

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-21 Сергієнко Сергій
(шифр, прізвище, ім'я, по батькові)

Перевірів

Головченко М.М.
(прізвище, ім'я, по батькові)

Київ 2023

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ	3
2	ЗАВДАННЯ	4
3	ВИКОНАННЯ	5
3.1	ПСЕВДОКОД АЛГОРИТМУ	5
3.2	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ	5
3.2.1	<i>Вихідний код</i>	5
	ВИСНОВОК	16

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. Достатньо штучно обмежити доступну ОП, для уникнення багатогодинних сортувань (наприклад використовуючи віртуальну машину).

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
23	Збалансоване багатопрохідне злиття

3 ВИКОНАННЯ

3.1 Псевдокод алгоритму

```
Procedure merge_series(input_files, output_files, size):
    values, last_values, buffer = None * length(input_files)
    series_ends = False * length(input_files)
    num_of_files = length(input_files)
    while true do:
        if num_of_files == 0:
            break
        for i = 0 to num_of_files do:
            if values[i] is None and not series_ends[i] do:
                if buffer[i] do:
                    line = buffer[i]
                    buffer[i] = None
                else do:
                    line = input_files[i].readline()
                if not line do:
                    num_of_files -= 1
                    delete inputs[i], values[i], last_values[i],
series_ends[i]
                    break
                else do:
                    values[i] = line
                    if last_values[i] is not None and values[i] <
last_values[i] do:
                        series_ends[i] = True
                        buffer[i] = values[i]
                        values[i] = None
                    else do:
                        last_values[i] = values[i]
            if all(series_ends) do:
                values, last_values = None * num_of_files
                series_ends = False * num_of_files
                outputs = outputs[1:] + [outputs[0]]
            if any(element in values) do:
                min_value = min(values)
                outputs[0].write(min_value')
                values[values.index(min_value)] = None
            end while
        if any(os.stat(file).st_size != 0 for file in
output_files[1:]) do:
            merge_series(output_files, input_files)
```

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
import time
from number_generator import NumberGenerator
```

```

from series_splitter import SeriesSplitter
from series_merger import SeriesMerger
import validators

def main():
    while True:
        file_size = input("Enter file size (megabytes): ")
        if not validators.validate_positive_integer(file_size):
            print("The input is not a valid positive integer. Please try
again.")
        else:
            file_size = int(file_size) * 1024 ** 2
            break
    input_file = "a.txt"
    output_files_1 = ["b1.txt", "b2.txt", "b3.txt"]
    output_files_2 = ["c1.txt", "c2.txt", "c3.txt"]

    number_generator = NumberGenerator(input_file, file_size)
    number_generator.generate() # генерація чисел

    validators.clear_files(output_files_1) # очистка допоміжних файлів
    series_splitter = SeriesSplitter(input_file, output_files_1)
    series_merger = SeriesMerger(output_files_1, output_files_2, file_size)
    start_time = time.time()
    series_splitter.split_series() # поділ на серії
    print("Series are split")
    series_merger.merge_series() # сортування
    end_time = time.time()
    print(f"The sorting took {end_time - start_time} seconds.")

if __name__ == "__main__":
    main()

import time
from number_generator import NumberGenerator
from chunk_sorter import ChunkSorter
from mod_series_merger import SeriesMerger
import validators

```

```

def main():
    while True:
        file_size = input("Enter file size (megabytes): ")
        if not validators.validate_positive_integer(file_size):
            print("The input is not a valid positive integer. Please try
again.")
        else:
            file_size = int(file_size) * 1024 ** 2
            break
    input_file = "a.txt"
    output_files_1 = ["b1.txt", "b2.txt", "b3.txt", "b4.txt"]
    output_files_2 = ["c1.txt", "c2.txt", "c3.txt", "c4.txt"]

    number_generator = NumberGenerator(input_file, file_size)
    number_generator.generate() # генерація чисел

    validators.clear_files(output_files_1) # очистка допоміжних файлів
    chunk_sorter = ChunkSorter(input_file, output_files_1)
    print("Chunks are sorted!")
    series_merger = SeriesMerger(output_files_1, output_files_2, file_size)
    start_time = time.time()
    chunk_sorter.sort_chunks() # поділ на серії розміром 1/10 вхідного
    series_merger.merge_series() # сортування
    end_time = time.time()

    print(f"The sorting took {end_time - start_time} seconds.")

