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**”

Виконав(ла) *ІП-14 Качмар А.Д* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Головченко М.М.* _____
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

2 ЗАВДАННЯ

Згідно варіанту (таблиця 2.1), розробити та записати алгоритм зовнішнього сортування за допомогою псевдокоду (чи іншого способу за вибором).

Виконати програмну реалізацію алгоритму на будь-якій мові програмування та відсортувати випадковим чином згенерований масив цілих чисел, що зберігається у файлі (розмір файлу має бути не менше 10 Мб, можна значно більше).

Здійснити модифікацію програми і відсортувати випадковим чином згенерований масив цілих чисел, що зберігається у файлі розміром не менше ніж двократний обсяг ОП вашого ПК. Досягти швидкості сортування з розрахунку 1Гб на 3хв. або менше.

Рекомендується попередньо впорядкувати серії елементів довжиною, що займає не менше 100Мб або використати інші підходи для пришвидшення процесу сортування.

Зробити узагальнений висновок з лабораторної роботи, у якому порівняти базову та модифіковану програми. У висновку деталізувати, які саме модифікації було виконано і який ефект вони дали.

Таблиця 2.1 – Варіанти алгоритмів

№	**Алгоритм сортування**
1	Пряме злиття
2	Природне (адаптивне) злиття
3	Збалансоване багатошляхове злиття
4	Багатофазне сортування
5	Пряме злиття
6	Природне (адаптивне) злиття
7	Збалансоване багатошляхове злиття
8	Багатофазне сортування
9	Пряме злиття

10	Природне (адаптивне) злиття
11	Збалансоване багатошляхове злиття
12	Багатофазне сортування
13	Пряме злиття
14	Природне (адаптивне) злиття
15	Збалансоване багатошляхове злиття
16	Багатофазне сортування
17	Пряме злиття
18	Природне (адаптивне) злиття
19	Збалансоване багатошляхове злиття
20	Багатофазне сортування
21	Пряме злиття
22	Природне (адаптивне) злиття
23	Збалансоване багатошляхове злиття
24	Багатофазне сортування
25	Пряме злиття
26	Природне (адаптивне) злиття
27	Збалансоване багатошляхове злиття
28	Багатофазне сортування
29	Пряме злиття
30	Природне (адаптивне) злиття
31	Збалансоване багатошляхове злиття
32	Багатофазне сортування
33	Пряме злиття
34	Природне (адаптивне) злиття
35	Збалансоване багатошляхове злиття

3 ВИКОНАННЯ

3.1 Псевдокод алгоритму

3.1.1 Псевдокод не оптимізованого алгоритму

```
generateNumbers (long length, string fileName)
    writer = new BufferedWriter(fileName)
    for i = 0 to length do
        num = randomNum()
        writer.append(num)
    end for
end

splitFiles (int batch)
    createFiles(batch,FILE_B)
    reader  = new BufferedReader(filePath)
    currentFile = 0
    currentLine = reader.readLine()
    previousLine = currentLine
    series[] = [currentLine]
    while ( currentLine not null) do
        if (series.size() > 1 and previousLine > current)
            writeLines to currentFile
            series.clear()
            currentFile++
        if (currentFile >= batch)
            currentFile = 0
        series.add(currentElement)
        previous = currentLine
    end while
    writeLines to currentFile
end
```

```
mergeAllFiles (int batch)
    direction = true
    createFiles(batch, FILE_C)
    while (not fullyMerged(FILE_B) or not fullyMerged(FILE_C)) do
        readFrom = direction ? FILE_B : FILE_C
        writeTo = direction ? FILE_B : FILE_C
        mergeFiles(batch, writeTo, inputSize)
        if fullyMerged(writeTo)
          clearEmptyFiles(readFrom)
          break
        direction = not direction
    end while
    removeEmptyFiles(FILE_B)
    removeEmptyFiles(FILE_C)
end
```

```
mergeFiles (int batch, readFrom, writeTo)
    readers[] = getReaders(readFrom)
    clearFiles(batch, writeTo)
    elements[] = []
    writeToIndex = 0
    while (not isAllFilesRead(readers)) do
        minElement = ∞
        readerIndex = null
        for reader to readers
            line = readLine(reader)
            if line not null
                current = line
                last = elements.last
                if elements.empty or current>=last and current<minElement
                    minElement = current
                    readerIndex = reader.index
        end for
        if readerIndex is null
            elements.add(minElement)
            readers.nextLine
        else
            writeLinesToFile(elements, filePath)
            elements.clear
            writeIndex++
    end while
    writeLinesToFile(elements, filePath)
    closeAllReaders(readers)
end
```

