UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Denkleiers • Leading Minds • Dikgopolo tša Dihlalefi

Department of Computer Science
Faculty of Engineering, Built Environment & IT
University of Pretoria

COS212 - Data structures and algorithms

Practical 8 Specifications - External Sorting

Release date: 06-05-2024 at 06:00
Due date: 10-05-2024 at 23:59
Total marks: 700

Contents

1 General Instructions

- ***Read the entire assignment thoroughly before you start coding.***
- This assignment should be completed individually; no group effort is allowed.
- **To prevent plagiarism, every submission will be inspected with the help of dedicated software.**
- Be ready to upload your assignment well before the deadline, as no extension will be granted.
- You may not import any of the built-in Java data structures. Doing so will result in a zero mark. You may only use native 1-dimensional arrays where applicable. If you require additional data structures, you will have to implement them yourself.
- If your code does not compile, you will be awarded a mark of zero. Only the output of your program will be considered for marks, but your code may be inspected for the presence or absence of certain prescribed features.
- If your code experiences a runtime error, you will be awarded a zero mark. Runtime errors are considered unsafe programming.
- Read the entire specification before you start coding.
- **Ensure your code compiles with Java 8**
- The usage of ChatGPT and other AI-Related software is strictly forbidden and will be considered as plagiarism.

2 Plagiarism

The Department of Computer Science considers plagiarism a serious offence. Disciplinary action will be taken against students who commit plagiarism. Plagiarism includes copying someone else's work without consent, copying a friend's work (even with consent), and copying material (such as text or program code) from the Internet. Copying will not be tolerated in this course. For a formal definition of plagiarism, the student is referred to http://www.library.up.ac.za/plagiarism/index.htm (from the main page of the University of Pretoria site, follow the Library quick link, and then choose the Plagiarism option under the Services menu). **If you have any form of question regarding this, please ask one of the lecturers to avoid any misunderstanding.** Also note that the OOP principle of code reuse does not mean that you should copy and adapt code to suit your solution.

3 Outcomes

On completion of this practical, you will have gained experience with the following:

- External sorting, focusing on multiway merging during external sorting, including:
  - Initial sorting
  - Repeated K-way merging
  - Temporary file manipulation

4 Introduction

4.1 Multiway Merge External Sorting

External sorting is a class of algorithms that handle massive amounts of data that do not fit into the main memory of a computer. These algorithms are crucial for database systems, large-scale data processing, and applications where data needs to be sorted in a constrained memory environment. Among the various external sorting techniques, Multiway Merge Sorting stands out due to its efficiency in dealing with vast datasets by reducing the number of disk I/O operations required during the sorting process.

Multiway Merge Sort employs a divide-and-conquer strategy that is adapted for external storage media such as hard drives. The algorithm can be broken down into two major phases:

- Splitting Phase: In this initial phase, the algorithm divides the entire dataset into manageable blocks that fit into the main memory. Each of these blocks is sorted in memory using an efficient internal sorting algorithm like Quicksort or Heapsort. After sorting, the blocks are written back to the disk as temporary sorted sequences.
- Merging Phase: The sorted sequences are then merged into a single sorted sequence. Unlike the traditional 2-way merging used in basic merge sort, Multiway Merge Sort can merge multiple sequences at once. This is achieved by using a k-way merge algorithm, where 'k' represents the number of sorted sequences that are merged simultaneously. The efficiency of this phase is significantly enhanced by utilizing a min-heap data structure that helps in quickly identifying the smallest element among the top elements of all sequences.

The main advantage of the Multiway Merge Sort is its ability to minimize the total number of passes over the data when compared to traditional 2-way merges. By increasing the number of sequences merged in each pass (i.e., increasing 'k'), the algorithm can reduce the number of merge passes required, thus speeding up the sorting process under disk-bound constraints.

In practical implementations, careful consideration must be given to the selection of 'k' and the size of the memory blocks to optimize the balance between CPU processing and disk I/O operations. This strategy is particularly effective in environments where disk access is a significant bottleneck, making Multiway Merge Sort a preferred choice for external sorting operations on large datasets.

5 Tasks

You are tasked with implementing an external sort using multiway merging. You have been given the skeleton for `MultiwayMergeExternalSort` as well as the implementation for the `Memory` class. The `Memory` class will be overridden during testing and must be used in your `MultiwayMergeExternalSort` instead of standard arrays. Your implementation must adhere to the given class diagram (Figure 1).

