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 Лисенко Анастасія* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов Олексій Олександрович* _____
(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

Розбиття початкового файлу на підфайли:

```
Функція separatingByFiles(fileName, amountOfFiles):
    filesB = []
    повторити для file від 0 до amountOfFiles
        filesB.append(open(path + str(file) + ".txt", "w"))
    все повторити

    currentList = []
    currentIndex = 0
    currentFile = 0

    initialFile = open(fileName, "r")
    a = np.fromfile(initialFile, dtype=np.uint32)
    initialFile.close()

    повторити для number в a:
        якщо( currentIndex != 0 та currentList[currentIndex - 1] >= number)
            то
                writeToFile(currentList, filesB[currentFile % amountOfFiles])
                currentFile += 1
                currentIndex = 0
                currentList.clear()
        все якщо
        currentIndex += 1
        currentList.append(number)
    все повторити

    writeToFile(currentList, filesB[currentFile % amountOfFiles])
    повторити для file в filesB:
        file.close()
    все повторити
    print("Initial file separated successfully!")


Функція merge_files(previous_names):
    amountOfFiles = len(previous_names)
    currentList = []
    CFiles = []
    BFiles = []
    fileSizes = [0] * amountOfFiles
    newSerialFileNames = []
    currentCFiles = 0
    amountOfFilesAfterMerge = amountOfFiles
```

```
print("\nNew phase of merging started...")
повторити для i, name в enumerate(previous_names):
    якщо exists(path + "B" + name + ".txt")
        то
            fileSizes[i] = os.path.getsize(path + "B" + fileNames[i] + ".txt")
            writeFiles.append(open(path + "C" + name + ".txt", "w"))
            readFiles.append(open(path + "B" + name + ".txt", "r"))
        інакше
            fileSizes[i] = os.path.getsize(path + "C" + fileNames[i] + ".txt")
            writeFiles.append(open(path + "B" + name + ".txt", "w"))
            readFiles.append(open(path + "C" + name + ".txt", "r"))
    все якщо
    symbol = BFiles[i].readline()
    fileSizes[i] -= len(symbol) + 1
    якщо( symbol != "\n" and symbol != "")
        то
            currentList.append(int(symbol))
        інакше
            fileSizes[i] = 0
            currentList.append(float('inf'))
            amountOfFilesAfterMerge -= 1
    все якщо
все повторити
повторити поки (x > 1 for x in fileSizes)
    повторити поки amountOfFilesAfterMerge > 0
        min_element = min(currentList)
        min_index = currentList.index(min_element)
        writeToFile(min_element, CFiles[currentCFiles % amountOfFiles])
        currentList.pop(min_index)
        symbol = BFiles[min_index].readline()
        fileSizes[min_index] -= len(symbol) + 1
        якщо (symbol != "\n" та symbol != "")
            то
                currentList.insert(min_index, int(symbol))
            інакше
                currentList.insert(min_index, float('inf'))
                amountOfFilesAfterMerge -= 1
        все якщо
    writeToFile(None, CFiles[currentCFiles % amountOfFiles], True)
    currentCFiles += 1
    currentList.clear()
    amountOfFilesAfterMerge = amountOfFiles
    повторити для i від 0 до amountOfFiles
        symbol = BFiles[i].readline()
        fileSizes[i] -= len(symbol) + 1
        якщо (symbol != "\n" та  symbol != "")
```

```
            то
                currentList.append(int(symbol))
            інакше
                fileSizes[i] = 0
        все якщо
            currentList.append(float('inf'))
            amountOfFilesAfterMerge -= 1
    все повторити
все повторити
повторити для file в CFiles:
    file.close()
все повторити
повторити для i, file в enumerate(BFiles):
    якщо currentCFiles - 1 >= i
        то
            newSerialFileNames.append(str(int((file.name[:-4])[35:]) + amountOfFiles))
    все якщо
    file.close()
    os.remove(file.name)
все повторити
print("Files merged(one of the multiple merges)...")
повернути newSerialFileNames
```

