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-13 Крупосій Вадим Сергійович* _____
(шифр, прізвище, ім'я, по батькові)

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

Київ 2022

ЗМІСТ

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

```
Метод natural_adaptive_merge_sort
ПОЧАТОК
    ПОКИ не відсортований файл
        відкрити файл A для читання
        відкрити файл B та C для читання
        розділити на файли файл A по серіях
        закрити файл A
        злити в файл A дані з файлу B та C
    КІНЕЦЬ ПОКИ
КІНЕЦЬ
Метод split_two_files
ПОЧАТОК
    ПОКИ читаємо файл A
        створюємо масив для серій і додаткову змінну буфер
        записуємо серії
        ПОКИ не закінчили читати файл A
            зчитуємо дані
            ЯКЩО попередній <= поточний
                записуємо серії
            ІНАКШЕ
                в буфер
     записуємо серії в допоміжні файли
КІНЕЦЬ
Метод merge_files
ПОЧАТОК
    відкриваємо для читання файли B та C
    відкриваємо для запису файл A
```

оголошуємо змінні prevNumC prevNumB

читаємо в змінні numB, numC

ПОКИ не кінець файлів

ЯКЩО numB > numC

записуємо в файл A numC

prevNumC = numC

зчитуємо numC

ЯКЩО numC < prevNumC

ПОВТОРИТИ

записуємо в файл A numB

prevNumB = numB

зчитуємо numB

ПОКИ numB >= prevNumB

КІНЕЦЬ ЯКЩО

КІНЕЦЬ ЯКЩО

ІНАКШЕ

записуємо в файл A numB

prevNumB = numB

зчитуємо numB

ЯКЩО numB < prevNumB

ПОВТОРИТИ

записуємо в файл A numC

prevNumC = numC

зчитуємо numB

ПОКИ numC >= prevNumC

КІНЕЦЬ ІНАКШЕ

записати в файл A min(numB, numC)

записати в файл A max(numB, numC)

ЯКЩО зчитуємо з B == true

ПОКИ зчитуємо з C == false

записати в файл A з файлу C

КІНЕЦЬ ПОКИ

ІНАКШЕ

ПОКИ зчитуємо з B == false

записати в файл A з файлу B

КІНЕЦЬ ПОКИ

КІНЕЦЬ ІНАКШЕ

ЗАКРИТИ ФАЙЛИ

КІНЕЦЬ

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
using System;
using System.Collections.Generic;
using System.IO;

namespace Laba1
{
    class Program
    {

        //Зовнішнє Природне (адаптивне) сортування

        static int count = 1;
        static string bufferL = null;
       // 184_000_000 = 1GB
       // 17_900_000 = 100MB
       // 1_790_000 = 10MB
        static void Main(string[] args)
        {
            Console.WriteLine("-----------------------------------------------------------------
------------------------------");
            Console.Write("What file you want to sort?
(test1GB.txt/test100MB.txt/test10MB.txt) ");
            string path;
            path = Console.ReadLine();
            if (path == "test10MB.txt")
            {
                DateTime time = DateTime.Now;
                natural_adaptive_merge_sort(path);
                DateTime time2 = DateTime.Now;
                Console.WriteLine("Time for sorting " + time2.Subtract(time).TotalSeconds +
" seconds");
            }
            else
            {
                DateTime time = DateTime.Now;
                optimized_adaptive_merge_sort(path);
                DateTime time2 = DateTime.Now;
                Console.WriteLine("Time for sorting " + time2.Subtract(time).TotalSeconds +
" seconds");
            }
            Console.Write("Do you want to print first 10000 elements in file? press y/n ");
            string mode;
            mode = Console.ReadLine();
            if (mode == "y")
            {
                printFile(path);
            }
            //createFile(path, 17_900_000);
            //DateTime time = DateTime.Now;
            //natural_adaptive_merge_sort(path);
            //optimized_adaptive_merge_sort(path);
            //printFile(path);
            //DateTime time2 = DateTime.Now;
            //Console.WriteLine("Time for sorting " + time2.Subtract(time).TotalSeconds + "
seconds");
        }
```

```
static void SplitTwoFilesOptimised(StreamReader streamReaderA)
{
    while (!streamReaderA.EndOfStream)
    {
        List<string> series = new List<string>();

        List<int> numbers = new List<int>();

        const int numbersToSort = 100000000;

        for (int i = 0; i < numbersToSort && !streamReaderA.EndOfStream; i++)
        {
            numbers.Add(int.Parse(streamReaderA.ReadLine()));
        }

        numbers.Sort();

        foreach (var number in numbers)
        {
            series.Add(number.ToString());
        }

        write_info_to_file(series);
    }

}

static void createFile(string path, int size)
{
    Random rand = new Random();
    using (StreamWriter sw = new StreamWriter(path, false))
    {
        for (int i = 0; i < size; i++)
        {
            sw.WriteLine(rand.Next(0, 10_000).ToString());
        }
    }
}

static void printFile(string path)
{
    using (StreamReader sw = new StreamReader(path, false))
    {
        for (int i = 0; i < 100000; i++)
        {
            Console.WriteLine(sw.ReadLine());
        }
    }

