UNIT - V

Similar to Binary tree except that the elements to the left Side of any node must be smaller than the node (parent) and the elements to the right must be greater or equal to the parent node Value

```
# include<stdio.h>
# include< stdlib.h>

struct tree
{
 struct tree  *left;
 char data;
 struct tree  *right;
}* root = NULL;

void insert(struct tree *temp, struct tree *nn)
{
  char ch;

  if ( nn -> data  <   temp -> data)
  {
      if (temp -> left  ==  NULL)
          temp -> left = nn;
      else
          ~~temp->~~ insert (temp -> left, nn);
  }
```

```
    if ( temp -> rigut == NULL)
        temp -> rigut == nn;
    else
        insert ( temp -> rigut , nn);

  }
} // end of insert function.


void Create ( )
{
  struct tree *nn;
  char ch;  char x;

  do
  {
    Printf ( " Enter data ");
    x = getchar ();

    nn=(struct tree *) malloc ( sizeof (struct tree));
    nn-> left = NULL;
    nn-> data = x;
    nn-> rigut = NULL;


    if ( root == NULL)
        root = nn;
    else
        insert ( root, nn);
  Printf("\n do you wish to Continue (y/n)");
  getchar();

  ch = getchar();
  getchar();
  }
 While( ch == 'y');

 }   // end of Create function.

// inorder Traversal
void  display( struct tree *temp)
{
   if (temp != NULL)
     {
       display( tem->left );
       Printf ("%c\n', temp->date);
       display( temp-> right);
     }
}


void   main( )
{
   Create();
   display( root);
}
```

AVL Trees

(Adelsion, velski & landis)

An AVL Tree is a binary search tree in which the difference of heights of left sub tree and right subtree for any node must be either -1, 0, +1.

inserting into AVL tree

Case (i) :- inserting a node into the left child of the left sub tree
(if tree unbalanced, use LL Rotation)

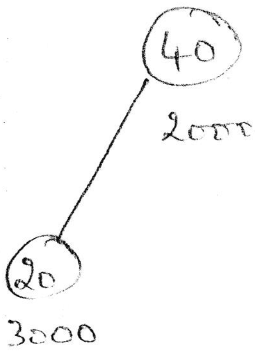
Case (ii) :- inserting a node into the right child of the left sub tree
(if tree unbalanced, use LR Rotation)

Case (iii) :- inserting a node into the right child of right sub tree
(if tree unbalanced,
use RR Rotation)

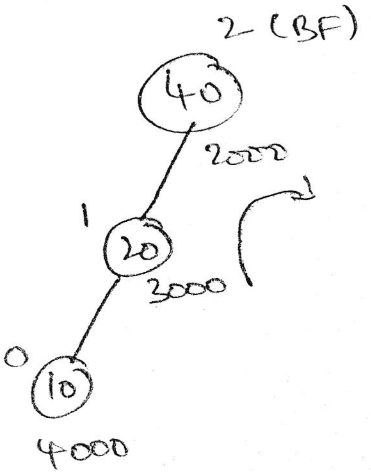
Case (iv) :- inserting a node into the left child of right subtree.
(if tree unbalanced,
use RL Rotation)

Scenario's for LL Rotation

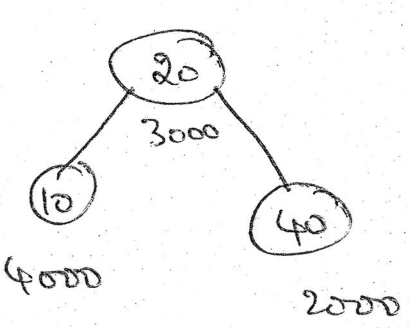
Ex 1 :-



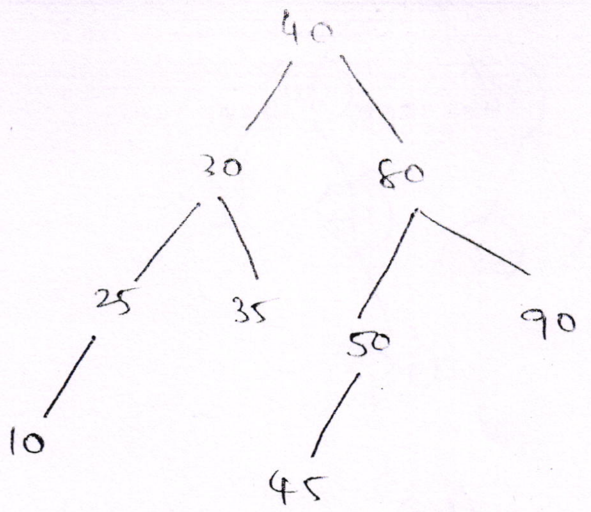
Insert 10.



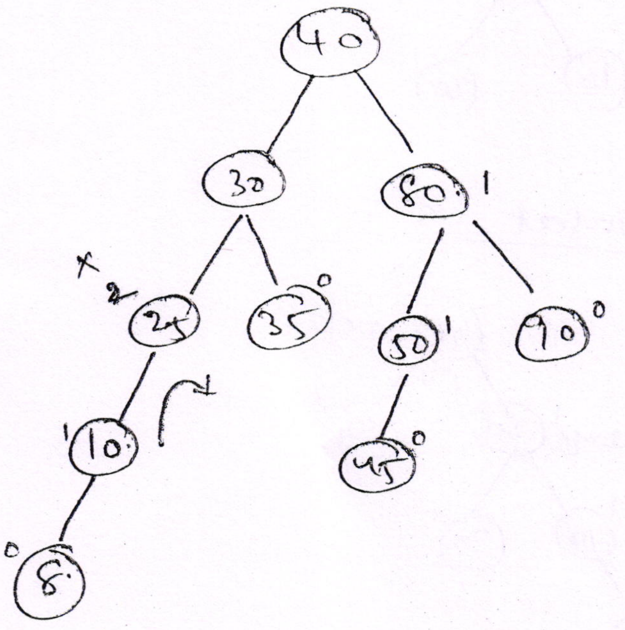
For the node at address 2000, the balance factor = 2 which is not accept ⇒ apply LL Rotation.
(ie Right Rotation)



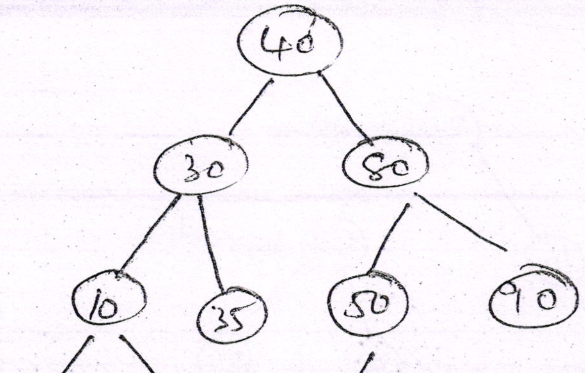
Ex 2 :



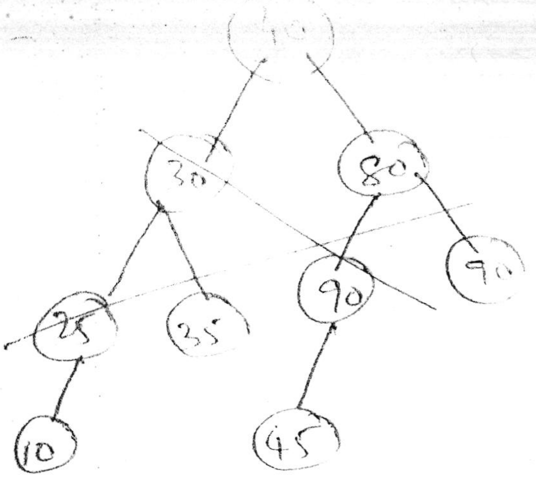
insert '8'.

