Problem Statement:

A Dictionary stores keywords & its meanings. Provide facility for adding new keywords, deleting keywords, updating values of any entry. Provide facility to display whole data sorted in ascending/ Descending order. Also find how many maximum comparisons may require for finding any keyword. Use Height balance tree and find the complexity for finding a keyword

```
#include <iostream>
using namespace std;

struct AVLnode {
   public:
   int cWord;    string
cMean;    AVLnode
*left,*right;
   int iHt;
};

class AVLtree {
```

```
public:
   AVLnode *Root;
   AVLtree () {
      Root = NULL;
   }
   AVLnode* insert (AVLnode*, int, string);
   AVLnode* deletE (AVLnode*, int);
   AVLnode* LL (AVLnode*);
   AVLnode* RR (AVLnode*);
   AVLnode* LR (AVLnode*);
AVLnode* RL (AVLnode*);
int height (AVLnode*);     int
bFactor (AVLnode*);     void
inOrder (AVLnode*);     void
preOrder (AVLnode*);
};


AVLnode* AVLtree::insert (AVLnode *root, int nWord, string nMean) {
   if (root == NULL) {        root = new
AVLnode;        root -> left = root ->
right = NULL;        root -> cWord =
nWord;        root -> cMean = nMean;
      root -> iHt = 0;
   }

   else if (root -> cWord != nWord) {        if (root ->
cWord > nWord)           root -> left = insert (root ->
left, nWord, nMean);

      else
```

```
        root -> right = insert (root -> right, nWord, nMean);
    }

    else
        cout << "\nRedundant AVLnode\n";

    root -> iHt = max(height(root -> left), height(root -> right)) + 1;
    if (bFactor (root) == 2) {        if (root
-> left -> cWord > nWord)            root
= RR (root);
        else            root =
LR (root);
    }

    if (bFactor (root) == -2) {        if (root
-> right -> cWord > nWord)            root
= RL (root);        else            root = LL
(root);
    }

    return root;
}


AVLnode *AVLtree::deletE (AVLnode *curr, int x) {
AVLnode *temp;
    if (curr == NULL) {        cout <<
"\nWord not present!\n";
        return curr;
    }
```

```
   else if (x > curr -> cWord)        curr ->
right = deletE (curr -> right, x);

   else if (x < curr -> cWord)        curr ->
left = deletE (curr -> left, x); else if
(curr -> right == NULL || curr -> left ==
NULL) {
      curr = curr -> left ? curr -> left : curr -> right;
cout << "\nWord deleted Successfully!\n";
   }

   else {        temp = curr -> right;        while (temp ->
left)           temp = temp -> left;        curr -> cWord =
temp -> cWord;        curr -> right = deletE (curr -> right,
temp -> cWord);
   }

   if (curr == NULL) return curr;

   curr -> iHt = max(height(curr -> left), height(curr -> right)) + 1;

   if (bFactor (curr) == 2) {        if
(bFactor (curr -> left) >= 0)
curr = RR (curr);        else
curr = LR (curr);
   }

   if (bFactor (curr) == -2) {        if
(bFactor (curr -> right) <= 0)
```

```
curr = LL (curr);        else
curr = RL (curr);
   }
   return (curr);
}


int AVLtree::height (AVLnode* curr) {
   if (curr == NULL)
return -1;     else
return curr -> iHt;
}


int AVLtree::bFactor (AVLnode* curr) {
   int lh = 0, rh = 0;
if (curr == NULL)
      return 0;
else
      return height(curr -> left) - height(curr -> right);
}


AVLnode* AVLtree::RR (AVLnode* curr) {     AVLnode* temp = curr -
> left;     curr -> left = temp -> right;     temp -> right = curr;     curr ->
iHt = max(height(curr -> left), height(curr -> right)) + 1;     temp ->
iHt = max(height(temp -> left), height(temp -> right)) + 1;     return
temp;
}
```

```
AVLnode* AVLtree::LL (AVLnode* curr) {     AVLnode* temp = curr -
> right;     curr -> right = temp -> left;     temp -> left = curr;     curr ->
iHt = max(height(curr -> left), height(curr -> right)) + 1;     temp ->
iHt = max(height(temp -> left), height(temp -> right)) + 1;     return
temp;
}


AVLnode* AVLtree::RL (AVLnode* curr) {
   curr -> right = RR (curr -> right);
return LL (curr);
}


AVLnode* AVLtree::LR (AVLnode* curr) {
   curr -> left = LL (curr -> left);
return RR (curr);
}


void AVLtree::inOrder (AVLnode* curr) {
   if (curr != NULL) {        inOrder (curr -> left);        cout <<
"\n\t" << curr -> cWord << "\t" << curr -> cMean;
      inOrder (curr -> right);
   }
}
void AVLtree::preOrder (AVLnode* curr) {
   if (curr != NULL) {
      cout << "\n\t" << curr -> cWord << "\t" << curr -> cMean;
      preOrder (curr -> left);
preOrder (curr -> right);
```

```
    }
}



int main () {
    int ch;
    AVLtree avl;
    AVLnode *temp = NULL;
    int word;
string mean;

    cout << "\n---------------------------------------";
    cout << "\n\tAVL TREE IMPLEMENTATION";
    cout << "\n---------------------------------------";    do
{       cout << "\n\t\tMENU";       cout <<
"\n1.Insert 2.Inorder 3.Delete 4.Exit";
       cout << "\n--------------------------------";
cout << "\nEnter your choice: ";
       cin >> ch;

       switch (ch) {
          case 1:
             cout << "\nEnter Word: ";              cin >>
word;              cout << "\nEnter Meaning: ";
cin >> mean;              avl.Root = avl.insert (avl.Root,
word, mean);               break;
          case 2:
             cout << "\nInorder Traversal:\n\tWORD\tMEANING";
             avl.inOrder (avl.Root);              cout << "\n\nPreorder
Traversal:\n\tWORD\tMEANING";
```

```
            avl.preOrder (avl.Root);
cout << '\n';               break;
         case 3:
            cout << "\nEnter the word to be deleted : ";
            cin >> word;              avl.Root =
avl.deletE (avl.Root, word);
            break;
         case 4:
            exit (0);
      }
   } while (ch != 4);

   return 0;
}

/*
---------------------------- OUTPUT ----------------------------

--------------------------------------
      AVL TREE IMPLEMENTATION
--------------------------------------
            MENU
1.Insert 2.Inorder 3.Delete 4.Exit
-------------------------------
Enter your choice: 1

Enter Word: 1

Enter Meaning: a

            MENU
```

