Comp-2540: Lab Assignment 1: Word Count I (4 points)

❀ **Your Marks (For TA Use Only)**

	Q1	Q2	Q3	Q4	Total	Marked by

You need to test your code on our submission site, fill in this PDF file, and submit the PDF file on BrightSpace. Check the following list before you submit:

Have signed the document

Have run *count_LINKED_LIST_BAD* and charted the results in section 5.1

Have tested *count_LINKED_LIST_GOOD* at our A1 site, and plotted the result in the chart in 5.1.

Have explained the time complexity of the BAD program correctly in 5.3.

Have tested your *countFAST* at our A1 site, and plotted the results in the chart in 5.1.

1 Due Date and Submission

Due date is January 20/22, depending on your lab section. To be submitted 15 minutes before the end of your lab section. You can also submit in the labs one week earlier. Before submitting your assignment, you must add your signature below to reaffirm that you followed Senate Bylaws 31 (click here for the document). Some PDF readers may not support digital signature well. We recommend you to use Acrobat Reader.

❀ **Fill and Sign The Form**

I, your name   , **verify that the submitted work is my own work**.

Date   Student Number   EMail ID

---

2 Objective

This is a motivating example to show why we need to understand the complexity of programs. You will see a seemingly harmless and straightforward code can stall your computer. A better understanding of its complexity can identify the problem, and improve the performance dramatically.

3 The Problem

Your task is to count the frequency of words in a text file, and return the most frequent word with its count. For example, given the following text,

```
there are two ways of constructing a software design one way
is to make it so simple that there are obviously no deficiencies
and the other way is to make it so complicated that there are no
obvious deficiencies
```

Your program should printout the following along with the milliseconds to finish the computing:

```
The most frequent word is "there" with 3 occurrences.
```

4   The Algorithms

The algorithm in pseudo code can be as follows. We leave out the data structure for ease of understanding. The algorithm scans the input tokens one by one, and update the *wordFreqList* depending whether it is already in the list or not. Algorithm 1 can be refined by adding more details, especially how to store the *wordFreqList*, and how to determine that *a token is in the wordFreqList*.

Input: Array of string tokens
Output: The most frequent word and its frequency
1 begin
2 | *wordFreqList =*empty;
3 | **foreach** *token in the input* **do**
4 | | **if** *token is in the wordFreqList* **then**
5 | | | increment the frequency of *token*
6 | | **end**
7 | | **else**
8 | | | insert *token* into the *wordFreqList* with freq=1;
9 | | **end**
10 | **end**
11 end

Algorithm 1: An algorithm without data structure.

Algorithm Based on LinkedList Algorithm 2 is based on the data structure LinkedList. It looks up a word in the list by scanning the list from the first position to the last by iterating through i. For each i, it gets the i-th element, and check whether it is the word we are looking for.

Algorithm using Arrays Algorithm 3 uses two arrays to keep track of the counts.

A template program written in Java is listed below in Appendix. It can be downloaded by right-clicking **this**.

Input: Array of string tokens
Output: The most frequent word and its frequency
1 begin
2 | wordFreqList =empty;
3 | **foreach** *token in the input* **do**
4 | | **for** *i =1; i<wordFreqList.length; i++* **do**
5 | | | **if** *wordFreqList.get(i) equals token* **then**
6 | | | | increment the freq of the word in wordFreqList;
7 | | | **end**
8 | | **end**
9 | | **if** *token not in wordFreqList* **then**
10 | | | insert token into the wordFreqList with freq=1;
11 | | **end**
12 | **end**
13 | Find the most frequent word from wordFreqList;
14 end

Algorithm 2: Using LinkedList

Input: Array of string tokens
Output: The most frequent word and its frequency
1 begin
2 | Initialize wordArray and countArray;
3 | **for** *each token in the input array* **do**
4 | | **if** *token in wordArray with index i* **then**
5 | | | increment countArray[i];
6 | | **end**
7 | | **else**
8 | | | find j the last position of wordArray;
9 | | | wordArray[j]=token;
10 | | | countArray[j]=1;
11 | | **end**
12 | **end**
13 | Find the most frequent word;
14 end