3.1.2 Псевдокод оптимізованого алгоритму

splitFiles *(int batch, filePath)*

createFiles(batch,FILE_B)
reader = new BufferedReader(filePath)
writers[] = createWriters(FILE_B)
currentFile = 0
line = reader.readLine
while (line **not** null) **do**
writer = writers.get(currentFile)
writer.write(line)
currentfile++
end while
closeAllWriters
end

sortFiles *(int batch, filePath)*

for i = 0 to batch **do**
lines = reader.readAllLines
lines = sortLines(lines)
writeToFile(lines)
end for
end

```
mergeFiles (int batch, readFrom, writeTo)
    readers[] = getReaders(readFrom)
    finishedBuffers = 0
    while (finishedBuffers.size not readers.size) do
        minElement = ∞
        readerIndex = null
        for reader to readers
            line = readLine(reader)
            if line not null and line < minElement
                minElement = line
                readerIndex = reader.index
            if line is null
                finishedBuffers++
        end for
        if readerIndex is not null
            writer.write(minElement)
    end while
    closeAllReaders(readers)
    writer.close
end
```

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код не оптимізованого алгоритму

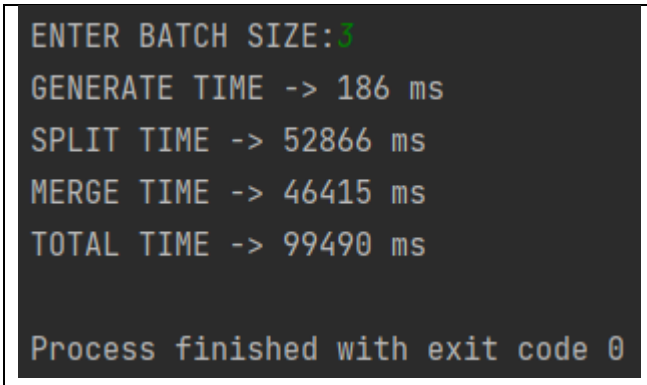
```
public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.print("ENTER BATCH SIZE:");
    int batch = scanner.nextInt();

    long startGen = System.currentTimeMillis();
    generateInputFile(SMALL_FILE, SMALL_FILE_PATH);
    long endGen = System.currentTimeMillis();
    printTime( method: "GENERATE", startGen, endGen);


    long startSplit = System.currentTimeMillis();
    splitFiles(batch);
    long endSplit = System.currentTimeMillis();
    printTime( method: "SPLIT", startSplit, endSplit);


    long startMerge = System.currentTimeMillis();
    mergeAllFiles(batch);
    long endMerge = System.currentTimeMillis();
    printTime( method: "MERGE", startMerge, endMerge);

    printTime( method: "TOTAL", startGen, endMerge);
}
```

