**COMP6521-ADVANCED DATABASE TECHNOLOGY AND APPLICATIONS FALL 2018**

**Project Report- Two Phase Mulitway Merge-Sort**

**Index**

**COMP 6521-FALL 2018**

**Project Report- Two Phase Mulitway Merge-Sort**

**Group Members:**


| Name | Admit Number |
|---|---|
| Amit Sachdeva | 40084627 |
| Pratheek Narula | 40091466 |
| Sagar Bhatia | 40076907 |
| Sibil Joe Kurian | 40070864 |


**DESCRIPTION OF CLASSES AND FUNCTIONS** (mergeClass)

In the project there is only one class called mergeClass. This class is used to perform two functions, intialSort() and mergeOperation()

**intialSort():** This method performs the phase-1 of this project .

Task

- Fetching the total number of tuples and size of the main memory
- Calculating the size of temporary file.
- Calling phase2

**mergeOperation():** This method performs phase-2 of this project.

Task

- Calculating the buffer size (limit of the multidimensional array)
- Sorting the multidimensional array
- Writing it into the output file

**ALGORITHM OF OUR PROGRAM**

**Pre-Requisite(Two-Phase Multiway Merge Sort)**

Consider a very large relation file R and limited memory M, which can be sorted in two phases.

Phase1: Divide the file into K chunks of M blocks and sort each chunk individually using the M buffers. Write the sorted chunks into temporary files

Phase2: Merge the sorted sub list of phase1, there should be M-1 sorted temporary files. Read the K sorted chunks and merge it into a single output buffer.

**The Two Phase Multiway Merge-sort works in two phases described as follows:-**

**Input File**

In this project we have a large relation file, first line of the file contains the number of records and it is followed by the amount of main memory. Second line, is left blank and from the third line tuple starts, which need to be sorted.

**Phase1**: During the initial phase, total number of tuples and amount of main memory are fetched from the input file and stored into variables named 'total' and 'mm' of type integer. Tuples which are fetched from the file are stored into an array named 'arr[]', where the limit of the array is calculated, by the formula **(((mm)main memory - 3 ) *10^6)/4**, this result is kept in a variable 'loa' of type integer. The limit of the array is checked with a variable 'count', when the count reaches the array limit, reading of the tuples from the file is stopped. In the next step, array is sorted and written into a temporary file. Input file will be divided on the basis of the formula mentioned above into many temporary file containing tuples of equal size.

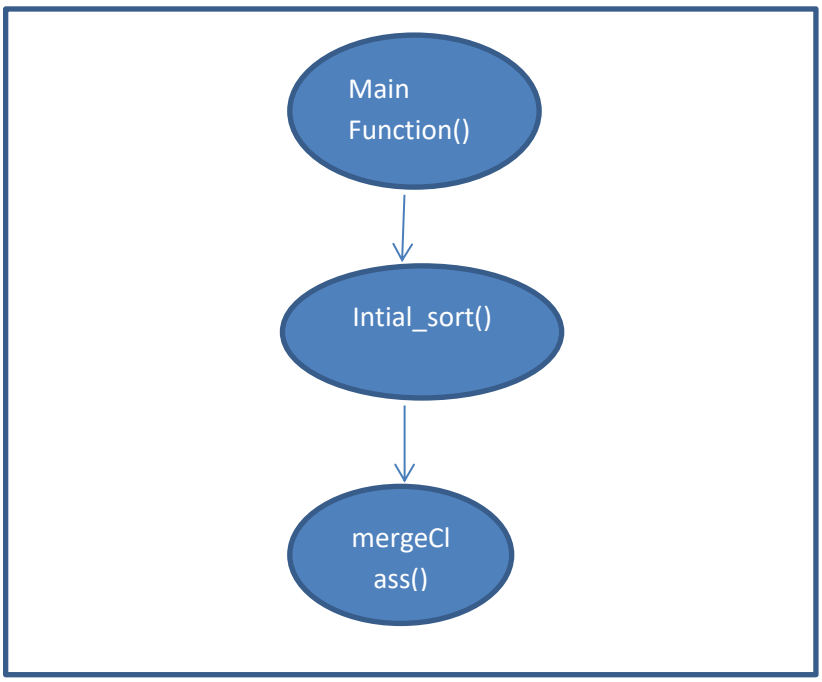
At last, phase2 is called with the following arguments total, mm and n. Here total stands for the whole number of tuples, mm is the amount of main memory and n is the total number of temporary files created

