**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)**

(шифр, прізвище, ім'я, по батькові)

*ІП-13 Лисенко Анастасія*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов Олексій Олександрович*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc109342184)

[2 ЗаВдання 4](#_Toc109342185)

[3 Виконання 6](#_Toc109342186)

[3.1 Псевдокод алгоритму 6](#_Toc109342187)

[3.2 Програмна реалізація алгоритму 6](#_Toc109342188)

[3.2.1 Вихідний код 6](#_Toc109342189)

[Висновок 7](#_Toc109342190)

[Критерії оцінювання 8](#_Toc109342191)

# Мета лабораторної роботи

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

# 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 |

# Виконання

## 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

## Програмна реалізація алгоритму

### 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:

Text

Description automatically generated

**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

Text

Description automatically generated

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%.