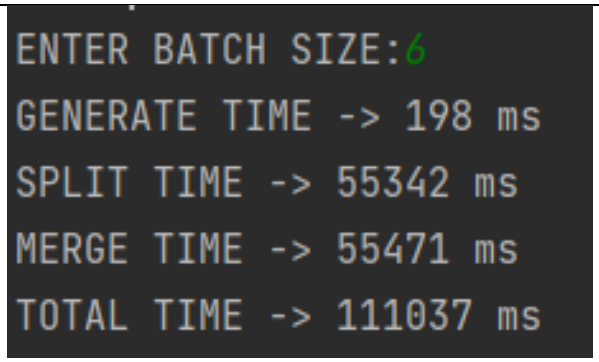
```
public static void generateInputFile(long length, String fileName) {
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(fileName,  append: true))) {
        for (int i = 0; i < length; i++) {
            String num = randomNumber();
            writer.append(num).append("\n");
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```java
public static void splitFiles(int batch) {
    try (BufferedReader bufferedReader = new BufferedReader(new FileReader(SMALL_FILE_PATH))) {
        createFiles(batch, FILE_B);

        int currentFile = 0;
        List<String> series = new ArrayList<>();

        String currentLine = bufferedReader.readLine();
        series.add(currentLine);
        String previous = currentLine;

        while ((currentLine = bufferedReader.readLine()) != null) {
            if (series.size() > 1 && !comparePrevious(previous, currentLine)) {
                writeToFileBulk(series, filePath: FILE_DIR + currentFile + FILE_B);
                series = new ArrayList<>();
                currentFile++;
            }
            if (currentFile >= batch) {
                currentFile = 0;
            }
            series.add(currentLine);
            previous = currentLine;
        }
        writeToFileBulk(series, filePath: FILE_DIR + currentFile + FILE_B);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
public static void mergeAllFiles(int batch) {
    try {
        boolean direction = true;
        createFiles(batch, FILE_C);
        while (!isFullyMerged(batch, FILE_B, inputSize) || !isFullyMerged(batch, FILE_C, inputSize)) {
            String readFrom = direction ? FILE_B : FILE_C;
            String writeTo = direction ? FILE_C : FILE_B;
            mergeFiles(batch, readFrom, writeTo);
            if (isFullyMerged(batch, writeTo, inputSize)) {
                clearFiles(batch, readFrom);
                break;
            }
            direction = !direction;
        }
        removeEmptyFiles(batch, FILE_B);
        removeEmptyFiles(batch, FILE_C);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```java
public static void mergeFiles(int batch, String readFrom, String writeTo) throws IOException {
    List<BufferedReader> readers = createReaders(batch, readFrom);
    clearFiles(batch, writeTo);
    List<String> elements = new ArrayList<>();
    int writeToIndex = 0;
    while (!isFilesRead(readers)) {
        Integer minElement = Integer.MAX_VALUE;
        Integer readerIndex = null;
        for (BufferedReader reader : readers) {
            String line = readLine(reader);
            if (line != null) {
                int current = Integer.parseInt(line);
                Integer last = elements.isEmpty() ? null : Integer.parseInt(elements.get(elements.size() - 1));
                if ((elements.isEmpty() || current >= last) && (current < minElement)) {
                    minElement = current;
                    readerIndex = readers.indexOf(reader);
                }
            }
        }
        if (readerIndex != null) {
            elements.add(String.valueOf(minElement));
            readers.get(readerIndex).readLine();
        } else {
            writeToFileBulk(elements, filePath: FILE_DIR + writeToIndex + writeTo);
            elements.clear();
            writeToIndex++;
            if (writeToIndex >= batch) {
                writeToIndex = 0;
            }
        }
    }
    writeToFileBulk(elements, filePath: FILE_DIR + writeToIndex + writeTo);
    closeReaders(readers);
}
```

Приклад роботи не оптимізованого алгоритму





Розмір файлу: 11,5 МБ	
Етап	**Час**
Генерація	0.186 sec
Розбиття	52.866 sec
Злиття	46.415 sec
Загальний час	**99.49 sec**

Розмір файлу: 11,5 МБ	
Етап	**Час**
Генерація	0.198 sec
Розбиття	55.342 sec
Злиття	55.471 sec
Загальний час	**111.037 sec**