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

Базовий алгоритм

MultiwayMerge.py

```
import os
import math
import time
from helperFunctions import writeToFile
import numpy as np
from constants import path


def separatingByFiles(fileName, amountOfFiles):
    try:
        os.remove("Result.txt")
    except FileNotFoundError:
        pass
    filesB = []
    for file in range(amountOfFiles):
        filesB.append(open(path + str(file) + ".txt", "w"))

    currentList = []
    currentIndex = 0
    currentFile = 0

    initialFile = open(fileName, "r")
```

```
    a = np.fromfile(initialFile, dtype=np.uint32)
    initialFile.close()

    for number in a:
        if currentIndex != 0 and currentList[currentIndex - 1] >= number:
            writeToFile(currentList, filesB[currentFile % amountOfFiles])
            currentFile += 1
            currentIndex = 0
            currentList.clear()
        currentIndex += 1
        currentList.append(number)

    writeToFile(currentList, filesB[currentFile % amountOfFiles])
    for file in filesB:
        file.close()
    print("Initial file separated successfully!")


def merge_files(fileNames):

    amountOfFiles = len(fileNames)
    currentList = []
    writeFiles = []
    readFiles = []
    fileSizes = [0] * amountOfFiles
    newSerialFileNames = []
    currentCFiles = 0
    amountOfFilesAfterMerge = amountOfFiles

    print("\nNew phase of merging started...")
    for i, name in enumerate(fileNames):
        if exists(path + "B" + name + ".txt"):
            fileSizes[i] = os.path.getsize(path + "B" + fileNames[i] + ".txt")
            writeFiles.append(open(path + "C" + name + ".txt", "w"))
            readFiles.append(open(path + "B" + name + ".txt", "r"))
        else:
            fileSizes[i] = os.path.getsize(path + "C" + fileNames[i] + ".txt")
            writeFiles.append(open(path + "B" + name + ".txt", "w"))
            readFiles.append(open(path + "C" + name + ".txt", "r"))

        symbol = readFiles[i].readline()
        fileSizes[i] -= len(symbol) + 1
        if symbol != "\n" and symbol != "":
            currentList.append(int(symbol))
        else:
            fileSizes[i] = 0
            currentList.append(float('inf'))
            amountOfFilesAfterMerge -= 1
    while any(x > 1 for x in fileSizes):
        while amountOfFilesAfterMerge > 0:
            min_element = min(currentList)
            min_index = currentList.index(min_element)
            writeToFile(min_element, writeFiles[currentCFiles % amountOfFiles])
            currentList.pop(min_index)
            symbol = readFiles[min_index].readline()
            fileSizes[min_index] -= len(symbol) + 1
            if symbol != "\n" and symbol != "":
                currentList.insert(min_index, int(symbol))
            else:
                currentList.insert(min_index, float('inf'))
                amountOfFilesAfterMerge -= 1
        writeToFile(None, writeFiles[currentCFiles % amountOfFiles], True)
        currentCFiles += 1
        currentList.clear()
```

```
        amountOfFilesAfterMerge = amountOfFiles
        for i in range(amountOfFiles):
            symbol = readFiles[i].readline()
            fileSizes[i] -= len(symbol) + 1
            if symbol != "\n" and symbol != "":
                currentList.append(int(symbol))
            else:
                fileSizes[i] = 0
                currentList.append(float('inf'))
                amountOfFilesAfterMerge -= 1
    for file in writeFiles:
        file.close()
    for i, file in enumerate(readFiles):
        if currentCFiles - 1 >= i:
            newSerialFileNames.append(str(int((file.name[:-4])[35:])))
        file.close()
        os.remove(file.name)
    print("Files merged(one of the multiple merges)...")
    return newSerialFileNames
```

