|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  | | --- | --- | --- | | **Project:** | CMP3005 – Analysis of Algorithms Term Project **Plagiarism Detector** | | | **Date:** | 13.01.2021 | | | **Students:** | **Name:** | **Student ID:** | | [REDACTED FOR PRIVACY] | [RFP] | | Ali Yağız İlban | 1731384 | | [REDACTED FOR PRIVACY] | [RFP] | |

[Algorithm/Data Structure 1](#_Toc7661)

[Big-O Complexity 1](#_Toc7662)

[Running time 1](#_Toc7663)

[Code Details 2](#_Toc7664)

[Header.h 2](#_Toc7665)

[Source.cpp 9](#_Toc7666)

[Example Output 10](#_Toc7667)

There are two source files in our project, these files are Header (Header.h) and Source (Source.cpp) files.

# Algorithm/Data Structure

We used in our Project 2-Dimensional Vector and Linear Search Algorithm.

# Big-O Complexity

O(n(n^2(n) + n^2 + n(n)))

* O(n^3(n) + n^2 + n^2)
* O(n^4 + n^2 + n^2)
* O(n^4) Worst-Case for all Char.

# Running time

In **debugger mode** 1 document is determined as the main document and the time for searching in 2 different documents is determined as **65 ms.**

On the **Release mode** this process made in **17 ms.**

The original document contains the first paragraph of Lorem Ipsum, the second paragraph of Lorem Ipsum in the first document to be compared, and the first paragraph of Lorem Ipsum in the second document to be compared.

# Code Details

## Header.h

This part is the method of separating sentences into words. The purpose of this method is to find sentences with ''istingstream iss (a [i]); ''. Then it separates the sentences into words and throws these words into the ''iss''.The vector string b holds the words we separated.

So (istringstream iss (a [i]);) first splits the 1st sentence of file one into words, and this method continues sequentially until the "iss" is done.

There is a for loop outside of the function, it increments the index each time it puts it into the function (line 132). It takes the sentence as iss (a [i]). If the sentence exists, it falls into a while loop.

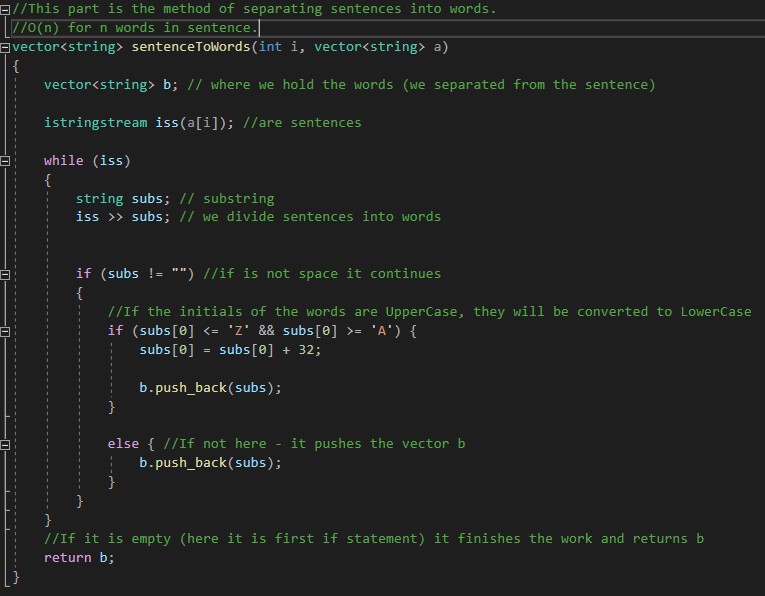
Is inside iss >> subs; are separated as subs, words are automatically assigned to subs.

If it does not detect the word as null, it puts it in a vector and stores it there. It will be repeated until the sentence is over.

We have a sentence divided into words. The word vector is returned, used by the variable that calls the function.

-> See: (vector <string> GetWord

This situation continues until the end and returns the vector string b. This situation continues until the end and returns the vector string b.

