03-60-254: Data Structures and Algorithms
Lab Assignment 2: Word Count II

1 Due date

In your lab section during the week October 15/17. To be submitted 15 minutes before the end of your lab section. Any submission within the last 15 minutes before the end of the lab session will not be accepted and a zero mark will be given. Late assignments will be given a zero mark. Submissions by email will not be accepted.

2 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.

3 Objective

The aim of this assignment is to understand the performance difference of various sorting algorithms.

4 The problem specification

The problem is the same as in *A1: Word Count I*, i.e., to count the frequency of words in a text file, and return the most frequent word with its count. Although the problem is the same, this time we will use a very different algorithm.

5 The algorithms

In this assignment, the counting method is based on sorting algorithms. A template of the algorithms is listed in Algorithm 1. The idea is to sort the tokens first, so that all the same tokens are located consecutively. For the sample input in A1, the sorting result can be:

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

When the tokens are sorted, it is rather easy to count their duplicates–we simply increment the count until a new token is encountered. The challenge is how to sort the tokens efficiently. We give three example sorting algorithms, i.e., Selection Sort, Insertion Sort, and Merge Sort. A template program written in Java can be downloaded by clicking **this**. You can also use other programming languages of your choice, such of Python, C, C++.

6 Tasks

Task 1 (2 marks). Demonstrate the performance of the algorithms You are required to run these three algorithms, and observe their performance differences. 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.

200 lines	Link to dblp200	**10k lines**	Link to dblp10k
500 lines	Link to dblp500	**100k lines**	Link to dblp100k
1k lines	Link to dblp1k	**1m lines**	Link to dblp1m
5k lines	Link to dblp5k	**3.6m lines**	Link to dblp

Report the experiments you run by adding markers on the following chart. x-axis is the data size, y-axis is the running time. For Selection-Sort and Insertion-Sort, the program won't be able to run the full gamut of the datasets. You plot the maximum data size that you can run on your computer.

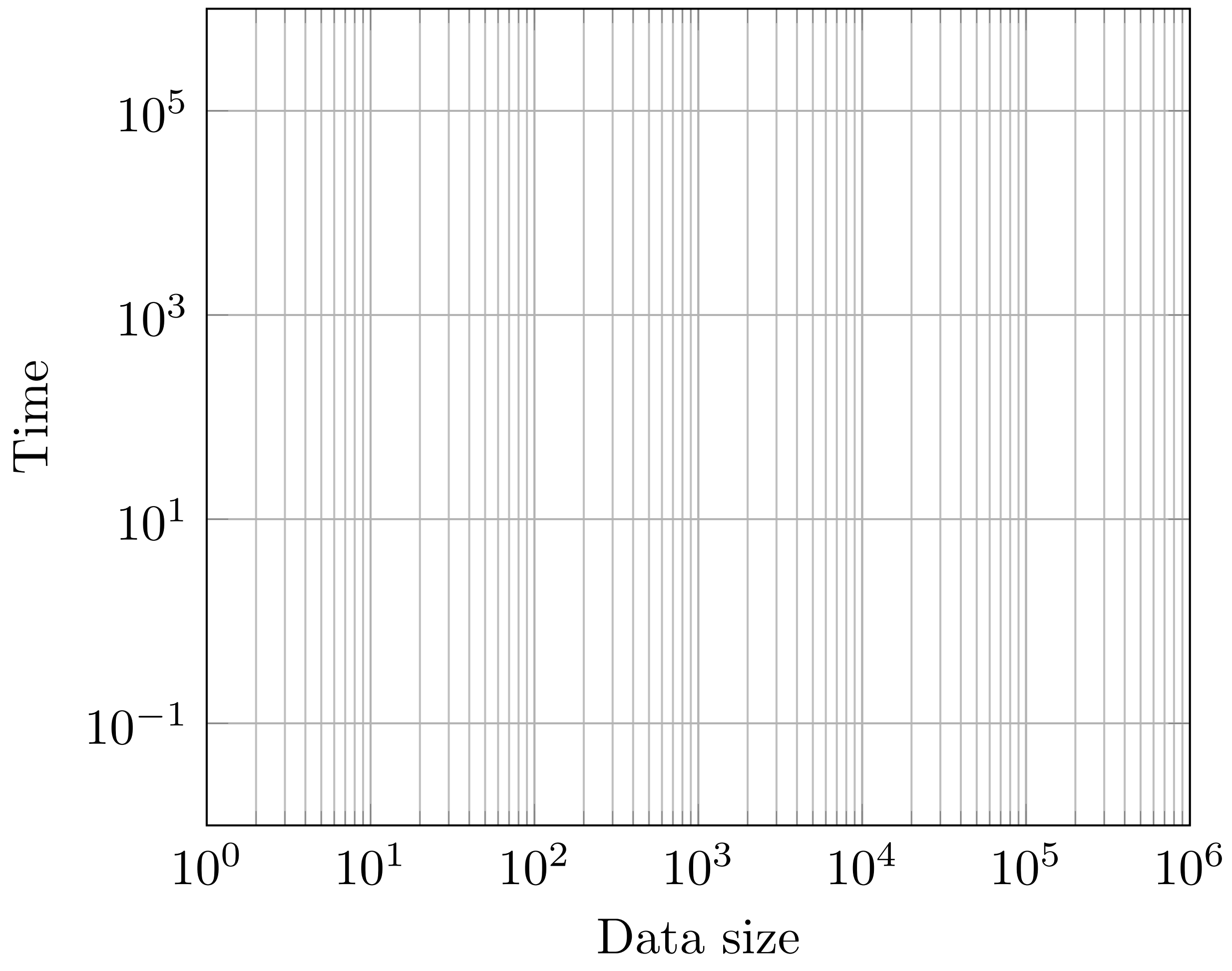
Task 2 (1 mark) Predict the run-time Since it is impossible to run Select-Sort on our 1Million data, we can only predict its approximate run-time. Give your estimate of run-time for Selection-Sort on data *dblp1m*.

Task 3 (2 marks) Algorithm analysis Give the complexities of these three algorithms. In particular, give the proof of the complexity of the MergeSort algorithm.

Bonus (1 mark) Implement a better sorting Algorithm] Although Merge Sort is very fast, there is room to improve it. Implement a better algorithm and demonstrate its performance.

```
Input: tokens: Array of string tokens
Output: The most frequent word and its frequency
begin
    sort tokens alphabetically;
    foreach t in tokens do
        if t is new then
            add t in wordArray;
            update wordFreqArray;
        end
        else
            increment wordFreqArray until it is different
        end
    end
    Find the most frequent word from wordFreqArray;
end
```

Algorithm 1: WordCount by sorting the tokens.


Time
10^5
10^3
10^1
10^-1
10^0
10^1
10^2
10^3
10^4
10^5
10^6
Data size