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 Шевцова Анастасія Андріївна*
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов О. О.*
(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**
27	Збалансоване багатошляхове злиття

3 ВИКОНАННЯ

3.1 Псевдокод алгоритму

Основний алгоритм

1. Зчитати кількість шляхів waysNum
2. Відкрити вихідний файл
3. Створити два масиви з допоміжних файлів
4. Розділити вихідний файл за допомогою функції SplitA
5. Застосувати алгоритм SortMerge
6. Скопіювати результат з допоміжного файлу у кінцевий
7. Закрити вихідний та кінцевий файли

Алгоритм SplitA

1. Відкрити вихідний файл та файли для запису
2. Індекс файлу для запису bIndex = 0
3. Створити список для зберігання серії series
4. ПОКИ не кінець вихідного файлу
   4.1. Зчитати наступне число
   4.2. Записати число в кінець файлу
   4.3. ЯКЩО current > next
      4.3.1. bIndex = (bIndex + 1) % waysNum
   4.4. КІНЕЦЬ ЯКЩО
5. КІНЕЦЬ ПОКИ
6. Закрити вихідний файл та файли для запису

Алгоритм SortMerge

1. Відкрити допоміжні файли
2. Індекс файлу для запису index = 0
3. Створити змінну prev_num зі значенням None
4. ПОКИ не зчитано повністю
   4.1. minNum = int.MaxValue
   4.2. minIndex = -1
   4.3. ДЛЯ i ВІД 0 ДО waysNum ПОВТОРИТИ

4.3.1. Зчитати наступне число number

4.3.2. ЯКЩО prev_num == None АБО number >= prev_num

4.3.2.1. ЯКЩО number <= minNum

4.3.2.1.1. minNum = number

4.3.2.1.2. minIndex = i

4.3.2.2. КІНЕЦЬ ЯКЩО

4.3.3. КІНЕЦЬ ЯКЩО

4.4. КІНЕЦЬ ПОВТОРИТИ

4.5. ЯКЩО minIndex == -1

4.5.1. prev_num = None

4.5.2. index = (index + 1) % waysNum

4.6. ІНАКШЕ

4.6.1. Записати minNum до файлу під індексом index

4.6.2. prev_num = minNum

4.6.3. Перемістити покажчик

4.7. КІНЕЦЬ ЯКЩО

5. КІНЕЦЬ ПОВТОРИТИ

6. Закрити допоміжні файли

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

lab1.py

```
from DefaultAlgorithm import balancedMultiwayMerging
from functions import fileGenerator
from functions import outputFile
import time

file_size = int(input("Enter size of file in MB: "))
numbers = int(file_size * pow(2, 20) / 4)
ways_num = int(input("Enter number of ways: "))

fileGenerator("A.bin", numbers)
print("File was successfully generated")

start = time.time()
balancedMultiwayMerging("A.bin", "Sorted.bin", ways_num)
finish = time.time()
```

```
print("Algorithm operation time: {0:10.3f}".format(finish - start))

output = input("Show result? [y / n]: ")
if output == "y":
    outputFile("Sorted.bin")
```