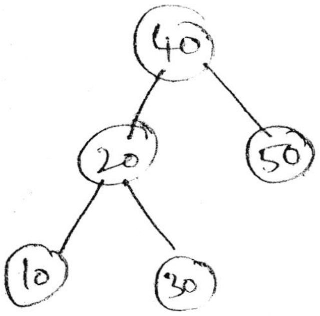


apply LL Rotation (



Ex3 -

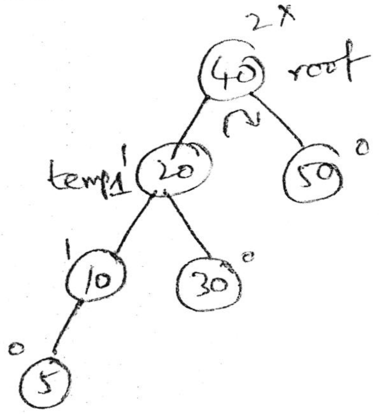




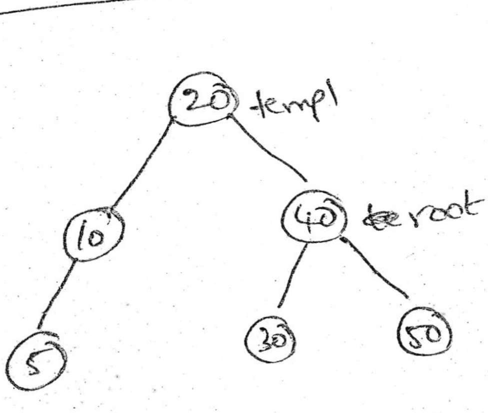
(a) insert 5

(b) insert 15

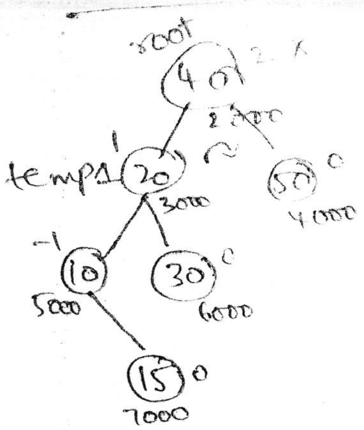
(a) insert 5



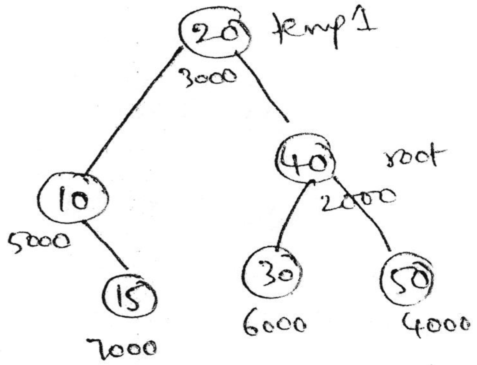
apply LL Rotation



(b) insert 15



apply LL Rotation.

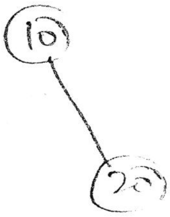


Code for LL Rotation

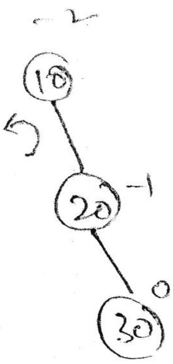
```
temp1 = root -> left
root -> left = temp -> right
temp1 -> right = root ,    root -> BF = 0.
 root = temp1

root -> BF = 0
```

Examples of RR Rotation

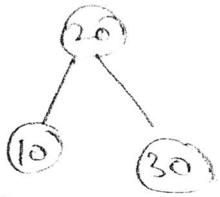
Ex1



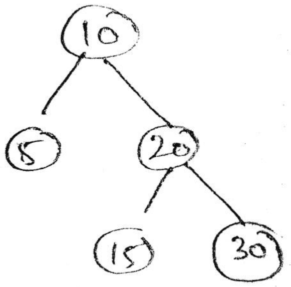
insert 30



Apply RR Rotation.



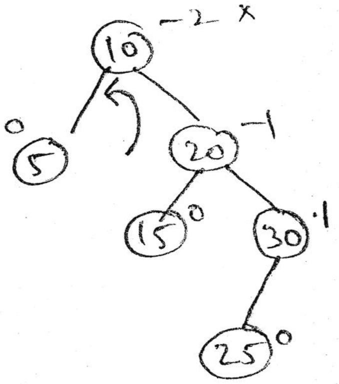
Ex2



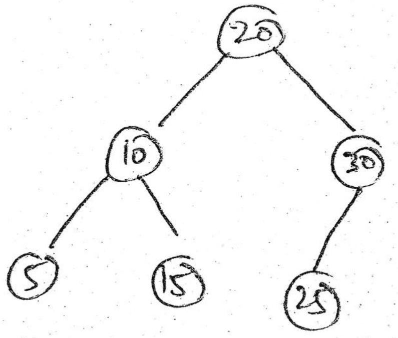
(a) insert 25

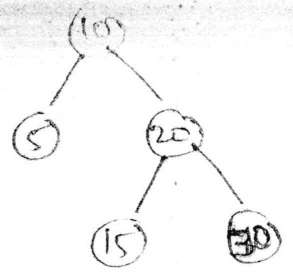
(b) insert 40

(a) insert 25

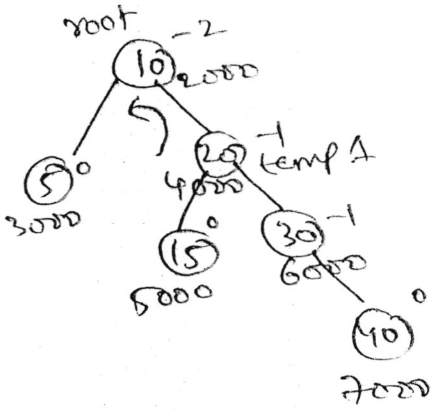


apply RR Rotation.





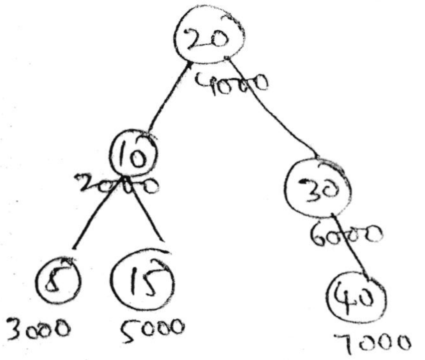
(b) Insert 40



apply

RR

Rotation

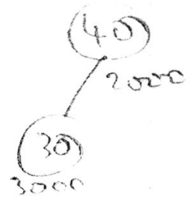


RR Rotation Code :-

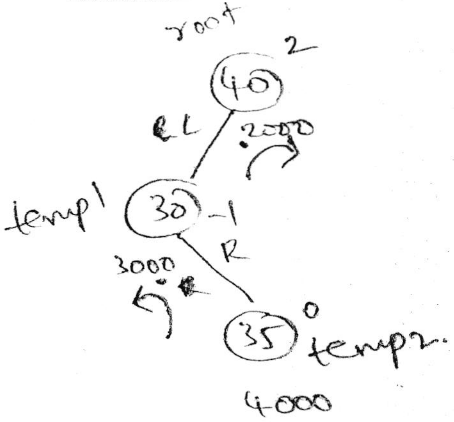
```
temp1 = root -> rigut·

root->rigut = temp1 ->left ;
temp1 ->left = root ;

root -> BF = 0 ;

root = temp1 ;

root -> BF = 0 ;
```

Examples of : LR Rotation

Ex1

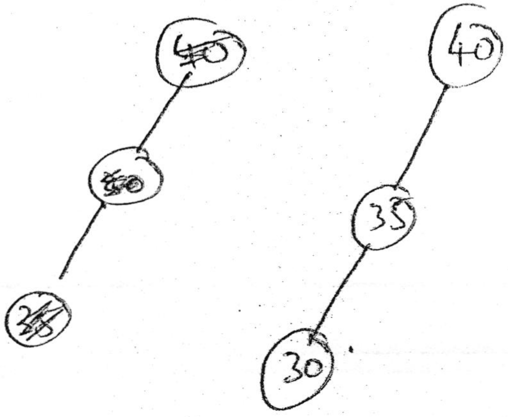


Insert 35.

