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-12 Бондарчук Анастасія* _____
(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

```
MergeSort(string fileName, out string sortedFileName, bool Modified, int numberOfFiles = 0)
	if (numberOfFIles < 2):
		Exception;
	end if;
	if (Modified == True):
		string[] BFilesArray = Range(i from 1 to numberOfFIles) "B{i}mod.bin";
		string[] CFilesArray = Range(i from 1 to numberOfFIles) "C{i}mod.bin";
		SplitFile(fileName, BFilesArray);
		SortHelper(BFilesArray, CFilesArray, out sortedFileName);
	else
		string[] BFilesArray = Range(i from 1 to numberOfFIles) "B{i}.bin";
		string[] CFilesArray = Range(i from 1 to numberOfFIles) "C{i}.bin";
		SplitFile(fileName, BFilesArray);
		SortHelper(BFilesArray, CFilesArray, out sortedFileName);
	end if;
end.
SplitFile(string fileName, string[] BFilesArray)
	List<File> writers = foreach file in BFilesArray (File.Open(file,
writeMode).ToList());
	File currentWriter = writers.First();
	FIle reader = File.OpenRead(fileName, readMode);
	int previous = int.MinValue;
	while (reader.SeekPosition  < reader.SeekPosition)
		int current = reader.ReadInt32();
		if (current >= previous)
			currentWriter.Write(current);
		else
			currentWriter = writers.Next(currentWriter);
			currentWriter.Write(current);
		end if;
		previous = current;
	end while;
	foreach w in writers(w.Dispose());
	reader.Close();
end.
SortHelper(string[] BFilesArray, string[] CFIlesArray, out string fileName)
	List<File> readers = foreach file in BFilesArray (File.Open(file,readMode).ToList());
	foreach r in readers( if (r.BaseStream.Position == r.BaseStream.Length): r.Dispose();
reader.Remove(R);
	if (readers.Count == 1)
		fileName = readers.First().Name;
		return;
	end if;
	List<File> writers = foreach file in CFIlesArray(File.Open(file,
writeMode)).ToList();
	File currentWriter = writers.First();
	File currentReader = readers.First();
	List<int> nums = new();
	List<int> nextNums = new();
```

```
        Dictionary readerAndPrevNum = readers.ToDictionary(r = r, _ = int.MinValue);
        while (readers.Count != 0)
                while (currentReader.BaseStream.Position == currentReader.BaseStream.Length)
                        File readerToRemove = currentReader;
                        currentReader = readers.Next(currentReader);
                        readers.Remove(readerToRemove);
                        readerAndPrevNum.Remove(readerToRemove);
                        readerToRemove.Dispose();
                        if (readers.Count == 0)
                                break;
                        end if;
                end while;
                if (readers.Count == 0)
                        nums.Sort();
                        foreach n in nums (currentWriter.Write(n));
                        currentWriter = writers.Next(currentWriter);
                        nextNums.Sort();
                        foreach n in nextNums (currentWriter.Write(n));
                        break;
                end if;
                int num = currentReader.ReadInt32();
                if (num >= readerAndPrevNum[currentReader])
                        nums.Add(num);
                        readerAndPrevNum[currentReader] = num;
                else
                        nextNums.Add(num);
                        readerAndPrevNum[currentReader] = num;
                        currentReader = readers.Next(currentReader);
                end if;
                if (nextNums.Count >= readers.Count)
                        nums.Sort();
                        foreach n in nums (currentWriter.Write(n));
                        currentWriter = writers.Next(currentWriter);
                        nums.Clear();
                        nums.AddRange(nextNums);
                        nextNums.Clear();
                end if;
                     readers.ForEach(r => r.Dispose());
                     writers.ForEach(w => w.Dispose());
                     SortHelper(CFIlesArray, BFilesArray, out fileName);
        end while;
end.
SortParts(string fileName, string resultFileName, int size, int shareSize)
        if (File.Exists(resultFileName))
                File.Delete(resultFileName);
        end if;
        int[] array = new int[shareSize];
        File reader = File.Open(fileName, readMode));
        File writer = File.Open(resultFileName, writeMode);
        for (int i = 0; i < size / shareSize; i++)
                for (int j = 0; j < shareSize; j++)
                        array[j] = reader.ReadInt32();
                end for;
```

```
                Array.Sort(array);
                for (int j = 0; j < shareSize; j++)
                        writer.Write(array[j]);
                end for;
            end for;
            reader.Close();
            writer.Close();
        end.
```

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
using System;
using System.Diagnostics;


namespace lab1
{

    internal class Program
    {
        static void Main(string[] args)
        {
            const int linesCount = 100_000_000;
            const int megabytes = 10;
            Console.WriteLine("Not modified");
            Normal(megabytes);
            Console.WriteLine("\nModified");
            Modified(linesCount);
            Console.ReadLine();
        }
        static void Normal(int megabytes)
        {
            string path = "A.bin";