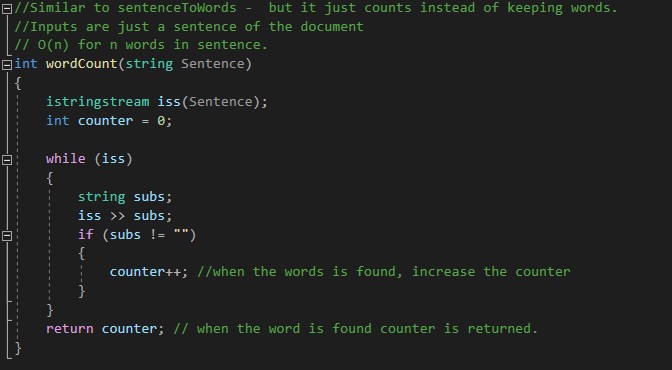


Here we have a function WordCount, and with this function, we send the sentence to

WordCount (Sentence). For example, we are sending the 1st sentence of the 2nd document,

WordCount and sentenceToWords are similar functions, but the purpose of sentenceToWords is to split sentences into words and returns the words.

WordCount is again dividing it into words 'but' counts the number of words and if the word is found it returns the number (return counter). counter++; when the word is found, increase the word count.



Our next function is **doesExist**. We give doesExist the word/-s to search, and the sentence it searches. Then this function includes found\_word\_counter (word counter of the founded word / -s) and found\_location (location of the founded word).

The word is searched in a sentence. The found (if it founded any word) word\_counter is incremented by 1, and the word is cut off from the sentence.

The process is repeated for the next word until we run out of words or the sentence is empty. *"for (int index = 0; index <Words.size (); index ++)"* takes the words in order, starting from 0, how many words are in the sentences, thanks to the index in the for loop using the find function "*found\_location = Sentence.find (Words [ index]);* " We are doing a word search normally.

The sentence can sometimes be empty or even a space, We speed it up by putting this if so it does not tire the system. So, if the search phrase is empty, return the found word counter. If the word is found, get the location after the word. If there is more than 1 of the same word in the sentence. If the word is found, we will remove the word from the string and here we used string before and after.

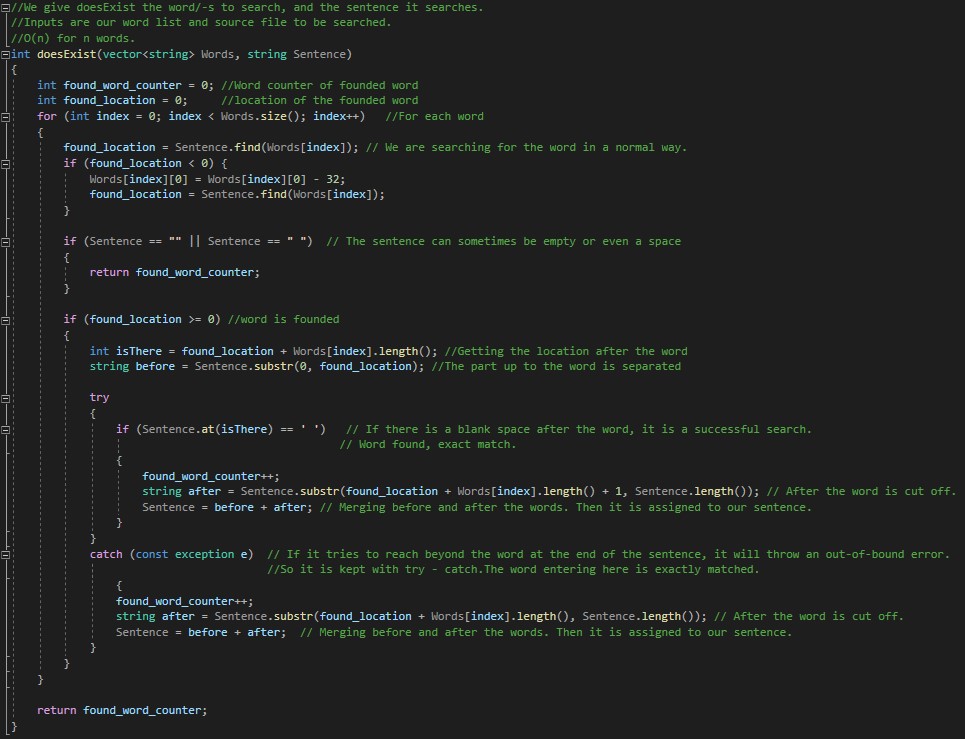
(isThere -> location of word + searched word length.)