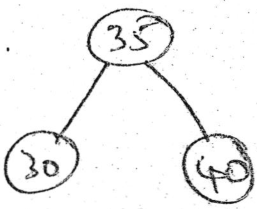


apply double Rotation. (LR)

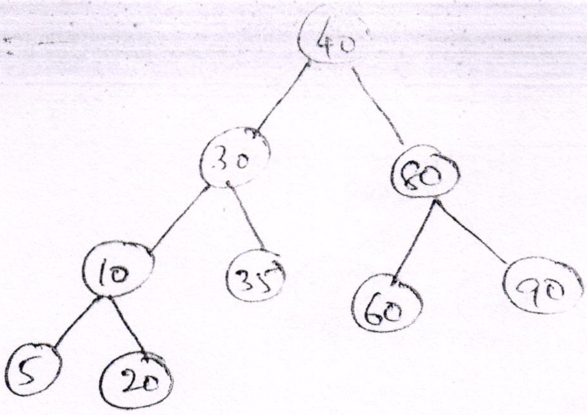
Step1 :- ~~apply~~



Step2



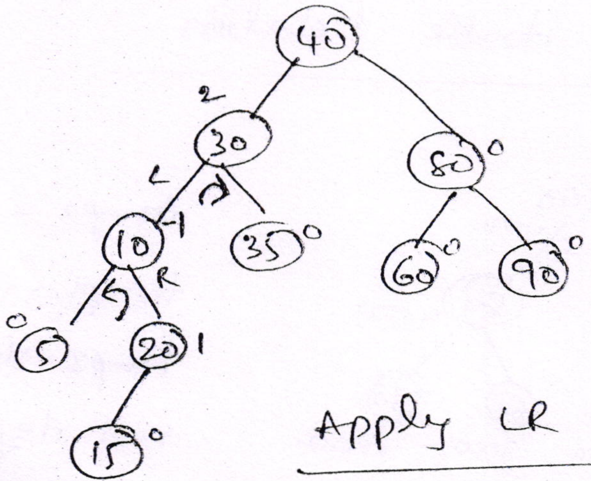
Ex2 :-



(a) insert 15

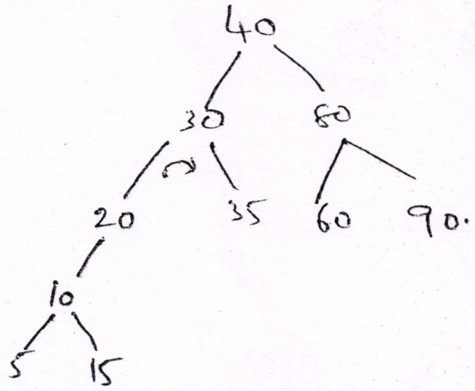
(b) insert 25

(a) insert 15

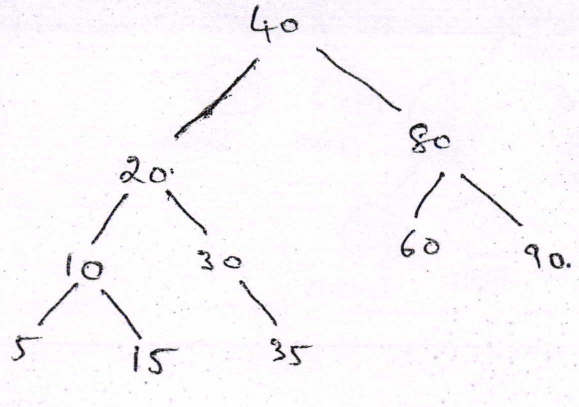


Apply LR Rotation

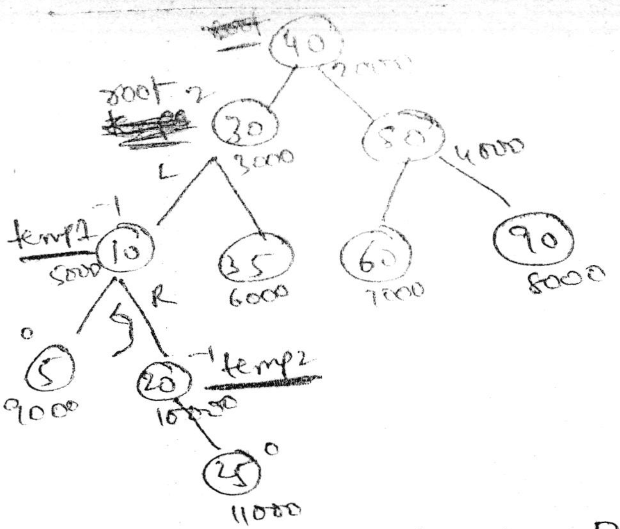
Step1 :-



Step2 :-



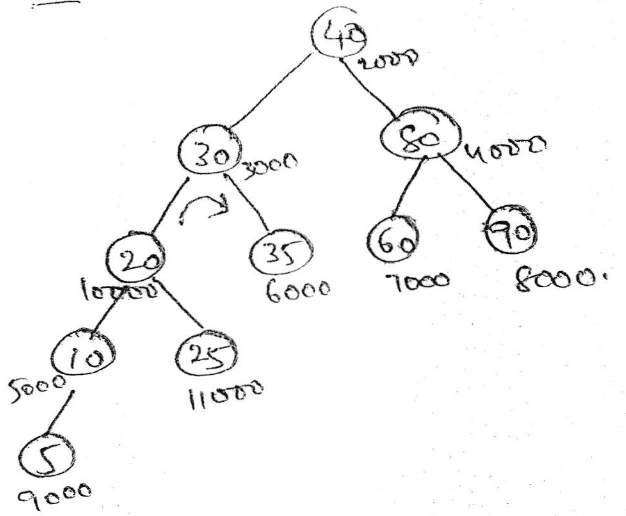
(1) Insert



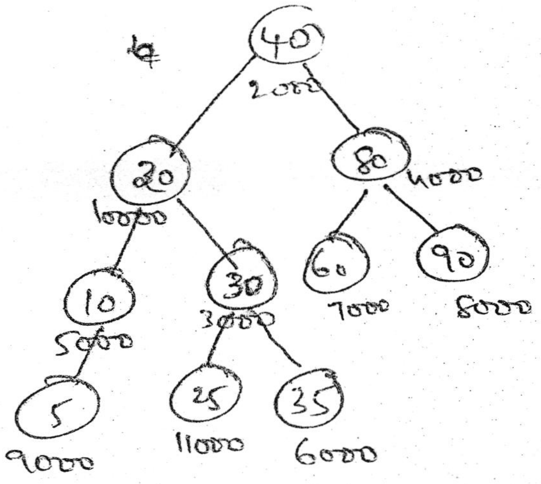
apply LR double Rotation

Code

Step1



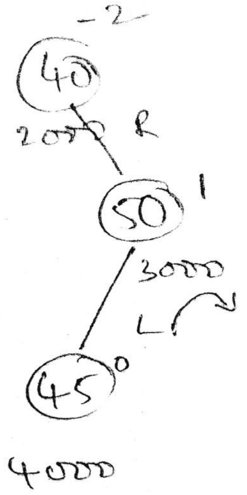
Step2



```
temp2 = temp1 -> right
temp1 -> right = temp2 -> left
temp2 -> left = temp1
root -> left = temp2 -> right
temp2 -> right = root;

{ if (temp2 -> BF == 1)
        root -> BF = -1;
{ else
        root -> BF = 0;

{ if (temp2 -> BF == -1)
        temp1 -> BF = 1;
  else
        temp1 -> BF = 0;

root = temp2;

root -> BF = 0.
```

