**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 1 з дисципліни

«Проектування алгоритмів»

„ **Проектування і аналіз алгоритмів зовнішнього сортування**”

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-41 Хижняк А. С.*

Київ 2025

**Лабораторна робота №1**

**«Проектування і аналіз алгоритмів зовнішнього сортування»**

**Варіант 25**

Мета роботи – вивчити основні алгоритми зовнішнього сортування та способи їх модифікації, оцінити поріг їх ефективності.

***Завдання***



***Псевдокод алгоритму***

**початок**(ExternalMergeSort)

sorted := хибно

**поки** (!sorted)

**повторити**

Distribute(a, b, c)

sorted := Merge(a, b, c)

seriesSize \*= 2

**все повторити**

**кінець**

**початок**(Distribute(File a, File b, File c))

n := 0

inFirstSeries := істинно

**поки**((line := a.readLine()) != null)

**повторити**

**якщо** (n == seriesSize)

inFirstSeries := !inFirstSeries

n := 0

**все якщо**

**якщо**( inFirstSeries )

**то**

b.println(line)

**інакше**

c.println(line)

**все якщо**

n++

**все повторити**

**кінець**

**початок**(Merge(File a, File b, File c))

seriesCount := 0

lineB := b.readLine

lineC := c.readLine

**поки**(lineB != null || lineC != null)

**повторити**

seriesSizeB := 0

seriesSizeC := 0

seriesCount++

**поки**((seriesSizeB < seriesSize && lineB != null) || (seriesSizeC < seriesSize && lineC != null))

**повторити**

**якщо**(seriesSizeB >= seriesSize || lineB == null)

**то**

a.println(lineC)

lineC := c.readLine

seriesSizeC++

**інакше якщо**(seriesSizeС >= seriesSize || lineС == null)

**то**

a.println(lineB)

lineB := b.readLine

seriesSizeB++

**інакше**

keyB := toInt(lineB.firstString)

keyC := toInt(lineC.firstString)

**якщо**(keyB >= keyC)

**то**

a.println(lineB)

lineB := b.readLine

seriesSizeB++

**інакше**

a.println(lineC)

lineC := c.readLine

seriesSizeC++

**все якщо**

**все якщо**

**все повторити**

**все повторити**

**повернути** (seriesCount == 1)

**кінець**

***Програмна реалізація алгоритму***

(Базова версія)

import java.io.\*;

import java.util.Scanner;

public class ExternalMergeSort{

private static int seriesSize = 1;

public static void ExternalSort(File a, File b, File c){

boolean sorted = false;

while(!sorted){

Distribute(a, b, c);

sorted = Merge(a, b, c);

seriesSize \*= 2;

}

}

public static void Distribute(File a, File b, File c){

boolean inFirstSeries = true;

String line;

int n = 0;

int seriesCount = 0;

try (

BufferedReader reader = new BufferedReader(new FileReader(a));

PrintWriter writerB = new PrintWriter(new FileWriter(b));

PrintWriter writerC = new PrintWriter(new FileWriter(c))

) {

while ((line = reader.readLine()) != null) {

if(n == seriesSize){

inFirstSeries = !inFirstSeries;

n = 0;

seriesCount++;

}

if (inFirstSeries) {

writerB.println(line);

}else writerC.println(line);

n++;

}

if(n > 0) {

seriesCount++;

}

} catch (Exception e){

throw new RuntimeException();

}

}

public static boolean Merge(File a, File b, File c){

int seriesCount = 0;

try (

PrintWriter writer = new PrintWriter(new FileWriter(a));

BufferedReader readerB = new BufferedReader(new FileReader(b));

BufferedReader readerC = new BufferedReader(new FileReader(c));

) {

String lB = readerB.readLine();

String lC = readerC.readLine();

while(lB != null || lC != null) {

int seriesSizeB = 0, seriesSizeC = 0;

seriesCount++;

while((seriesSizeB < seriesSize && lB != null) || (seriesSizeC < seriesSize && lC != null)){

if(seriesSizeB >= seriesSize || lB == null){

writer.println(lC);

lC = readerC.readLine();

seriesSizeC++;

} else if(seriesSizeC >= seriesSize || lC == null){

writer.println(lB);

lB = readerB.readLine();

seriesSizeB++;

} else {

int keyB = Integer.parseInt(lB.trim().split("\\s+")[0]);

int keyC = Integer.parseInt(lC.trim().split("\\s+")[0]);

if(keyB >= keyC){

writer.println(lB);

lB = readerB.readLine();

seriesSizeB++;

} else {

writer.println(lC);

lC = readerC.readLine();

seriesSizeC++;