string before -> The part up to the word is separated. It works from the point in the found\_location (select from the beginning (0) to the beginning of the word). If there is a blank space after the word, it is a successful search. That means the word is found it is an exact match. When the word we are looking for is found, we increase the found\_word\_counter. string after -> We add the word to the location (at the end). We say that the part after, (also including the space), until the end of the sentence (Sentence.length ()); )

*Sentence = before + after;* Merging before and after the words. Then it is assigned to our sentence again.

Before and after is when we cut the word from the sentence. Then we merge it, and it is assigned to the sentence.

If it tries to reach beyond the word at the end of the sentence, it will throw an out-of-bounds error. So it is kept with try-catch. The word entering here is exactly matched.



The inputs in this function are the Main Document and the file to be compared. This function connects to other functions, takes values from them, assigns to another function it does, return, evaluates etc.

We start by creating a 2D Vector. The sentences of the original document are sent one by one to sentenceToWords and used in word form.

The Words, the document to compare ?. (1. Or 2. etc.) sent to doesExist together with the sentence. From there the founded words will be returned.

Similarity is being calculated. The calculation will be: Founded Words count/word count of the sentence (copied sentence).

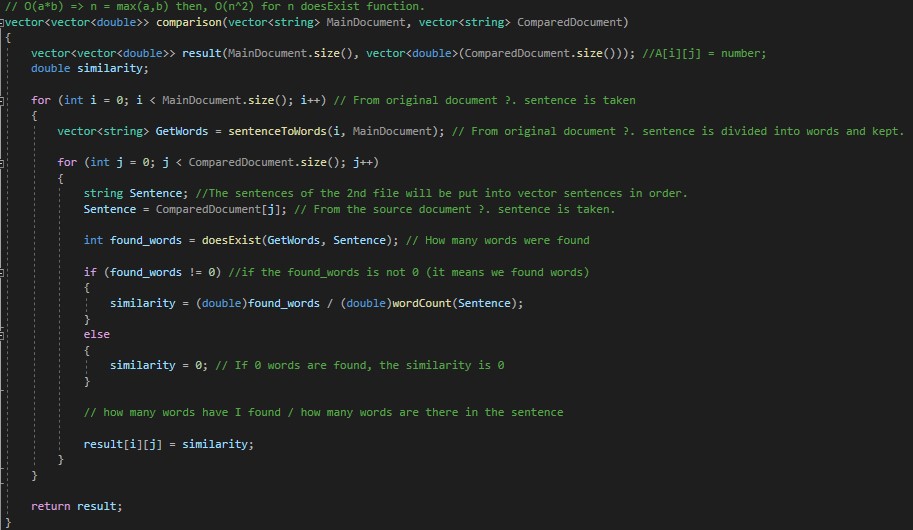
After returning (return b) the Vector String. Now we have separated versions of the words in the sentence (GetWords).

*vector <string> Sentence;* Here we have a vector called Sentence. The sentences of the 2nd file will be put into vector sentences in order.

*int found\_words = doesExist (GetWords, Sentence);* Then we say go to the doesExist method we give to this method the words (GetWords) and sentence that we want to find (Sentence).

If the found\_words is not 0, it means we found words then we will calculate the similarities between the files. This calculation is "how many words have I found / how many words are there in the sentence" which gives us the rate.

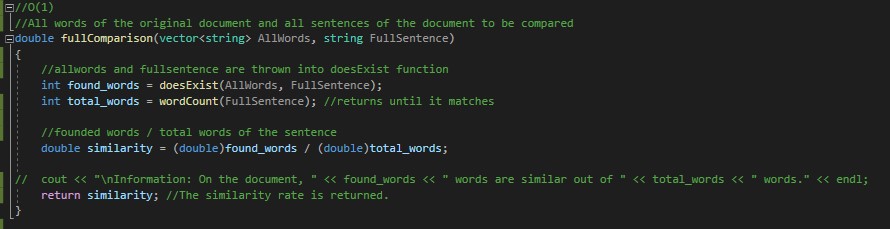
For example, if there are six words in an 8-word sentence, found\_words returns six. In this section, we will find similarities between sentences. Since Found\_words is integer, we converted all of them to double.



