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”**
Варіант 1

Виконав: *ІП-13 Ал Хадам Мурат Резгович* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив: *Сопов Олексій Олександрович* _____
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ 3

2 ЗАВДАННЯ 4

3 ВИКОНАННЯ 6

3.1 Псевдокод алгоритму 6

3.2 Програмна реалізація алгоритму 8

3.2.1 Вихідний код 8

ВИСНОВОК 12

КРИТЕРІЇ ОЦІНЮВАННЯ 13

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

Процедура **distrubute(**serie_len**)**:

counter = 0

Поки a.curr:

Якщо counter % 2 == 0:

Для всіх i від 0 до serie_len **повторити:**

file_b.write(next(a))

Все повторити

Інакше:

Для всіх i від 0 до serie_len **повторити:**

file_c.write(next(a))

Все повторити

Все якщо

counter += 1

Все поки

Все процедура

Процедура **merge()**:

b = FileReader(self.file_b)

c = FileReader(self.file_c)

Поки no c.is_eof():

counter_b = 0

counter_c = 0

```
        Поки counter_b < serie_len and counter_c < serie_len:
            Якщо b.curr <= c.curr:
                binary_writer.write(next(b))
                counter_b += 1

                Поки counter_c < serie_len:
                    binary_writer.write(next(c))
                    counter_c += 1
                Все поки
            Інакше:
                binary_writer.write(next(c))
                counter_c += 1

                Поки counter_b < serie_len:
                    binary_writer.write(next(b))
                    counter_b += 1
                Все поки
            Все якщо
        Все поки
    Все поки

Поки counter_b < serie_len:
    binary_writer.write(next(b))
    counter_b += 1
Все поки
Все процедура
```

Процедура **straightMerge()**:

serie_len = 1

Поки not self.is_sorted():

self.**distrubute**(serie_len)

self.**merge**(serie_len)

serie_len *= 2

Все поки

Все процедура

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

main.py

```
from StraightMerge import StraightMerge


def main():
    file_name = input("Enter file to sort: ")

    sorter = StraightMerge(f"D:/asd1-files/{file_name}.txt",
                           r"D:\\asd1-files\\b_file.bin",
                           r"D:\\asd1-files\\c_file.bin")

    sorter.sort()
    print(sorter)


if __name__ == "__main__":
    main()
```

file_creator.py

```
from random import randint


def main():

    file_name = r"D:\\asd1-files\\" + input("Enter the filename: ") + ".txt"
    number_amount = int(eval(input("Enter amount of number to generate: ")))

    lower_bound, upper_bound = 1, 1000000

    with open(file_name, "wb") as a:
        # multiply by 32 for file size
        for i in range(number_amount):
            a.write(randint(lower_bound, upper_bound).to_bytes(32,
byteorder="big"))
```

FileReader.py

```
class FileReader:
    def __init__(self, file_path: str):
        self.path = file_path
        self.file = open(file_path, "rb")
        self.curr = self.file.read(32)
        self.next_num = self.file.read(32)

    def __iter__(self):
        return self

    def __next__(self):
        temp = self.curr
        self.curr = self.next_num
        self.next_num = self.file.read(32)
        return temp

    def read_32(self):
        self.file.read(32)

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

    def is_eof(self):
        if self.file.read(1) == b'':
            return True
        return False
```

StraightMerge.py

```
rom FileReader import FileReader
from time import time, sleep


class StraightMerge:
    def __init__(self, file_a, file_b, file_c):
        self.start = time()

        self.file_a = file_a
        self.file_b = file_b
        self.file_c = file_c

    def __str__(self):
        print(f"Sorting finished in {time() - self.start}")

    @staticmethod
    def clear_file(file):
        with open(file, "wb") as f:
            pass

    def is_sorted(self):
        file = FileReader(self.file_a)

        while file.next_num:
            if file.next_num < file.curr:
                file.close()
                return False
            next(file)
        return True

    def distribute(self, serie_len):
```

```
        counter = 0

        a = FileReader(self.file_a)
        b = open(self.file_b, "wb")
        c = open(self.file_c, "wb")

        while a.next_num:
            if counter % 2 == 0:
                for i in range(serie_len):
                    b.write(a.curr)
            else:
                for i in range(serie_len):
                    c.write(a.curr)

            next(a)
            counter += 1

        a.close()
        b.close()
        c.close()

    def merge(self, serie_len):
        b = FileReader(self.file_b)
        c = FileReader(self.file_c)

        with open(self.file_a, "wb") as binary_writer:
            # while is not EOF
            while not c.is_eof():
                counter_b = 0
                counter_c = 0

                # собираю в зависимости от длины
                while counter_b < serie_len and counter_c < serie_len:
                    if b.curr <= c.curr:
                        binary_writer.write(next(b))
                        counter_b += 1

                        while counter_c < serie_len:
                            binary_writer.write(next(c))
                            counter_c += 1
                    else:
                        binary_writer.write(next(c))
                        counter_c += 1

                        while counter_b < serie_len:
                            binary_writer.write(next(b))
                            counter_b += 1

            while b.next_num:
                binary_writer.write(next(b))

        binary_writer.close()
        b.close()
        c.close()

    def sort(self):
        serie_len = 1

        # for _ in range(1, int(log2(self.file_a_size / 32))):
        while not self.is_sorted():
            self.distribute(serie_len)
            print(f'Distrubion with the {serie_len} length successfully
finish...')
            sleep(10)
```

```
            self.merge(serie_len)
            print(f'Merging with the {serie_len} length successfully finish...')

            serie_len *= 2
```

ВИСНОВОК

При виконанні даної лабораторної роботи я засвоїв теоретичні відомості про основні типи зовнішнього сортування та практично реалізував сортування злиттям, а саме пряме сортування мовою програмування Python. Протягом роботи для тестування були згенеровані файли різної розмірності. Основний принцип роботи базового алгоритму прямого злиття полягяє в тому, що вхідний файл невпорядкованих байтів буде розділений в залежності від порядку серії в файлі, серією будемо вважати відсортовану послідовність елементів у розмірі степені числа 2. Файл B буде заповнений серіями/числами, які стоять на непарних позиціях (1/3/5...), а файл C відповідно серіями парними за індексом (2/4/6...). Поки вхідний файл в результаті повторюваних злиттів файлів не буде відсортованний будемо повторювати операцію.

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%.