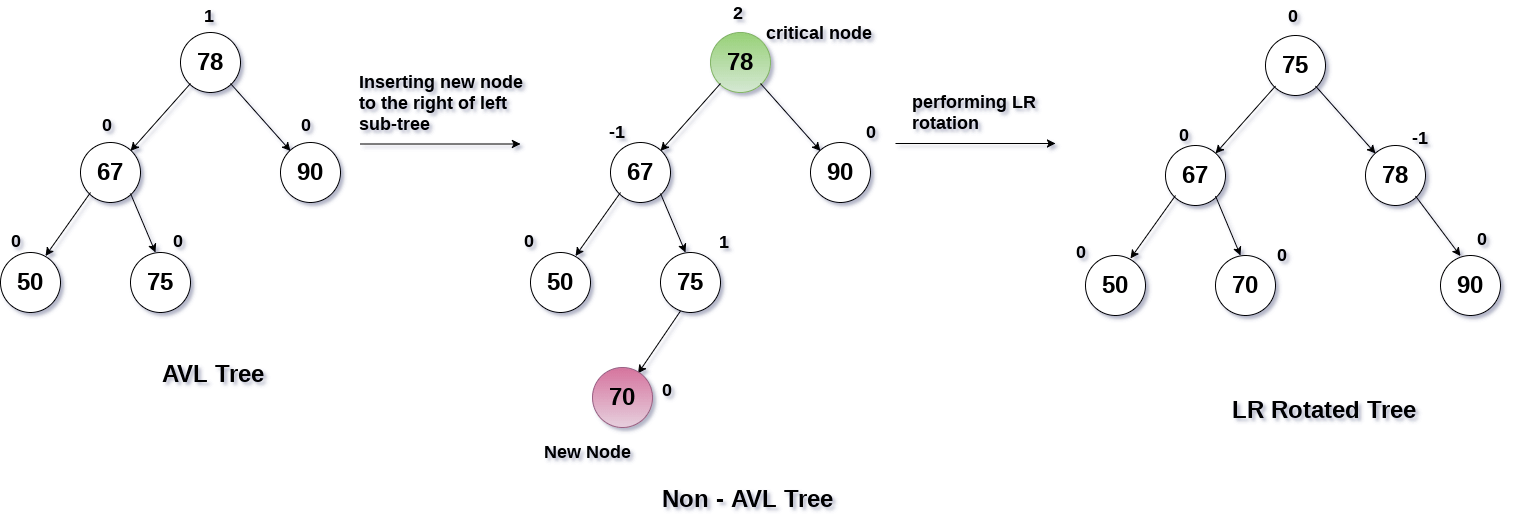
RR Rotation

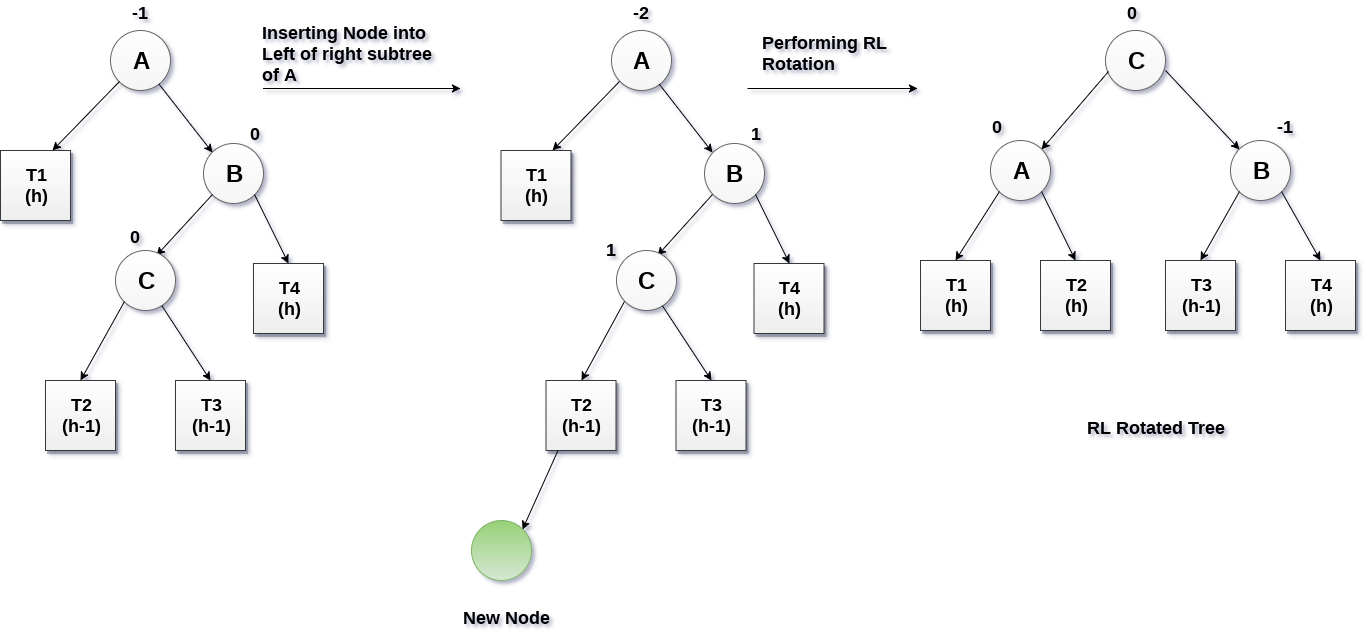


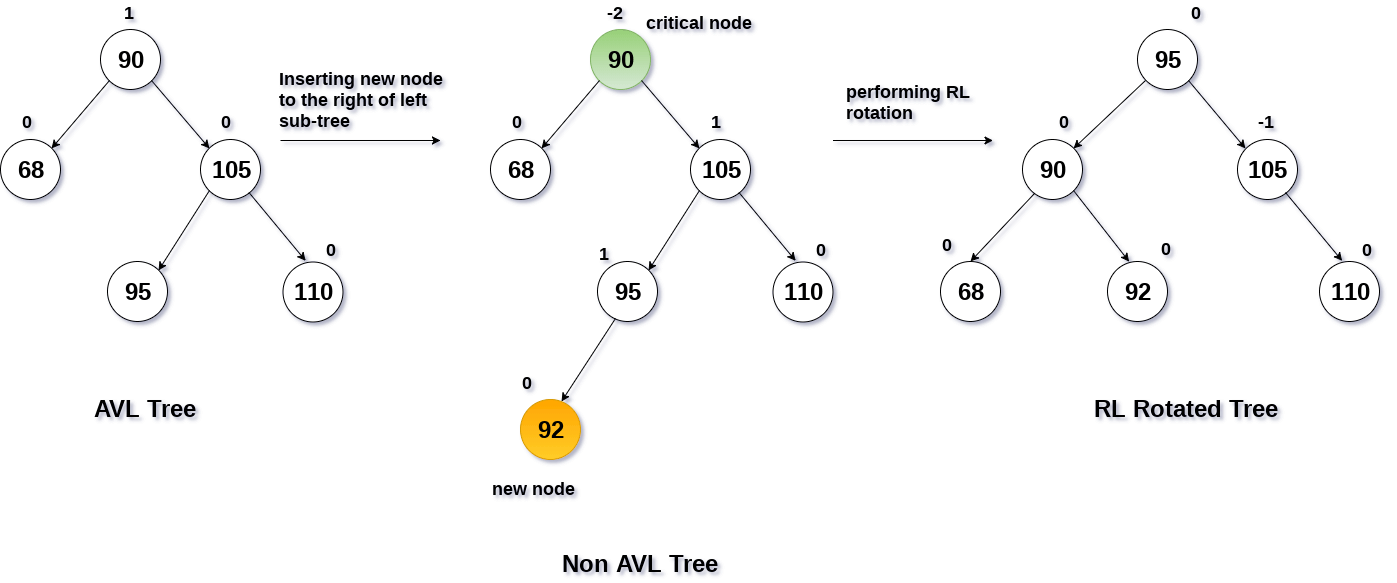


LR Rotation









EX- Creation of AVL TREE

**Construct an AVL tree by inserting the following elements in the given order.**

**63, 9, 19, 27, 18, 108, 99, 81**

