Algorithm 1: Splay Function

```
Input: root: node, key: int
Output: The root node after splaying
Function splay(root, key):
    if root is null or root's key is equal to the given key then
        return root;
    key_value ← key;
    if root's key is greater than key_value then
        if root has no left child then
            return root;
        if root's left child's key is greater than key_value then
            root→left→left ← splay(root→left→left, key);
            root ← rightRotate(root);
        else if root's left child's key is less than key_value then
            root→left→right ← splay(root→left→right, key);
            if root's left right child exists then
                root→left ← leftRotate(root→left);
        return (root→left exists)? rightRotate(root):root
        else
            if root has no right child then
                return root;
            if root's right child's key is greater than key_value then
                root→right→left ← splay(root→right→left, key);
                if root's right left child exists then
                    root→right ← rightRotate(root→right);
            else if root's right child's key is less than key_value then
                root→right→right ← splay(root→right→right, key);
                root ← leftRotate(root);
            return (root→right exists)? leftRotate(root):root
```

Algorithm 2: Insert Function

```
Data: root: node, key: int
Result: Root node after insertion and splay operation
Function insert(root, key):
    parent ← NULL;
    temp ← root;
    while temp is not NULL do
        parent ← temp;
        if temp→key < key then
            temp ← temp→right;
        else if temp→key < key then
            temp ← temp→left;
        else
            return splay(root, key);
    insertNode ← createnode(key);
    if parent is NULL then
        root ← insertNode;
    else if parent→key < key then
        parent→right ← insertNode;
    else
        parent→left ← insertNode;
    return splay(root, key);
```

Algorithm 3: Deletenode Function

```
Data: root: node, key: int
Result: Root node after deletion and splay operation
Function deletenode(root, key):
    temp ← find(root, key);
    if temp is NULL then
        return root;
    if temp→left is NULL then
        root ← temp→right;
        free(temp);
    else if temp→right is NULL then
        root ← temp→left;
        free(temp);
    else
        maxleft ← findMaxleft(temp);
        maxleft→right ← temp→right;
        maxleft→left ← temp→left;
        root ← maxleft;
        free(temp);
    return root;
```

Algorithm 4: FindMaxleft Function

```
Data: root: node
Result: Node with maximum left value
Function findMaxleft(root):
    temp ← root→left;
    P ← NULL;
    while temp→right do
        P ← temp;
        temp ← temp→right;
    if P is NULL then
        root→left ← temp→left;
        return temp;
    P→right ← temp→left;
    return temp;
```

Algorithm 5: Find Function

```
Data: root: node, key: int
Result: Root node after splay operation
Function find(root, key):
    temp ← root;
    while temp is not NULL do
        if temp→key < key then
            temp ← temp→right;
        else if temp→key > key then
            temp ← temp→left;
        else
            ;
    if temp is NULL then
        Output: Not found;
        return root;
    return splay(root, key);
```