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-з21 Клименко М. М.* _____
(шифр, прізвище, ім'я, по батькові)

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

Київ 2024

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета лабораторної роботи полягає у детальному вивченні алгоритму балансованого багатошляхового злиття, одного з ключових методів зовнішнього сортування. Вона включає аналіз теоретичних основ цього алгоритму, його практичну реалізацію, а також оптимізацію для роботи з великими обсягами даних. Особливий акцент робиться на оптимізації використання дискового простору та пам'яті, а також на оцінці ефективності алгоритму в різних умовах. Ця робота має на меті забезпечити глибоке розуміння принципів та технік, що лежать в основі зовнішнього сортування, і дослідити їх практичне застосування.

2 ЗАВДАННЯ

Згідно варіанту 3 (таблиця 2.1), розробити та записати алгоритм зовнішнього сортування за допомогою псевдокоду.

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**
3	Збалансоване багатошляхове злиття

3 ВИКОНАННЯ

3.1 Псевдокод алгоритму

Алгоритм збалансоване багатошляхове злиття

Вхід: input_file - шлях до вхідного файлу
output_file - шлях до вихідного файлу
block_size_mb - розмір блоку в мегабайтах

1. Розділення великого файлу на сортовані блоки
```
blocks = пустий список
Відкрити input_file для читання
Поки не досягнуто кінця файлу:
    numbers = пустий список
    Читати числа з input_file до заповнення блоку розміром block_size_mb
    Якщо numbers не порожній:
        Сортувати numbers
        temp_filename = створити назву для тимчасового файлу
        Відкрити temp_filename для запису
        Записати відсортовані числа в temp_filename
        Додати temp_filename до списку blocks
Закрити input_file
```

2. Багатошляхове злиття
```
Відкрити output_file для запису
files = список, який містить відкриті файли для читання з кожного блоку
heap = пуста мінімальна купа
Для кожного файлу у files:
    Читати перше число з файлу
    Додати пару (число, індекс файлу) до heap
Процес злиття:
    Поки heap не порожня:
        min_val, file_idx = вилучити найменше число і індекс файлу з heap
        Записати min_val у output_file
        Читати наступне число з файлу з індексом file_idx
        Якщо число існує:
            Додати пару (число, file_idx) до heap
Закрити всі файли у files
Видалити тимчасові файли
Закрити output_file
```

Кінець Алгоритму

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код базової програми

```
import heapq
import os
import random
import time

def generate_large_file(filename, size_mb, number_range):
    num_numbers = (size_mb * (1024**2)) // 4
    with open(filename, 'w') as file:
        for _ in range(num_numbers):
            file.write(f"{random.randint(*number_range)}\n")

def external_sort(input_file, output_file, block_size_mb):
    start_time = time.time()  # Початок засікування часу

    # Крок 1: Розбиття на блоки та їх сортування
    blocks = []
    with open(input_file, 'r') as file:
        while True:
            numbers = []
            for _ in range(block_size_mb * 250000):
                line = file.readline()
                if line:
                    numbers.append(int(line))
                else:
                    break

            if not numbers:
                break

            numbers.sort()
            temp_filename = f"temp_{len(blocks)}.txt"
            with open(temp_filename, 'w') as temp_file:
                for num in numbers:
                    temp_file.write(f"{num}\n")
            blocks.append(temp_filename)

    # Крок 2: Багатошляхове злиття
    with open(output_file, 'w') as out_file:
        files = [open(block, 'r') for block in blocks]
```

```
            heap = [(int(file.readline()), i) for i, file in enumerate(files) if not file.closed]
            heapq.heapify(heap)

            while heap:
                min_val, file_idx = heapq.heappop(heap)
                out_file.write(f"{min_val}\n")
                next_val = files[file_idx].readline()
                if next_val:
                    heapq.heappush(heap, (int(next_val), file_idx))

            for file in files:
                file.close()
                os.remove(file.name)

        end_time = time.time()  # Кінець засікування часу
        print(f"Сортування завершено! Час сортування: {end_time - start_time:.2f} секунд.")

    # Приклад використання
    generate_large_file("large_file.txt", 10, (0, 100000))
    external_sort("large_file.txt", "sorted_file.txt", 1)
```

