Submitted by- Isha Gupta(2020BTechCSE035)

QUESTION:

As discussed in lab regarding implementation of External Merge Sort for a large file containing random huge numbers up to 50000. Considering RAM to accommodate only 2000 elements at a time. Apply efficient External Merge sort.

SOLUTION

```
package com.ds;

import java.io.*;
import java.util.*;

public class ExternalMergeSort {
    private static final int CHUNK_SIZE = 2000;

    public static void sort(String inputFile, String outputFile) throws IOException {
        // Divide the input file into smaller chunks
        int numChunks = (int) Math.ceil((double) new File(inputFile).length() /
(double) CHUNK_SIZE / 4);
        List<String> chunkFiles = new ArrayList<>();
        BufferedReader br = new BufferedReader(new FileReader(inputFile));
        for (int i = 0; i < numChunks; i++) {
            List<Integer> chunk = new ArrayList<>();
            String chunkFile = "chunk" + i + ".txt";
            chunkFiles.add(chunkFile);
            PrintWriter pw = new PrintWriter(new FileWriter(chunkFile));
            String line;
            int count = 0;
            while ((line = br.readLine()) != null && count < CHUNK_SIZE) {
                chunk.add(Integer.parseInt(line));
                count++;
            }
            Collections.sort(chunk);
            for (int num : chunk) {
                pw.println(num);
            }
            pw.close();
        }
        br.close();

        // Merge the sorted chunks together
        PriorityQueue<BufferedReader> queue = new PriorityQueue<>(numChunks, new
Comparator<BufferedReader>() {
            public int compare(BufferedReader br1, BufferedReader br2) {
                try {
                    int num1 = Integer.parseInt(br1.readLine());
                    int num2 = Integer.parseInt(br2.readLine());
                    return Integer.compare(num1, num2);
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
        });
        for (String chunkFile : chunkFiles) {
            BufferedReader brChunk = new BufferedReader(new FileReader(chunkFile));
            queue.add(brChunk);
        }
        PrintWriter pw = new PrintWriter(new FileWriter(outputFile));
        while (!queue.isEmpty()) {
            BufferedReader brChunk = queue.poll();
            String line = brChunk.readLine();
            if (line != null) {
                pw.println(line);
                queue.add(brChunk);
            } else {
                brChunk.close();
            }
        }
        pw.close();

        // Delete the temporary chunk files
        for (String chunkFile : chunkFiles) {
            new File(chunkFile).delete();
        }
    }

    public static void main(String[] args) throws IOException {
        String inputFile =
"C:\\Users\\gupta\\IdeaProjects\\LeetCodePractice\\src\\com\\ds\\input.txt";
        String outputFile = "output.txt";
        sort(inputFile, outputFile);
    }
}
```