if __name__ == "__main__":
    main()

class SeriesSplitter:
    def __init__(self, input_file, output_files):
        self.input_file = input_file
        self.output_files = output_files

    def split_series(self):
        current_series = 0
        last_num = None

```

```

        outputs = [open(output_file, 'w') for output_file in
self.output_files]

    with open(self.input_file, 'r') as f:
        for line in f:
            try:
                num = int(line.strip())
            except ValueError:
                for file in outputs:
                    file.close()
                raise ValueError(f"Invalid value in input file: {line}")
                if last_num is not None and num < last_num: # якщо
попереднє число більше наступного, серія закінчилась
                    current_series = (current_series + 1) % len(outputs)
                outputs[current_series].write(f"{num}\n")
                last_num = num

    for file in outputs:
        file.close()

import os

class SeriesMerger:
    def __init__(self, input_files, output_files, size):
        self.input_files = input_files
        self.output_files = output_files
        self.size = size

    def merge_series(self):
        inputs = [open(input_file, 'r') for input_file in self.input_files]
        outputs = [open(output_file, 'w') for output_file in
self.output_files]

        num_of_files = len(inputs)
        values = [None] * num_of_files # зчитані значення
        last_values = [None] * num_of_files # попередні зчитані значення
        series_ends = [False] * num_of_files # чи закінчилась n-та серія у
файлі

        buffer = [None] * num_of_files # буфер зчитаних значень (якщо серія
скінчилась, то значення потрапляє сюди)

```



```

while True:
    if num_of_files == 0:
        break
    for i in range(num_of_files):
        if not values[i] and not series_ends[i]:
            if buffer[i]: # якщо у буфері щось є, то воно має бути
                зчитане першим
                line = str(buffer[i])
                buffer[i] = None
            else:
                line = inputs[i].readline()
                if not line.strip(): # якщо зчитано пустий рядок,
                    значить файл закінчився

                num_of_files -= 1
                inputs[i].close()
                del inputs[i]
                del values[i]
                del last_values[i]
                del series_ends[i]
                break
            else:
                try:
                    values[i] = int(line)
                except ValueError:
                    for file in inputs + outputs:
                        file.close()
                        raise ValueError(f"Invalid value in file
{self.input_files[i]}: {line}")

                if last_values[i] and values[i] < last_values[i]: #
                    перевірка на кінець серії

                    series_ends[i] = True
                    buffer[i] = values[i]
                    values[i] = None
                else:
                    last_values[i] = values[i]
    if all(series_ends):
        values = [None] * num_of_files
        last_values = [None] * num_of_files
        series_ends = [False] * num_of_files
        outputs = outputs[1:] + [outputs[0]]
        if any(element for element in values): # запис у файл

```

```

        min_value = min(v for v in values if v)
        outputs[0].write(str(min_value) + '\n')
        values[values.index(min_value)] = None

    for file in inputs + outputs:
        file.close()

        if any(os.stat(file).st_size != 0 for file in
self.output_files[1:]):
            # якщо b2-bn або c2-cn не пусті, то зливаємо знову
            self.input_files, self.output_files = self.output_files,
self.input_files
            self.merge_series()

import os
import mmap

class SeriesMerger:
    def __init__(self, input_files, output_files, size):
        self.input_files = input_files
        self.output_files = output_files
        self.size = size

    def merge_series(self):
        files = [open(input_file, 'r') for input_file in self.input_files]
        inputs = [mmap.mmap(f.fileno(), 0, access=mmap.ACCESS_READ) for f,
input_file in zip(files, self.input_files) if
os.stat(input_file).st_size != 0] # відображення файла у
пам'ять (модифікація)
        outputs = [open(output_file, 'w') for output_file in
self.output_files]

        num_of_files = len(inputs)
        values = [None] * num_of_files # зчитані значення
        last_values = [None] * num_of_files # попередні зчитані значення
        series_ends = [False] * num_of_files # чи закінчилась n-та серія у
файлі

        buffer = [None] * num_of_files # буфер зчитаних значень (якщо серія
скінчилась, то значення потрапляє сюди)

```

```

        chunk_size = self.size // 8
        chunk = [] # буфер для запису чисел у доп. файли (модифікація); 1/8
показало оптимальні значення
        while True:
            if num_of_files == 0: # якщо закінчились всі файли, виписуємо
все з буферу

                outputs[0].write('\n'.join(chunk))
                if len(chunk) != 0:
                    outputs[0].write('\n')
                chunk.clear()
                break
            for i in range(num_of_files):
                if not values[i] and not series_ends[i]:
                    if buffer[i]: # якщо у буфері щось є, то воно має бути
зчитане першим

                        line = str(buffer[i])
                        buffer[i] = None
                    else:
                        line = inputs[i].readline()
                        if not line.strip(): # якщо зчитано пустий рядок,
значить файл закінчився