helperFunctions.py

```
import time
import os
import math
import numpy as np
import glob
from sys import getsizeof
import random
from constants import MB

def inputSize():
    filename = str(input("Enter your filename: "))
    size = int(input("Enter the size in MB: "))
    amountOfAdditionalFiles = 8 + int(math.log2(size)) if math.log2(size) > 0
else 5
    sortType = str(input("Enter the sort type(o - for optimized version and n -
for normal): "))
    return filename, size, amountOfAdditionalFiles, sortType


def writeToFile(currentList, file, appendNewline=False):
    newline = "\n"
    textCurrentList = ""
    if appendNewline:
        file.write(newline)
        return
    if type(currentList) == list:
        for number in currentList:
            textCurrentList += str(number) + newline
        file.write(textCurrentList + newline)
    else:
        file.write(str(currentList) + newline)

def generateNumbers(fileName, size):
    print("Starting the generation of file...\n")

    start = time.process_time()
    with open(fileName, 'wb') as fout:
        fout.write(os.urandom(size*MB))
    end = time.process_time()
```

```
    print("Generation succeeded!")
    return end-start

def txtGenerator(fileName, size):
    start = time.process_time()
    generateNumbers("test.bin", size)
    f = open("test.bin", 'r')
    a = np.fromfile(f, dtype=np.uint32)
    with open(fileName, 'w') as fout:
        for num in a:
            fout.write(str(num) + '\n')
    end = time.process_time()
    return (end-start)

def delete_dir(path):
    files = glob.glob(path)
    for f in files:
        os.remove(f)

def partition(array, low, high):
    # choose the rightmost element as pivot
    pivot = array[high]

    # pointer for greater element
    i = low - 1

    # traverse through all elements
    # compare each element with pivot
    for j in range(low, high):
        if array[j] <= pivot:
            # If element smaller than pivot is found
            # swap it with the greater element pointed by i
            i = i + 1

            # Swapping element at i with element at j
            (array[i], array[j]) = (array[j], array[i])

    # Swap the pivot element with the greater element specified by i
    (array[i + 1], array[high]) = (array[high], array[i + 1])

    # Return the position from where partition is done
    return i + 1


# function to perform quicksort


def quickSort(array, low, high):
    if low < high:
        # Find pivot element such that
        # element smaller than pivot are on the left
        # element greater than pivot are on the right
        pi = partition(array, low, high)

        # Recursive call on the left of pivot
        quickSort(array, low, pi - 1)

        # Recursive call on the right of pivot
        quickSort(array, pi + 1, high)
```

main.py

```
from helperFunctions import inputSize, generateNumbers, delete_dir
from MultiwayMerge import separatingByFiles, merge_files
from optimization import merge_files_optimize, divide_input_file
import time, os
from constants import path, path_to_folder, MAX_SIZE_OF_CHUNK


fileName, size, amountFiles, sortType = inputSize()
tm = generateNumbers(fileName, size)

print("Time spent(in seconds) on generations: ", tm)
if sortType == "n":
    print("Sorting starts...")
    separatingByFiles(fileName, amountFiles, size)
    start = time.time()
    namesOfFiles = merge_files([str(i) for i in range(amountFiles)])
    flag = 'B'
    while len(namesOfFiles) > 1:
        flag = 'C' if flag == 'B' else 'B'
        namesOfFiles = merge_files(namesOfFiles)
    flag = 'C' if flag == 'B' else 'B'
    os.rename(path + flag + str(namesOfFiles[0]) + ".txt", "Result.txt")
    delete_dir(path_to_folder)
    end = time.time()
    print("\nSorting ended, you can check your result file")
    print("Time spent on sorting: ", str(end - start), "seconds")
if sortType == "o":
    print("Sorting starts...")
    start = time.time()
    namesOfFiles = divide_input_file(fileName, MAX_SIZE_OF_CHUNK)
    merge_files_optimize(namesOfFiles)
    end = time.time()
    print("\nSorting ended, you can check your result file")
    print("Time taken: ", str(end - start), "seconds / ", str((end - start) /
60), "minutes")
```

constants.py

```
path = "D:\\PA-LABS-2-1-SEMESTR\\LAB1\\files\\"
path_to_folder = "D:/PA-LABS-2-1-SEMESTR/LAB1/files/*"
MB = 1048576
MAX_SIZE_OF_CHUNK = 1171875
```

Приклад сортування:

```
Enter the size in MB: 15
Enter the sort type(o - for optimized version and n - for normal): n
Starting the generation of file...

Generation succeeded!
Time spent(in seconds) on generations:  0.015625
Sorting starts...
Initial file separated successfully!

New phase of merging started...
Files merged(one of the multiple merges)...

New phase of merging started...
Files merged(one of the multiple merges)...

New phase of merging started...
Files merged(one of the multiple merges)...

New phase of merging started...
Files merged(one of the multiple merges)...

New phase of merging started...
Files merged(one of the multiple merges)...

New phase of merging started...
Files merged(one of the multiple merges)...

New phase of merging started...
Files merged(one of the multiple merges)...

Sorting ended, you can check your result file
Time spent on sorting:  64.67369222640991 seconds
```