3.2.2 Вихідний код оптимізованого алгоритму

```
public static void splitFiles(int batch, String filePath) {
    try (BufferedReader bufferedReader = new BufferedReader(new FileReader(filePath))) {
        createFiles(batch, FILE_B);
        List<BufferedWriter> writers = createWriters(batch, FILE_B);

        int currentFile = 0;
        String line;
        while ((line = bufferedReader.readLine()) != null) {
            BufferedWriter currentWriter = writers.get(currentFile);
            currentWriter.write( str: line + "\n");
            currentFile++;
            if (currentFile >= batch) {
                currentFile = 0;
            }
        }
        closeWriters(writers);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```
public static void sortFiles(int batch, String filePath) {
    try {
        for (int i = 0; i < batch; i++) {
            String path = FILE_DIR + i + filePath;
            BufferedReader bufferedReader = new BufferedReader(new FileReader(path));
            List<String> lines = bufferedReader.lines() Stream<String>
                    .map(Integer::parseInt) Stream<Integer>
                    .sorted().map(String::valueOf) Stream<String>
                    .collect(Collectors.toList());
            rewriteToFile(lines, path);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```java
public static void mergeAllFiles(int batch) {
    String writeTo = FILE_C;
    try {
        createFiles(batch, writeTo);
        mergeFiles(batch, FILE_B, writeTo);
        removeEmptyFiles(batch, FILE_C);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```java
public static void mergeFiles(int batch, String readFrom, String writeTo) throws IOException {
    List<BufferedReader> readers = createReaders(batch, readFrom);
    BufferedWriter writer = new BufferedWriter(new FileWriter( fileName: FILE_DIR + 0 + writeTo,  append: true));

    while (!isFilesRead(readers)) {
        int min = Integer.MAX_VALUE;
        Integer minIndex = null;
        for (BufferedReader reader : readers) {
            String line = readLine(reader);
            if (line != null && Integer.parseInt(line) < min) {
                min = Integer.parseInt(line);
                minIndex = readers.indexOf(reader);
            }
        }
        if (minIndex != null) {
            writer.write( str: min + "\n");
            readers.get(minIndex).readLine();
        }
    }
    closeReaders(readers);
    writer.close();
}
```

Приклад роботи оптимізованого алгоритму

<table>
<tr><th>Розбиття по 10 файлах</th><th>Розбиття по 5 файлах</th></tr>
<tr><td>

```
ENTER BATCH SIZE:10
GENERATE TIME -> 12603 ms
SPLIT TIME -> 17145 ms
SORT TIME -> 112774 ms
MERGE TIME -> 198743 ms
TOTAL TIME -> 341289 ms

Process finished with exit code 0
```

</td><td>

```
ENTER BATCH SIZE:5
GENERATE TIME -> 13991 ms
SPLIT TIME -> 20637 ms
SORT TIME -> 123082 ms
MERGE TIME -> 84809 ms
TOTAL TIME -> 242544 ms
```

</td></tr>
<tr><td>

Розмір файлу: 1GB	
Етап	**Час**
Генерація	12.603 sec
Розбиття	17.145 sec
Сортування	112.774 sec
Злиття	198.743 sec
Загальний час	**341.289**

</td><td>

Розмір файлу: 1GB	
Етап	**Час**
Генерація	13.991 sec
Розбиття	20.637 sec
Сортування	123.082 sec
Злиття	84.809 sec
Загальний час	**242.544**

</td></tr>
</table>

ВИСНОВОК

При виконанні даної лабораторної роботи було вивчено основні алгоритми зовнішнього сортування та способи їх модифікації. А саме було досліджено роботу алгоритму збалансованого багатошляхового злиття. Було реалізовано базову та оптимізовану версію даного алгоритму. Для кожної реалізації було проведено виміри часу виконання певних блоків алгоритму.

Базова реалізація алгоритму сортує вхідний файл розміром 11,5 MB в середньому за 1 хв 45 с.

Оптимізована реалізація алгоритму сортує вхідний файл розміром 1 GB в середньому за 4 хв 50 с.

Під оптимізації алгоритму було внесено наступні зміни

1. Вхідний файл порівно розбивається на під файли
2. Кожен з отриманих файлів сортується алгоритмом внутрішнього сортування
3. При злитті файлів одна серія = одному файлу адже усі елементи впорядковані. Тому злиття відбувається за один прохід в один результуючий файл

Число проходів в алгоритмі збалансованого багатошляхового злиття оцінюється як O(log n) де n – кількість елементів вхідного файлу. Отже було вивчено основні алгоритми зовнішнього сортування та способи їх модифікації.

КРИТЕРІЇ ОЦІНЮВАННЯ

У випадку здачі лабораторної роботи до 09.10.2022 включно максимальний бал дорівнює – 5. Після 09.10.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- псевдокод алгоритму – 15%;
- програмна реалізація алгоритму – 40%;
- програмна реалізація модифікацій – 40%;
- висновок – 5%.