            RandomGenerator.GenerateBySize(path, megabytes);
            Console.WriteLine("Not modified file is generated");

            var sw = Stopwatch.StartNew();
            KwaySort.MergeSort(path, out string sortedFileName, false);
            sw.Stop();
            Console.WriteLine($"Sorted,    file:    {sortedFileName},    seconds:
{sw.Elapsed.TotalSeconds}");
        }

        static void Modified(int linesNumber)
        {
            string path = "Amod.bin";
```

```
                    RandomGenerator.GenerateByLinesCount(path, linesNumber);
                    Console.WriteLine("Modified file is generated"); ;
                    var sw = Stopwatch.StartNew();
                    KwaySort.SortParts(path, "Result.bin", linesNumber, linesNumber / 8);
                    KwaySort.MergeSort("Result.bin", out string sortedFileName, true);
                    sw.Stop();
                    Console.WriteLine($"Sorted,     file:     {sortedFileName},     seconds:
{sw.Elapsed.TotalSeconds}");
                    Helper.ShowFile(sortedFileName, 20);
                }
            }
        }

        using System;
        using System.Collections.Generic;
        using System.IO;
        using System.Linq;

        namespace lab1
        {
            public class KwaySort
            {
                public static void MergeSort(string fileName, out string sortedFileName,
bool Modified, int numberOfFIles = 5)
                {
                    if (numberOfFIles < 2)
                        throw new ArgumentException(null, nameof(numberOfFIles));
                    if (Modified)
                    {
                        string[] BFilesArray = Enumerable.Range(1, numberOfFIles).Select(i
=> $"B{i}mod.bin").ToArray();
                        string[] CFilesArray = Enumerable.Range(1, numberOfFIles).Select(i
=> $"C{i}mod.bin").ToArray();
                        SplitFile(fileName, BFilesArray);
                        SortHelper(BFilesArray, CFilesArray, out sortedFileName);
                    }
                    else
                    {
                        string[] BFilesArray = Enumerable.Range(1, numberOfFIles).Select(i
=> $"B{i}.bin").ToArray();
                        string[] CFilesArray = Enumerable.Range(1, numberOfFIles).Select(i
=> $"C{i}.bin").ToArray();
                        SplitFile(fileName, BFilesArray);
                        SortHelper(BFilesArray, CFilesArray, out sortedFileName);
                    }
                }
```

```
            public static void SortParts(string fileName, string resultFileName, int
size, int shareSize)
            {
                if (File.Exists(resultFileName))
                {
                    File.Delete(resultFileName);
                }

                int[] array = new int[shareSize];
                var reader = new BinaryReader(File.Open(fileName, FileMode.Open));
                var     writer     =     new     BinaryWriter(File.Open(resultFileName,
FileMode.OpenOrCreate));
                for (int i = 0; i < size / shareSize; i++)
                {
                    for (int j = 0; j < shareSize; j++)
                    {
                        array[j] = reader.ReadInt32();
                    }

                    Array.Sort(array);
                    for (int j = 0; j < shareSize; j++)
                    {
                        writer.Write(array[j]);
                    }
                }

                reader.Close();
                writer.Close();
            }

            private static void SortHelper(string[] BFilesArray, string[] CFIlesArray,
out string fileName)
            {
                var       readers       =       BFilesArray.Select(f       =>       new
BinaryReader(File.OpenRead(f))).ToList();
                readers.Where(r          =>          (r.BaseStream.Position          ==
r.BaseStream.Length)).ToList().ForEach(r =>
                {
                    r.Dispose();
                    readers.Remove(r);
                });

                if (readers.Count == 1)
                {
                    fileName = ((FileStream) readers.First().BaseStream).Name;
                    return;
```

```
                }

                var writers = CFIlesArray.Select(f => new BinaryWriter(File.Open(f,
FileMode.Create))).ToList();
                var currentWriter = writers.First();
                var currentReader = readers.First();
                var nums = new List<int>();
                var nextNums = new List<int>();
                var readerAndPrevNum = readers.ToDictionary(r => r, _ => int.MinValue);

                while (readers.Count != 0)
                {
                    while (currentReader.BaseStream.Position ==
currentReader.BaseStream.Length)
                    {
                        var readerToRemove = currentReader;
                        currentReader = readers.Next(currentReader);
                        readers.Remove(readerToRemove);
                        readerAndPrevNum.Remove(readerToRemove);
                        readerToRemove.Dispose();
                        if (readers.Count == 0)
                        {
                            break;
                        }
                    }

                    if (readers.Count == 0)
                    {
                        nums.Sort();
                        foreach (int n in nums)
                        {
                            currentWriter.Write(n);
                        }

                        currentWriter = writers.Next(currentWriter);
                        nextNums.Sort();
                        foreach (int n in nextNums)
                        {
                            currentWriter.Write(n);
                        }

                        break;
                    }

                    int num = currentReader.ReadInt32();
                    if (num >= readerAndPrevNum[currentReader])
                    {
```

