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-14 Бабіч Денис* _____

(шифр, прізвище, ім'я, по батькові)

Перевірив *Ахаладзе Ілля Елдарійович* _____

(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

Варіант 2

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**
2	Природне (адаптивне) злиття

3 ВИКОНАННЯ

3.1 Псевдокод алгоритму

```
const DEFAULT = 0
const MODIFICATION = 1
pathA = "A.txt"
pathB = "B.txt"
pathC = "C.txt"

//Заповнення файлу, який потім буде сортуватися випадковими величинами
GenerateFile(pathA, 2_000_000, 1, 2_000_000)

if (ChooseApproach() == MODIFICATION)
        //Попереднє сортування числових послідовностей файлу
        PreSort(pathA, GetSize(pathA))

//Поки файл не відсортовано
while (NOT IsFileSorted(pathA)) DO
        //Розподіл файлу по підфайлам
        ManageData(pathA, pathB, pathC)
        //Злиття підфайлів в основний файл
        MergeData(pathA, pathB, pathC)

File.Delete(pathB)
File.Delete(pathC)
```

```
function GenerateFile(path, quantity, min, max) returns void
        File.WriteAllText(path, String.Empty)
        using (sw = StreamWriter(File.OpenWrite(path)))
                Random random = new Random()
                for (i = 0, i < quantity, i += 1) do
                        sw.Write(random.Next(min, max))
                endfor

function ChooseApproach() returns int
        do
                if (Int32.TryParse(Console.ReadLine(), out value)) then
                        if (value > 1 || value < 0) then
                                continue
                        endif
                        return value
                else
                        Console.WriteLine("Введіть коректні дані!")
                endif
        while (true)
```

```
function GetSize(path) returns int
	long fileSize = new FileInfo(path).Length
	do
		if (Int32.TryParse(Console.ReadLine(), out size)) then
			if (size > fileSize) then
				continue
			endif
		return size /= sizeof(Int32)
		else
			Console.WriteLine("Введіть коректні дані.")
		endif
	while (true)

function TryGetNumber(reader, out number) returns bool
	if (reader.Peek() == -1) then
		number = -1
		return false
	endif
	string temp = String.Empty
	while (reader.Peek() != ' ' AND reader.Peek() != -1) do
		temp += (char)reader.Read()
		if (reader.Peek() != -1) then
			reader.Read()
	number = Int32.Parse(temp)
	return true
```

```
function ManageData(pathA, pathB, pathC) returns void
    File.WriteAllText(pathB, string.Empty)
    File.WriteAllText(pathC, string.Empty)
    using (sr = StreamReader(File.OpenRead(pathA)))
            using (sw1 = StreamWriter(File.OpenWrite(pathB)),
                    sw2 = StreamWriter(File.OpenWrite(pathC)))
                        numberOfSeries = 0
                        TryGetNumber(sr, out previousNumber)
                        sw1.Write("{previousNumber} ")
                        while (TryGetNumber(sr, out currentNumber)) do
                                if (previousNumber > currentNumber) then
                                        numberOfSeries += 1
                                endif
                                previousNumber = currentNumber
                                if (numberOfSeries % 2 == 0) then
                                        sw1.Write(currentNumber)
                                else
                                        sw2.Write(currentNumber)
                                endif
                        endwhile
```

```
function MergeData(pathA, pathB, pathC) returns void
    File.WriteAllText(pathA, string.Empty)
    using (readerB = new StreamReader(File.OpenRead(pathB)),
            readerC = new StreamReader(File.OpenRead(pathC)))
                numbers.AddRange(GetNumbers(readerB, out lastB))
                numbers.AddRange(GetNumbers(readerC, out lastC))
                numbers.Sort()
                using (sw = StreamWriter(File.OpenWrite(pathA)))
                        for (i = 0, i < numbers.Count, i += 1) do
                                sw.Write(numbers[i])
                        endfor
                        while (true) do
                                numbers.Clear()
                                if (lastB != -1) then
                                        numbers.Add(lastB)
                                        lastB = -1
                                endif
                                numbers.AddRange(GetNumbers(readerB, out lastB))
                                if (lastC != -1) then
                                        numbers.Add(lastC)
                                        lastC = -1
                                endif
                                numbers.AddRange(GetNumbers(readerC, out lastC))
                                if (numbers.Count == 0) then
                                        break
                                numbers.Sort()
                                for (i = 0, i < numbers.Count, i += 1) do
                                        sw.Write(numbers[i])
                                endfor
```

```
function GetNumbers(reader, out lastNumber) returns List
    lastNumber = -1
    List numbers
    if (NOT TryGetNumber(reader, out previousNumber))
            return numbers
    numbers.Add(previousNumber)
    while (TryGetNumber(reader, out currentNumber)) do
            if (previousNumber <= currentNumber) then
                    numbers.Add(currentNumber)
            else
                    lastNumber = currentNumber
                    break
            endif
            previousNumber = currentNumber;
    return numbers


function IsFileSorted(string path) returns bool
    using (sr = StreamReader(File.OpenRead(path)))
            TryGetNumber(sr, out previousNumber)
            while (TryGetNumber(sr, out currentNumber)) do
                    if (previousNumber > currentNumber)
                            return false
                    endif
                    previousNumber = currentNumber
    return true
```

```
function PreSort(string path, int size) returns void
        tempPath = path.Replace("A.txt", "Temp.txt")
        using (sr = StreamReader(File.OpenRead(path)))
                using (sw = StreamWriter(File.OpenWrite(tempPath)))
                        sequence = List(size)
                        counter = 0
                        while (true) do
                                while (counter < size) do
                                        if (TryGetNumber(sr, out currentNumber)) then
                                                TryGetNumber(sr, out currentNumber))
                                        endif
                                        counter += 1
                                sequence.Sort()
                                for (i = 0, i < sequence.Count, i += 1)
                                        sw.Write(sequence[i])
                                sequence.Clear()
                                counter = 0
                                if (sr.Peek() == -1)
                                        break
          File.Delete(path)
          File.Move(tempPath, path)
```

