ASSIGNMENT 4

SORT MERGE JOIN AND HASH JOIN

Name: Chirag Shilwant – 2020201061

A. Configuration of the System

- **Processor**: Intel(R) Core (TM) i7-8550U CPU @ 1.80GHz
- **Memory**: 7.7 GB
- **Hard disk**: 1.8 GB left
- **OS**: Ubuntu 20.04.1 LTS

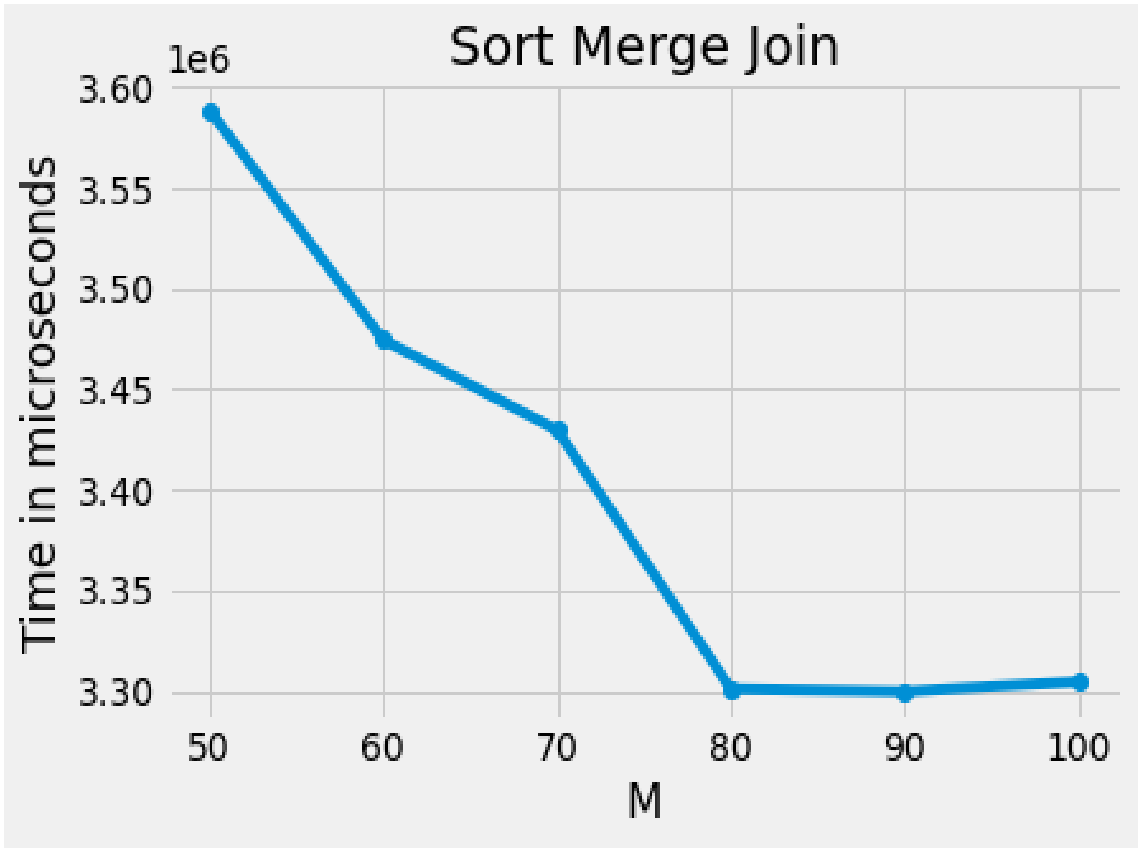
B. INPUT DATA FILES

Input files inputR.txt and inputS.txt were created with 50000 records (50 thousand records each), and sort merge and hash join procedures were performed, with the times measured.

1. Sort Merge Join

During the sort merge join, we created M blocks in each intermediate file. These files were sorted in the middle. The common keys were extracted and written to the output file after the files were opened. The files could be located anywhere.