All words of the original document [ordered in a vector string] and all sentences of the document to be compared [one long string].

The working principle of this function is as follows. It has a very similar structure to the comparison function. All words are thrown into the doesExist function, and also a full sentence. The word count returns until it matches.

All the words of the 2nd document are calculated with the wordCount function. "Similarity is calculated".

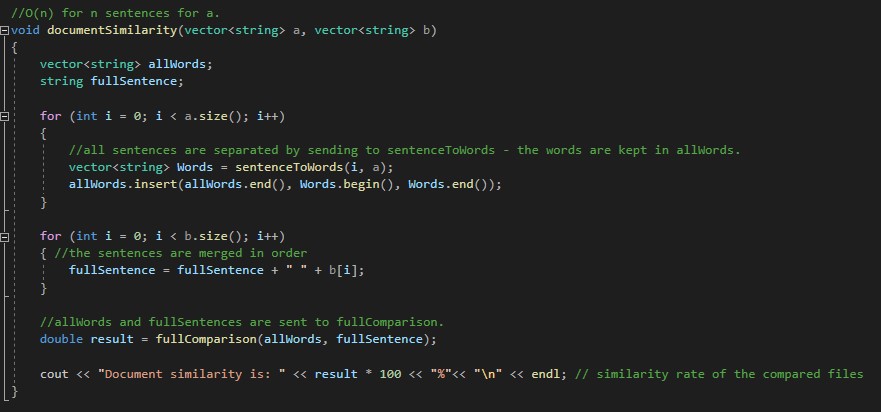


In this function, the file (the file we want to compare) will be compared with the original file.

It allows the original document to be separated into all words. In the for loop, all sentences are separated by sending to sentenceToWords, and the words are kept in allWords.

The sentences of the document to be compared are merged in the For loop in order.

All words and full sentences are sent to fullComparison. The similarity rate returns from there. The similarity rate is printed on the screen, and the process is finished.



Our 2-dimensional Vector with similarity rates (coming from the comparison function), the original document vector - the vector of the document to be compared.

Find the position with the greatest value, print the sentences. Set the similitude value of the location to 0. Repeat this process 5 times. It will print the 5 most similar sentences on the screen.

**Why?**

Why did we follow such a path when we found the 5 most similar sentences?

1. We are using 2D Vector. We can sort this Vector as either an Array or a Vector (thanks to #include <algorithm>), but the problem is that this Vector is 2-dimensional. It is necessary to turn it into one dimension.

1. 2D Vector should not be converted to 1D Vector. //A[i][j] i->original document,

j-> when the source to be compared is converted to a one - dimensional one, it loses the j parameter and makes the sentence retrieval process complicated.

