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-з11 Химич Володимир Леонідович*
(шифр, прізвище, ім'я, по батькові)

Перевірив *Голосинцю 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.

Зовнішнє сортування прямим злиттям - метод, який сортує частини, кожна з яких входить у оперативну пам'ять, а потім об'єднує відсортовані частини.

Особливості методу: сортування в два кроки: спочатку сортування, а потім об'єднання не завжди буде ефективним. Обмеження для одиничного злиття полягає в тому, що при збільшенні кількості фрагментів пам'ять буде розділена на більшу кількість частин, тому кожна з них буде меншою. Це спричиняє велику кількість процесів читання малих частин даних, ніж меншу кількість більших. Таким чином, для сортування великої кількості даних, використовувати одиничне злиття неефективно: пошук на диску необхідний для заповнення вхідних фрагментів даних займає більшу частину часу. Використання двох злиттів або більше, дозволяє вирішити проблему. Часова складність алгоритму - O (n log n)

Модифікації / Альтернативи:

- Природне злиття
- Багатошляхове злиття O(log n)
- Багатофазне сортування

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 прямого злиття:

Не модифікований, власноруч написаний алгоритм прямого злиття. Для легкого використання для роботи з файлами, його не було реалізовано рекурсивно. До того ж, рекурсивна реалізація алгоритму буде неефективною при роботі з великою кількістю даних:

```
A = input array
maxPowerOfArrayLength = 0
i = 2
n = 1
//Look for the power of 2, which is bigger than array length
while i < A.length do
   i = i * 2
   n = n + 1
end while

for j = 0 to n do

   //SPLIT ARRAY

   splitedNumbersCount = 0
   currentStep = 1
   stepLength = Math.pow(2, j)
   B = new array[A.lenth / 2]
   C = new array[A.lenth - A.lenth / 2]
   k = 0
   m = 0
   toNextStepCounter = 0
   while (splitedNumbersCount < A.length) do
      if(currentStep % 2 == 1)
         B[k] = A[splitedNumbersCount]
         k = k + 1
      else
         C[m] = A[splitedNumbersCount]
         m = m + 1
      end if
      splitedNumbersCount = splitedNumbersCount + 1
      if(toNextStepCounter != 0 && toNextStepCounter % stepLength == 0)
         currentStep++
         toNextStepCounter = 0
      end if
   end while


   //MERGE ARRAY

   mergedNumbersCount = 0
   leftStep = 1
   rightStep = 1
   leftReadCount = 0
   rightReadCount = 0
   while (mergedNumbersCount < A.length) do
      if (leftStep == rightStep) {
         if (B[leftReadCount] < C[rightReadCount]) {
            if(leftReadCount < B.lenth - 1)
               A[mergedNumbersCount] = B[leftReadCount]
```

```
            leftReadCount = leftReadCount + 1
            leftStep = Math.ceil(leftReadCount / stepLength)
         else
            leftStep = leftStep + 1;
         end if
      else
         if(rightReadCount < C.lenth - 1)
            A[mergedNumbersCount] = C[rightReadCount]
            rightReadCount = rightReadCount + 1
            rightStep = Math.ceil(rightReadCount / stepLength)
         else
            rightStep = rightStep + 1;
         end if
   else if (leftStep > rightStep)
      if(rightReadCount < C.lenth - 1)
         A[mergedNumbersCount] = C[rightReadCount]
         rightReadCount = rightReadCount + 1
         rightStep = Math.ceil(rightReadCount / stepLength)
      end if
   else
      if(leftReadCount < B.lenth - 1)
         A[mergedNumbersCount] = B[leftReadCount]
         leftReadCount = leftReadCount + 1
         leftStep = Math.ceil(leftReadCount / stepLength)
      else
         leftStep = leftStep + 1
      end if
   end if
   mergedNumbersCount = mergedNumbersCount + 1
end while
```

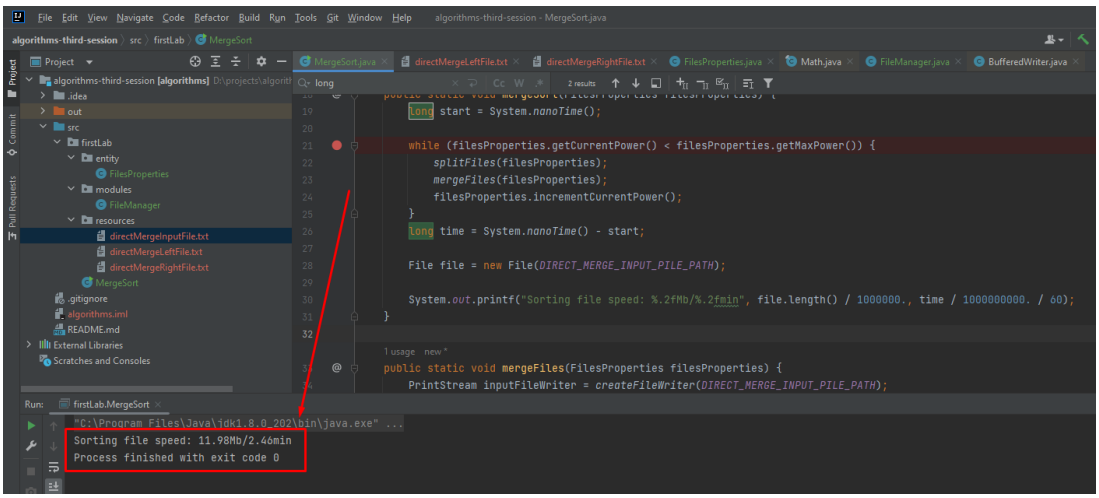
Після кожного спліту файлу буде відбуватись його мердж.

