PROJECT REPORT

IMPLEMENTING A SORTED FILE

MILESTONE 1:

THE BIGQ CLASS

Name: Gloria Gilbert Nazareth
UFID: 8221-8035
GatorLink: gnazareth@ufl.edu

Steps to compile and execute:
1. make ./test.out : this command builds the program and creates the required object files
2. ./test.out: this will execute the program, and ask for user input in order to perform required functionality.
3. ./runTestCases.sh: this will run the test cases mentioned in the file runTestCases.sh and will write the ouput to output1.txt
4. Make ./gtest.out : this command will build the program and create the required object files
5. ./runGtest.sh : this will execute the gtest. We are executing it through a script file, as the yy_parse() function used to test sorting and priority queue requires user input.
   [PS: Bin files need to be present before compiling the code, else it will result it error]

Description of the Implemented Function:
The BigQ class has the following functions:

1. BigQ (Pipe &in, Pipe &out, OrderMaker &sortorder, int runlen)
   This is the constructor of the class. It declares a worker pthread and starts it. It also passes the workerRoutine to the pthread to perform the necessary Two Pass Multiple Merge (TPMM) algorithm. The constructor terminates if the runlen is zero.

2. void* WorkerRoutine(void* arg)
   This function is called when the pthread is declared. This procedure is responsible for the TPMM algorithm. The function calls two other functions divideRuns and mergeRuns needed for the algorithm.

3. void divideRuns(Pipe *in, int runlen, map<int,Page*>& overflow, OrderMaker* sortorder, File& f, int& pageCount)
   This function reads the records from the in pipe and writes it to pages. Once a page is full, it appends it to the page vector by calling the addRecords function. It also counts the number of pages written. Once the number of pages written is equal to runlength (runlen) it calls the sortRunlens method and then writes the sorted pages in the runlen to a temporary file.

4. void addRecords(vector<Record*>& recVector, vector<Page*>& pageVector)
   This function adds the records from the pages in the pageVector to the recordVector.

5. int sortRunlens(vector<Record*>& recVector, OrderMaker* sortorder)
   This function sorts the records present in the record vector using the sortorder comparator.

6. int writeSortedRun(map<int,Page*>& overflow, File& f, vector<Record*> recVector, int runlen, int& pageCount)
   This function writes the sorted records in the record vector onto pages and appends it to a file. It also checks for overflows and maps those pages in a overflow map, which is used in merging.

7. void mergeRuns(int pageCount, Pipe* out, int runlen, File& f, OrderMaker* sortorder, map<int,Page*> overflow)
This function merges the sorted runlens to create a sorted file. It does this using a priority queue. The algorithm used is similar to K-way merging of lists. It calculates the number of runs by dividing the total pages by run length. A Map (recRunIndexMap) is used to map the records to a run number. An array (pageArray) is used to map the current page number to a run number. After all the records are sorted, i.e the queue is empty, the out pipe is shutdown.
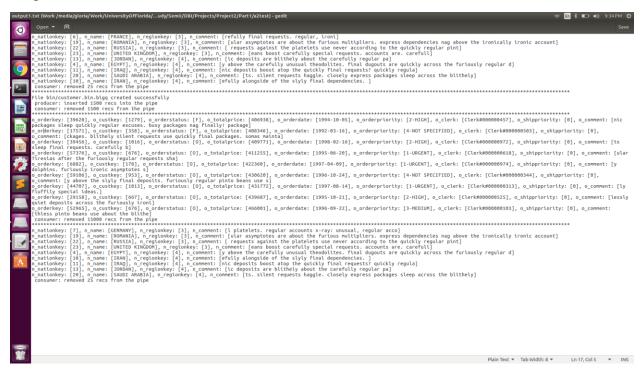
Output:

10 Mb data:

Console:
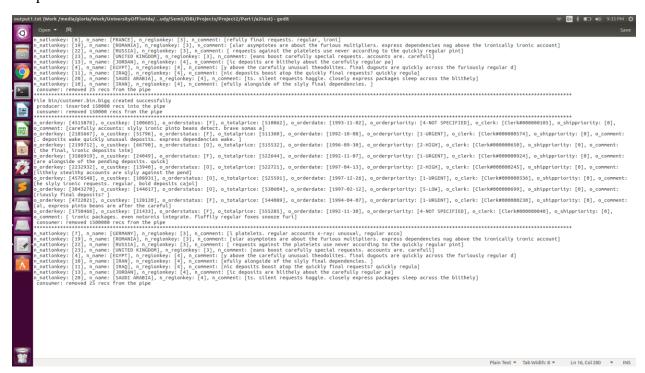

Output1.txt:

1 GB data:

Console:

Output1.txt:

Gtest Results:

1. TEST(writeSortedRuns,test1)
   This function will read records from a file, write it to a record vector. It then tests the writeSortedRun function in BigQ class. Here we check it for a file with records less than a single page.

2. TEST(writeSortedRunsMultiple,test2)
   This function will read records from a file, write it to a record vector. It then tests the writeSortedRun function in BigQ class. Here we check it for a file with records more than a single page.

3. TEST(sortRun, test3)
   This function tests the sorting of a records vector. Here, we read the records from an unsorted bin file and a sorted bin file and push it to two different record vectors. We then sort the vector containing the records of the unsorted bin file using the SortRunlens function of BigQ. After this we compare the records one by one. The assert equates to 0 if unsorted record vector is equal to the sorted record vector. Else 1.

4. TEST(checkPriorityQueue, test4)
   This function checks if the priority queue has the records in a sorted manner. This function tests the sorting of a records vector. Here, we read the records from an unsorted bin file and a sorted bin file and push it to two different record vectors. We then create a priority queue with a particular Merge_sort comparator. It then pops the records one by one and we cross check it with the sorted record vector. If it exactly matches it returns 0, else 1.