}

}

}

}

return (seriesCount == 1);

} catch (Exception e){

e.printStackTrace();

throw new RuntimeException();

}

}

}

(Модифікована версія)

import java.io.File;

import java.io.PrintWriter;

import java.io.FileWriter;

import java.io.BufferedReader;

import java.io.FileReader;

import java.util.ArrayList;

import java.util.List;

import java.nio.file.Files;

import java.nio.file.StandardCopyOption;

public class ModifiedExternalMergeSort{

private static long seriesSize = 100L \* 1024 \* 1024;

public static void ExternalSort(File a, File b, File c){

InitialSort(a);

boolean sorted = false;

while(!sorted){

Distribute(a, b, c);

sorted = Merge(a, b, c);

//seriesSize \*= 2;

}

}

private static int GetKey(String line){

return Integer.parseInt(line.substring(0, line.indexOf(' ')));

}

public static void InitialSort(File a) {

File tmp = new File("temp.txt");

try (

BufferedReader reader = new BufferedReader(new FileReader(a));

PrintWriter writer = new PrintWriter(new FileWriter(tmp))

) {

List<String> list = new ArrayList<>();

String line;

long currentSize = 0;

while ((line = reader.readLine()) != null) {

list.add(line);

currentSize += estimateSize(line);

if (currentSize >= seriesSize) {

SortAndWrite(list, writer);

currentSize = 0;

}

}

if (!list.isEmpty()) {

SortAndWrite(list, writer);

}

} catch (Exception e) {

throw new RuntimeException(e);

}

try {

Files.move(tmp.toPath(), a.toPath(), StandardCopyOption.REPLACE\_EXISTING);

} catch (Exception e) {

throw new RuntimeException("Couldn`t replace file: " + e.getMessage(), e);

}

}

private static void SortAndWrite(List<String> list, PrintWriter writer) {

list.sort((s1, s2) -> {

int k1 = GetKey(s1);

int k2 = GetKey(s2);

return Integer.compare(k2, k1);

});

for (String s : list) {

writer.println(s);

}

writer.println("#");

list.clear();

}

public static void Distribute(File a, File b, File c){

try( BufferedReader reader = new BufferedReader(new FileReader(a));

PrintWriter writerB = new PrintWriter(new FileWriter(b));

PrintWriter writerC = new PrintWriter(new FileWriter(c));

){

boolean inFirstSeries = true;

String line;

while ((line = reader.readLine()) != null) {

if (inFirstSeries) {

writerB.println(line);

} else {

writerC.println(line);

}

if (line.equals("#")) {

inFirstSeries = !inFirstSeries;

}

}

}

catch (Exception e){

throw new RuntimeException(e);

}

}

public static boolean Merge(File a, File b, File c){

try( PrintWriter writer = new PrintWriter(new FileWriter(a));

BufferedReader readerB = new BufferedReader(new FileReader(b));

BufferedReader readerC = new BufferedReader(new FileReader(c));

) {

int seriesCount = 0;

String lineB = readerB.readLine();

String lineC = readerC.readLine();

while (lineB != null || lineC != null) {

seriesCount++;

while ((lineB != null && !lineB.equals("#")) ||

(lineC != null && !lineC.equals("#"))) {

if (lineC == null || lineC.equals("#")) {

writer.println(lineB);

lineB = readerB.readLine();

}

else if (lineB == null || lineB.equals("#")) {

writer.println(lineC);

lineC = readerC.readLine();

}

else {

int keyB = GetKey(lineB);

int keyC = GetKey(lineC);

if (keyB >= keyC) {

writer.println(lineB);

lineB = readerB.readLine();

} else {

writer.println(lineC);

lineC = readerC.readLine();

}

}

}

if (lineB != null && lineB.equals("#")) {

lineB = readerB.readLine();

}

if (lineC != null && lineC.equals("#")) {

lineC = readerC.readLine();

}

writer.println("#");

}

return (seriesCount == 1);

}

catch (Exception e){

throw new RuntimeException(e);

}

}

private static int estimateSize(String line) {

if (line == null) return 0;