1.Insert 2.Inorder 3.Delete 4.Exit

Enter your choice: 1

Enter Word: 2

Enter Meaning: b

MENU

1.Insert 2.Inorder 3.Delete 4.Exit

Enter your choice: 1

Enter Word: 3

Enter Meaning: c

MENU

1.Insert 2.Inorder 3.Delete 4.Exit

Enter your choice: 2

Inorder Traversal:

	WORD	MEANING
1	a	
2	b	
3	c	

Preorder Traversal:

WORD	MEANING
2	b

```
    1    a
    3    c


          MENU
1.Insert 2.Inorder 3.Delete 4.Exit
--------------------------------
Enter your choice: 1

Enter Word: 4

Enter Meaning: d

          MENU
1.Insert 2.Inorder 3.Delete 4.Exit
--------------------------------
Enter your choice: 3

Enter the word to be deleted : 3

Word deleted Successfully!

          MENU
1.Insert 2.Inorder 3.Delete 4.Exit
--------------------------------
Enter your choice: 2

Inorder Traversal:
    WORD    MEANING
1    a
2    b
    4    d
```

Preorder Traversal:

WORD MEANING

2 b

1 a

4 d

MENU

1.Insert 2.Inorder 3.Delete 4.Exit

Enter your choice: 1

Enter Word: 2

Enter Meaning: x

Redundant AVLnode

MENU

1.Insert 2.Inorder 3.Delete 4.Exit

Enter your choice: 3

Enter the word to be deleted : 2 Word deleted Successfully!

MENU

1.Insert 2.Inorder 3.Delete 4.Exit

Enter your choice: 2

Inorder Traversal:

WORD MEANING

1 7

4 8

Preorder Traversal: 9

WORD MEANING 4 9

1 a

MENU

1.Insert 2.Inorder 3.Delete 4.Exit

Enter your choice: 4

*/