COL362 - Assignment 3

Aman Kumar - 2019CS10324

Design choices

In the first step, the input file is split into many runs. The number of runs is determined by the size of the input file as well as the amount of RAM that is available. The quick sorting method is then used to each run. In the second step, k separate runs are picked out and then combined into one. Every string that contains exactly k runs is appended in a sorted fashion to the output buffer. After the output buffer has been written to an intermediate file, we will once again begin reading strings from the selected k runs and continue doing so until all of the runs have been cleared of their contents. The following k runs will be chosen for the merging stage when this one is finished. In the end, we will get a single output file that is already arranged in the correct order.

Optimizing Disk I/O

We read the entirety of the input strings in a single run, and then we broke the data up into many runs so that we could limit the amount of disc I/Os.
Again, as we are writing the merged output, we are first saving the data in the output buffer, and then when the buffer is full, we are writing it into the output file. This helps us limit the number of disc I/O operations that are required.

Data-structure

> vectors are used.
> I also tried one implementation with inbuilt priority queue but final implementation using vectors is faster.

Observed performance over released datasets

random.list:

2	237.9s

8	156.3s
16	91.5s

english.list:

2	19.4s
8	14.9s
16	12.6s