Exercise 1: Code analysis

Question 1: In the Word class, when is it possible to assign the str attribute?

It is private, it can only be assigned (or modified) inside the class implementation, not directly from outside. We can use at construction, inside member functions of the class, by copy/move assignment or copy/move construction.

```
Word w("hello");  // str = "hello"
void set(const QString& newStr) { str = newStr; }

Word a("hi");

Word b("bye");

a = b;  // str of a is assigned from str of b
```

Question 2: In the Word class, which operators are overloaded?

These two are overloaded.

```
bool operator<(const QString &cmp);

bool operator==(const QString &cmp);
```

Question 3: If w is a pointer to Word, how do you evaluate whether the attribute str of w is less than “bridge” and not equal to “false”?

```
if ( (*w < "bridge") && !(*w == "false") ) {
   // do something }
```

Question 4: Study the BST::insert() function. How is the data structure constructed? Draw an example after inserting the elements {”apple”,”bear”,”cat”, ”dog”}.

The data is constructed in a linked-list manner.

```
APPLE
  \
   BEAR
```

```
\
 CAT
  \
   DOG.
```

Question 5: Study the BST::search() method and explain how the while loop works. What is the complexity of this algorithm?

The search() function begins at the current node and continues traversing down the left subtree, incrementing the counter n at each step. This represents the worst-case traversal scenario.

Question 6: If we consider that the words in the data file are sorted in ascending order, what optimization can be applied to this algorithm? What is the effect on complexity?

If the words in the file are already sorted in ascending order, inserting them one by one into a standard BST will create a right-skewed tree (like a linked list).

Optimization:

- Use a Balanced Binary Search Tree (e.g., AVL tree or Red-Black tree) or
- Insert words using a divide-and-conquer approach:
- Take the middle element as the root,
- Recursively insert left and right halves.

Effect on complexity:

- Without optimization: insertion/search → O(n) (skewed tree)
- With optimization (balanced tree): insertion/search → O(log n)

Exercise 2: Insertion and search method in a BST

Question 1: Create a method bool search(const QString &src,int &n) that returns if a node that constains src exists.

```
bool BST::search(const QString &src, int &n) {
    n++; // visiting this node
    if (nodeWord == nullptr) {
        return false;
    }


    if (*nodeWord == src) {
        return true;
    } else if (*nodeWord < src) {
        // go right
        return (right != nullptr) ? right->search(src, n) : false;
    } else {
        // go left
        return (left != nullptr) ? left->search(src, n) : false;
    }
}
```

Question 2: Theinsertion procedure (BST *insert(const QString &word)) is quite similar to the previous searching method. If the node is empty it fill it with word and return the node pointer. In general cases, it starts at the root and recursively goes down the tree searching for a location in a BST to insert a new node. If the element to be inserted is already in the tree, we are done (we do not insert duplicates, and the method returns the node pointer). Otherwise create a new node and insert it at the good branch.

```
BST* BST::insert(const QString &word) {
    if (nodeWord == nullptr) {
        nodeWord = new Word(word);
        return this;
    }


    if (*nodeWord == word) {
        // already exists, do nothing
        return this;
    } else if (*nodeWord < word) {
        // go right
        if (right == nullptr) {
            right = new BST(word, this, nullptr, nullptr);
        } else {
            right->insert(word);
        }
    } else {
        // go left
        if (left == nullptr) {
            left = new BST(word, this, nullptr, nullptr);
        } else {
            left->insert(word);
        }
    }
    return this;
}
```

Question 3: Insert a list of ordered elements, for example: {”apple”,”bear”,”cat”, ”dog, ”elephant”}. What is the shape of the generated BST? Is is efficient for the searching algorithm?

```
apple
    bear
        cat
            dog
                elephant
```

Exercise 3: Equilibrate the BST

Question 1: Write the method int BST::depth() that returns the depth of a branch.

```
int BST::depth() const {
    int leftDepth = left ? left->depth() : 0;
    int rightDepth = right ? right->depth() : 0;
    return 1 + std::max(leftDepth, rightDepth);
}
```

Question 2: Complete the previous method to compute the balance coefficient at each node and run the appropriate rotation if necessary.

```
" "bear" " (balance: -1 )
" "dog" " (balance: 0 )
" "elephant" " (balance: 0 )
" "cat" " (balance: 0 )
" "apple" " (balance: 0 )
```

Question 3: Insertalist of ordered elements to test your new code, for example: {”apple”,”bear”,”cat”, ”dog, ”elephant”}. What is the new shape of the generated BST?