                            num_of_files -= 1
                            inputs[i].close()
                            del inputs[i]
                            del values[i]
                            del last_values[i]
                            del series_ends[i]
                            break
                    else:
                        try:
                            values[i] = int(line)
                        except ValueError:
                            for file in files + outputs:
                                file.close()
                                raise ValueError(f"Invalid value in file
{self.input_files[i]}: {line}")
                            if last_values[i] and values[i] < last_values[i]: #
перевірка на кінець серії

                                series_ends[i] = True
                                buffer[i] = values[i]
                                values[i] = None

```

```

        else:
            last_values[i] = values[i]
    if all(series_ends):
        values = [None] * num_of_files
        last_values = [None] * num_of_files
        series_ends = [False] * num_of_files
        outputs[0].write('\n'.join(chunk))
        if len(chunk) != 0:
            outputs[0].write('\n')
        chunk.clear()
        outputs = outputs[1:] + [outputs[0]]
    if any(element for element in values): # запис у буфер
        min_value = None
        min_index = None
        for i, x in enumerate(values):
            if x is not None and (not min_value or x < min_value):
                min_value = x
                min_index = i
        values[min_index] = None
        chunk.append(str(min_value))
        if len(chunk) > chunk_size: # якщо буфер переповнено,
записуємо у файл
            outputs[0].write('\n'.join(chunk))
            outputs[0].write('\n')
            chunk.clear()

    for file in files + outputs:
        file.close()

        if any(os.stat(file).st_size != 0 for file in
self.output_files[1:]):
            self.input_files, self.output_files = self.output_files,
self.input_files
            self.merge_series() # якщо b2-bn або c2-cn не пусті, то
зливаємо знову

def clear_files(files):
    for file in files:
        try:
            with open(file, 'w'):
                pass

```

```

except FileNotFoundError:
    print(f"The file '{file}' does not exist.")
except Exception as e:
    print(f"An error occurred while clearing '{file}': {str(e)}")

def validate_positive_integer(n):
    try:
        value = int(n)
        if value > 0:
            return True
        else:
            return False
    except ValueError:
        return False

import mmap
import os

class ChunkSorter:
    def __init__(self, input_file, output_files):
        self.input_file = input_file
        self.output_files = output_files

    def sort_chunks(self):
        # вхідний файл ділиться на 10 блоків, кожен блок сортується і
        ділиться на 10 частин. у файл блоки записуються по
        # цим частинам. після запису кожної, перевіряється чи розмір не
        перевищив допустимого. виходить швидко і більш
        # менш рівні за розміром файли
        with open(self.input_file, 'r') as f:
            if os.stat(self.input_file).st_size != 0:
                mmapmed_file = mmap.mmap(f.fileno(), 0,
access=mmap.ACCESS_READ)
            else:
                raise ValueError(f"Cannot memory map an empty input file")
        file_size = os.path.getsize(self.input_file)
        chunk_size = file_size // 10
        part_size = 10
        max_output_file_size = file_size // len(self.output_files)

```

```

        if chunk_size == 0:
            chunk = mmaped_file[0:len(mmaped_file)]
            lines = chunk.decode().splitlines()
            non_empty_lines = [line for line in lines if line]
            sorted_chunk = sorted(non_empty_lines, key=int)
            with open(self.output_files[0], 'a') as f_out:
                f_out.write('\n'.join(sorted_chunk))
                f_out.write('\n')
            return
        for i in range(0, file_size, chunk_size):
            chunk = mmaped_file[i:i + chunk_size]
            lines = chunk.decode().splitlines()
            non_empty_lines = [line for line in lines if line]
            sorted_chunk = sorted(non_empty_lines, key=int)
            if len(sorted_chunk) // part_size != 0:
                chunk_parts = [sorted_chunk[j:j + len(sorted_chunk) //
part_size] for j in
                                range(0, len(sorted_chunk),
                                len(sorted_chunk) // part_size)]
            else:
                chunk_parts = sorted_chunk

            for chunk_part in chunk_parts:
                for output_file in self.output_files:
                    if os.path.getsize(output_file) +
max_output_file_size * (
                                1 / part_size ** 2) < max_output_file_size:
                        with open(output_file, 'a') as f_out:
                            f_out.write('\n'.join(chunk_part))
                            f_out.write('\n')
                        break

import random

class NumberGenerator:
    def __init__(self, filename, size):
        self.filename = filename
        self.size = size