**Phase2**: Initially, number of buffer elements are measured using the formula **(((mm-3)*10^6)/n+1)/4** and it is stored in the variable 'b_elements' which is of type integer. Here, a multidimensional array is declared with the limit of temporary files plus one for the output file and the second index with the limit of buffer elements. Count of files 'ncf ' is declared to make sure all the temporary files data are gone to the multidimensional array. An array 'p' with the limit, number of files is declared to keep the pointer of the multidimensional array. When the last row of the multidimensional array is filled it is written to the output file.

## PROGRAM ARCHITECTURE

**Flow of control and program architecture**

**MergeClass**




**Sequence of execution**

**Mainfunction() –> intialsort()->mergeClass()**

## DIFFICULTIES FACED

**1. Scaling the main memory for size of operation (Under the condition-Meeting the wall clock)**

Solution: The code is tested with different types of test cases. These test cases and results are specified in table. Analysis of test cases lead to the finding that, as the size of operation increases time taken to sort the input file reduces.

**2. Space and time complexity (Under the condition-Meeting the wall clock)**

Solution: To optimize the program and for getting a better space and time complexity, it was found that main memory space must be divided on the basis of giving more space for operations and less space for the data. This can be verified from the test cases.

**3. Restricting the size of temporary file**

Solution: The size of the temporary file should not exceed 1 million tuples as the read operation will consume a lot of time.

**4. Finding the proper data structure in phase 2.**

Solution: We made a comparison of three data structures such as linked list, array list and arrays for selecting the phase 2 data structure. The final data structure we selected was array as, the rest of the two were taking too much memory when compared to arrays.

**TEST CASES**

| No .of Tuples | Allowed Memory Space | Operation Memory Size | Time taken |
|---|---|---|---|
| 10million | 10mb | 4mb | 2min |
| 10million | 10mb | 3mb | 4.30min |
| 10milion | 10mb | 2mb | 5.20min |
| 10million | 5mb | 3mb | 3.05min |
| 10million | 5mb | 2mb | 5.23min |
| 10million | 7mb | 3mb | 4.15min |
| 10million | 7mb | 4mb | 1.27min |
| 10million | 7mb | 2mb | 7.10min |
| | | | |
| 1million | 10mb | 4mb | 5sec |
| 1million | 10mb | 3mb | 5sec |
| 1million | 10mb | 2mb | 5sec |
| 1milion | 5mb | 3mb | 7sec |
| 1million | 5mb | 3mb | 26sec |
| 1million | 7mb | 2mb | 17sec |
| 1million | 7mb | 3mb | 15sec |
| 1million | 7mb | 4mb | 8sec |
| | | | |
| 1.5million | 10mb | 4mb | 16sec |
| 1.5million | 10mb | 3mb | 24sec |
| 1.5million | 10mb | 2mb | 1.min10sec |
| 1.5million | 5mb | 3mb | 13sec |
| 1.5miilion | 5mb | 2mb | 42sec |
| 1.5million | 7mb | 2mb | 1min30sec |
| 1.5million | 7mb | 3mb | 40sec |
| 1.5million | 7mb | 4mb | 14sec |
| | | | |
| 3million | 10mb | 4mb | 42sec |

| | | | |
|---|---|---|---|
| 3million | 10mb | 3mb | 2min |
| 3million | 10mb | 2mb | 5min |
| 3million | 5mb | 3mb | 28sec |
| 3million | 5mb | 2mb | 1.41min |
| 3million | 7mb | 2mb | 3.05min |
| 3million | 7mb | 3mb | 1.09min |
| 3million | 7mb | 4mb | 16sec |

**CONCLUSION**

Analysis of the table shows that as the operation memory is being increased, the time taken for execution gets reduced drastically in many of the cases. This trend helped to arrive at the aforementioned conclusion and to plan the algorithm accordingly.