**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-Васильєв Єгор Костянтинович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О.О.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc115883999)

[2 Завдання 4](#_Toc115884000)

[3 Виконання 6](#_Toc115884001)

[3.1 Псевдокод алгоритму 6](#_Toc115884002)

[3.2 Програмна реалізація алгоритму 8](#_Toc115884003)

[3.2.1 Вихідний код 8](#_Toc115884004)

[4 Модифікація 12](#_Toc115884005)

[4.1 Теоретичні відомості 12](#_Toc115884006)

[Висновок 13](#_Toc115884007)

[Критерії оцінювання 14](#_Toc115884008)

# 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

Варіант 7 «Збалансоване багатошляхове злиття»

## Псевдокод алгоритму

Функція поділу початкового файлу на задану кількість інших файлів шляхом виділення у ньому впорядкованих серій та їх почерговому запису у нові менші файли.

**divide\_in\_files(path, numbernumber\_of\_files):**

**start**

tmp\_lst = []

file\_handler = []

current\_index = 0

current\_file = 0

**повторити для** file **у діапазоні**(0, number\_of\_files):

file\_handler.append(open(str(file) + ".txt", "w"))

with open(path, "r") as input\_file:

**повторити для** line **у** input\_file:

**якщо** current\_index != 0 **та** tmp\_lst[current\_index - 1] >= int(line):

write\_to\_file(tmp\_lst, file\_handler[current\_file % number\_of\_files])

current\_file += 1

current\_index = 0

tmp\_lst.clear()

current\_index += 1

tmp\_lst.append(int(line))

write\_to\_file(tmp\_lst, file\_handler[current\_file % number\_of\_files])

**повторити для** file **у** file\_handler:

file.close()

**end**

Функція злиття серій з previous\_names файлів у нові більші серій у нові файли, причому кількість нових файлів менша або дорівнює кількості попередніх. Створюється масив з перших чисел у кожному файлі, потім обирається найменше з них та записується у вихідний файл, на його місце у масиві стає наступне число у поточному файлі. Процедура відбувається доти, доки хоча б в одному з початкових фалів присутні не оброблені числа з поточної серії. Далі процедура повторюється для наступної серії до тих пір, поки усі серії в початкових файлах не будуть оброблені.

**merge\_files(previous\_names):**

**start**

num\_of\_files = len(previous\_names)

curr\_elements = []

new\_file\_handler = []

file\_handler = []

sizes\_of\_files = [0] \* num\_of\_files

new\_names = []

current\_new\_file = 0

real\_length = num\_of\_files

**повторити для** i, name **у** enumerate(previous\_names):

sizes\_of\_files[i] = os.path.getsize(previous\_names[i] + ".txt")

new\_file\_handler.append(open(str(int(name) + num\_of\_files) + ".txt", "w"))

file\_handler.append(open(name + ".txt", "r"))

symbol = file\_handler[i].readline()

sizes\_of\_files[i] -= len(symbol) + 1

**якщо** symbol != "\n" **та** symbol != "":

curr\_elements.append(int(symbol))

**інакше**:

sizes\_of\_files[i] = 0

curr\_elements.append(float('inf'))

real\_length -= 1

**поки** any(x > 1 for x in sizes\_of\_files):

**поки** real\_length > 0:

min\_element = min(curr\_elements)

min\_index = curr\_elements.index(min\_element)

write\_to\_file(min\_element,new\_file\_handler[current\_new\_file%num\_of\_files])

curr\_elements.pop(min\_index)

symbol = file\_handler[min\_index].readline()

sizes\_of\_files[min\_index] -= len(symbol) + 1

**якщо** symbol != "\n" **та** symbol != "":

curr\_elements.insert(min\_index, int(symbol))

**інакше**:

curr\_elements.insert(min\_index, float('inf'))

real\_length -= 1

write\_to\_file(None,new\_file\_handler[current\_new\_file%num\_of\_files],True)

current\_new\_file += 1

curr\_elements.clear()

real\_length = num\_of\_files

**повторити для** i **у діапазоні** (num\_of\_files):

symbol = file\_handler[i].readline()

sizes\_of\_files[i] -= len(symbol) + 1

**якщо** symbol != "\n" **та** symbol != "":

curr\_elements.append(int(symbol))

else:

sizes\_of\_files[i] = 0

curr\_elements.append(float('inf'))

real\_length -= 1

**повторити для** file **у** new\_file\_handler:

file.close()