Algorithm 3: Using Array
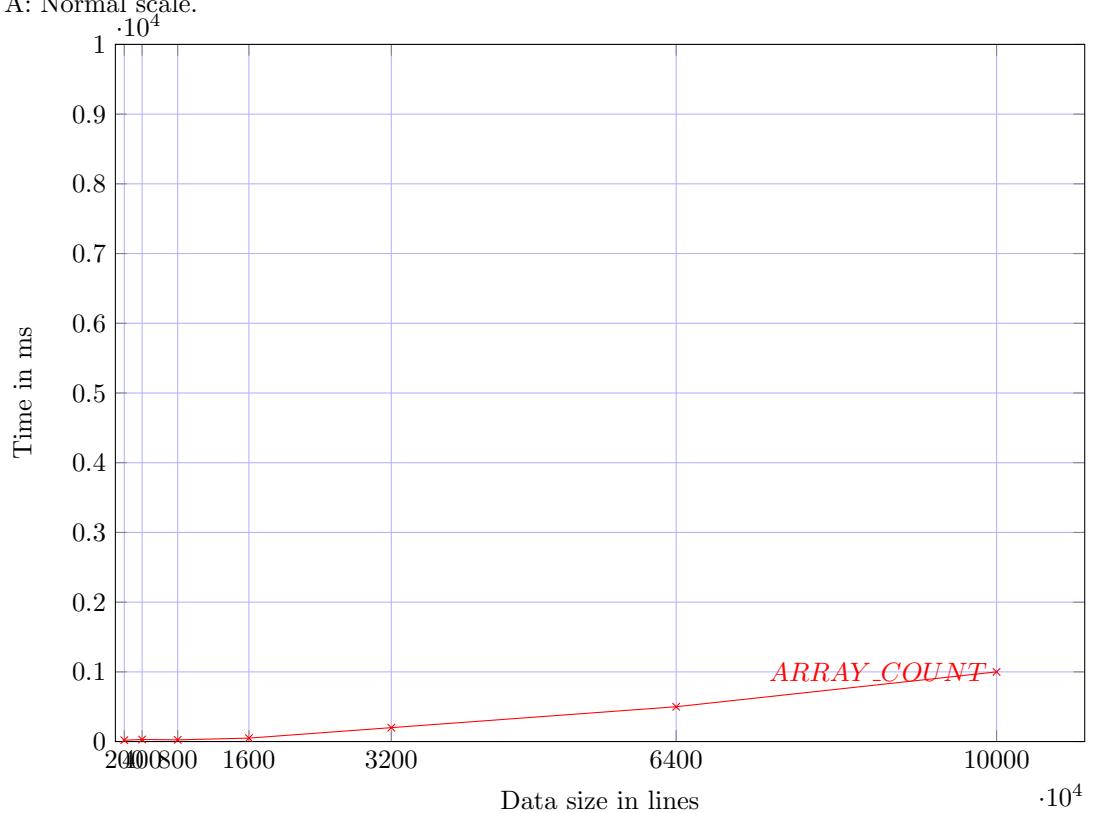
5   Tasks

5.1   Fill in Charts (1 mark)

Fill in the charts for the run time of *count_LINKED_LIST_BAD* by running the program for data sizes 200, 400, 800, 1600, 3200. Beyond 3200 the program will run for very long time. An example is done for you for the *ARRAY_COUNT* method. The first chart is in normal scale, the second plots the same data but in loglog scale. For example, when the data size is 10k (lines), the running time is roughly 1000 ms. You can add your plot using *comment* in Adobe Reader. If your marker goes beyond the scope of the Y-axis, you can mark the position roughly outside of the chart. Before adding markers in the charts, please record the running time in the following boxes:

| Data size(x-axis)	200	400	800	1600	3200	6400
Time (ms)(y-axis)						

A: Normal scale.