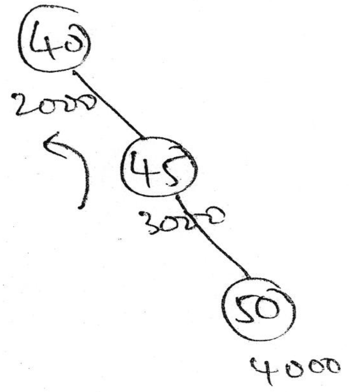
Example for RL Rotation :-

Ex 1 :-

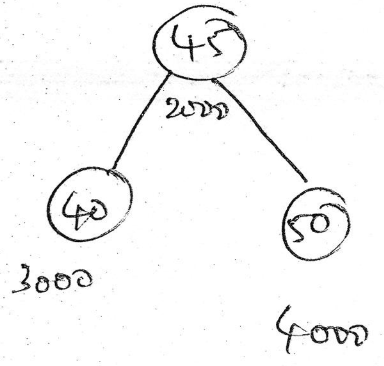


apply double Rotation.
(RL Rotation)

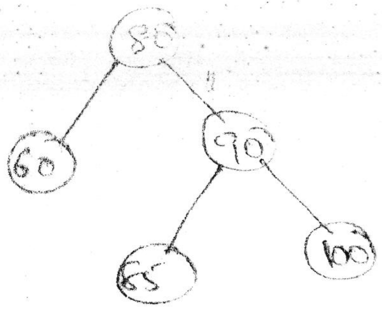
Step 1



Step 2



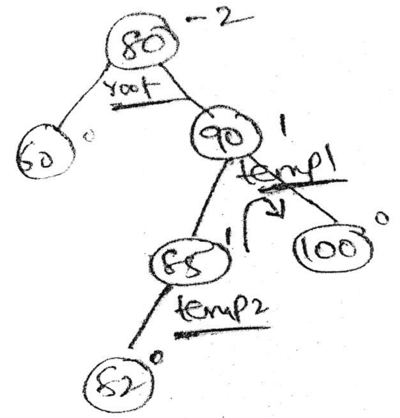
Ex 2



(a) insert 82

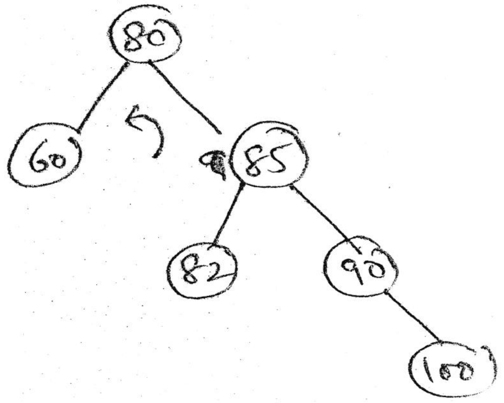
(b) insert 86

(a) insert 82

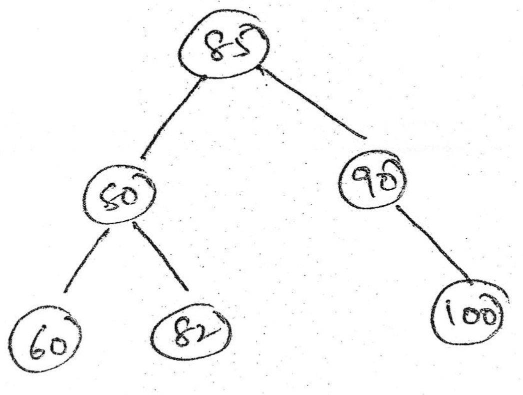


Apply RL Rotation (Rouble Rotation)

Step1 :-



Step2 :-



Code for RL Rotation

```
temp1 = root -> right

temp2 = temp1 -> left ;

temp1 -> left = temp2 -> right ;

temp2 -> right = temp1 ;

root -> right = temp2 -> left ;

temp2 -> left = root ;

if (temp2 -> BF == -1)
        root -> BF = 1 ;
else
        root -> BF = 0 ;

if ( temp2 -> BF == 1)
        temp1 -> BF = -1 ;
  else
        temp1 -> BF = 0 ;

root = temp2

root -> BF = 0 ;
```

Task 1 :-

Insert the following elements into the AVL Tree.

40, 30, 20, 60, 50, 80, 15, 28, 25.

Task 2 :-

Insert the following elements into the AVL Tree

A, V, L, T, R, E, I, S, O, K.

AVL Tree implementation

```
#include< stdio.h>
#include < stdlib.h>

struct  avlnode
{
   struct  avlnode  *left;
   int  data;
   int  bf;
   struct  avlnode  *right;

};

typedef   struct  avlnode  node;

node   * root  ;

node *    Insert ( node  *root, int data, Int *Curren

{
 node  *temp1, *temp2;

 if ( root   == NULL)
   {
    root =  (node *) malloc( sizeof(node));

    root -> data = data;
    root -> left = NULL;
    root -> right =NULL;
    root -> bf  = 0;

    *Current =1;
     return   root;
   }
```

```
if ( data > root -> data)
{
 root -> rigut = insert (root -> right, data, current);
 if (* Current == 1)
 {
   Switch (root -> bf)
   {
 Case -1 : temp1 = root -> rigut;
           if ( temp1 -> bf == -1)
           {
            Printf("\n Single Rotation : RR\n");
            root -> rigut = temp1 -> left;
            temp1 -> left = root;
            root -> bf = 0;
            root = temp1;
            }
          else
            {
             Printf ("\n double Rotation RL\n");
             temp2 = temp1 -> left;
             temp1 -> left = temp2 -> rigut;
             temp2 -> rigut = temp1;
             root -> rigut = temp2 -> left;
             temp2 -> left = root;

             if ( temp2 -> bf == -1)
                   root -> bf = 1;
             else
                  root -> bf = 0;
if ( temp2 -> bf ==1)
      root -> bf = -1;
else
      root -> bf =0;

 if ( temp2 -> bf == -1)
       temp1 -> bf =1;
 else
      temp1 -> bf = 0;.
 root = temp2;
 }
root -> bf = 0;
* Current = 0;
break;
Case  0  :  root -> bf =1;
            break;


 Case  -1 :  root -> bf = 0;
             * Current = 0;


   }

   }

   }
```

```
if ( data > root -> data)
{
 root -> right = insert (root -> right, data, current);
 if (* current == 1)
 {
  switch ( root -> bf)
  {
 case  -1 : temp1 = root -> right;
           if ( temp1 -> bf == -1)
           {
            printf ("\n Single Rotation : RR\n");
           root -> right = temp1 -> left;
           temp1 -> left = root;
           root -> bf = 0;
           root = temp1;
            }
          else
           {
            printf ("\n double Rotation RL\n");
            temp2 = temp1 -> left;
            temp1 -> left = temp2 -> right;
            temp2 -> right = temp1;
            root -> right = temp2 -> left;
            temp2 -> left = root;

            if ( temp2 -> bf == -1)
                 root -> bf = 1;
            else
                 root -> bf = 0;
```

```
      if (temp2 -> bf == -1)
            temp1 -> bf = -1;
      else
            temp1 -> bf = 0;

      root = temp2;

      }
     root -> bf = 0;
     * Current = 0;
      break;

     Case  0 :   root -> bf = -1;
                 break;


     Case  1 :   root -> bf = 0;
                * Current = 0;


      }
      }
    }
   return  root;
   }


  void  display ( node  * temp)
  {
        if ( temp != NULL)
          {
             display (temp -> left);
             Printf ("\n %d", temp -> data);
             display (temp -> right);
          }
  }
```

```
Void  main( )
{
 int  Current=1; int  i,x ;

 root = NULL;


 for ( i=1 ;  i<=15 ; i++)
{
  Printf ("Enter data to insert ");
 scanf (" %d ', & x);

 root = insert (root, x , & Current);

}

display ( root);

}.
```

