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

struct Node {
    string keyword;
    string meaning;
    Node* left;
    Node* right;
    Node(string k, string m) : keyword(k), meaning(m), left(NULL), right(NULL) {}
};

class BSTDictionary {
public:
    Node* root;
    BSTDictionary() : root(NULL) {}

    Node* insert(Node* root, string keyword, string meaning) {
        if (root == NULL) return new Node(keyword, meaning);
        if (keyword < root->keyword)
            root->left = insert(root->left, keyword, meaning);
        else if (keyword > root->keyword)
            root->right = insert(root->right, keyword, meaning);
        else
            root->meaning = meaning;  // Update meaning if keyword exists
        return root;
    }

    Node* minValueNode(Node* node) {
        Node* current = node;
        while (current && current->left != NULL)
            current = current->left;
        return current;
    }

    Node* deleteNode(Node* root, string keyword) {
        if (root == NULL) return root;
        if (keyword < root->keyword)
            root->left = deleteNode(root->left, keyword);
        else if (keyword > root->keyword)
            root->right = deleteNode(root->right, keyword);
        else {
            if (root->left == NULL) {
                Node* temp = root->right;
                delete root;
                return temp;
            } else if (root->right == NULL) {
                Node* temp = root->left;
                delete root;
                return temp;
            }
            Node* temp = minValueNode(root->right);
            root->keyword = temp->keyword;
            root->meaning = temp->meaning;
            root->right = deleteNode(root->right, temp->keyword);
        }
        return root;
    }

    void inorder(Node* root) {
        if (root != NULL) {
            inorder(root->left);
            cout << root->keyword << " : " << root->meaning << endl;
            inorder(root->right);
        }
    }

    void reverseInorder(Node* root) {
        if (root != NULL) {
            reverseInorder(root->right);
            cout << root->keyword << " : " << root->meaning << endl;
            reverseInorder(root->left);
        }
    }

    int searchComparisons(Node* root, string keyword) {
        int comparisons = 0;
        while (root != NULL) {
            comparisons++;
            if (keyword == root->keyword)
                return comparisons;
            else if (keyword < root->keyword)
                root = root->left;
            else
                root = root->right;
        }
        return -1;  // Return -1 if keyword is not found
    }
};

int main() {
    BSTDictionary dict;
    int choice;
    string keyword, meaning;

    do {
        cout << "\n1. Add keyword\n2. Delete keyword\n3. Update keyword\n4. Display (Ascending)\n5. Display (Descending)\n6. Search keyword comparisons\n7. Exit\nEnter choice: ";
        cin >> choice;
        switch(choice) {
            case 1:
                cout << "Enter keyword: "; cin >> keyword;
                cout << "Enter meaning: "; cin.ignore(); getline(cin, meaning);
                dict.root = dict.insert(dict.root, keyword, meaning);
                break;
            case 2:
                cout << "Enter keyword to delete: "; cin >> keyword;
                dict.root = dict.deleteNode(dict.root, keyword);
                break;
            case 3:
                cout << "Enter keyword to update: "; cin >> keyword;
                cout << "Enter new meaning: "; cin.ignore(); getline(cin, meaning);
                dict.root = dict.insert(dict.root, keyword, meaning);
                break;
            case 4:
                dict.inorder(dict.root);
                break;
            case 5:
                dict.reverseInorder(dict.root);
                break;
            case 6:
                cout << "Enter keyword to search: "; cin >> keyword;
                {
                    int comparisons = dict.searchComparisons(dict.root, keyword);
                    if (comparisons == -1)
                        cout << "Keyword not found!\n";
                    else
                        cout << "Comparisons required to search: " << comparisons << endl;
                }
                break;
            case 7:
                cout << "Exiting...\n";
                break;
            default:
                cout << "Invalid choice!\n";
        }
    } while(choice != 7);

    return 0;
}

```