B: Loglog plot
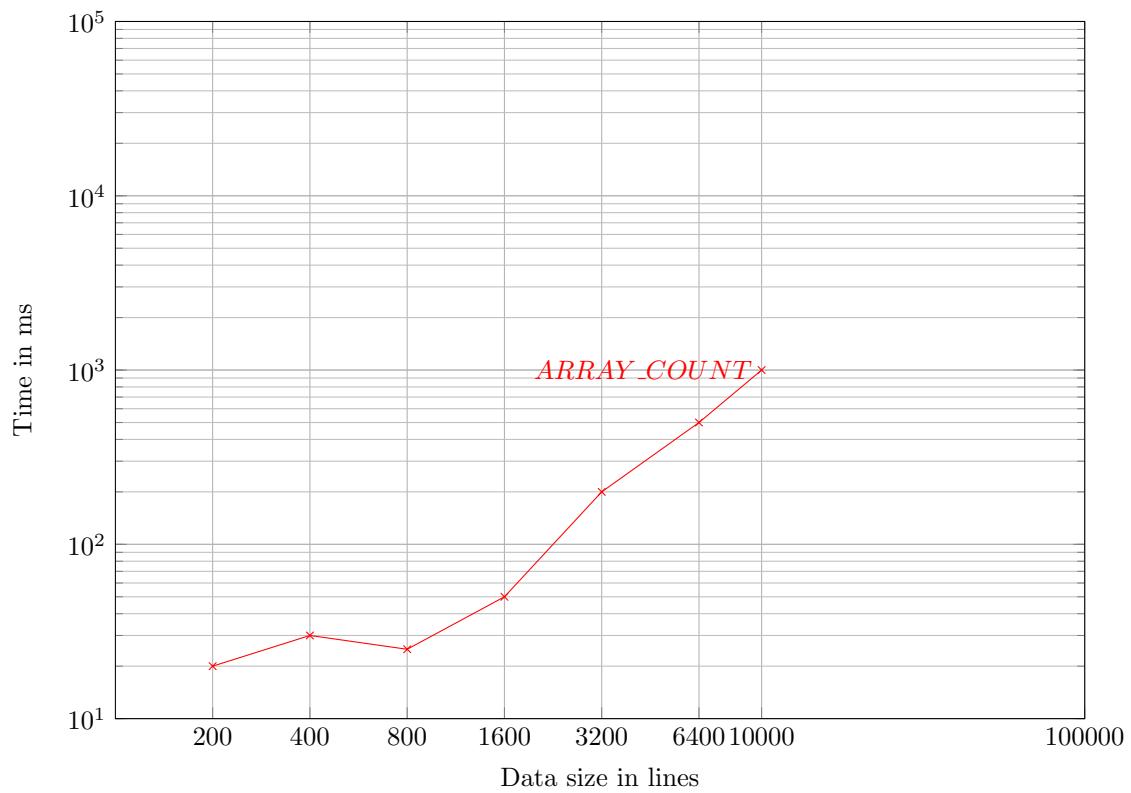
Figure 1: Running time as a function of data size.

