Assignment - 3
COL362

Ashok Mali (2019EE30561)
Ayush Verma (2019MT60749)

➢ *Strategies & Techniques Used by Us*

- We used Radix Sort for doing in-memory sorting of the initial runs. For Small inputs, there is no significance in performance so we kept it. But when we used it on a large dataset, we saw significant performance differences and it is for some reason slowing the process. So we removed it and decided to keep the inbuilt sorting algorithm (sort()).

- During the merge pass, reading in each run one block at a time leads to a large number of seeks. So to reduce the number of seeks, we decide to read or write a larger number of blocks at a time and allocated equal buffer memory space to all input runs and output, in accordance with the memory available.

- We also used some utility/helper functions to facilitate some of the intermediate steps for us. Those are :

Name of function	Description
create_initial_runs	Generates sorted initial runs according to the buffer memory available.
intermediate _merge	It merges the previous sorted runs taking at most k runs at a time. If total_runs generated by the previous runs is less than <=k then the next merge is written directly to the output.

- ❖ We used a priority queue for the k-way merging step and stored the first element of every sorted run in it and then wrote the minimum of the queue in the output and removed it from the queue and replaced it with the next element from that run.

➢ *Testing & Verification*

We tested the external K-way merge sort program on datasets and files of different sizes and types and found the following performance results.

⇒ The intermediate & final output is verified to be correct and the mid-stopping function (num_merges != 0) is also working correctly as asked in the assignment.

⇒ The running performance results are as follows:

Dataset	Time Taken (for k = 16)
english-subset.txt (~21 MB)	~ 3 sec
random.txt (~500 MB)	~ 12 sec
Random_strings.txt (~10 GB)	~ 4 min 45 sec

Here, the *english-subset.txt* and *random.txt* files are provided by the professor for testing and *Random_strings.txt* is made by us by running a Python script which produces random strings of size <= 1024 bytes and containing non-control ASCII characters.

—----------XXX—----------XXX—----------