3.2 Програмна реалізація алгоритму

Алгорим було реалізовано на мові програмування Java.

Демонстрація реалізації алгоритму

Псевдокод, описаний вище, не відображає реалізацію алгоритму на 100%, оскільки частково було використано ООП, розбиття на функції, а замість масивів довелось працювати із файлами. Але догіка виконання алгоритму після написання коду не змінилась.



Ефективність даної реалізації алгоритму могла бути і кращою. За 2.46 хвилин було упорядковано лише 11.98Мб

3.2.1 Вихідний код

Код також можна глянути за посиланням: https://github.com/vovik541/algorithms-third-session/tree/master/src

```
import firstLab.entity.FilesProperties;

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

import static firstLab.entity.FilesProperties.*;
import static firstLab.modules.FileManager.*;

public class ExternalMergeSort {

    public static void main(String[] args) {
        FilesProperties filesProperties = createDirectMergeInputFile();
        mergeSort(filesProperties);
    }
```

```
    public static void mergeSort(FilesProperties filesProperties) {
        long start = System.nanoTime();

        while (filesProperties.getCurrentPower() < filesProperties.getMaxPower()) {
            splitFiles(filesProperties);
            mergeFiles(filesProperties);
            filesProperties.incrementCurrentPower();
        }
        long time = System.nanoTime() - start;

        File file = new File(DIRECT_MERGE_INPUT_PILE_PATH);

        System.out.printf("Sorting file speed: %.2fMb/%.2fmin", file.length() / 1000000., time
/ 1000000000. / 60);
    }

    public static void mergeFiles(FilesProperties filesProperties) {
        PrintStream inputFileWriter = createFileWriter(DIRECT_MERGE_INPUT_PILE_PATH);
        Scanner leftReader = createFileScanner(DIRECT_MERGE_LEFT_FILE_PATH);
        Scanner rightReader = createFileScanner(DIRECT_MERGE_RIGHT_FILE_PATH);

        int mergedNumbersCount = 0;
        double stepLength = Math.pow(2, filesProperties.getCurrentPower());
        double leftStep = 1;
        double rightStep = 1;
        double leftReadCount = 1;
        double rightReadCount = 1;

        int leftNumber;
        int rightNumber;

        if (leftReader.hasNext() && rightReader.hasNext()) {
            leftNumber = leftReader.nextInt();
            rightNumber = rightReader.nextInt();
            while (mergedNumbersCount < filesProperties.TOTAL_INPUT_ARRAY_LENGTH) {
                if (leftStep == rightStep) {
                    if (leftNumber < rightNumber) {
                        inputFileWriter.println(leftNumber);
                        if (leftReader.hasNext()) {
                            leftNumber = leftReader.nextInt();
```

```
                    ++leftReadCount;
                    leftStep = Math.ceil(leftReadCount / stepLength);
                } else {
                    leftStep++;
                }
            } else {
                inputFileWriter.println(rightNumber);
                if (rightReader.hasNext()) {
                    rightNumber = rightReader.nextInt();
                    ++rightReadCount;
                    rightStep = Math.ceil(rightReadCount / stepLength);
                } else {
                    rightStep++;
                }
            }
        } else if (leftStep > rightStep) {
            inputFileWriter.println(rightNumber);
            if (rightReader.hasNext()) {
                rightNumber = rightReader.nextInt();
                ++rightReadCount;
                rightStep = Math.ceil(rightReadCount / stepLength);
            }

        } else {
            inputFileWriter.println(leftNumber);
            if (leftReader.hasNext()) {
                leftNumber = leftReader.nextInt();
                ++leftReadCount;
                leftStep = Math.ceil(leftReadCount / stepLength);
            } else {
                leftStep++;
            }

        }
        ++mergedNumbersCount;
    }
} else {
    while (leftReader.hasNext()) {
        inputFileWriter.println(leftReader.nextInt());
    }
    while (rightReader.hasNext()) {
```

```
                inputFileWriter.println(rightReader.nextInt());
            }
        }

        inputFileWriter.close();
        leftReader.close();
        rightReader.close();
    }

    public static void splitFiles(FilesProperties filesProperties) {
        Scanner inputFileReader = createFileScanner(DIRECT_MERGE_INPUT_PILE_PATH);
        PrintStream leftWriter = createFileWriter(DIRECT_MERGE_LEFT_FILE_PATH);
        PrintStream rightWriter = createFileWriter(DIRECT_MERGE_RIGHT_FILE_PATH);

        filesProperties.nullLeftWriteArraysLength();

        int splitedNumbersCount = 0;

        int currentStep = 1;
        int toNextStepCounter = 0;
        int stepLength = (int) Math.pow(2, filesProperties.getCurrentPower());

        while (splitedNumbersCount < filesProperties.TOTAL_INPUT_ARRAY_LENGTH) {
            if (currentStep % 2 == 1) {
                leftWriter.println(inputFileReader.nextInt());
                filesProperties.incrementLeftLength();
            } else {
                rightWriter.println(inputFileReader.nextInt());
                filesProperties.incrementRightLength();
            }
            ++splitedNumbersCount;
            ++toNextStepCounter;
            if (toNextStepCounter != 0 && toNextStepCounter % stepLength == 0) {
                currentStep++;
                toNextStepCounter = 0;
            }
        }

        inputFileReader.close();
        leftWriter.close();
        rightWriter.close();
```

```
        }
    }
```

