

**COMP - 6521**

**Advanced Database Technology and Applications**

**Project Report**

**On**

**Lab Assignment 1: TPMMS & Duplicate Elimination**

**Professor**

**Dr. Nematollaah Shiri**

**Team Members**

|  |  |  |
| --- | --- | --- |
| **Student Name** | **Student Id** | **Email Id** |
| Yash Pandya | 40119272 | y\_pandy@encs.concordia.ca |
| Himen Sidhpura | 40091993 | h\_sidhpu@encs.concordia.ca |
| Sucheta Sudhakumari | 40080543 | s\_ijaya@encs.concordia.ca |

**Index**

1. Steps to run the program 3
2. Program Description 3
3. Experiment Results 3
4. Architecture Diagrams 6
5. Algorithm 7
6. Technical Details 7
7. Coding Standard 7
8. Class Description 8
9. Group Member Contribution 8
10. Results 9
11. **Steps to run the program:**

* Place the data sets in the ‘InputFiles’ folder and set up the eclipse environment.
* Setup main memory size.
* Run the program with ‘ProgramController.java’ class.
* The program will read the input files based on memory size, sort them into sublists, merge that sublists into final sorted relation while eliminating the duplicates.
* The sub lists can be seen in the ‘blocks’ folder.
* The output which is the final sorted list without duplicates will be generated in the output files folder.

1. **Program Description:**

* In Phase 1, the program begins by clearing the output and sublist folders. Followed by this the program takes the data sets T1(sample1.txt) and T2(sample2.txt) one after the other and sorts them. The sub-lists created in phase 1 can be seen in the blocks folder.
* In phase 2, the sublists are read into the 2 input buffers iteratively and compared against each other to find the smallest one which will then be written into the output buffer. Elimination of duplicate tuples takes place amidst this comparison. Once there is no more sublists remaining the final output is written back to the disk.

1. **Experiment Results:**

The experiment was conducted by changing the allocated main memory size the result of which has been consolidated below:

**Experiment 1:**

**Main Memory Size = 5 MB**

|  |  |
| --- | --- |
| **Tuple Count** | **20,00,064** |
| Number of Disk IO for phase 1 | 1,00,004 |
| Time to sort T1 | 3.633 sec |
| Time to sort T2 | 4.008 sec |
| Number of Blocks created | 50,001 |
| Number of Disk IO for phase 2 | 1,46,581 |
| Time to merge Sublists | 5.015 sec |
| Total program execution time | 12.656 sec |
| **Main Memory Size = 10 MB** | |
| **Tuple Count** | **20,00,064** |
| Number of Disk IO for phase 1 | 1,00,004 |
| Time to sort T1 | 3.103 sec |
| Time to sort T2 | 3.626 sec |
| Number of Blocks created | 50,001 |
| Number of Disk IO for phase 2 | 1,06,230 |
| Time to merge Sublists | 2.831 sec |
| Total program execution time | 9.56 sec |
| **Main Memory Size = 20 MB** | |
| **Tuple Count** | **20,00,064** |
| Number of Disk IO for phase 1 | 1,00,004 |
| Time to sort T1 | 2.494 sec |
| Time to sort T2 | 2.404 sec |
| Number of Blocks created | 50,001 |
| Number of Disk IO for phase 2 | 77,350 |
| Time to merge Sublists | 1.745 sec |
| Total program execution time | 6.642 sec |

**Experiment 2:**

**Main Memory Size = 5 MB**

|  |  |
| --- | --- |
| **Tuple Count** | **12,00,064** |
| Number of Disk IO for phase 1 | 60,004 |
| Time to sort T1 | 2.506 sec |
| Time to sort T2 | 2.239 sec |
| Number of Blocks created | 30,001 |
| Number of Disk IO for phase 2 | 88,210 |
| Time to merge Sublists | 2.918 sec |
| Total program execution time | 7.663 sec |
| **Main Memory Size = 10 MB** | |
| **Tuple Count** | **12,00,064** |
| Number of Disk IO for phase 1 | 60,004 |
| Time to sort T1 | 2.011 sec |
| Time to sort T2 | 1.933 sec |
| Number of Blocks created | 30,001 |
| Number of Disk IO for phase 2 | 63,738 |
| Time to merge Sublists | 1.839 sec |
| Total program execution time | 5.783 sec |
| **Main Memory Size = 20 MB** | |
| **Tuple Count** | **12,00,064** |
| Number of Disk IO for phase 1 | 60,004 |
| Time to sort T1 | 1.55 sec |
| Time to sort T2 | 1.5 sec |
| Number of Blocks created | 30,001 |
| Number of Disk IO for phase 2 | 46,384 |
| Time to merge Sublists | 0.963 sec |
| Total program execution time | 4.013 sec |