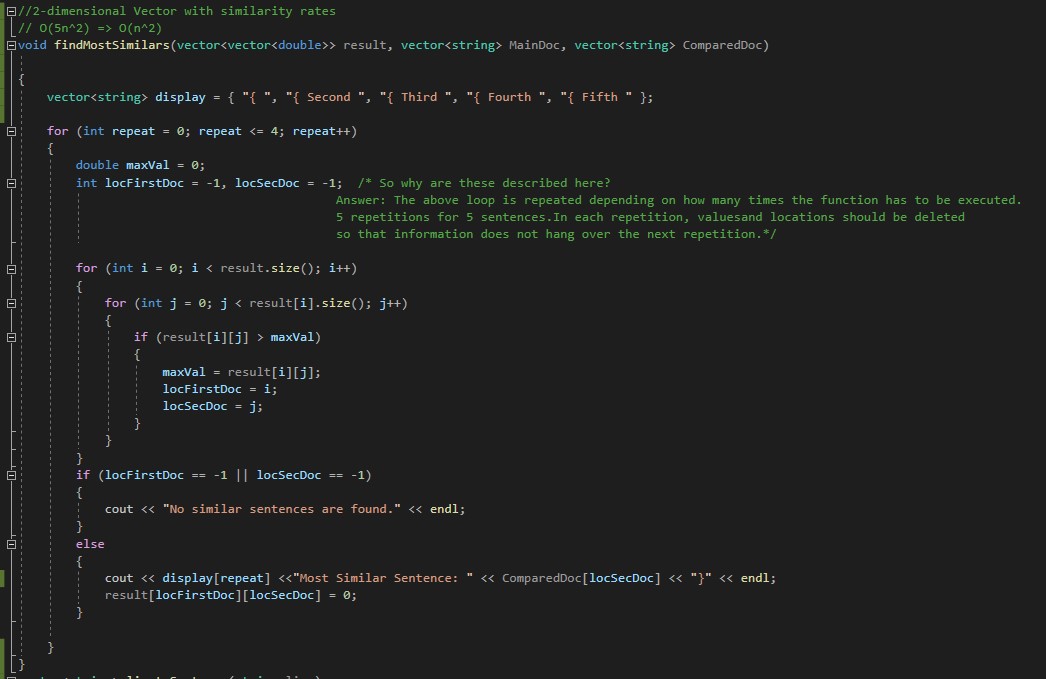
1. The search we wrote using Insertion Sort requires a one-dimensional vector/array. So, we had two options because neither the sort in my library nor the sort we were going to write externally would not work properly for the 2D vector.

1. We were going to create a copy array, find the position with the largest value 5 times and print it.

What it is wrong with putting the copy into the array.Inside the Copy Array, insert A [1. document ?. sentence] [2. document ?. sentence ].

We could not find how to throw it without breaking the structure - even so, the iand j were breaking down again.

1. In a simple way, we just had to find the largest value, print it out in the process, and equate the similarity value to 0 in the location. It won't be able to find that sentence again.



Input is each line read from the document.The line is taken in '.' searched. If there is no or if the dot is at the end, this is a single sentence, it is recorded. If the location is inside;

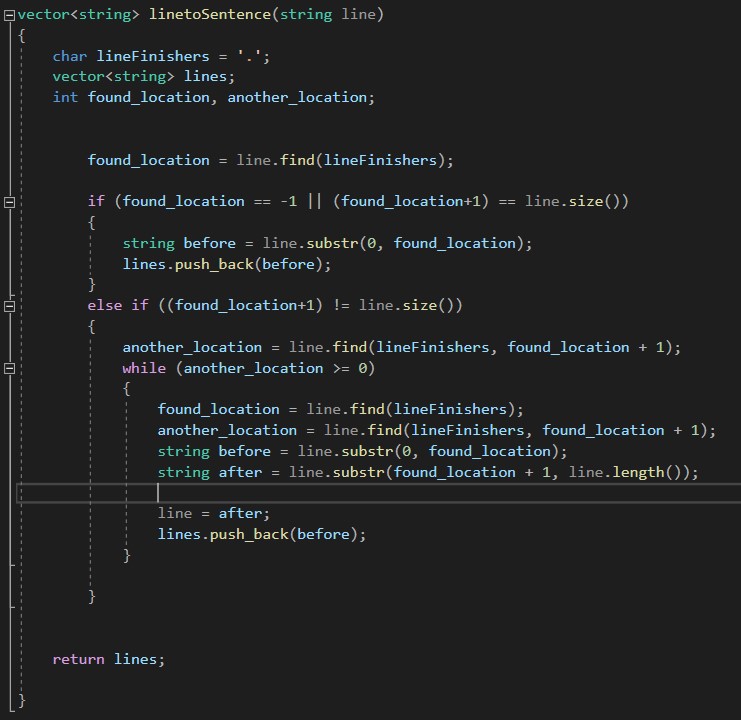
The position of the next point is searched. If there is a next point (While) the sentence is split into two parts. The first part is the part from the beginning to the first point. It is saved in the sentence vector.

The second part takes the part from the end of the first part to the end of the whole line. The second part is saved as a "line". Returns until the position of point 2 is not found in the while loop.