**повторити для** i, file **у** enumerate(file\_handler):

**якщо** current\_new\_file - 1 >= i:

new\_names.append(str(int(file.name[:-4]) + num\_of\_files))

file.close()

os.remove(file.name)

**якщо** len(new\_names) < num\_of\_files:

for i in range(num\_of\_files - len(new\_names)):

os.remove(str(int(new\_names[-1]) + i + 1) + ".txt")

**повернути** new\_names

**end**

input\_file\_path = "input.txt"

amount\_of\_files=log2(input\_file\_path.size())

divide\_in\_files(input\_file\_path, amount\_of\_files)

file\_names = merge\_files([str(i)**повторити для** i **у діапазоні**(0, amount\_of\_files)])

**поки** len(file\_names) > 1:

file\_names = merge\_files(file\_names)

## Програмна реалізація алгоритму

### Вихідний код

import os  
import math  
import time  
  
  
def write\_to\_file(arr, handler, add\_enter=False):  
 *"""function to write to a file: a number, a list, or just a newline"""* str\_arr = ""  
 if add\_enter:  
 handler.write("\n")  
 return  
 if type(arr) == list:  
 for i in arr:  
 str\_arr += str(i) + "\n"  
 handler.write(str\_arr + "\n")  
 else:  
 handler.write(str(arr) + "\n")  
  
  
def divide\_in\_files(path, number\_of\_files=5):  
 *"""function for dividing the initial file into series and writing them into new files"""* tmp\_lst = []  
 file\_handler = []  
 current\_index = 0  
 current\_file = 0  
 for file in range(number\_of\_files):  
 file\_handler.append(open(str(file) + ".txt", "w"))  
 with open(path, "r") as input\_file:  
 for line in input\_file:  
 if current\_index != 0 and tmp\_lst[current\_index - 1] >= int(line):  
 write\_to\_file(tmp\_lst, file\_handler[current\_file % number\_of\_files])  
 current\_file += 1  
 current\_index = 0  
 tmp\_lst.clear()  
 current\_index += 1  
 tmp\_lst.append(int(line))  
 write\_to\_file(tmp\_lst, file\_handler[current\_file % number\_of\_files])  
 for file in file\_handler:  
 file.close()  
 print("Divided successful")  
  
  
def merge\_files(previous\_names):  
 *"""function for merging small series into larger series"""* num\_of\_files = len(previous\_names)  
 curr\_elements = []  
 new\_file\_handler = []  
 file\_handler = []  
 sizes\_of\_files = [0] \* num\_of\_files  
 new\_names = []  
 current\_new\_file = 0  
 real\_length = num\_of\_files  
 for i, name in enumerate(previous\_names):  
 # opening and calculating file sizes to track their end  
 sizes\_of\_files[i] = os.path.getsize(previous\_names[i] + ".txt")  
 new\_file\_handler.append(open(str(int(name) + num\_of\_files) + ".txt", "w"))  
 file\_handler.append(open(name + ".txt", "r"))  
 symbol = file\_handler[i].readline()  
 sizes\_of\_files[i] -= len(symbol) + 1  
 # 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  
 sizes\_of\_files[i] = 0  
 curr\_elements.append(float('inf'))  
 real\_length -= 1  
 while any(x > 1 for x in sizes\_of\_files): # outer loop running until all numbers in all files have been processed  
 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  
 write\_to\_file(min\_element, new\_file\_handler[current\_new\_file % num\_of\_files])  
 curr\_elements.pop(min\_index)  
 # replacing this number with the next one from the same file  
 symbol = file\_handler[min\_index].readline()  
 sizes\_of\_files[min\_index] -= len(symbol) + 1  
 if symbol != "\n" and symbol != "":  
 curr\_elements.insert(min\_index, int(symbol))  
 else:  
 curr\_elements.insert(min\_index, float('inf'))  
 real\_length -= 1  
 write\_to\_file(None, new\_file\_handler[current\_new\_file % num\_of\_files], True)  
 current\_new\_file += 1  
 curr\_elements.clear()  
 real\_length = num\_of\_files  
 # repeating the procedure for the next series and the next output file  
 for i in range(num\_of\_files):  
 symbol = file\_handler[i].readline()  
 sizes\_of\_files[i] -= len(symbol) + 1  
 if symbol != "\n" and symbol != "":  
 curr\_elements.append(int(symbol))  
 else:  
 sizes\_of\_files[i] = 0  
 curr\_elements.append(float('inf'))  
 real\_length -= 1  
 for file in new\_file\_handler:  
 file.close()  
 for i, file in enumerate(file\_handler):  
 if current\_new\_file - 1 >= i:  
 new\_names.append(str(int(file.name[:-4]) + num\_of\_files))  
 file.close()  
 os.remove(file.name)  
 if len(new\_names) < num\_of\_files:  
 for i in range(num\_of\_files - len(new\_names)):  
 os.remove(str(int(new\_names[-1]) + i + 1) + ".txt")  
 print(new\_names)  
 return new\_names  
  