Допоміжні класи:

```
package firstLab.entity;

import java.nio.file.Paths;

public class FilesProperties {
    public static final String PROJECT_PATH = Paths.get("").toAbsolutePath().toString();
    public static final String DIRECT_MERGE_INPUT_PILE_PATH = PROJECT_PATH +
"\\src\\firstLab\\resources\\directMergeInputFile.txt";
    public static final String DIRECT_MERGE_LEFT_FILE_PATH = PROJECT_PATH +
"\\src\\firstLab\\resources\\directMergeLeftFile.txt";
    public static final String DIRECT_MERGE_RIGHT_FILE_PATH = PROJECT_PATH +
"\\src\\firstLab\\resources\\directMergeRightFile.txt";
    public final int TOTAL_INPUT_ARRAY_LENGTH;
    private long leftArrayLength = 0;
    private long rightArrayLength = 0;
    private int currentPower = 0;

    private int maxPower = 0;

    public FilesProperties(int TOTAL_INPUT_ARRAY_LENGTH) {
        this.TOTAL_INPUT_ARRAY_LENGTH = TOTAL_INPUT_ARRAY_LENGTH;

        while (Math.pow(2, maxPower) < TOTAL_INPUT_ARRAY_LENGTH){
            maxPower++;
        }
    }

    public void nullLeftWriteArraysLength(){
        this.leftArrayLength = 0;
        this.rightArrayLength = 0;
    }

    public void incrementCurrentPower(){
        this.currentPower++;
    }

    public void incrementLeftLength(){
        this.leftArrayLength++;
```

```
    }
    public void incrementRightLength(){
        this.rightArrayLength++;
    }
    public long getLeftArrayLength() {
        return leftArrayLength;
    }

    public long getRightArrayLength() {
        return rightArrayLength;
    }

    public int getMaxPower() {
        return maxPower;
    }

    public int getCurrentPower() {
        return currentPower;
    }
}
package firstLab.modules;

import firstLab.entity.FilesProperties;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintStream;
import java.util.Random;
import java.util.Scanner;

public class FileManager {
    public static Scanner createFileScanner(String filePath) {
        try {
            return new Scanner(new File(filePath));
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    public static FilesProperties createDirectMergeInputFile() {
```

```
        int numbersCount = 0;

        try (PrintStream ps = createFileWriter(FilesProperties.DIRECT_MERGE_INPUT_PILE_PATH)) {
            for (int i = 0; i < 10000000; i++) {
                ps.println(new Random().nextInt());
                ++numbersCount;
            }
        }

        return new FilesProperties(numbersCount);
    }

    public static PrintStream createFileWriter(String filePath) {
        try {
            return new PrintStream(filePath);
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        }
    }
}
```

- ВИСНОВОК

При виконанні даної лабораторної роботи я на абстрактному рівні зрозумів, як працюють різні алгоритми сортування із великими об’ємами даних, та на прикладі із Merge Sort. Шукаючи рішення для оптимізації алгоритму сортування, я познайомився із різними його реалізаціями. Я зрозумів, як працє Natural Merge Sort, але в силу відсутності часу, не встиг переписати своє додаткове рішення під роботу з файлами й виконати добровільне завдання. Особливо корисно було дізнатись про існування memory mapped files в java, які допомгагають отримати вмістимість файлу напряму із пам’яті. До того ж, із ними можна без проблем працювати до 128МБ, без зауваження перних негативних змін у ефективності алгоритму. Я також познайомився із Priority Queue. Ще важливо зауважити, що якщо ми хочемо написати ефективний алгоритм сортування великих файлів, то потрібно знайти баланс між розмірами файлів, на які ми будемо його ділити, та їх кількістю. В своїх спробах написати оптимізлваний алгоритм, я пробував підібрати різні розміри файлів (зупинився на 1024 записах) та створював нові запити роботи алгоритму “буфери”. Якщо у мене було більше, ніж половина вільною пам’яті віртуальної машини, я старався підняти кількість обрахункових процесів. Коли ж я доходив до “потолку”, то пробував їх понизити.

Я витратив багато часу на цю лабораторну роботу, не зробив її досконало, але отримав багато задоволення від процесу програмування та освоєння ідеї мого алгоритму.

- 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%.