1. **Architecture Diagram:**

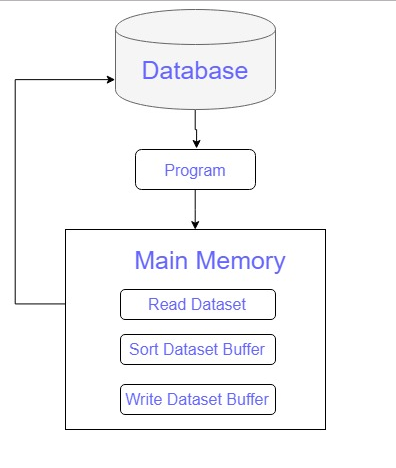


Figure 1: Phase 1

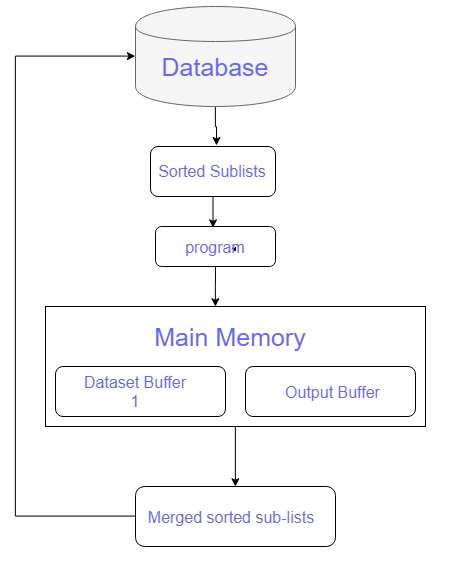
**

Figure 2: Phase 2

1. **Algorithms:**

**PHASE - 1 (TPMMS Sort):**

* Start.
* Allocate main memory to read sublists.
* Reset output folder and sublist folder.
* Calculate the block size which is 10% of total memory available.
* Load the input buffer.
* Update the count of I/O operation.
* Apply Quick sort to sort the tuples within input buffer.
* Write the sorted list to the sublist folder.
* Update the count of I/O operations.
* Update the count of sublists created.
* Repeat the above steps from 5 to 10 for calculated number of memory fills.
* Calculate the time taken to sort the relation.
* End.

**PHASE - 2 (TPMMS Sort):**

* Start
* Read the input blocks into 2 string variables and check for their validity.
* Compare the strings against id and date to identify the smallest and non duplicated record.
* Write the record into the output buffer.
* Update the disk IO count and time taken for one merge iteration.
* Continue the operation until both buffers are emtpy.
* Stop

1. **Technical Detail:**

The program executes in two phases

* Phase-1 involves reading input file data and saving sorted data in sublist.
* Phase-2 includes the collection of sublist records and the merging of the sorted list into the output file by removing the duplicate tuple and contains tuple with the latest date.

The above process is followed for memory size of 10 MB and 20 MB, then the execution period is measured for executing the entire operation that involves the sorting and the duplicate removal in the merging.

1. **Coding Standards:**

The most general coding conventions were followed while the codes were developed as follows,

* The class name begins with an uppercase word.
* E.g.: ProgramController.java
* Constants are called with characters in the upper case
* The variable name is descriptive and is rendered in lower case including a capital letter to separate words.
* The procedure name begins with a lowercase character and uses the uppercase characters to separate words.

1. **Class Description:**

**Constants.java:**  This class stores the constant values required for program execution like file IO paths, block size etc.

**PhaseOne.java:**  This class has the methods for reading tuples into main memory, sorting them and creating the sublists. It also handles the IO calculation for phase one.

**PhaseTwo.java:** This class has the methods for combining the sublists into a final sorted list, while eliminating duplicates.

**ProgramController.java:** This class has the main method from where program execution starts. It builds and clears the output directories and prints the timing outputs. It also creates the objects for other classes and trigger their execution.

**QuickSort.java:**  This class performs the quicksort algorithm on the blocks read into memory.

1. **Group Member Contribution:**

Member participation was uniform across all stages of the project development . We had meetings once a week to discuss on design changes and individual progress .This ensured that everyone is on the same page. We also adopted pair programming strategy which let us help each other with our areas of expertise , thereby developing efficient code.  
The documentation part was split into sections and assigned to each team mate as a part of even work distribution .