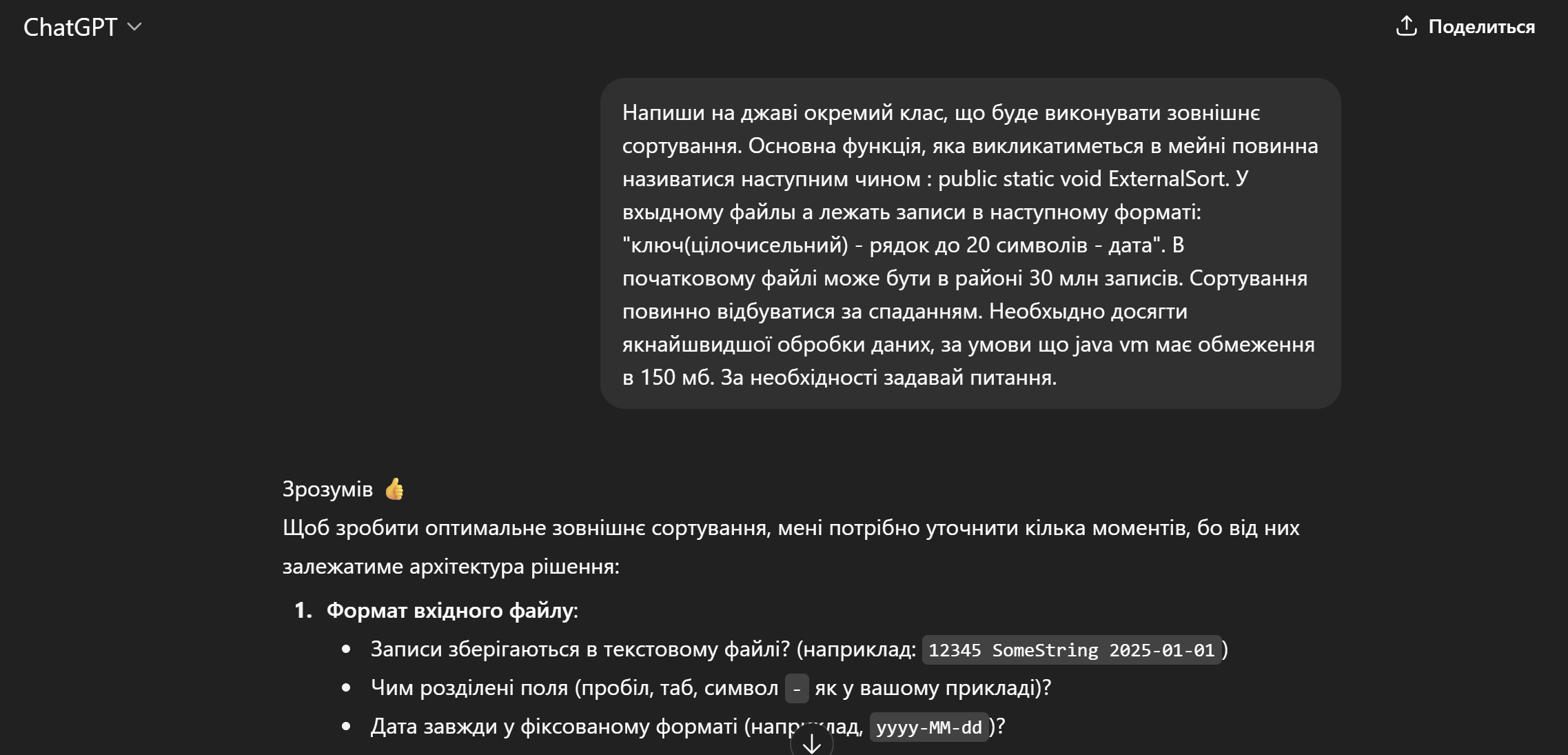
return line.getBytes().length + 40;

