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-13 Нещерет В. О.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов О. О.* _____
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ 3

2 ЗАВДАННЯ 4

3 ВИКОНАННЯ 6

3.1 Псевдокод алгоритму 6

3.2 Програмна реалізація алгоритму 8

3.2.1 Вихідний код 8

ВИСНОВОК 14

КРИТЕРІЇ ОЦІНЮВАННЯ 15

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

Основний алгоритм:

1. Роздітити вхідний файл на серії та записати їх у файли B.
    1.1. Створити список set, що позначатиме серію
    1.2. Ініціалізувати змінну counter = 0
    1.3. ПОКИ не кінець вхідного файла
        1.3.1. Зчитати наступний елемент elem
        1.3.2. ЯКЩО set пустий, або elem >= set.last()
            1.3.2.1. Додати elem до set
        1.3.3. ІНАКШЕ
            1.3.3.1. Записати set до файла B з номером counter
            1.3.3.2. Очистити set та записати у нього elem
            1.3.3.3. Збільшити counter на 1
        1.3.4. КІНЕЦЬ ЯКЩО
    1.4. КІНЕЦЬ ПОКИ
2. Ініціалізувати змінну BC = true
3. ПОКИ не відсортовано повністю
    3.1. ЯКЩО BC = true
        3.1.1. Викликати функцію merge для злиття файлів B у файли C
    3.2. ІНАКШЕ
        3.2.1. Викликати функцію merge для злиття файлів C у файли B
    3.3. КІНЕЦЬ ЯКЩО
4. BC = ! BC
5. КИНЕЦЬ ПОКИ
6. ЯКЩО вихідний масив у файлі B1
    6.1. Скопіювати дані з файлу B1 у вихідний файл
7. ІНАКШЕ
    7.1. Скопіювати дані з файлу C1 у вихідний файл
8. КІНЕЦЬ ЯКЩО
9. Очистити вміст допоміжних файлів

Алгоритм злиття:

1. Очистити вміст вихідних файлів
2. Ініціалізувати змінну counter = 0
3. ПОКИ не закінчились дані у всіх вхідних файлах
    3.1. Ініціалізувати масив для відсортованих чисел sortedList
    3.2. Ініціалізувати масив tempList
    3.3. ЦИКЛ і ВІД 0 ДО кількість файів
        3.3.1. Зчитати наступне значення із файлу і у змінну line
        3.3.2. ЯКЩО line містить число
            3.3.2.1. Зберегти line до tempList
        3.3.3. КІНЕЦЬ ЯКЩО
    3.4. КІНЕЦЬ ЦИКЛА
    3.5. ПОКИ не закінчилась поточна серія у всіх файлах
        3.5.1. Ініціалізувати змінні minValue та minIndex
        3.5.2. ЯКЩО tempList не пустий
            3.5.2.1. ЦИКЛ і ВІД 0 ДО розмір tempList
                3.5.2.1.1. ЯКЩО tempList[i] <= minValue
                    3.5.2.1.1.1. minValue = tempList[i]
                    3.5.2.1.1.2. minIndex = i
                3.5.2.1.2. КІНЕЦЬ ЯКЩО
            3.5.2.2. КІНЕЦЬ ЦИКЛА
            3.5.2.3. Додати значення minValue до sortedList
            3.5.2.4. Видалити елемент minValue з tempList
            3.5.2.5. Зчитати наступне значення з файлу minIndex та додати до tempList
        3.5.3. ІНАКШЕ
            3.5.3.1. Вивести у вихідний файл з номером counter масив sortedList
            3.5.3.2. Очистити sortedList
            3.5.3.3. Counter = (counter + 1) % кількість файлів
        3.5.4. КІНЕЦЬ ЯКЩО

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
fun main() {
    val inputFileName = "input.txt"
    var fileLength: ULong = 0uL
    var blockLength = 0

    when (inputFileType()) {
        "small" -> {
            fileLength = 10uL * 1024uL * 1024uL
            blockLength = 500
        }
        "normal" -> {
            fileLength = 500uL * 1024uL * 1024uL
            blockLength = 50_000
        }
        "large" -> {
            fileLength = 8uL * 1024uL * 1024uL * 1024uL
            blockLength = 500_000
        }
    }

    val startGeneratingTime = System.currentTimeMillis()

    generateFile(inputFileName, fileLength, blockLength)

    val startSortingTime = System.currentTimeMillis()
    val totalTimeOfGenerating = startSortingTime - startGeneratingTime
    println(
        "Generating time ${"%02d".format(totalTimeOfGenerating / (60_000))}" +
                ":${"%02d".format(totalTimeOfGenerating / 1000 % 60)}" +
                ".${totalTimeOfGenerating % 1000}"
    )
```

```
    newMultiwayMergeSort(inputFileName, "output.txt")
/*    externalSort(inputFileName, 3)*/

    val totalSortingTime = System.currentTimeMillis() - startSortingTime
    println(
        "Sorting time ${"%02d".format(totalSortingTime / (60_000))}" +
                ":${"%02d".format(totalSortingTime / 1000 % 60)}" +
                ".${"%03d".format(totalSortingTime % 1000)}"
    )
}

import java.io.File
import kotlin.random.Random

fun inputFileType(): String {
    var fileType: String
    print(
        "Enter length of the file (small/normal/large) " +
                "\nSmall - 10 MB " +
                "\nNormal - 500 MB " +
                "\nLarge - 8 GB \n"
    )
    do {
        fileType = readln().lowercase()
    } while (!setOf("small", "normal", "large").contains(fileType))

    return fileType
}

fun generateFile(name: String, length: ULong, blockLength: Int) {
    val file = File(name)
    file.writeText("")
    do {
        val tempList = MutableList(blockLength) { Random.nextInt() }
        file.appendText(tempList.joinToString(separator = "\n", postfix = "\n"))
```