  
try:  
 os.remove("Result.txt")  
except FileNotFoundError:  
 pass  
input\_file\_path = "input\_medium.txt"  
amount\_of\_files = 8 + int(math.log2(os.path.getsize(input\_file\_path) / 1000000)) if math.log2(  
 os.path.getsize(input\_file\_path) / 1000000) > 0 else 5  
divide\_in\_files(input\_file\_path, amount\_of\_files)  
start = time.time()  
file\_names = merge\_files([str(i) for i in range(amount\_of\_files)])  
while len(file\_names) > 1:  
 file\_names = merge\_files(file\_names)  
os.rename(str(file\_names[0]) + ".txt", "Result.txt")  
end = time.time()  
print("Time taken: ", str(end - start), "seconds / ", str((end - start) / 60), "minutes")

# Модифікація

## Теоретичні відомості

Для покращення часу сортування змінимо функцію виділення серій з початкового файлу на функцію його поділу на менші файли, кожен з яких має розмір n елементів, яка при цьому буде внутрішньо сортувати ці елементи. Також модифікуємо функцію злиття, прибравши з неї зайвий цикл, призначений для обробки усіх серій (оскільки зараз кожен файл містить лише одну серію. Окрім того функція тепер викликатиметься лише один раз, а не декілька, оскільки усі проміжні файли вже будуть відсортовані, то вистачить лише одного злиття.

## Програмна реалізація

import os  
  
  
def write\_to\_file(arr, handler, add\_enter=False):  
 *"""function to write to a file: a number, a list, or just a newline"""* str\_arr = ""  
 if add\_enter:  
 handler.write("\n")  
 return  
 if type(arr) == list:  
 for i in arr:  
 str\_arr += str(i) + "\n"  
 handler.write(str\_arr + "\n")  
 else:  
 handler.write(str(arr) + "\n")  
  
  
def divide\_input\_file(path, chunk):  
 with open(path, "r") as numbers\_from\_file:  
 num\_line = 0  
 num\_file = 0  
 tmp\_lst = []  
 line = numbers\_from\_file.readline()  
 while line != "":  
 tmp\_lst.append(int(line))  
 num\_line += 1  
 line = numbers\_from\_file.readline()  
 if num\_line == chunk:  
 with open(f"{num\_file}.txt", "w") as numbers\_to\_file:  
 tmp\_lst.sort()  
 write\_to\_file(tmp\_lst, numbers\_to\_file)  
 num\_file += 1  
 tmp\_lst.clear()  
 num\_line = 0  
 print("Created file number:", num\_file)  
 return [f"{i}" for i in range(num\_file)]  
  
  
def merge\_files(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  
 write\_to\_file(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  
 write\_to\_file(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)  
  
  
MAX\_SIZE\_OF\_CHUNK = 625000  
names = divide\_input\_file("input\_big.txt", MAX\_SIZE\_OF\_CHUNK)  
merge\_files(names)

Висновок

При виконанні даної лабораторної роботи було розглянуто випадки, у яких неможливо здійснити внутрішнє сортування даних, коли їх обсяг значно перевищує обсяги оперативної пам’яті. Для таких випадків було досліджено різні види зовнішнього сортування, зокрема було розроблено алгоритм збалансованого багатошляхового злиття у вигляді псевдокоду та була здійснена його реалізація програмна реалізація на Python. Була здійснена модифікація зазначеного алгоритму, яка поєднала у собі внутрішнє та зовнішнє сортування, для зменшення часу виконання.

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%.