**High‐level description of the overall solution strategy**

The C++ program initially reads a text file line by line, removing all punctuations and converting all words into lowercase and putting them into char array *pool* (string pool) since it avoids the overhead of dynamic memory and can access any string quickly and efficiently using the start and end indexes. Each of the word is then inserted into binary search tree (BST) using start and end indexes since it has the property to keep the values in sorted order.

A BST tree is created where each of the node contains a *struct* of int start variable, int last variable, int count variable and int left and int right child variables holding the index of their children (default value would be -1). Nodes will be inserted by doing comparison of words, making it the left child if the word comes before its parent node or else the right child. If the same word appears again count is incremented by 1. At the end of this, each of the node in this tree would have start and end indexes of the word from string pool and its total occurrences found in the file.

Now the above BST tree is sorted again, this time doing comparison done on its count value, making the new node a left child if is greater than its parent node, else it will become the right child. If in case the counts are equal, nodes are inserted by again comparing the word with the parent node. The new node becomes the left child if the word comes before its parent node or else the right child.

First ten and last ten sorted words by their decreasing counts are displayed using algorithm of BST inorder traversal.

**List of all of the data structures used, where they are used and the reasons for their choice.**

**Linear arrays**

Where used: -

* Storing characters after reading from the file.
* Implementation of BST using array.

Reasons: -

* Easy to store and access element in array.
* Both store and access takes constant time.
* Arrays take linear (O(n)) space in the number of elements n that they hold.

**Binary Search Tree**

Where used: -

* Searching for similar words in a file.
* Sorting words alphabetically and their occurrences.

Reasons: -

* Trees provide an efficient insertion and searching.
* Doing inorder traversal gives the sorted output.
* It has an average time complexity of O (n log n) in insertion and searching, if the tree is balanced and worst case complexity of O (n), if the tree is unbalanced

**List of standard algorithms used, where they are used and why they are used.**

Binary Search Tree: -

**Insertion: -** Used for storing words arranged alphabetically in a tree. Once the whole tree is constructed, it is again arranged based on its occurrences. If the occurrences are same, words are compared and again stored alphabetically. If the tree is balanced, it has an O (n log n) complexity in searching and insertion, which is relativity faster. Also the words in the file won’t be alphabetically ordered, so the binary search tree will be sensibly balanced.

**Inorder Traversal:** - This algorithm was required to display top and last ten words and their counts. The inorder traversal of a binary search tree gives sorted output.