```
    } while (file.length().toULong() < length)
}

import java.io.BufferedReader
import java.io.File
import java.io.FileReader

fun externalSort(fileName: String, numOfFiles: Int) {
    val inputFile = File(fileName)
    val filesB = List(numOfFiles) { i -> File("B${i + 1}.txt").also { it.writeText("") } }
    val filesC = List(numOfFiles) { i -> File("C${i + 1}.txt").also { it.writeText("") } }

    splitFile(inputFile, filesB, numOfFiles)

    var BC = true

    while (!isFullySorted(File(fileName), filesB[0], filesC[0])) {
        if (BC) {
            merge(filesB, filesC, numOfFiles)
        } else {
            merge(filesC, filesB, numOfFiles)
        }
        BC = !BC
    }

    if (filesB[0].length() == inputFile.length()) {
        filesB[0].copyTo(File("output.txt"), true)
    } else {
        filesC[0].copyTo(File("output.txt"), true)
    }
    for (i in 0 until numOfFiles) {
        filesB[i].writeText("")
        filesC[i].writeText("")
    }
}
```

```
fun splitFile(inputFile: File, outputFiles: List<File>, numOfFiles: Int) {
    var set = mutableListOf<Int>()
    val reader = BufferedReader(FileReader(inputFile))
    var counter = 0

    while (!isEOFs(listOf(reader))) {
        val element = reader.readLine().toInt()

        if (set.isEmpty() || element >= set.last()) set.add(element)
        else {
            outputFiles[counter].appendText(set.joinToString(separator = "\n", postfix = "\n"))
            set = mutableListOf(element)
            counter = (counter + 1) % numOfFiles
        }
    }
    outputFiles[counter].appendText(set.joinToString(separator = "\n", postfix = "\n"))
}

fun merge(inputFiles: List<File>, outputFiles: List<File>, numOfFiles: Int) {
    for (file in outputFiles) file.writeText("")
    val buffReaders = List(numOfFiles) { i -> BufferedReader(FileReader(inputFiles[i])) }
    var outputFileCounter = 0
    while (!isEOFs(buffReaders)) {
        val sortedList = mutableListOf<Int>()
        val tempList = mutableListOf<Pair<Int, Int>>()
        for (i in buffReaders.indices) {
            val line = checkLine(buffReaders[i])
            if (line != null && line != "") {
                tempList.add(Pair(i, line.toInt()))
            }
        }
        var flag = true
        while (flag) {
            var minValue = Int.MAX_VALUE
```

```
                var minIndex = -1
                if (tempList.isNotEmpty()) {
                    var indexForDel = 0
                    for (i in tempList.indices) {
                        if (tempList[i].second <= minValue) {
                            minValue = tempList[i].second
                            minIndex = tempList[i].first
                            indexForDel = i
                        }
                    }
                    sortedList.add(minValue)
                    tempList.removeAt(indexForDel)
                    buffReaders[minIndex].readLine()
                    val line = checkLine(buffReaders[minIndex])
                    if (line != null && line != "" && line != "-") {
                        if (sortedList.isEmpty() || line.toInt() >= sortedList.last()) {
                            tempList.add(Pair(minIndex, line.toInt()))
                        }
                    }
                } else {
                    outputFiles[outputFileCounter].appendText(sortedList.joinToString(separator = "\n", postfix = "\n"))
                    sortedList.clear()
                    outputFileCounter = (outputFileCounter + 1) % numOfFiles
                    flag = false
                }
            }

        }
    }

    fun checkLine(reader: BufferedReader): String? {
        reader.mark(50)
        val line = reader.readLine()
        reader.reset()
```

```
    return line
}

fun isFullySorted(fileA: File, fileB: File, fileC: File): Boolean {
    return fileA.length() == fileB.length() || fileA.length() == fileC.length()
}

fun isEOFs(buffReaders: List<BufferedReader>): Boolean {
    for (i in buffReaders) {
        if (checkLine(i) !in setOf("", null)) return false
    }
    return true
}
```

ВИСНОВОК

При виконанні даної лабораторної роботи я реалізував алгоритм збалансованого багатошляхового злиття для зовнішнього сортування. В результаті, алгоритм проводить сортування файла розміром 10 магабайт за 44 секунди, використовуючи при цьому 3 допоміжні файли. Цей час, звісно, завеликий для файлів розміром кілька гігабайт. Проте, програмну реалізацію можна вдосконалити декількома способами. Наприклад, варто змінити процес зчитування данних з початкового файла, визначаючи серії не порівнянням елементів, а сортуючи в оперативній пам’яті деяку зчитану послідовність чисел заданої довжини. Також, час запису данних до файлів можна зменшити, якщо використовувати об’єкт BufferedWriter, який зберігає в буфері деяку чистину файла і тим самим зменшує кількість звертань до жорсткого диску.

Загалом, я дізнався багато нового про зовнішнє сортування, спробував себе у проєктуванні алгоритмів та покращив знання щодо роботи з файловою системою.

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%.