}

static void natural_adaptive_merge_sort(string path)
{

    while (!is_Sorted(path))
    {
        StreamReader read_A = new StreamReader(path);
        File.WriteAllText("B.txt", "");
        File.WriteAllText("C.txt", "");

        split_two_files(read_A);
        read_A.Close();
        merge_files(path, "B.txt", "C.txt");
    }
```

```
}

static void optimized_adaptive_merge_sort(string path)
{
    int i = 0;
    // Optimised part
    StreamReader streamReaderA = new StreamReader(path);
    File.WriteAllText("B.txt", "");
    File.WriteAllText("C.txt", "");
    SplitTwoFilesOptimised(streamReaderA);

    streamReaderA.Close();
    merge_files(path, "B.txt", "C.txt");

    while (!is_Sorted(path))
    {
        Console.WriteLine($"Iteration №{++i}");
        streamReaderA = new StreamReader(path);
        File.WriteAllText("B.txt", "");
        File.WriteAllText("C.txt", "");

        split_two_files(streamReaderA);
        streamReaderA.Close();
        merge_files(path, "B.txt", "C.txt");
    }
}

static void split_two_files(StreamReader read_A)
{

    while (!read_A.EndOfStream)
    {
        List<string> series = new List<string>();

        string line = bufferL;
        if (line == null) { line = read_A.ReadLine(); }
        series.Add(line);


        while (!read_A.EndOfStream)
        {
            string prevL = line;
            line = read_A.ReadLine();
            int intline = Convert.ToInt32(line);
            int intprevLine = Convert.ToInt32(prevL);
            if (intprevLine <= intline)
            {
                series.Add(line);
            }
            else
            {
                bufferL = line;
                break;
            }

        }
        write_info_to_file(series);
    }
}
static bool is_Sorted(string path)
{
    StreamReader read_from = new StreamReader(path);

    int num1 = Convert.ToInt32(read_from.ReadLine());
    int num2 = Convert.ToInt32(read_from.ReadLine());
```

```
    while (!read_from.EndOfStream)
    {

        if (num1 > num2)
        {
            read_from.Close();
            return false;
        }
        num1 = num2;
        num2 = Convert.ToInt32(read_from.ReadLine());

    }
    read_from.Close();
    return true;
}
static void write_info_to_file(List<string> series)
{
    if (count % 2 == 1)
    {
        StreamWriter writer = new StreamWriter("B.txt", true);
        foreach (string line in series)
        {
            writer.WriteLine(line);
        }
        writer.Flush();
        writer.Close();
    }
    else
    {
        StreamWriter writer = new StreamWriter("C.txt", true);
        foreach (string line in series)
        {
            writer.WriteLine(line);
        }
        writer.Flush();
        writer.Close();
    }
    count++;
}

static void merge_files(string pathA, string pathB, string pathC)
{
    StreamReader read_from_B = new StreamReader(pathB);
    StreamReader read_from_C = new StreamReader(pathC);
    StreamWriter write_to_A = new StreamWriter(pathA);
    int prevNumC, prevNumB;
    int numB = Convert.ToInt32(read_from_B.ReadLine());
    int numC = Convert.ToInt32(read_from_C.ReadLine());
    while (!read_from_B.EndOfStream && !read_from_C.EndOfStream)
    {

        if (numB > numC)
        {
            write_to_A.WriteLine(numC);
            prevNumC = numC;
            numC = Convert.ToInt32(read_from_C.ReadLine());
            if (numC < prevNumC)
            {
                do
                {
                    write_to_A.WriteLine(numB);
                    prevNumB = numB;
                    numB = Convert.ToInt32(read_from_B.ReadLine());
                } while (numB >= prevNumB);
            }
```

```
            }
            else
            {
                write_to_A.WriteLine(numB);
                prevNumB = numB;
                numB = Convert.ToInt32(read_from_B.ReadLine());
                if (numB < prevNumB)
                {
                    do
                    {
                        write_to_A.WriteLine(numC);
                        prevNumC = numC;
                        numC = Convert.ToInt32(read_from_C.ReadLine());
                    } while (numC >= prevNumC);
                }
            }
        }
        write_to_A.WriteLine(Math.Min(numB, numC));
        write_to_A.WriteLine(Math.Max(numB, numC));
        if (read_from_B.EndOfStream)
        {
            while (!read_from_C.EndOfStream)
            {
                write_to_A.WriteLine(Convert.ToInt32(read_from_C.ReadLine()));
            }
        }
        else
        {
            while (!read_from_B.EndOfStream)
            {
                write_to_A.WriteLine(Convert.ToInt32(read_from_B.ReadLine()));
            }
        }
        read_from_B.Close();
        read_from_C.Close();
        write_to_A.Flush();
        write_to_A.Close();
    }
}
   }
```

ВИСНОВОК

При виконанні даної лабораторної роботи було досліджено алгоритм зовнішнього сортування “Природне (адаптивне) злиття”. Було написано псевдокод алгоритму, програмна реалізація алгоритму зовнішнього сортування, також програмна реалізація оптимізованого алгоритму зовнішнього сортування “Природне злиття”. При досліджені цих алгоритмів було виявлено, що оптимізована версія сортує значно швидше, ніж не оптимізована. До прикладу, неупорядкованих 10мб звичайний алгоритм відсортував за 6-8хв., а 100мб оптимізований за 6 секунд хв., а 1ГБ за 69-76 секунд. Також було відсортовано файл розміром в х2 оперативної пам’яті.

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%.