AVL Tree Code Tracing

Insert the following elements into the AVL Tree & Display in Inorder.

40, 50, 30, 60, 70, 45.

(i) insert 40

```
main()                                  c [ 1 ]
{                                        5555(Address)
  root = NULL;

  c = 1;

  root = insert(NULL, 40, 5555); ——> goto (a)
                               ↑
                               └————————— (b)

  // root becomes 2000. after the Call

}
```

(a) ——> insert(NULL, 40, 5555)

```
root = NULL

=> root = 2000 ( Create a new node
                 Let the address allocated is 2000)


2000 -> data =10
2000 -> left = NULL                     |N|40|0|N|
2000 -> right = NULL.                      2000
2000 -> BF = 0.

c=1

return  2000  ——> (b)
```

(ii) insert 50

```
main( )
{
2000 = insert (2000, 50, 5555);  → (a)
}
```

(a) → insert (2000, 50, 5555)

(50 > 40)   data > root → data

root → right = insert (root → right, data, c)

2000 → right = insert (NULL, 50, 5555) → (b)

(c) →

2000 → right = 3000

2000 → bf = 0 ⇒ Case 0 ∴ 2000 → bf = -1
break.

retur 2000 → (d)

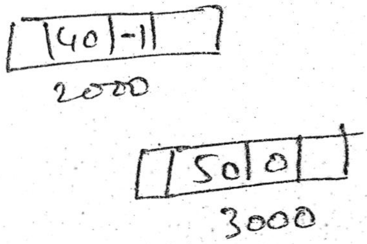
(b) → insert (NULL, 50, 5555)

root = NULL

Create a newnode with addr '3000'

store data as 50, bf = 0, c = 1

return 3000 → (c)



(iii) Insert 30

```
main()
{
2000 = insert (2000, 30, 5555);  ⟶ (a)
            ↑_______________________ (d)
}
```

(a) → insert (2000, 30, 5555)

30 < 40   data < root → data

2000 → left = insert (2000 → left, 30, 5555) ⟶ (b)

⟵ (c)

2000 → left = 4000

c = 1

2000 → bf = 1 - 1, ⇒ 2000 → bf = 0.

return 2000 ⟶ (d)

(b) → Insert (NULL, 30, 5555)

root = NULL

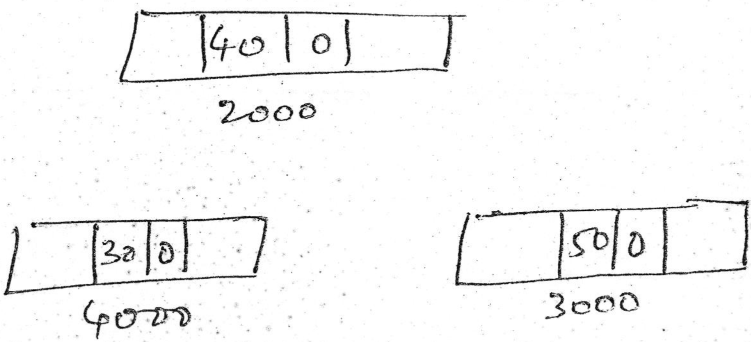
new node is Created with addr 4000

30 stored as data

bf = 0

c = 1

return 4000 ⟶ (c)



(iv) insert 60

main( )
{
2000 = insert (2000, 60, 5555); → (a)
↑ ← (f)
}

(a) → insert (2000, 60, 5555)

(60 > 40) data > root → data

2000 → right = insert (3000, 60, 5555) → (b)
← (e)

2000 → right = 3000

c = 1, 2000 → bf = 0

Case 0 ⇒ 2000 → bf = -1,

return 2000 → (f)

(b) → insert (3000, 60, 5555)

(60 > 50) data > root → data

3000 → right = insert (3000 → right, 60, 5555) → (c)
← (d)

3000 → right = 5000

c = 1, 3000 → bf = 0.

Case 0 ⇒ 3000 → bf = -1

return 3000 → (e)

(c) → insert (NULL, 60, 5555)

root = NULL

Creates a newnode with addr 5000

Put data as 60, bf = 0, c = 1

return 5000 → b (d)

4000	40	-1	3000

2000

N	50	-1	6000

3000

N	30	0	N

4000

N	60	0	N

6000

(V). insert 70

main( )

{

2000 = insert (2000, 70, 5555) → (a)

↑ (h)

}

(a) → insert (2000, 70, 5555)

data > root → data

2000 → right = insert (3000, 70, 5555) → (b)

← (g)

2000 → right = 5000

c = 0, return 2000 → (h)

(b) → insert (3000, 70, 5555)

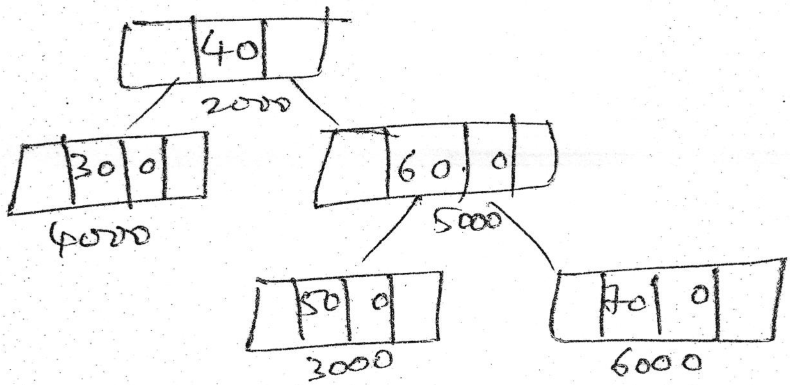
data > root → data.

3000 → right = insert (5000, 70, 5555) → (c)

← (f)

3000 → right = 5000, c = 1, 3000 → bf = -1

Case -1 ⇒ Right Rotation.



3000 → bf = 0

5000 → bf = 0 & root = 5000

return 5000 → (g).