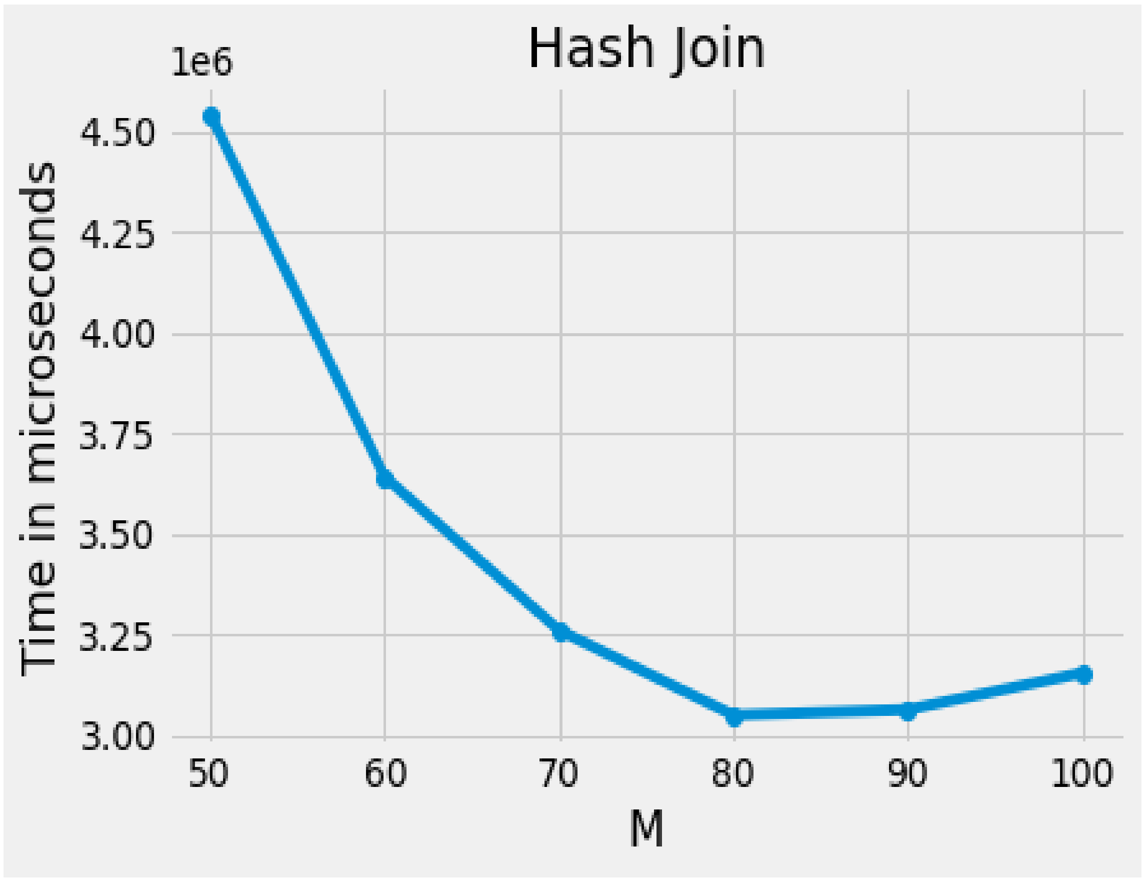
M	Number of Rows	Time Taken (in microseconds)
50	49846	3588765
60	49846	3474455
70	49846	3429744
80	49846	3300998
90	49846	3299734
100	49846	3304456



2. Hash Join

The intermediate files in a hash join are created in such a way that the same hashed keys are placed in each intermediate file. As a consequence, we create about M-1 intermediate files. The array is then loaded with the smaller data among R and S intermediate files, followed by the join calculation and storage in the output register.

M	**Number of Rows**	**Time Taken (in microseconds)**
50	49846	4541784
60	49846	3640387
70	49846	3258918
80	49846	3048207
90	49846	3061738
100	49846	3152514



C. Conclusion

Finally, Sort Merge join tends to function best for small M values; however, if a join key is repeated more than 100 times, we may need to hold a temporary copy of all the matches to match the join key (a version of which has been implemented in the code provided). Hash Join functions well for big M values. However, due to its dependency on the hashing functions used, hashing is vulnerable to failure for certain datasets. To finish this assignment, I used a rolling hash function.