5.2 Improve *count_LINKED_LIST_BAD* (1 marks)

As demonstrated above, *count_LINKED_LIST_BAD* is very slow. The code is listed below and the complete program can be accessed here. Obviously there is something very wrong. The program is an implementation of Algorithm 2. The key is how to judge whether the word is in *wordFreqList*. The implementation logic is very simple–it scans the word frequency list one by one until it reaches the end of *wordFreqList* or finds the word. The problem is the *get(i)* method–it iterates through the list to get the *i*-th element. Your task is to improve this algorithm while **keeping the LinkedList** data structure. A hint is to avoid to use *get(i)* by changing the way to iterate the elements in the list.

```
public static Entry<String, Integer> count_LINKED_LIST_BAD(String[] tokens) {
  LinkedList<Entry<String, Integer>> list = new LinkedList<Entry<String, Integer>>();
  for (int j = 0; j < tokens.length; j++) {
        String word = tokens[j];
        boolean found = false;
        for (int i = 0; i < list.size(); i++) {
                Entry<String, Integer> e = list.get(i);
                if (word.equals(e.getKey())) {
                        e.setValue(e.getValue() + 1);
                        list.set(i, e);
                        found = true;
                        break;
                }
        }
        if (!found)
                list.add(new AbstractMap.SimpleEntry<String, Integer>(word, 1));
}
```

Please paste your method *count_LINKED_LIST_GOOD* in the following box. Make sure that you test your code at our site before you paste your code, record the running time in the table below, and fill in the charts in 5.1 correspondingly.

Data size(x-axis)	800	1600	3200	6400	10k	100k
Time (ms)(y-axis)						

5.3 Write and explain the time complexity of *count_LINKED_LIST_BAD* (1 marks)

For a given text file, suppose that there are n number of tokens, and m number of distinct words (i.e., the vocabulary size). In big O notation, write the time complexity of *count_LINKED_LIST_BAD*.

❀**Example Analysis for our *ARRAY_COUNT Method***

The time complexity is $O(nm)$. The justification is as follows: The outer loop repeats n times. For each repetition, we need to search in the word frequency list to see whether the word is there. In the worst case, the cost of doing this is m, the length of the vocabulary. Hence, the total cost is *upper* bounded by

$$\sum_{i=1}^{n} m = nm$$

Please write and explain the time complexity for *count_LINKED_LIST_BAD* below. Make sure that you test your code at our site before you paste in the form below.

5.4   Improve *count_ARRAY* (1 marks)

Although *count_ARRAY* is fast when compared with the LinkedList implementation, its time complexity is not good enough. Write a program that can improve the time complexity that is much better than *count_ARRAY* (i.e., in orders of magnitude). One hint of improvement is to sort the data before counting. Copy/paste your method named *countFast* in the following fillable form, and answer the questions in below the code. Make sure that you test your code at our site before you paste your code, and fill in the charts in 5.1.

Data size(x-axis)	1600	3200	6400	10k	100k	1m
Time (ms)(y-axis)						

Time complexity:

The largest file you can run: Running time in ms:

Most frequent word: Frequency of the word:

6 Data

We prepared a set of test datasets of different sizes. You can click the following data sets to download them. Try to run the data sets in increasing order up to the point when it is unbearably long. Report the performance upto the biggest data you can run. Here are the links to to the datasets. You can right-click the link then use *save as*. The number indicates the number of lines in the text file. 100, 200, 400, 800, 1600, 3200, 6400, 10k, 100k, 1m, 3m.

> ✿**Download Programs and Datasets**
>
> You can download those files using unix commands like:
>
> ```
> wget https://jlu.myweb.cs.uwindsor.ca/2540/dblp200.txt
> ```
>
> If you use OSX, you can use the command line
>
> ```
> curl https://jlu.myweb.cs.uwindsor.ca/2540/dblp200.txt > dblp200.txt
> ```

7 Regulations

This assignment must be done individually. Any similarity found between your code/answers and another student's, or a carbon-copy of an answer found in the web will be considered cheating. For the part where coding is required, you must implement your own program.

8 Bonus

The top two program (the fastest ones) in Task 5.4 will receive one bonus mark. Running time and scoreboard will be displayed after your submission. The website for submitting your code is at A1 Bonus Site.

9 Appendix: Code template

Here is the code template written in Java. You can download the code by clicking on this (use right-click then save as. Copying/pasting from html page may get unwanted characters not visible). Note that you may need to adjust the program, e.g., when running large data, CAPACITY may need to be increased.