(c) → insert (5000, 10, 5555)

data > root → data (70 > 60)

5000 → right = insest (NULL, 70, 5555) → (d)

← (e)

5000 → right = 6000

c = 1

5000 → bf = 0

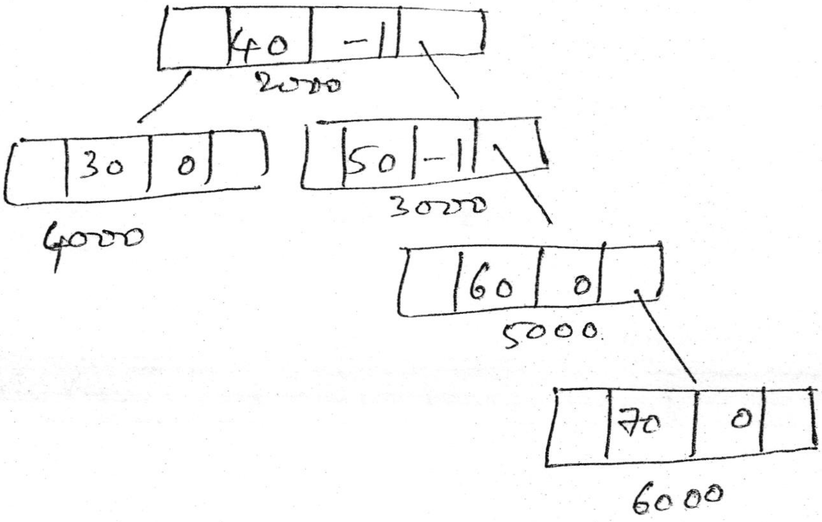
Case 0 ⇒ 5000 → bf = -1

return 5000 → (f)

(d) → <u>Insest (NULL, 70, 5555)</u>

root = NULL, Create new node with address 6000

data as 70, bf = 0, c = 1.



return 6000 → (e)

(vi) insert 45

```
main()
{
 2000 = Insert(2000, 45, 5555);  ⟶ (a)
}              ↑ ―――――――――――――――― (h)
```

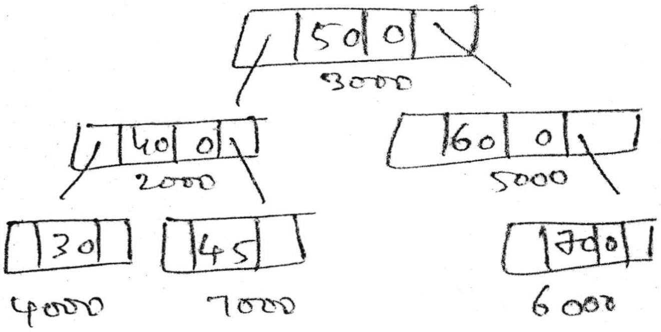
(a) Insert (2000, 45, 5555)

(45 > 40) data > root → data

2000 → right = Insert(5000, 45, 5555) → (b)

← (g)

c = 1, 2000 → bf = -1 ⇒ Case -1 (RL Rotation)



3000 → bf = 1 ⇒ 2000 → bf = 0, 5000 → bf = -1

root = temp2 = 3000, ⇒ 3000 → bf = 0.

c = 0., return 3000 ⟶ (h)

(b) → insert (5000, 45, 5555)

45 < 60

5000 → left = insert (3000, 45, 5555) → (c)

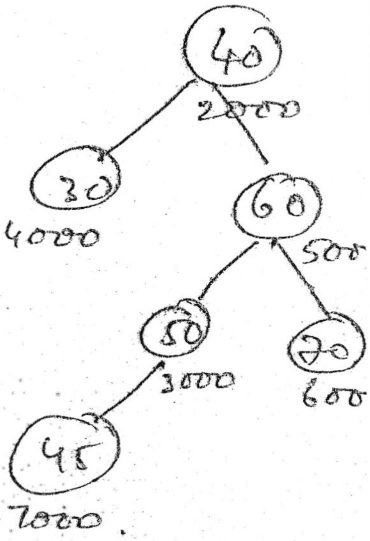
← (f)

5000 → left = 3000

c = 1, 5000 → bf = 0..

⇒ 5000 → bf = 1

return 5000 ⟶ (g)

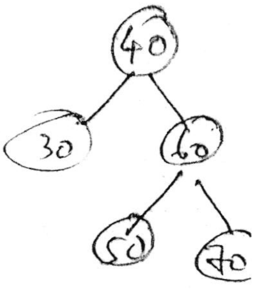


(c) → Insert (2000, 45, 5555)

45 < 50.

3000 → left = insert (3000 → left, 45, 5555) ——→ (d)

← (e)



3000 → left = 7000

c = 1

3000 → bf = 0 ⇒ 3000 → bf = 1

return 3000 → (f)

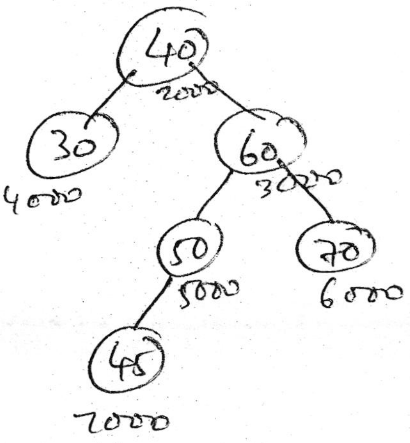
(d) → insert (NULL, 45, 5555)

root = NULL

Create a new node & let the addr is

7000.

data as 45, bf = 0, c = 1



return 7000 ——→ (e)

B-Tree

A B-tree of order m, is an m-way Search tree with the following Properties

(i) Root must have atleast two children

(ii) All the leaf nodes must be on the bottom level.

(iii) All the ~~leaf nod~~ Internal nodes except leaf nodes must have atleast Ceil (m/2) non empty Children.

(iv) if the node has 'n' children, then it must have n-1 keys.

Task 1 :-

Insert the following values into the B-tree
3, 14, 7, 1, 8, 5, 11, 17, 13, 6, 23, 12, 20, 26, 4, 16, 18, 24, 25, 19.