```
                    nums.Add(num);
                    readerAndPrevNum[currentReader] = num;
                }
                else
                {
                    nextNums.Add(num);
                    readerAndPrevNum[currentReader] = num;
                    currentReader = readers.Next(currentReader);
                }

                if (nextNums.Count >= readers.Count)
                {
                    nums.Sort();
                    foreach (int n in nums)
                    {
                        currentWriter.Write(n);
                    }

                    currentWriter = writers.Next(currentWriter);
                    nums.Clear();
                    nums.AddRange(nextNums);
                    nextNums.Clear();
                }
            }

            readers.ForEach(r => r.Dispose());
            writers.ForEach(w => w.Dispose());
            SortHelper(CFIlesArray, BFilesArray, out fileName);

        }


        private static void SplitFile(string fileName, string[] BFilesArray)
        {
            var writers = BFilesArray.Select(f => new BinaryWriter(File.Open(f,
FileMode.Create))).ToList();
            var currentWriter = writers.First();
            var reader = new BinaryReader(File.OpenRead(fileName));
            int previous = int.MinValue;

            while (reader.BaseStream.Position < reader.BaseStream.Length)
            {
                int current = reader.ReadInt32();
                if (current >= previous)
                {
                    currentWriter.Write(current);
                }
```

```
                    else
                    {
                        currentWriter = writers.Next(currentWriter);
                        currentWriter.Write(current);
                    }

                    previous = current;
                }

                writers.ForEach(w => w.Dispose());
                reader.Close();
            }
        }
    }

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

    namespace lab1
    {
        public static class Helper
        {
            public static long MegabytesToBytes(int megabytes) => (long) Math.Pow(2, 20)
* megabytes;

            public static T Next<T>(this List<T> source, T item)
            {
                int indexOfItem = source.IndexOf(item);
                return indexOfItem < source.Count - 1 ? source[indexOfItem + 1] :
source[0];
            }

            public static int[] GetArrayOfParts(int start, int size, string fileName)
            {
                var array = new int[size];
                var reader = new BinaryReader(File.OpenRead(fileName));
                for (int i = 0; i < start; i++)
                {
                    reader.ReadInt32();
                }

                for (int i = 0; i < size; i++)
                {
                    array[i] = reader.ReadInt32();
                }
                reader.Close();
```

```
                return array;
            }

            public static void ShowFile(string fileName, int size)
            {
                var array = GetArrayOfParts(0, size, fileName);
                Console.Write("[ ");
                foreach (var i in array)
                {
                    Console.Write(i + ", ");
                }
                Console.WriteLine("]");
            }
        }
    }

    using System;
    using System.IO;

    namespace lab1
    {
        public class RandomGenerator
        {
            public static void GenerateBySize(string fileName, int megabytes, int minNum
= 0, int maxNum = 1_000_000_000)
            {
                Random random = new Random();
                var writer = new BinaryWriter(File.Open(fileName, FileMode.Create));

                for (int i = 0; i % 10_000 != 0 || !(new FileInfo(fileName).Length >=
Helper.MegabytesToBytes(megabytes)); i++)
                {
                    writer.Write(random.Next(minNum, maxNum));
                }
                writer.Close();
            }
            public static void GenerateByLinesCount(string fileName, long linesCount,
int minNum = 0, int maxNum = 1_000_000_000)
            {
                Random random = new Random();
                var writer = new BinaryWriter(File.Open(fileName, FileMode.Create));
                int currentCount = 0;

                while (currentCount++ != linesCount)
                {
                    writer.Write(random.Next(minNum, maxNum));
                }
```

```
                writer.Close();
            }
        }
    }
```

ВИСНОВОК

У ході виконання лабораторної роботи було досліджено роботу алгоритму зовнішнього сортування «Збалансоване багатошляхове злиття» (K-way Merge Sort). В результаті було створено два алгоритми: стандартний та модифікований.

Стандартний алгоритм сортування складається з наступних кроків: розподіл вхідних даних на файли, зчитування серій та їх зливання в інші файли і повторення цих дій до закінчення сортування. Ознака закінчення сортування – один непорожній файл, що й міститиме у собі відсортовані дані.

Модифікований алгоритм сортування складається з наступних кроків: розподіл вхідних даних на великі файли, внутрішнє сортування цих файлів, їх злиття до закінчення сортування. Ознака закінчення сортування – один непорожній файл, що й міститиме у собі відсортовані дані.

Причина різниці швидкості роботи алгоритмів полягає у більшій швидкодії внутрішнього сортування, що застосоване в модифікації, а також у значно зменшеній кількості рекурсивних викликів для повторних злиттів файлів.

Стандартний алгоритм сортує 1ГБ цілочисельних даних в середньому за 54 хвилини, модифікований – 18 хвилин. Зважаючи на системні можливості ПК, можна вважати такі результати задовільними.

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%.