1. **Results:**

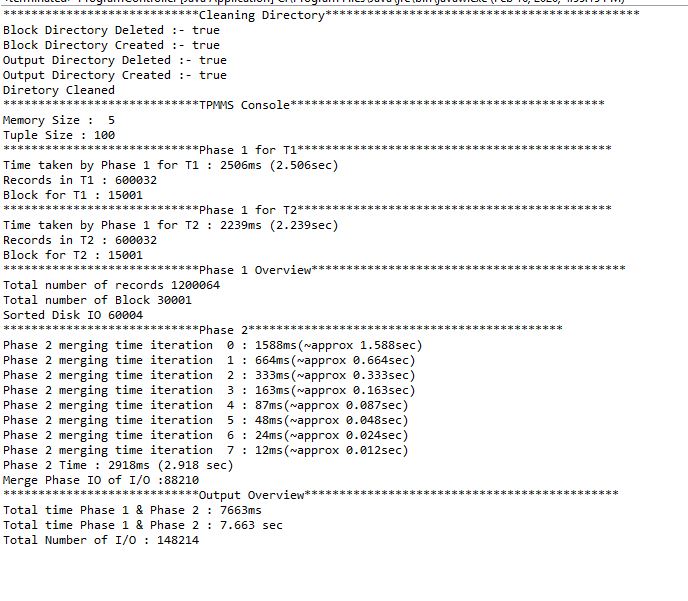
****

Figure 3: 5 MB – 12,00,064 Tuple Count

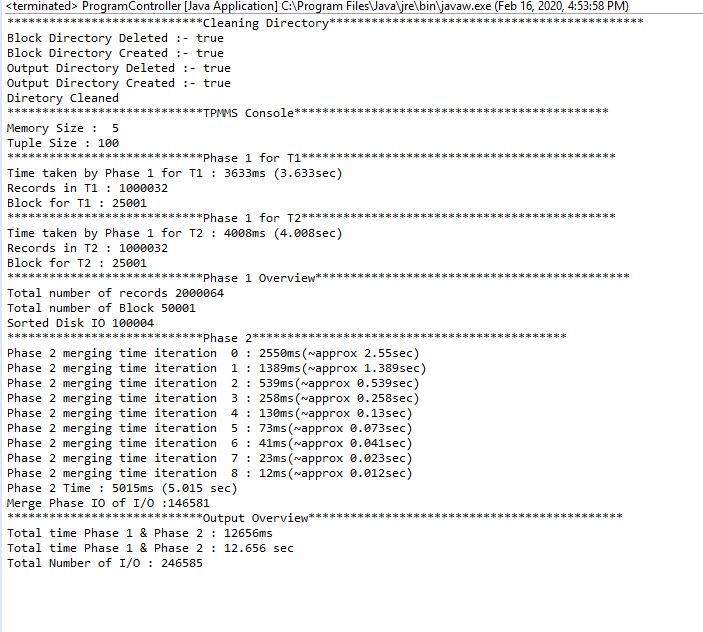


Figure 4: 5MB – 20,00,064 Tuple Counts

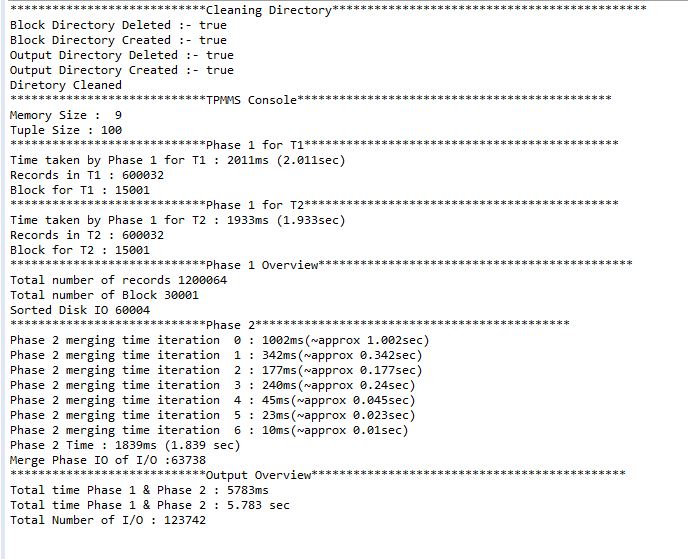
****

Figure 5: 10MB – 12,00,064 Tuple Count

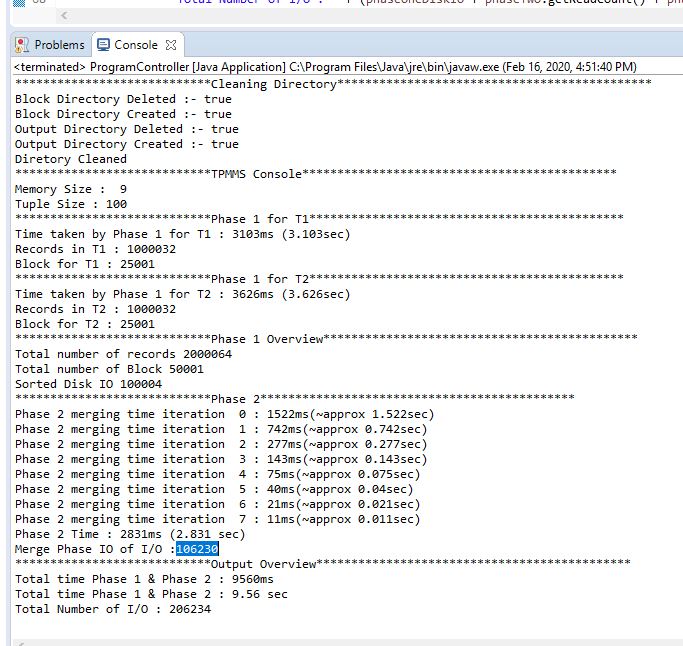
****

Figure 6: 10MB – 20,00,064 Tuple Count

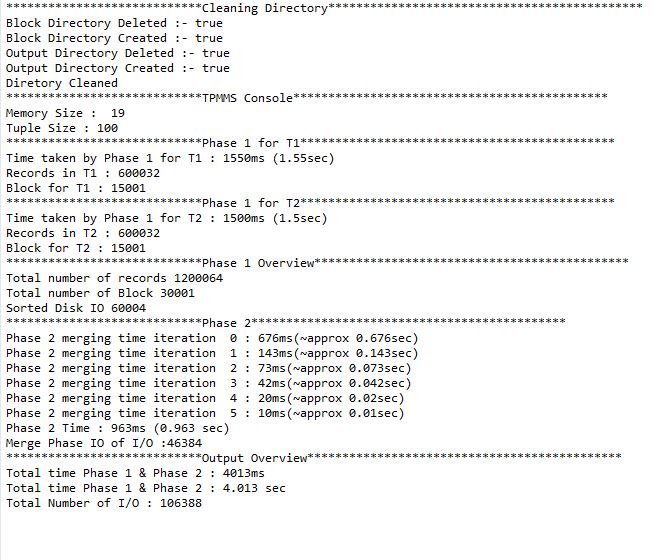
****

Figure 7: 20MB – 12,00,064 Tuple Count

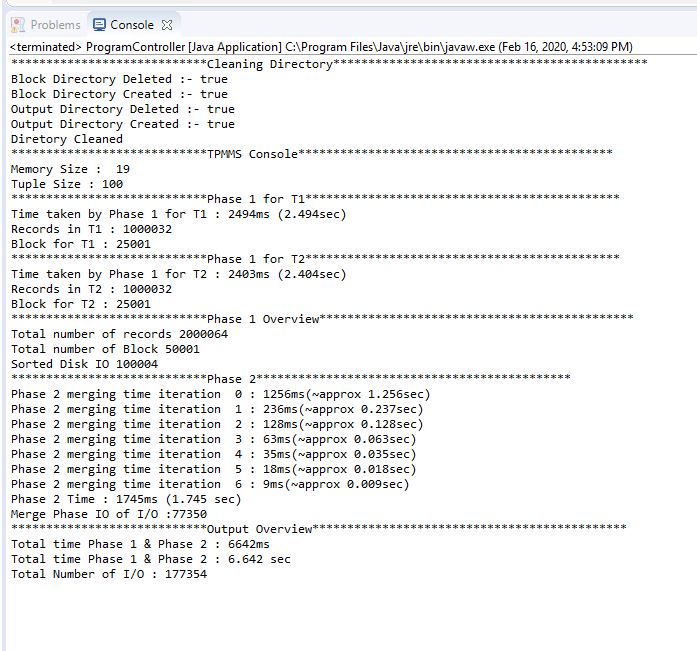
****

Figure 8: 10MB – 20,00,064 Tuple Count

**References**

1. <https://www.geeksforgeeks.org/java-program-for-quicksort/>
2. <https://examples.javacodegeeks.com/core-java/nio/bytebuffer/write-append-to-file-with-byte-buffer/>
3. <https://www.geeksforgeeks.org/bytebuffer-get-method-in-java-with-examples/>
4. <http://rosettacode.org/wiki/Binary_search#Java>
5. <https://en.wikipedia.org/wiki/Sorting_algorithm>
6. <https://crunchify.com/increase-eclipse-memory-size-to-avoid-oom-on-startup/>
7. <https://github.com/sagarvetal/ADB_Project_1_TPMMS>
8. <https://howtodoinjava.com/java/io/how-to-check-if-file-exists-in-java/>
9. <https://www.javadevjournal.com/java/java-copy-file-directory/>
10. <https://mkyong.com/java/how-to-delete-directory-in-java/>
11. <https://www.tutorialspoint.com/how-to-create-a-new-directory-by-using-file-object-in-java>
12. <https://www.tutorialspoint.com/java/io/file_isfile.htm>
13. <http://www.mathcs.emory.edu/~cheung/Courses/554/Syllabus/4-query-exec/2-pass=TPMMS.html>
14. <http://www.mathcs.emory.edu/~cheung/Courses/554/Syllabus/4-query-exec/TPMMS=join2.html>