The output is a fragmented sentence vector.

**BIG NOTE**: The part that cuts a sentence. , !, ? , ; Of the parts like,: just had to be reduced to.

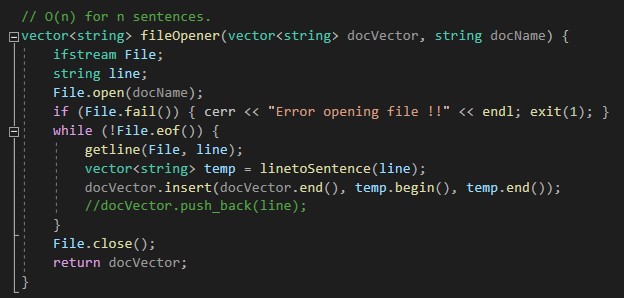
Ex: If before the position of the found point ! , ? - If there are terminators like it could go unnoticed and pass.



The string vector variable where the sentences of the file to be opened are stored and the name of the file to be opened.

The file with the file name sent is opened, the lines inside it are added to the vectore until the file is finished and the file is closed.

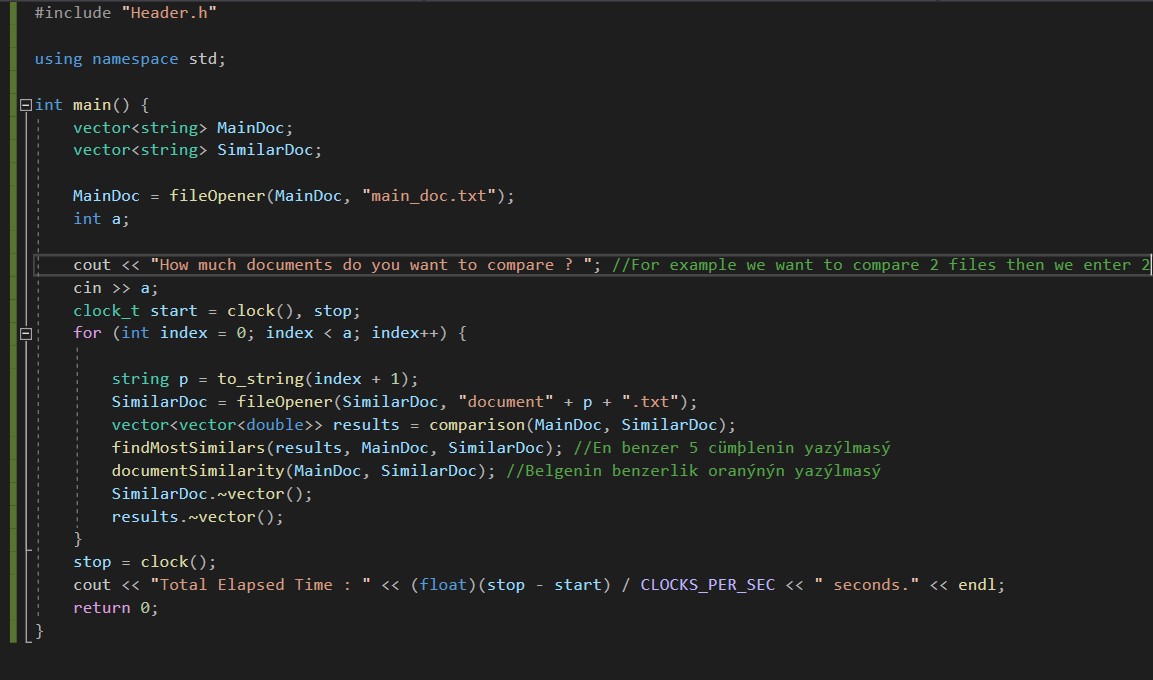
Error output will be shown if the file cannot be opened.



## Source.cpp

A string vector is created for main\_doc and files to be compared.

The Sentences in "main\_doc" are separated by the fileOpener function. While the documents are compared, and Most Similar sentences are found and written to the screen. The comparison continues while the file series is finished.



# Example Output

This output is an output according to our own txt files. Apart from the Main Document, we used 3 more files for comparison.