Solutions :-

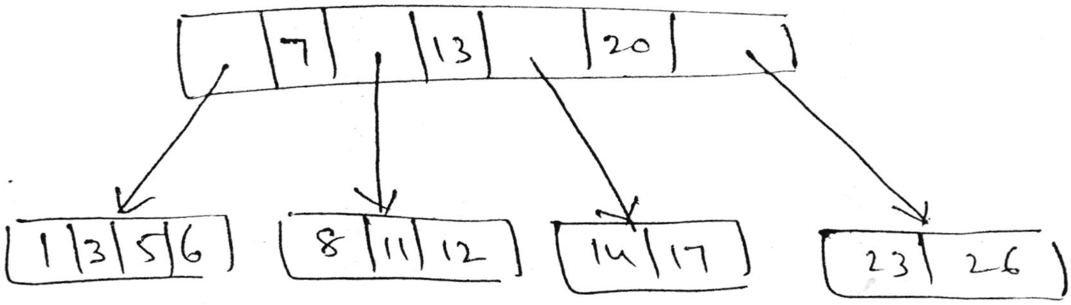
inserting 3 | | 3 | |

inserting 14 | | 3 | 14 | |

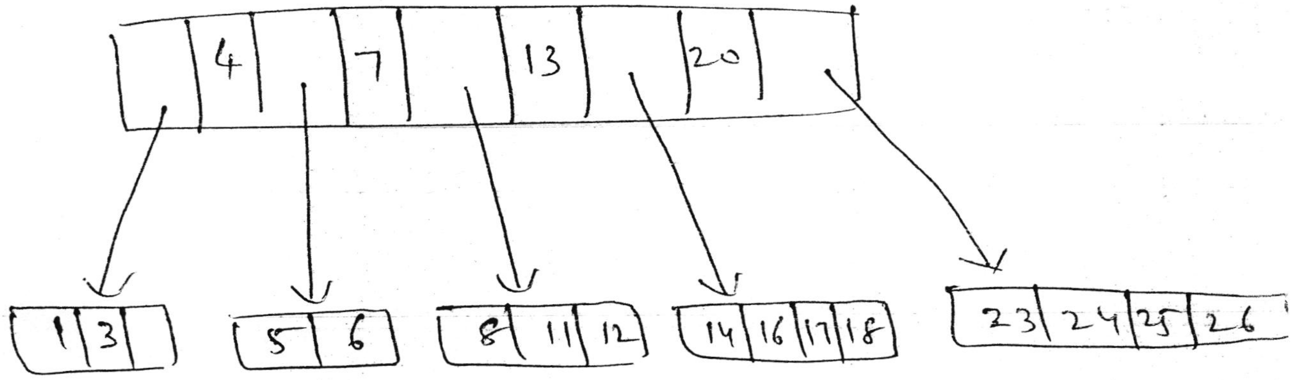
inserting 7 | | 3 | 7 | 14 | |

inserting 1 | | 1 | 3 | 7 | 14 | |

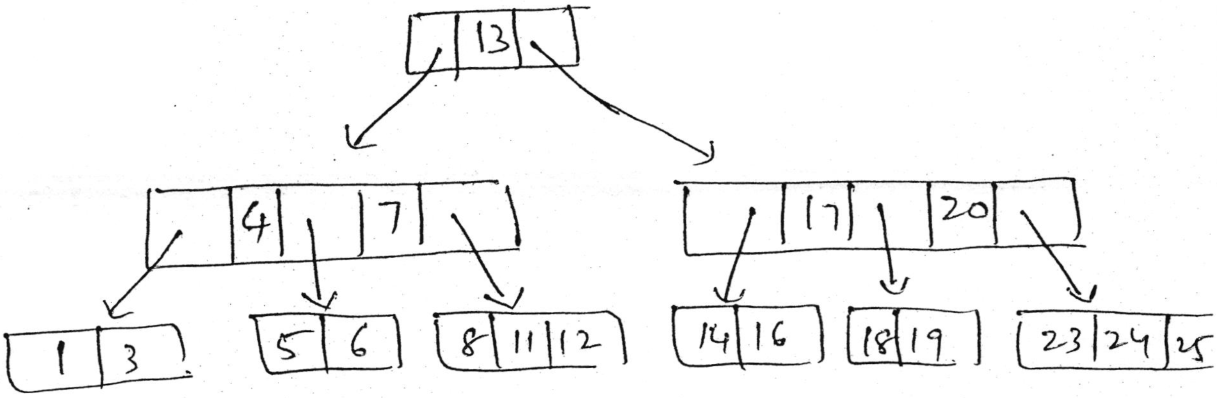
Insert 26



Insert 4



Insert 19



Implementation of B tree

```
# include <stdio.h>
# include <stdlib.h>

# define   MAX   4
# define   MIN   2

struct  treenode
{
   int  Count ;
   int  keys [MAX +1] ;
   struct  treenode   * links [MAX +1] ;

} ;


typedef  struct  treenode  node ;


int  search (int, node *, int *) ;

void  insertin ( int, node * , node * , int) ;

node *   insert (int, node *) ;

int   movedown ( int, node * , int * , node ** ) ;

void   split ( int, node *,  node *, int,
            int *, node ** ) ;

void   inorder ( node * ) ;
```

```
int search (int key, node *current, int *pos)
{
    if (key < current -> keys [1])
      {
        *pos=0;
        return 0;

      }
    else {

    for (*pos= current -> count;
        key < current -> keys[*pos] && *pos >1;
                (*pos) -- );

    if (key  ==  current -> keys[*pos])
            return 1;
    else
            return 0;

        }
    }
------------------------------------------------------------
void insertin(int med, node *medright, node *Curren
                                        int pos)
{   int i;

    for( i= current -> count ;  i > pos ;  i--)
    {
  current -> keys [i+1] = current -> keys [i];
  current -> links [i+1] = current -> links [i];
    }
  current -> keys [pos+1] =med;
  current -> links [pos+1] = medright;

  current -> count ++;

}
```

```
node *  insert (int  x, node  *temp)
{
  int  medentry;  node  *medright, *newnode;
  if (movedown (x, temp, &medentry, &medright))
  {
  newnode = (node *) malloc(sizeof(struct treenode));
  newnode -> count = 1;
  newnode -> keys [1] = medentry;
  newnode -> links [0] = temp;
  newnode -> links [1] = medright;
  return  newnode;
   }
  return  temp;
  }
```

```
int   movedown (int  x,  node  *current, int  *med,
                                  node   ** medright)
{
   int  pos;
   if ( current == NULL)
     {
        *med = x ;
        * medright = NULL;
        return  1 ;
     }
   else
     {
        if (search ( x, current, &pos) )
           printf (" duplicate  key ");
   if (movedown(x, current->links[pos], med, medright))
   {
  if (current -> count  <  MAX)
   {
     insertin(*med, *medright, current, pos);
     return   0 ;
   }
  else {
     split (*med, *medright, current, pos, med, medright);
     return  1 ;
     } }  return  0 ; }}
```

```
void split(int med, node *medright, node *current,
          int pos, int *newmedian, node **newright)

{
  int i;
  int median;

  if(pos <= MIN)
     median = MIN;
  else
     median = MIN+1;

 *newright = (node *) malloc(sizeof(struct treenode));


 for(i = median +1 ; k<= MAX ; l++)
 {
(*newright) -> keys[i-median] = current -> keys[i];
(*newright) -> links[i-median] = current -> links[i];
 }


(*newright) -> Count = MAX - median;

 current -> Count = median;


 if (Pos <= MIN)
    insertIn(med, medright, current, pos);
 else
    insert in( med, medright, *newright, pos-median);

*newmedian = Current -> keys[current -> Count];

(*newright) -> links[0] = current -> links [current -> Count];

current -> Count --;

 }
```

```
void inorder (node *temp)
{
  int pos ;
  if (temp)
  {
    inorder ( temp->links[0]);
    for( pos=1 ; pos <= temp->count ; pos++)
    {
     pf( "%d", temp->keys [ pos] );
     inorder(temp->links [ pos] );
    }
  }
}


void main( )
{
 node *root; int i,x ;
 root = NULL;

 for(i=1; i<=20 ; i++)
 {
pf("Enter the data to insert ");
sf("%d", & x);
root = insert( x, root);
 }
 inorder( root);
 }
```

(i) insert 3

```
main()
{    node  *root = NULL;
root = insert(3, NULL)  ⟶ ⓐ
                        ⟵ ⓓ
}
```

ⓐ ⟶ insert(3, NULL)

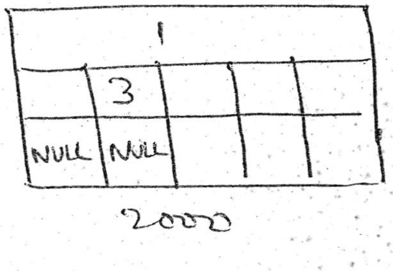
```
if (movedown(3, NULL, &me, &mr))  ⟶ ⓑ
        ⟵ ⓒ
```

Creates new node, let the address = 2000

```
newnode -> ~~entry~~ keys [1] = me = 3
newnode -> link [0] = root = NULL
newnode -> link [1] = mr = NULL
return 2000 ⟶ ⓓ
```

ⓑ ⟶ movedown(3, NULL, &me, &mr)

```
cur = NULL
me = x
mr = NULL
return 1 ⟶ ⓒ
```



(ii) insert

main()
{
root = insert (14, 2000) → (a)
↑ ← (h)
}

(a) → insert (14, 2000)

if (movedown(14, 2000, &me, &mr)) → (b)

← (g)

return 2000 → (h)

(b) → movedown(14, 2000, &me, &mr)

CUR != NULL

Search~~node~~ (14, 2000, &pos) ⇒ pos = 1

if (move down (14, 2000 → link[1]; me, mr) ; → (c)

