Assignment 3

Sreemanti Dey
Aryan Dua

April 2023

1 Algorithm

We divided the algorithm into 2 steps: In the first step, we form the temp.0.x files, which are the runs, as defined in external merge-sort algorithm. In the second step, we bring the temp.i.x files from memory and merge them k at a time, where k is given as the input. This merging is done using a priority queue such that each time, we pop off an element from the priority queue, we add the next element to the priority queue, this ensures only k elements stay in the queue at a time, and hence retrieval of smallest element is fastest and only takes log(k) time.

2 Data structures used for in-memory sorting of strings

We tried various data structures for in place sorting of strings. We used trie, merge sort, quicksort and insertion sort for in place sorting of strings but the best we found was via quicksort, which is exactly the function used by std::sort in C++ STL.

3 Strategy used for optimizing disk I/O

We tried various compression and decompression techniques to help in optimizing disk i/o performance, we tried using zlib but zlib uses a linking flag which will not be provided at compile time, hence we did not proceed with this. We used stream operators to read and write data easily and we also used seekg to get to the desired position in the file where we had stopped earlier, this helped in improving the time taken to run the code.

4 Datasets used for performance evaluation

We generated a few large datasets by the following method. We copied the file random.txt and some other larger files several times to get files of very large

size. We ran this python file to generate more random data.

```
f1 = open("big_random.txt", "r")
f2 = open("big_big_big_big_random.txt", "a")
for i in range(1000000):
    x = f1.readline()
    f2.write(x)
    f2.write(x[-2::-1])
    f2.write(x[1:])
    f2.write(x[::2])
    f2.write(x[-2::-2])
```

5 Performance evaluation

We generated a Virtual Machine Ubuntu 20.04 with 1 CPU and 1 GB of RAM and ran our code on that machine to check the actual times our code would take if run on test conditions.

5.1 on released datasets

The following table shows the avg time taken by our code to run on different dataset sizes (averaged over k=2,8,16)

Dataset name	Time taken(s)
english-subset.txt	0.324
random.txt	3.834

5.2 on our generated datasets

The following table shows the avg time taken by our code to run on different dataset sizes (averaged over k=2,8,16)

Dataset size (GB)	Time taken(s)
0.512	3.834
2.90	18.563
10.34	170.3432
20.67	380.4341