Модифікація алгоритму

```
import math
import os
import time

import numpy as np

from LAB1.constants import MB
from helperFunctions import writeToFile
```

```
def divide_input_file(fileName, chunk):
    try:
        os.remove("Result.txt")
    except FileNotFoundError:
        pass

    currentFile = 0

    byteSize = os.path.getsize(fileName)
    amn = math.floor(byteSize / (10 * MB))
    left_amn = byteSize - (10 * MB) * amn

    for i in range(0, amn):
        a = np.fromfile(fileName, count=chunk, offset=i * chunk * 4,
dtype=np.uint32)
        with open(f"{currentFile}.txt", "w") as numbers_to_file:
            a.sort()
            for number in a:
                writeToFile(number, numbers_to_file)
            currentFile += 1
            print("Created file number:", currentFile)
    a = np.fromfile(fileName, count=int(left_amn / 4), offset=amn * chunk * 4,
dtype=np.uint32)
    with open(f"{currentFile}.txt", "w") as numbers_to_file:
        a.sort()
        for number in a:
            writeToFile(number, numbers_to_file)
        currentFile += 1
        print("Created file number:", currentFile)
    return [f"{i}" for i in range(currentFile)]


def merge_files_optimize(previous_names):
    """function for merging small series into larger series"""
    print("Merging started")
    num_of_files = len(previous_names)
    output_file = open("Result.txt", "w")
    curr_elements = []
    file_handler = []
    real_length = num_of_files
    for i, name in enumerate(previous_names):
        # opening files
        file_handler.append(open(name + ".txt", "r"))
        symbol = file_handler[i].readline()
        # reading the first characters of each file and adding them to the list
        if symbol != "\n" and symbol != "":
            curr_elements.append(int(symbol))
        else:  # if the file runs out of numbers, decrease the number of files
with characters
            curr_elements.append(float('inf'))
            real_length -= 1
    while real_length > 0:  # internal loop that works as long as at least one
file contains numbers related to the current series
        min_element = min(curr_elements)
        min_index = curr_elements.index(min_element)
        # writing the smallest number from the list of first numbers in each
file to the output file
        writeToFile(min_element, output_file)
        curr_elements.pop(min_index)
        # replacing this number with the next one from the same file
        symbol = file_handler[min_index].readline()
        if symbol != "\n" and symbol != "":
            curr_elements.insert(min_index, int(symbol))
```

```
        else:
            curr_elements.insert(min_index, float('inf'))
            real_length -= 1
    writeToFile(None, output_file, True)
    curr_elements.clear()
    real_length = num_of_files
    # repeating the procedure for the next series
    for i in range(num_of_files):
        symbol = file_handler[i].readline()
        if symbol != "\n" and symbol != "":
            curr_elements.append(int(symbol))
        else:
            curr_elements.append(float('inf'))
            real_length -= 1
    output_file.close()
    for file in file_handler:
        file.close()
        os.remove(file.name)
```

Приклад сортування

```
Enter your filename: output_filename.bin
Enter the size in MB: 15
Enter the sort type(o - for optimized version and n - for normal): o
Starting the generation of file...

Generation succeeded!
Time spent(in seconds) on generations:  0.015625
Sorting starts...
Created file number: 1
Created file number: 2
Merging started

Sorting ended, you can check your result file
Time taken:  8.475780487060547 seconds /  0.14126300811767578 minutes
```

ВИСНОВОК

При виконанні даної лабораторної роботи було досліджено деякі алгоритми зовнішнього сортування, зокрема алгоритм збалансованого багатошляхового злиття та його модифікація. Був написаний псевдокод до алгоритму та програмна реалізація мовою Python. Код був протестований на бінарному файлі розміром 10 та більше мегабайт.

Саме сортування умовно можна розділити на два етапи: перший – розділення початкового файлу на серії і розподіл серій по файлам; другий – злиття серій із одних файлів у інші.

Ця лабораторна робота допомогла нам розвинути більш глибокі знання щодо зовнішніх сортувань та покращила наші знання у сфері проектування 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%.