If you are new to Java, you should focus on the language constructs that you can understand. You only need to understand loops and if statements so that you can improve the code. Those languages constructs are universal for every programming language. Ignore the part that you can not fully understand for now.

That being said, you need to check language libraries. For example, the JavaDoc for LinkedList is here. One thing that you may not be familiar with is the generics in Java–The $Entry < String, Integer >$ in the code is an entry that can store a pair of string and integer. Inside the angled brackets are data types that are allowed for the pair. It is called generics in Java.

```java
import java.io.File;
import java.util.Scanner;
import java.util.Map.Entry;
import java.util.AbstractMap;
import java.util.LinkedList;

public class WordCountLinkedList2540 {
        public static Entry<String, Integer> count_ARRAY(String[] tokens) {
                int CAPACITY = 10000;
                String[] words = new String[CAPACITY];
                int[] counts = new int[CAPACITY];
                for (int j = 0; j < tokens.length; j++) {
                        String token = tokens[j];
                        for (int i = 0; i < CAPACITY; i++) {
                                if (words[i] == null) {
                                        words[i] = token;
                                        counts[i] = 1;
                                        break;
                                } else if (words[i].equals(token))
                                        counts[i] = counts[i] + 1;
                        }
                }

                int maxCount = 0;
```

```java
        String maxWord = "";
        for (int i = 0; i < CAPACITY & words[i] != null; i++) {
                if (counts[i] > maxCount) {
                        maxWord = words[i];
                        maxCount = counts[i];
                }
        }
        return new AbstractMap.SimpleEntry<String, Integer>(maxWord, maxCount);
}

public static Entry<String, Integer> count_LINKED_LIST(String[] tokens) {
        LinkedList<Entry<String, Integer>> list = new LinkedList<Entry<String, Integer>>();
        for (int j = 0; j < tokens.length; j++) {
                String word = tokens[j];
                boolean found = false;
                for (int i = 0; i < list.size(); i++) {
                        Entry<String, Integer> e = list.get(i);

                        if (word.equals(e.getKey())) {
                                e.setValue(e.getValue() + 1);
                                list.set(i, e);
                                found = true;
                                break;
                        }
                }
                if (!found)
                        list.add(new AbstractMap.SimpleEntry<String, Integer>(word, 1));
        }

        int maxCount = 0;
        String maxWord = "";
        for (int i = 0; i < list.size(); i++) {
                int count = list.get(i).getValue();
                if (count > maxCount) {
                        maxWord = list.get(i).getKey();
                        maxCount = count;
                }
        }
        return new AbstractMap.SimpleEntry<String, Integer>(maxWord, maxCount);
}

static String[] readText(String PATH) throws Exception {
        Scanner doc = new Scanner(new File(PATH)).useDelimiter("[^a-zA-Z]+");
        int length = 0;
        while (doc.hasNext()) {
                doc.next();
                length++;
        }

        String[] tokens = new String[length];
        Scanner s = new Scanner(new File(PATH)).useDelimiter("[^a-zA-Z]+");
        length = 0;
        while (s.hasNext()) {
                tokens[length] = s.next().toLowerCase();
                length++;
        }
        doc.close();

        return tokens;
}

public static void main(String[] args) throws Exception {

        String PATH = "dblp200.txt";
        String[] tokens = readText(PATH);
        long startTime = System.currentTimeMillis();
        Entry<String, Integer> entry = count_LINKED_LIST(tokens);
        long endTime = System.currentTimeMillis();
        String time = String.format("%12d", endTime - startTime);
        System.out.println("time\t" + time + "\t" + entry.getKey() + ":" +
            entry.getValue());

        tokens = readText(PATH);
        startTime = System.currentTimeMillis();
        entry = count_ARRAY(tokens);
        endTime = System.currentTimeMillis();
```

```java
        time = String.format("%12d", endTime - startTime);
        System.out.println("time\t" + time + "\t" + entry.getKey() + ":" +
            entry.getValue());
}
```