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
namespace ExternalSorting
{
   public sealed class ExternalSorting
   {
      private const int DEFAULT = 0;
      private const int MODIFICATION = 1;

      private static void Main()
      {
         Console.InputEncoding = System.Text.Encoding.Unicode;
         Console.OutputEncoding = System.Text.Encoding.Unicode;

         string pathA = "A.txt";
         string pathB = "B.txt";
         string pathC = "C.txt";

         GenerateFile(pathA, 2_000_000, 1, 2_000_000);

         Console.WriteLine($"Можна переглянути вміст згенерованого файлу за відповідним шляхом: {pathA}");

         if (ChooseApproach() == MODIFICATION)
            PreSort(pathA, GetSize(pathA));

         while (!IsFileSorted(pathA))
         {
            ManageData(pathA, pathB, pathC);
```

```
            MergeData(pathA, pathB, pathC);
        }

        File.Delete(pathB);
        File.Delete(pathC);

        Console.WriteLine("Файл відсортовано.");
    }

    private static void GenerateFile(string path, int quantity, int min, int max)
    {
        //if (File.Exists(path))
            File.WriteAllText(path, String.Empty);

        using (StreamWriter sw = new StreamWriter(File.OpenWrite(path)))
        {
            Random random = new Random();

            for (int i = 0; i < quantity; ++i)
                sw.Write($"{random.Next(min, max)} ");
        }
    }

    private static int ChooseApproach()
    {
        int value;

        do
        {
```

```
            Console.Write($"Оберіть спосіб сортування. Звичайне зовнішнє сортування ({DEFAULT}), або зовнішнє сортування з модифікацією ({MODIFICATION}): ");

            if (Int32.TryParse(Console.ReadLine(), out value))
            {
                if (value > 1 || value < 0)
                {
                    Console.WriteLine($"Допустимі значення лише {DEFAULT} або {MODIFICATION}!");
                    continue;
                }

                return value;
            }
            else
                Console.WriteLine("Введіть коректні дані!");

        } while (true);
    }

    private static int GetSize(string path)
    {
        long fileSize = new FileInfo(path).Length;
        int size;

        do
        {
            Console.Write("Скільки байтів файлу Ви бажаєте попередньо впорядкувати: ");
```

```
            if (Int32.TryParse(Console.ReadLine(), out size))
            {
                if (size > fileSize)
                {
                    Console.WriteLine("Розмір послідовності має бути меншим за розмір
файлу!");
                    continue;
                }

                return size /= sizeof(Int32);
            }
            else
                Console.WriteLine("Введіть коректні дані.");

        } while (true);
    }

    private static bool TryGetNumber(StreamReader reader, out int number)
    {
        if (reader.Peek() == -1)
        {
            number = -1;
            return false;
        }

        string temp = String.Empty;

        while (reader.Peek() != ' ' && reader.Peek() != -1)
            temp += (char)reader.Read();
```