3.2.2 Вихідний код модифікованої програми

```
import heapq
import os
import random
from datetime import datetime

# Генерація великого файлу з випадковими числами
def generate_large_file(filename, size_mb, number_range):

    size_in_bytes = size_mb * (1024**2)  # Налаштування розміру до 1000 МБ
    current_size = 0
    with open(filename, 'w') as file:
        while current_size < size_in_bytes:
            number = random.randint(*number_range)
            line = f"{number}\n"
            file.write(line)
            current_size += len(line.encode('utf-8'))

# Зовншішнє сортування великого файлу
```

```
def external_sort(input_file, output_file, block_size_mb):
    start_time = datetime.now()
    # Крок 1: Розбиття на блоки та сортування
    blocks = []
    with open(input_file, 'r') as file:
        while True:
            numbers = []
            try:
                for _ in range(block_size_mb * 250000):  # Збільшення розміру блоку
                    line = file.readline()
                    if line:
                        numbers.append(int(line))
                    else:
                        break
            except MemoryError:
                break

            if not numbers:
                break

            numbers.sort()
            temp_filename = f"temp_{len(blocks)}.txt"
            with open(temp_filename, 'w') as temp_file:
                for num in numbers:
                    temp_file.write(f"{num}\n")
            blocks.append(temp_filename)

    # Крок 2: Багатошляхове злиття
    with open(output_file, 'w') as out_file:
        files = [open(block, 'r') for block in blocks]
        heap = [(int(file.readline()), i) for i, file in enumerate(files) if not file.closed]
        heapq.heapify(heap)

        while heap:
            min_val, file_idx = heapq.heappop(heap)
            out_file.write(f"{min_val}\n")
            next_val = files[file_idx].readline()
            if next_val:
                heapq.heappush(heap, (int(next_val), file_idx))

        for file in files:
            file.close()
            os.remove(file.name)
```

```
    end_time = datetime.now()
    duration = end_time - start_time
    minutes = duration.seconds // 60
    seconds = duration.seconds % 60

        print(f"Сортування завершено! Час початку:
{start_time.strftime('%H:%M:%S')}, Час закінчення:
{end_time.strftime('%H:%M:%S')}, Загальний час сортування: {minutes}
хвилин {seconds} секунд")

        # Генерація файлу розміром 1000 МБ
        generate_large_file("large_file.txt", 1000, (0, 1000000))

        # Сортування файлу зі збільшеним розміром блоку
        external_sort("large_file.txt", "sorted_file.txt", 400)  # Збільшення розміру
блоку для кращої продуктивності
```

3.2.3 Результати роботи програм

На рисунку 3.2.1 показано результат сортування 15 мб базовою програмою.

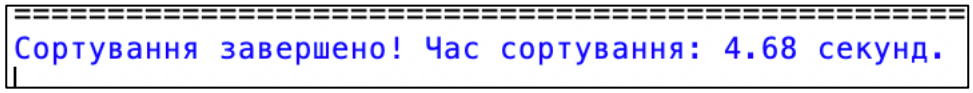

Рис. 3.2.1 Результат сортування базовою програмою

На рисунку 3.2.2 показано результат сортування 1 гб модифікованою програмою.

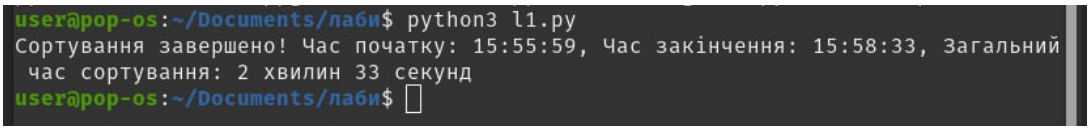

Рис. 3.2.1 Результат сортування модифікованою програмою

ВИСНОВОК

При виконанні даної лабораторної роботи з вивчення та порівняння базової та модифікованої програм зовнішнього сортування було встановлено, що внесені зміни значно покращують продуктивність та надійність системи. Основні модифікації, які були виконані в програмі, включають збільшення розміру блоку з 1 MB до 400 MB, що дозволяє ефективніше обробляти великі обсяги даних, зменшуючи кількість необхідних проходів для сортування файлу. Також додано обробку помилок пам'яті, що забезпечує більшу стабільність програми при роботі з великими файлами.

Ефективність модифікованої програми була підтверджена під час тестування: базова програма змогла сортувати 15 MB файл за 4.68 секунди, тоді як модифікована програма справилася з 1 GB файлом за 2 хвилини 33 секунди. Це свідчить про значне покращення продуктивності, особливо при врахуванні збільшеного обсягу оброблюваних даних.

Таким чином, виконання лабораторної роботи підтвердило, що внесені модифікації ефективно покращують здатність програми обробляти великі масиви даних, роблячи її більш придатною для задач, де необхідна швидка та надійна обробка великих об'ємів інформації.

КРИТЕРІЇ ОЦІНЮВАННЯ

У випадку здачі лабораторної роботи до 08.10.2022 включно максимальний бал дорівнює – 5. Після 08.10.2022 максимальний бал дорівнює – 4,5.

Критерії оцінювання у відсотках від максимального балу:

- псевдокод алгоритму – 15%;
- програмна реалізація алгоритму – 20%;
- програмна реалізація модифікацій – 20%;
- робота з git – 40%;
- висновок – 5%.