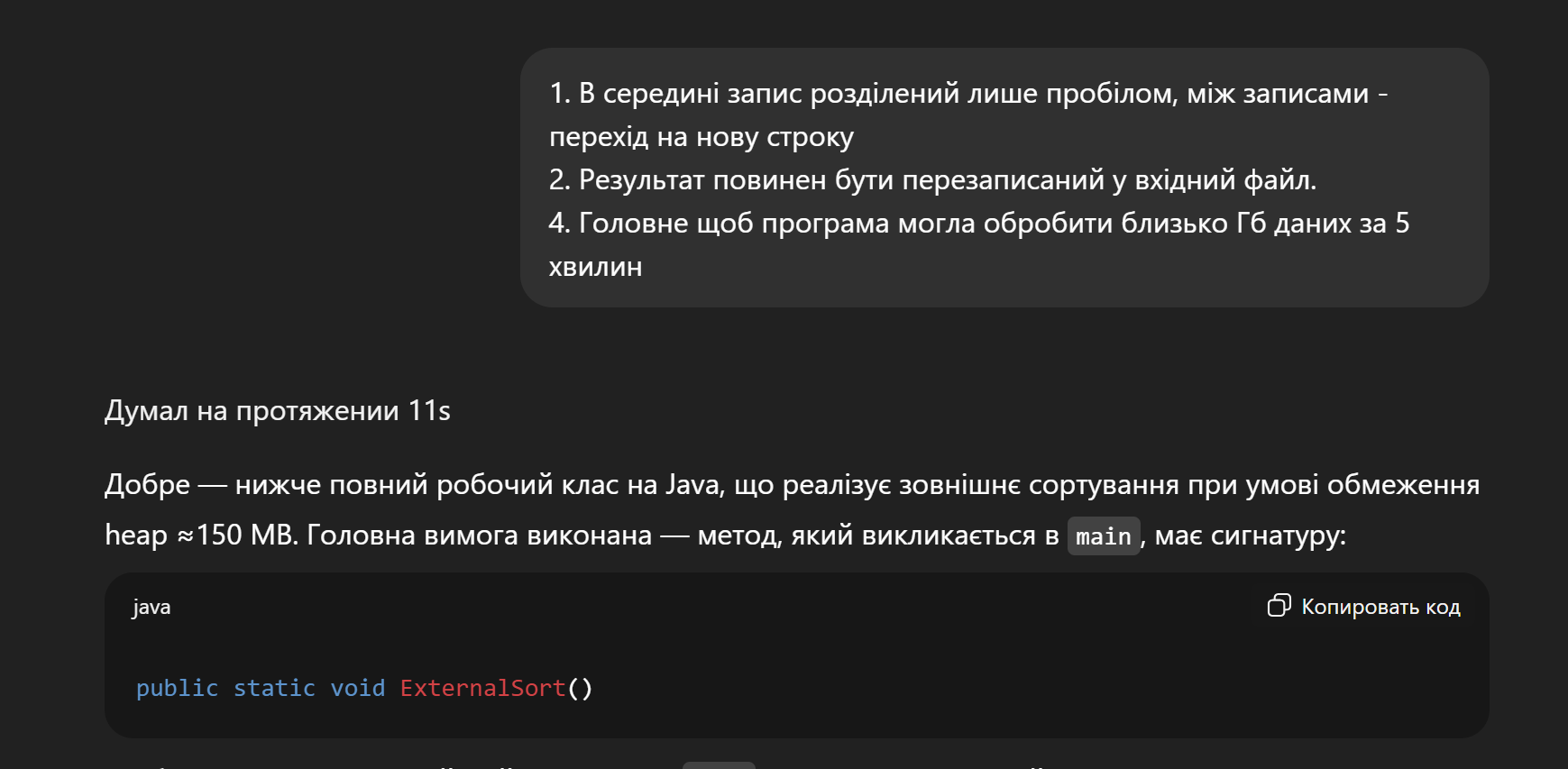
}

}

***Запити до чат-боту***

В першому запиті було описано суть завдання, а саме необхідність написати алгоритм зовнішнього сортування, використовуючи 150 Мб пам’яті, і досягти якнайкращої можливої швидкості.



Далі Chat GPT задав декілька навідних питань, перед початком роботи:  


В результаті роботи Chat GPT написав програму, яка зчитує з вхідного файлу серії по 100 Мб. Кожна серія після сортування записується в окремий файл. Після чого створюється черга з пріоритетом, в яку записується перший рядок з кожного файлу. Коли з черги дістається найбільший елемент і записується у вихідний файл, зчитується наступний рядок з того самого файлу, що й найбільший.

­

Порівняльна таблиця

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Спосіб сортування | 10 тисяч записів | 100 тисяч записів  (3 MB) | 1 мільйон записів  (30 MB) | 10 мільйонів записів  (300 MB) | 30 мільйонів записів  (900 MB) |
| Зовнішнє сортування злиттям | 252 ms | 1 sec 728 ms | 18 sec 481 ms | 4 min 6 sec 205 ms | 11 min 49 sec 522 ms |
| Модифіковане зовнішнє сортування злиттям | 76 ms | 230 ms | 2 sec 184 ms | 31 sec 675 ms | 1 min 58 sec 255 ms |
| Алгоритм згенерований чат-ботом | 86 ms | 185 ms | 1 sec 631 ms | 16 sec 988 ms | 48 sec 493 ms |

Висновок: Базова програма підходить для роботи з мінімальними витратами пам’яті, проте має дуже погану швидкість. Власноруч модифікована версія краще опрацьовує великі об’єми даних, оскільки вона попередньо впорядковує серії, що зменшує кількість ітерацій злиття. Написана чат-ботом програма добре справляється з великою кількістю вхідних даних, але створює дуже велику кількість допоміжних файлів, що є досить незручним.