PSUEDO CODE:-

```
##Structure Definition:
struct Node23
{
    int val1                                        // stores min. of rightc in twoNode and min. of
midc in threeNode
    int val2                                        // stores min. of rightc in threeNode,
unessential in twoNode
    int height                                      // Height of tree
    Node23* leftc                                   // left child
    Node23* rightc                                  // right child
    Node23* midc                                    // middle child
}

##Merge Function:
Merge (T1,T2)                       // returns T = T1 U T2 , T1 and T2 are the roots of the two trees
{
    Node23* T;

    if(T1->height <= T2->height)
        T = insertRoot(T2,T1,0)     // third argument tells information about the comparison of height of
the two trees
    else
        T = insertRoot(T1,T2,1)
}

#Insertion Functions:
insertRoot (N, V , flag)            // returns V inserted at the root of N, flag tells about the values
of elements
{
    if(flag==0)                     // for flag=0, All elements of V smaller than elements of N
    {
        if(N==nil)                                      // Checking for trivial cases first
            return V
        if(V==nil)
            return N

        if(isLeaf(N))
            T = twoNode(N->val1,V,N)
            T.height = N->height + 1
            return T

        Let P,Q,x = Insert(N,V,flag, N->height-V->height)    // For non-trivial cases

        if(Q==nil)                                      // Used when N is a two Node
            T = P
            T->height = N->height
            return T
        else                                            // Used when N is a three Node
            T = twoNode(x,P,Q)
            T->height = N->height + 1
            return T
    }

    else                                  // for flag=1, All elements of V greater than elements of N
    {
        if(N==nil)
            return V
        if(V==nil)
            return N

        Let P,Q,x = Insert(N,V,flag, N->height-V->height)    // For non-trivial cases

        if(Q==nil)                                      // Used when N is a two Node
            T = P
            T->height = N->height
```

```
            return T
        else                                                // Used when N is a three Node
            T = twoNode(x,P,Q)
            T->height = N->height + 1
            return T
    }
}

Insert (N,V,flag,gap)                           // Helper function of insertRoot, returns values for non-
trivial cases
{
    if(flag==0)
    {
        if(gap==0)
            return ( V, N, FindMin(N) )

        if(isTwoNode(N))
        {
            Let A,B,t = Insert(N->leftc,V,flag,gap-1)

            if(B==nil)
                return (twoNode(N->val1,A,N->rightc), nil, 0)
            else
                return ( threeNode(t ,N->val1 , A , B , N->rightc) , nil , 0 )
        }

        else                                                // case for threenode returns two twoNodes to
insertRoot
        {
            Let A,B,t = Insert(N->leftc,V,flag,gap-1)

            if(B==nil)
                return ( threeNode(N->val1,N->val2,A,N->midc,N->rightc) , nil ,0 )
            else
                return ( twoNode(t,A,B) , twoNode(N->val2,N->midc,N->rightc) , N->val1)
        }
    }

    else
    {
        if(gap==0)
            return ( N, V, FindMin(V) )

        if(isTwoNode(N))
        {
            Let A,B,t = Insert(N->rightc,V,flag,gap-1)

            if(B==nil)
                return (twoNode(N->val1,N->leftc,A), nil, 0)
            else
                return ( threeNode(N->val1, t , N->leftc , A , B) , nil , 0 )
        }

        else                                                // case for threenode returns two twoNodes to
insertRoot
        {
            Let A,B,t = Insert(N->rightc,V,flag,gap-1)

            if(B==nil)
                return ( threeNode(N->val1,N->val2,N->leftc,N->midc,A) , nil ,0 )
            else
                return ( twoNode(N->val1,N->leftc,N->midc) , twoNode(t,A,B) , N->val2)
        }
    }

}

##Node Creator/Helper Functions:
isLeaf(Node)                                        // checks if a given Node is a leaf
```

```
{
    if(Node->leftc==nil and Node->rightc==nil and Node->midc==nil)
        return True
    else
        return False
}

istwoNode(Node)                                          // checks if a given Node has two children
{
    if(Node->leftc!=nil and Node->rightc!=nil and Node->midc==nil)
        return True
    else
        return False
}

twoNode(x,P,Q)                                           // returns a twoNode formed using number x and
Nodes P,Q
{
    Node23* Root = new Node23
    Root->val1   = x
    Root->val2   = nil                                   // unessential in Leaf and twoNode
    Root->leftc  = P
    Root->rightc = Q
    Root->midc   = nil

    return Root
}

threeNode(x,y,P,Q,R)                                     // returns a threeNode formed using numbers x,y
and Nodes P,Q,R
{
    Node23* Root = new Node23
    Root->val1   = x
    Root->val2   = y                                     // unessential in twoNode
    Root->leftc  = P
    Root->midc   = Q
    Root->rightc = R

    return Root
}

FindMin(Node)                                            // Finds minimum of subtree rooted at argument
Node
{
    if(isleaf(Node))
        return Node->val1
    else
        return FindMin(Node->leftc)                      // Minimum for a 2-3 tree occurs in the left
subtree
}
```