```
            if (reader.Peek() != -1)
                reader.Read();

            number = Int32.Parse(temp);
            return true;
        }

        private static void ManageData(string pathA, string pathB, string pathC)
        {
            File.WriteAllText(pathB, string.Empty);
            File.WriteAllText(pathC, string.Empty);

            using (StreamReader sr = new StreamReader(File.OpenRead(pathA)))
            {
                using (StreamWriter sw1 = new StreamWriter(File.OpenWrite(pathB)), sw2 =
new StreamWriter(File.OpenWrite(pathC)))
                {
                    int numberOfSeries = 0;
                    int previousNumber, currentNumber;

                    TryGetNumber(sr, out previousNumber);
                    sw1.Write($"{previousNumber} ");

                    while (TryGetNumber(sr, out currentNumber))
                    {
                        if (previousNumber > currentNumber)
                            ++numberOfSeries;
```

```
                previousNumber = currentNumber;

                if (numberOfSeries % 2 == 0)
                    sw1.Write($"{currentNumber} ");
                else
                    sw2.Write($"{currentNumber} ");
            }
        }
    }
}

private static void MergeData(string pathA, string pathB, string pathC)
{
    File.WriteAllText(pathA, string.Empty);

    using (StreamReader readerB = new StreamReader(File.OpenRead(pathB)),
readerC = new StreamReader(File.OpenRead(pathC)))
    {
        List<int> numbers = new List<int>();
        int lastB, lastC;

        numbers.AddRange(GetNumbers(readerB, out lastB));
        numbers.AddRange(GetNumbers(readerC, out lastC));
        numbers.Sort();

        using (StreamWriter sw = new StreamWriter(File.OpenWrite(pathA)))
        {
            for (int i = 0, length = numbers.Count; i < length; ++i)
                sw.Write($"{numbers[i]} ");
```

```
while (true)
{
    numbers.Clear();

    if (lastB != -1)
    {
        numbers.Add(lastB);
        lastB = -1;
    }

    numbers.AddRange(GetNumbers(readerB, out lastB));

    if (lastC != -1)
    {
        numbers.Add(lastC);
        lastC = -1;
    }

    numbers.AddRange(GetNumbers(readerC, out lastC));

    if (numbers.Count == 0)
        break;

    numbers.Sort();

    for (int i = 0, length = numbers.Count; i < length; ++i)
        sw.Write($"{numbers[i]} ");
}
```

```
    }
}

List<int> GetNumbers(StreamReader reader, out int lastNumber)
{
    lastNumber = -1;
    int currentNumber;
    int previousNumber;
    List<int> numbers = new List<int>();

    if (!TryGetNumber(reader, out previousNumber))
        return numbers;

    numbers.Add(previousNumber);

    while (TryGetNumber(reader, out currentNumber))
    {
        if (previousNumber <= currentNumber)
            numbers.Add(currentNumber);
        else
        {
            lastNumber = currentNumber;
            break;
        }

        previousNumber = currentNumber;
    }

    return numbers;
```

```
    }
}

private static bool IsFileSorted(string path)
{
    using (StreamReader sr = new StreamReader(File.OpenRead(path)))
    {
        int previousNumber, currentNumber;

        TryGetNumber(sr, out previousNumber);

        while (TryGetNumber(sr, out currentNumber))
        {
            if (previousNumber > currentNumber)
                return false;

            previousNumber = currentNumber;
        }
    }

    return true;
}

private static void PreSort(string path, int size)
{
    string tempPath = path.Replace("A.txt", "Temp.txt");

    using (StreamReader sr = new StreamReader(File.OpenRead(path)))
    {
```

```
using (StreamWriter sw = new StreamWriter(File.OpenWrite(tempPath)))
{
    List<int> sequence = new List<int>(size);

    int counter = 0;
    int currentNumber;

    while (true)
    {
        while (counter < size)
        {
            if (TryGetNumber(sr, out currentNumber))
                sequence.Add(currentNumber);

            ++counter;
        }

        sequence.Sort();

        for (int i = 0, length = sequence.Count; i < length; ++i)
            sw.Write($"{sequence[i]} ");

        sequence.Clear();

        counter = 0;

        if (sr.Peek() == -1)
            break;
    }
```

```
            }
        }

        File.Delete(path);
        File.Move(tempPath, path);
    }
  }
}
```

4 ВИСНОВОК

При виконанні даної лабораторної роботи дослідив алгоритм сортування природне (адаптивне) злиття. Переконався у його працездатності на завчасно підготованих текстових файлах. При перевірці виконання роботи файл розміром 10 мб був відсортований приблизно за 13 с, а файл розміром 1 гб було відсортовано за 2.5 хв, враховуючи те, що початковий файл було попередньо відсортовано серіями розмірами по 500 м.

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 алгоритму – 40%;
- висновок – 5%.