Balanced BST: The top parent is bear then apple and dog are their child and dog have cat and elephant child, it is also same on terminal.

```
        apple
         /
  bear        cat
         \      /
           dog
               \ elephant
```

Exercise 4: Complexity

Question 1: How complex was the algorithm provided?

Before balancing, the insertion and search operations used a standard binary search tree (BST):

Search: Start at the root and follow left or right child pointers until the desired node is found.

Time complexity (unbalanced BST): In the worst case, when the tree is completely skewed (like a linked list), the search takes O(n) time.

Question 2: What is the maximum number of nodes you would need to examine to perform any search? Considering a complete tree, what are the probabilities of crossing 1,2,..H node?

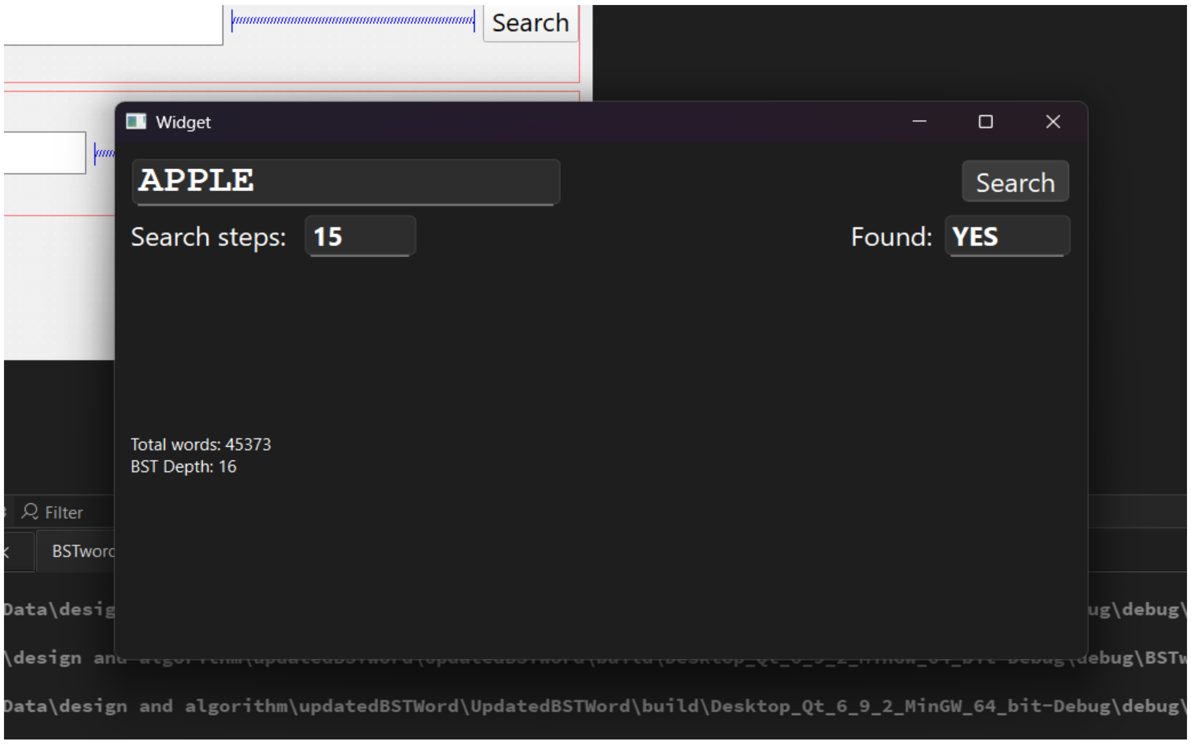
The maximum number of nodes that may need to be examined corresponds to the height of the BST. For a well-balanced tree, most searches will traverse significantly fewer nodes on average.

Question 3: Deduce the complexity of this new solution?

Balanced BST has complexity: O(log n)

Question 4: Considering the file contains 45373 different words, what is the theoretical size of the BST?

$H=\log_2(n) = \log_2(45373) = 15.46 \Longrightarrow H=16$



APPLE

Search steps: 15

Found: YES

Total words: 45373
BST Depth: 16

```
Data\design and algorithm\updatedBSTWord\UpdatedBSTWord\build\Desktop_Qt_6_9_2_MinGW_64_bit-Debug\debug\
```