← (d)

2000 → Count < MAX then

insertIn(14, NULL, 2000, 1) → (e)

← (f)

return 0 → (g)

(c) → movedown (14, NULL, me, mr)

CUR == NULL me = 14 mr = NULL return 1 → (d)

(e) → insert in (14, NULL, 2000, 1)

places 14 as 2nd key & link [2] = NULL, Count ++

2				
	3	14		
N	N	N		

return ; → (f)

(iii) insert 7

```
main()
{
root = insert (7, 2000) ————→ (a)
         ↑ ———————————————— (h)
}
```

~~insert~~ ((a) → <u>insert (7, 2000)</u>

if (movedown(7, 2000, &me, &mr)) ————→ (b)
←—— (g)

return 2000 ——→ (h)

(b) → <u>movedown(7, 2000, &me, &mr)</u>

CUR != NULL   search(7, 2000, &pos) ⇒ pos = 1

if (movedown(7, 2000 → link [1], me, mr)) ——→ (c)
←— (d)

2000 → Count < MAX Then
insert in(7, N, 2000, 1) ——→ (e)
←— (f)
return 0 ——→ (g)

(c) → <u>movedown(7, NULL, me, mr)</u>

CUR = NULL , me = 7, mr = NULL , return 1 ——→ (d)

(e) → <u>insertin(7, NULL, 2000, 1)</u>

move 2000 → link [2] , 2000 → ~~Keys~~[2] to Right

place 2000 → ~~Keys~~[2] = 7, 2000 → link [2] = NULL.

3				
	3	7	14	
N	N	N	N	

return ; ——→ (f)

<u>(iv) insert 1</u>

o/p

4				
	1	3	7	14
N	N	N	N	N

(iv) Insert 8

```
main()
{
  root = insert (8, 2000) → (a)
         ←                  (j)
}
```

(a) → insert (8, 2000)

if (movedown(8, 2000, &me, &mr)) → (b)
← (i)

Creates new node, let the address is 4000, Count = 1

4000 → Keys[1] = 7, 4000 → links[0] = 2000,

4000 → links[1] = mr = 3000, return 4000 → (j)

(b) → movedown(8, 2000, &me, &mr)

CUR != NULL, Search (8, 2000, &Pos) ⇒ pos = 3

if (movedown(8, 2000 → link[3], me, mr) → (c)
← (d)

2000 → Count < MAX False split(8, NULL, 2000, 3, me, mr) → (e)
← (h)

return 1 → (i)

(c) → movedown(8, NULL, me, mr)

CUR = NULL, me = 8, mr = NULL, return 1 → (d)

(e) → split(8, NULL, 2000, 3, me, mr)

Pos <= MIN, False ⇒ median = 3

newright = new node (let the addr is 3000)

move entries after median to newright (ie 3000)

newright → Count = 1

2000 → Count = 3

Pos <= MIN False

insertin (8, NULL, 3000, Pos-median) → (f)

(g)

newmedian = curr → entry [curr → count]

3000 → link [0] = 2000 → link [3] = NULL

2000 → count -- ; 2000 → Count = 2

updations :- me = 7

mr = 3000

return ; ⟶ (h)

insertin(8, NULL, 3000, 0)

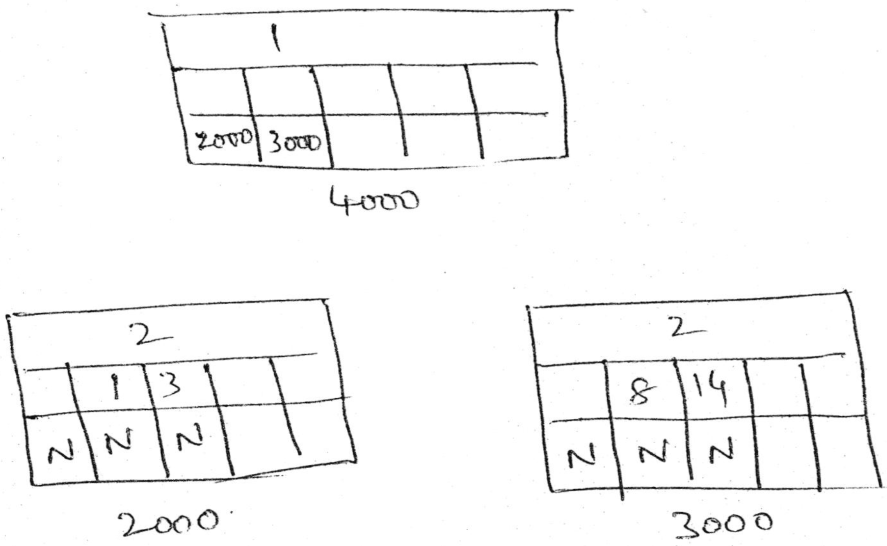
move 3000 → keys[1]

3000 → link[1] to right

&

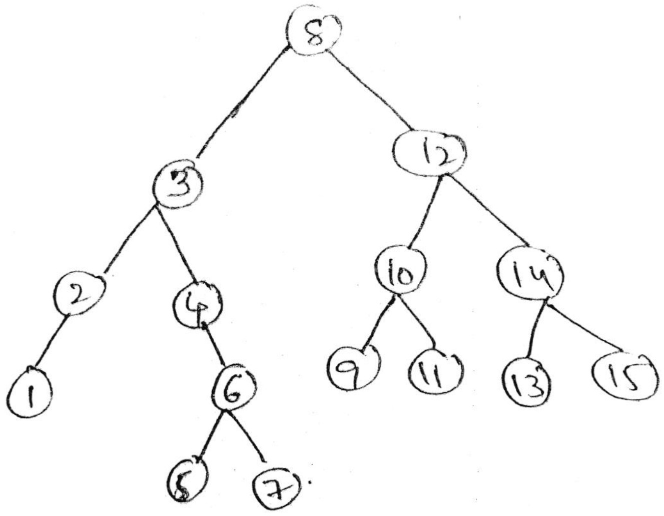
Store 3000 → keys[1] = 8

3000 → link[1] = NULL.

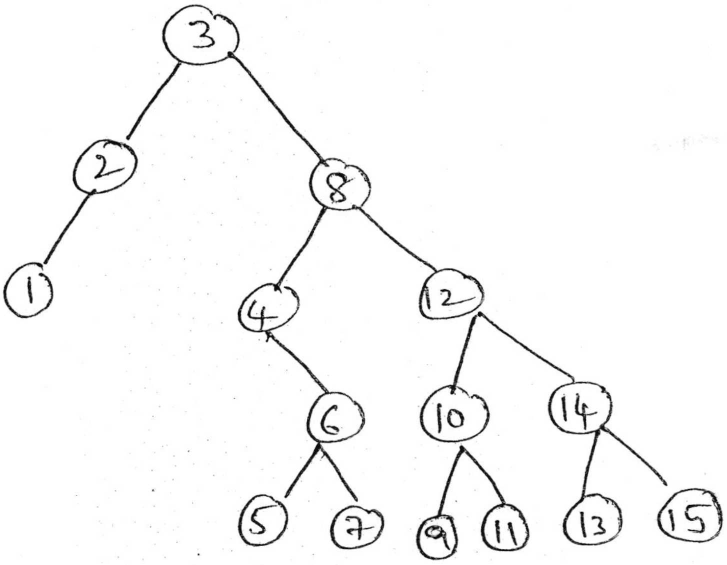


return ; ⟶ (g).

node 3, parent, grand parent involved in splay operation indicate zig-zag (LR rotation)



To bring '3' to Root, we need Zig (L Rota n)


3
2
8
1
4
12
6
10
14
5
7
9
11
13
15