DefaultAlgorithm.py

```
import os
import shutil
from functions import *


# основний алгоритм злиття
def balancedMultiwayMerging(pathNotSorted: str, pathSorded: str, waysNum: str):
    pathB = []
    pathC = []

    for i in range(waysNum):
        pathB.append(f"B{i + 1}.bin")
        pathC.append(f"C{i + 1}.bin")

    for path in pathC:
        with open(path, "wb") as file:
            pass

    splitA(pathNotSorted, pathB, waysNum)
    flag = True
    while not isSorted(pathNotSorted, pathB[0], pathC[0]):
        if flag:
            sortMerge(pathB, pathC, waysNum)
        else:
            sortMerge(pathC, pathB, waysNum)
        flag = not flag

    if os.path.getsize(pathNotSorted) == os.path.getsize(pathB[0]):
        shutil.copy(pathB[0], pathSorded)
    else:
        shutil.copy(pathC[0], pathSorded)

    for path in pathB:
        if os.path.exists(path):
            os.remove(path)
    for path in pathC:
        if os.path.exists(path):
            os.remove(path)


# розділення початкового файлу
def splitA(pathNotSorted: str, pathB: list, waysNum: str):
    fileA = FileReader(pathNotSorted)
    filesB = []
    for path in pathB:
        filesB.append(open(path, "wb"))

    bIndex = 0
    while fileA.current:
        filesB[bIndex].write(fileA.current)
```

```
        if fileA.current > fileA.next:
            bIndex = (bIndex + 1) % waysNum
        next(fileA)

    fileA.close()
    for file in filesB:
        file.close()


# злиття n файлів у інші n файлів
def sortMerge(pathsInputs: list, pathsOutputs: list, waysNum: int):
    readers = []
    writers = []

    index = 0
    prev_num = None

    for path in pathsInputs:
        readers.append(FileReader(path))

    for path in pathsOutputs:
        writers.append(open(path, "wb"))

    while (not isEOFReached(readers)):
        minNum = MAX_INT
        minIndex = -1

        for i in range(waysNum):
            if readers[i].current:
                number = int.from_bytes(readers[i].current, 'big')
                if prev_num == None or number >= prev_num:
                    if number <= minNum:
                        minNum = number
                        minIndex = i

        if minIndex == -1:
            prev_num = None
            index = (index + 1) % waysNum
        else:
            writers[index].write(minNum.to_bytes(4, 'big'))
            prev_num = minNum
            next(readers[minIndex])

    for file in readers:
        file.close()
    for file in writers:
        file.close()


# первіряє чи прочитані всі файли
def isEOFReached(readers):
    for file in readers:
        if file.current:
            return False
    return True


# перевіряє чи сортування завершилося
def isSorted(pathA: str, pathB: str, pathC: str) -> bool:
```

```
    return os.path.getsize(pathA) == os.path.getsize(pathB) or os.path.getsize(pathA) ==
os.path.getsize(pathC)
```

functions

```
from random import randint
MAX_INT = 2147483647

#заповнює файл випадковими числами
def fileGenerator(path: str, numbers: int):
    with open(path, "wb") as file:
        for i in range(numbers):
            file.write(randint(1, MAX_INT).to_bytes(4, "big"))


#виводить перші 50 чисел з файлу
def outputFile(path: str):
    list = []
    with open(path, "rb") as file:
        for i in range(50):
            number = int.from_bytes(file.read(4), 'big')
            list.append(number)
    print(" ".join(map(str, list)))


#допоміжний клас для зчитування з файлу
class FileReader:
    def __init__(self, path: str):
        self.path = path
        self.file = open(path, "rb")
        self.current = self.file.read(4)
        self.next = self.file.read(4)

    def __next__(self):
        temp = self.current
        self.current = self.next
        self.next = self.file.read(4)
        return temp

    def close(self):
        self.file.close()
```

ВИСНОВОК

При виконанні даної лабораторної роботи було реалізовано алгоритм Збалансованого багатошляхового злиття з використанням засобів мови C#. Також був написаний псевдокод алгоритму.

Роботу алгоритму протестовано на файлі розміром 10 та більше мегабайт.

Для сортування великих об’ємів даних потрібно зробити деяку модифікацію алгоритму. Наприклад, зчитувати не одне число, а використовувати буфер, для зменшення кількості звертань до жорсткого диску. Окрім цього, серії, записані до буфера, можна заздалегідь відсортувати за допомогою алгоритму внутрішнього сортування. Таким чином час виконання алгоритму значно зменшується.

Ця лабораторна допомогла мені поглибити знання щодо алгоритмів зовнішнього сортування та у проєктуванні алгоритмів.

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%.