    def generate(self):

```

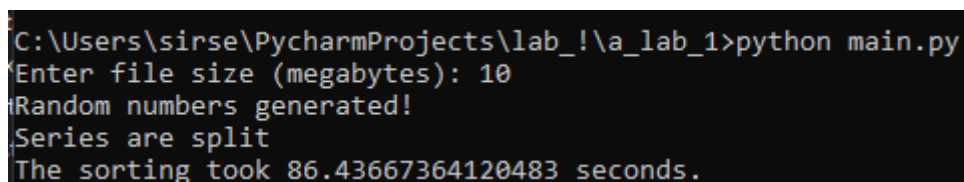
```
numbers_per_chunk = self.size // (10 ** 6)
chunk = ''
with open(self.filename, 'w') as f:
    while f.tell() < self.size:
        chunk += '\n'.join(str(random.randint(1, 1000)) for _ in
range(numbers_per_chunk)) + '\n'
        f.write(chunk)
        chunk = ''
print("Random numbers generated!")
```

Висновок

При виконанні даної лабораторної роботи я ознайомився з основними алгоритмами зовнішнього сортування, дослідив способи їхньої модифікації та оцінив їхню ефективність.

У цій лабораторній роботі я розробив алгоритм збалансованого багатошляхового злиття мовою Python. У базовій версії вхідний файл спочатку розбивається на серії – впорядковані підпоследовності последовності чисел. Таким чином він записується у 3 перші допоміжні файли. Далі використовуючи другі 3 допоміжні файли, дані кілька раз зливаються та сортуються. Таким чином, у один з допоміжних файлів після кількох злиттів записуються відсортовані вхідні дані. У модифікованій версії спочатку файл розбивається на декілька серій елементів, які сортуються вбудованим внутрішнім алгоритмом сортування та записуються у перші допоміжні файли. Серед інших модифікацій я реалізував відображення файлів у пам'ять та буферизацію запису при злитті.

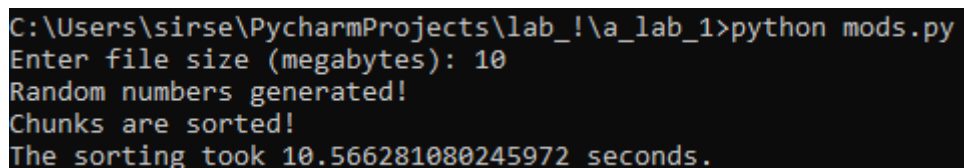
Базовий алгоритм сортування з вхідними даними обсягом 10 Мб виконується приблизно за 86 секунд (рис. 1).



```
C:\Users\sirse\PycharmProjects\lab_!\a_lab_1>python main.py
Enter file size (megabytes): 10
Random numbers generated!
Series are split
The sorting took 86.43667364120483 seconds.
```

Рис. 1 Базовий алгоритм сортування з вхідними даними 10 Мб

Модифікований алгоритм сортування з вхідними даними обсягом 10 Мб виконується приблизно за 10 секунд (рис. 2).

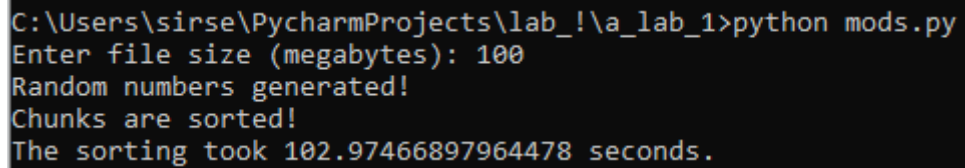


```
C:\Users\sirse\PycharmProjects\lab_!\a_lab_1>python mods.py
Enter file size (megabytes): 10
Random numbers generated!
Chunks are sorted!
The sorting took 10.566281080245972 seconds.
```

Рис. 2 Модифікований алгоритм сортування з вхідними даними 10 Мб

Таким чином, можна підсумувати, що модифікації пришвидшили роботу базового алгоритму більш ніж у 8 разів.

Для вхідних даних обсягом 100 Мб модифікований алгоритм сортування виконується приблизно за 102 секунди (рис. 3).



```
C:\Users\sirse\PycharmProjects\lab_!\a_lab_1>python mods.py
Enter file size (megabytes): 100
Random numbers generated!
Chunks are sorted!
The sorting took 102.97466897964478 seconds.
```

Рис. 3 Модифікований алгоритм сортування з вхідними даними 100 Мб

На жаль, досягти бажаної швидкості сортування модифікованого алгоритму 1 Гб / 3 хв не вдалося досягти, і алгоритм сортує такий вхідний файл у межах 20 хв. Це можна пояснити тим, що мова Python є інтерпретованою, що зумовлює її повільність.

Крім того, у цій роботі я використовував систему контролю версій Git та модульне тестування.