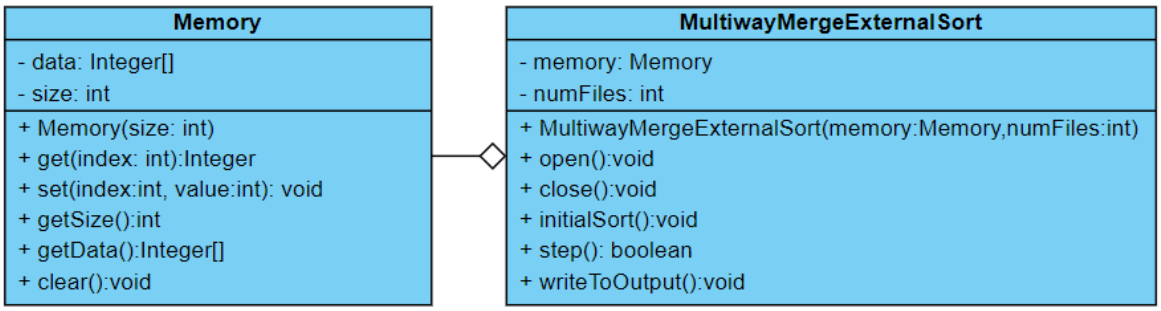

Figure 1: ReplacementSelectionSort UML class diagram

5.1 MultiwayMergeExternalSort

- Members:
  - memory: Memory
    - A reference to a `Memory` object.
    - The `Memory` object is used to simulate limited memory.
    - You should not use other arrays to store data in your sort. Only the memory object should be used for data to be sorted.
  - numFiles: int
    - The number of files that are used for the external multiway merge sort.
    - This number refers to the number of input/output files using the algorithm.
    - So if `numFiles` is 2 then there are 2 input and 2 output files. i.e $T_{a1}, T_{a2}, T_{b1}, T_{b2}$
- Methods:
  - MultiwayMergeExternalSort(memory:Memory,numFiles:int)
    - A constructor which sets the memory and the number of files
    - Any other setup can be done here.
  - open(intputFile:String,outputFile:String):void
    - This function creates a way to read from an input file and write to an output file.
    - This function can also create ways to interact with temporary files.
    - It is recommended to use `BufferedReader` and `BufferedWriter`
  - close(data: T): void
    - This function closes all open resources such as `BufferedReader`s and `BufferedWriter`s.
  - initialSort(): vois
    - This function performs the initial split and sort based on memory size and number of files.

    * In this practical all sorting should be done in ascending order.
    * Initially the data should be read from input in chunks of size `memory.getSize)`, sorted and written into the **b** set of files.

- step(): boolean
    * This function does one full merge and transfer from the **b** set of files to the **a** set of files or vice versa.
    * You must do a multiway merge as in the textbook (section 7.12.4). This means you will need to implement your own priority queue and keep track of the file from which each item came. Your priority queue does not need to use the `Memory` object.
    * **NB!** If you do not do the following correctly you will get no marks.
        · All temporary files (like T_{a1}) need to be made, read and manipulated in a directory called `tmp`. You do not need to make `tmp` directory in your code or makefile. One already exists in the FF testing environment.
        · All temporary files must end in `.tmp`
        · Temporary files must be named in the following format: "T_a/b_n.tmp". For example, T_{b2} should be named "T_b_2.tmp" and T_{a0} should be named "T_a_0.tmp"
    * This function returns true if no more steps have to be done to sort the data (i.e. all the data is completely sorted and in the file indexed 0). This function returns false if more steps need to be done to sort the data.

- writeToOutput():String
    * Writes the sorted array to the output file as given in the `open` method.
    * Each value should be written on a new line.
    * Do not add any empty lines. If there are 500 values the output file should have exactly 500 lines.

6 Testing

As testing is a vital skill that all software developers need to know and be able to perform daily, approximately 10% of the assignment marks will be assigned to your testing skills. To do this, you will need to submit a testing main (inside the Main.java file) that will be used to test an Instructor-provided solution. You may add any helper functions to the main.cpp file to aid your testing. In order to determine the coverage of your testing the jacoco tool. The following set of commands will be used to run jacoco:

```
javac *.java
rm -Rf cov
mkdir ./cov
java -javaagent:jacocoagent.jar=excludes=org.jacoco.*,destfile=./cov/output.exec
    -cp ./ Main
mv *.class ./cov
java -jar ./jacococli.jar report ./cov/output.exec --classfiles ./cov --html
    ./cov/report
```

This will generate output which we will use to determine your testing coverage. The following coverage ratio will be used:

$$\frac{\text{number of lines executed}}{\text{number of source code lines}}$$

and we will scale this ratio according to the size of the class.
The mark you will receive for the testing coverage task is determined using table 1:

Coverage ratio range	% of testing mark
0%-5%	0%
5%-20%	20%
20%-40%	40%
40%-60%	60%
60%-80%	80%
80%-100%	100%

Table 1: Mark assignment for testing

Note the top boundary for the Coverage ratio range is not inclusive except for 100%. Also, note that only the function stipulated in this specification will be considered to determine your mark. Remember that your main will be testing the instructor-provided code and as such it can only be assumed that the functions outlined in this specification are defined and implemented.

7 Upload checklist

The following files should be in the root of your archive

- Main.java
- MultiwayMergeExternalSort.java

8 Allowed libraries

- java.io.*

9 Submission

You need to submit your source files on the FitchFork website (https://ff.cs.up.ac.za/). All methods need to be implemented (or at least stubbed) before submission. Place the above-mentioned files in a zip named uXXXXXXXX.zip where XXXXXXXX is your student number. Your code must be able to be compiled with the Java 8 standard.

For this practical, you will have 5 upload opportunities and your best mark will be your final mark. Upload your archive to the appropriate slot on the FitchFork website well before the deadline. **No late submissions will be accepted!**