Kattinisetty Venkata Sravya
IBM18CSOH11.

K.V. Sravya

Insertion on AVL tree

here n is not node

```
Node* insertion(Node* n, int val)
{
    if (n == NULL)
      return getnode(val);
    if (val < n->value)
        n->left = insertion(n->right, val);
    else
      return n
  update the height of the parent node and check for
 the balance condition.
  int b = get the balance of 'n'.
 if the node in unbalanced then,
    if (b > 1 && val < n->left->value)
        return Rotate_right(n);          // right rotation
    if (b < -1 && val > n->right->value)
        return Rotate_left(n);           // left rotation
    if (b > 1 && val > n->left->value)
        return Rotate_right(n);          // left-right rotation
    if (b < -1 && val < n->right->value)
        return Rotate_left(n);           // right-left rotation
  return n;
}
```

Kattinisetty Venkata Sravya

1BM18CS041

K. V. Sravya

here n is root

```
Node* delete (Node* n, int val)
{   if (n == NULL)
      return n;
    if (val < n->value)
      n->left = delete (n->left, val);
    else
      if (val > n->val)
        n->right = delete (n->right, val);
      else
        { // one child
          if (n->left == NULL || n->right == NULL)
            { Node* p = n->left ? n->left : n->right;
              if (p == NULL)
                {
                  p = n;
                  n = NULL;
                }
              else
                *n = *p;
              free (p);
            }
          else
            { // two children
              Node *p = smallest in the right subtree;
              n->val = p->val;
              n->right = delete (n->right, p->val);
            }
        }
```

update height and get balance.

int b = get the balance of 'n'.

Kattimsetty Venkata Sravya
IBM18CS0111

K.V. Sravya

```
if (b>1 && balance of n->left <0)
    {
        n->left = Rotate_left(n->left);
        return Rotate_Right(n);          // left-right
    }

if (b<-1 && balance of n->right>0)
    {
      n->right = Rotate_right(n->right);
      return Rotate_left(n);             // right_left
    }
if (b>1 && balance of n->left >=0)
    {
      return Rotat_right(n);             // left
    }

if (b<-1 && balance of n->right <=0)
    {
    return Rotate_left(n);               // right
    }

return n;
}
```