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-*_____ _____
(шифр, прізвище, ім'я, по батькові)

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

Київ 2024

ЗМІСТ

1. Мета лабораторної роботи

Метою лабораторної роботи є вивчення алгоритму зовнішнього сортування методом прямого злиття та його програмна реалізація. Робота передбачає розробку псевдокоду, написання програми, а також сортування випадковим чином згенерованого масиву цілих чисел, що зберігається у файлі великого розміру. Крім того, необхідно виконати модифікацію алгоритму для сортування файлів, розмір яких перевищує обсяг доступної оперативної пам’яті, з метою досягнення високої швидкості сортування. Завершальним етапом є аналіз ефективності базового та модифікованого алгоритмів із подальшим узагальненням отриманих результатів і оцінкою впливу модифікацій на продуктивність.

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. 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.

Варіант 21

№	**Алгоритм сортування**
21	Пряме злиття

3. Виконання

3.1 Псевдокод алгоритму

```
АЛГОРИТМ Пряме_злиття
   ВХІД: Шлях до файлу з масивом чисел (inputFilePath), шлях до файлу для збереження результату (outputFilePath), розмір файлу в Мб (sizeInMb)
   ВИХІД: Відсортований масив чисел, збережений у файлі

   ПОЧАТИ
      Ініціалізувати таймер для вимірювання часу виконання

      ВИКЛИК GenerateRandomNumbersToFile(inputFilePath, sizeInMb)
      МАСИВ array ← ReadNumbersFromFile(inputFilePath)
      МАСИВ sortedArray ← MergeSort(array)
      ВИКЛИК WriteNumbersToFile(outputFilePath, sortedArray)

      Вивести час виконання
   КІНЕЦЬ

ПІДАЛГОРИТМ GenerateRandomNumbersToFile(filePath, sizeInMb)
   ВІДКРИТИ файл filePath для запису
   Розрахувати totalNumbers = sizeInMb * 1024 * 1024 / 4
   ДЛЯ і від 0 ДО totalNumbers - 1
      ЗГЕНЕРУВАТИ випадкове число і записати його у файл
   ЗАКРИТИ файл
КІНЕЦЬ

ПІДАЛГОРИТМ ReadNumbersFromFile(filePath)
   ЗЧИТАТИ всі рядки з filePath
   ПЕРЕТВОРИТИ рядки в числа та зберегти у масив numbers
```

```
    ПОВЕРНУТИ numbers
КІНЕЦЬ

ПІДАЛГОРИТМ WriteNumbersToFile(filePath, array)
    ВІДКРИТИ файл filePath для запису
    ДЛЯ кожного елемента в array
        ЗАПИСАТИ елемент у файл
    ЗАКРИТИ файл
КІНЕЦЬ

ПІДАЛГОРИТМ MergeSort(array)
    ЯКЩО довжина array <= 1
        ПОВЕРНУТИ array
    РОЗДІЛИТИ array на дві частини: left і right
    ВИКЛИК MergeSort(left)
    ВИКЛИК MergeSort(right)
    ПОВЕРНУТИ Merge(left, right)
КІНЕЦЬ

ПІДАЛГОРИТМ Merge(left, right)
    ІНІЦІАЛІЗУВАТИ порожній масив result
    ПОКИ left і right не порожні
        ЯКЩО перший елемент left ≤ перший елемент right
            ДОДАТИ перший елемент left до result
            ВИДАЛИТИ цей елемент з left
        ІНАКШЕ
            ДОДАТИ перший елемент right до result
            ВИДАЛИТИ цей елемент з right
    ДОДАТИ залишки з left і right до result
    ПОВЕРНУТИ result
КІНЕЦЬ
```

3.2 ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ

3.2.1 Вихідний код

```
using System;
using System.Diagnostics;

class Program
{

    static void Main()
    {

         Stopwatch stopwatch = new Stopwatch();
         stopwatch.Start();

        string inputFilePath = "P:\\KPI-Works
3\\algorithm\\Code\\Laba1\\large_array.txt";
        string outputFilePath = "P:\\KPI-Works
3\\algorithm\\Code\\Laba1\\sorted_large_array.txt";

        GenerateRandomNumbersToFile(inputFilePath, 50);

        int[] array = ReadNumbersFromFile(inputFilePath);

        array = MergeSort(array);

        WriteNumbersToFile(outputFilePath, array);

        Console.WriteLine($"Sorted file is saved at {outputFilePath}");

        stopwatch.Stop();
        Console.WriteLine($"\n Sorting completed in
{stopwatch.Elapsed.TotalSeconds} seconds");

        Console.WriteLine($"Total time: {stopwatch.Elapsed.Hours} hours
{stopwatch.Elapsed.Minutes} minutes {stopwatch.Elapsed.Seconds} seconds");
    }

    static void GenerateRandomNumbersToFile(string filePath, int sizeInMb)
    {
        Random random = new Random();
        using (StreamWriter writer = new StreamWriter(filePath))
        {
            long totalNumbers = sizeInMb * 1024 * 1024 / 4;
            for (long i = 0; i < totalNumbers; i++)
            {
                writer.WriteLine(random.Next(-1000000000, 1000000000));
            }
        }
        Console.WriteLine($"Generated file with random numbers at
{filePath}");
    }

    static int[] ReadNumbersFromFile(string filePath)
    {
```

```
        string[] lines = File.ReadAllLines(filePath);
        int[] numbers = new int[lines.Length];

        for (int i = 0; i < lines.Length; i++)
        {
            numbers[i] = int.Parse(lines[i]);
        }

        return numbers;
    }

    static void WriteNumbersToFile(string filePath, int[] array)
    {
        using (StreamWriter writer = new StreamWriter(filePath))
        {
            foreach (int number in array)
            {
                writer.WriteLine(number);
            }
        }
    }



    static int[] MergeSort(int[] array)
    {
        if (array.Length <= 1)
            return array;

        // Разделяем массив на две части
        int mid = array.Length / 2;
        int[] left = new int[mid];
        int[] right = new int[array.Length - mid];

        Array.Copy(array, 0, left, 0, mid);
        Array.Copy(array, mid, right, 0, array.Length - mid);

        left = MergeSort(left);
        right = MergeSort(right);

        return Merge(left, right);
    }

    static int[] Merge(int[] left, int[] right)
    {
        int[] result = new int[left.Length + right.Length];
        int i = 0, j = 0, k = 0;

        while (i < left.Length && j < right.Length)
        {
            if (left[i] <= right[j])
            {
                result[k] = left[i];
                i++;
            }
            else
            {
```

```
                result[k] = right[j];
                j++;
            }
            k++;
        }

        while (i < left.Length)
        {
            result[k] = left[i];
            i++;
            k++;
        }

        while (j < right.Length)
        {
            result[k] = right[j];
            j++;
            k++;
        }

        return result;
    }
}
```

ВИСНОВОК

У ході виконання лабораторної роботи було досліджено алгоритм зовнішнього сортування за допомогою методу прямого злиття. Алгоритм дозволяє ефективно сортувати великі об'єми даних, які не можуть поміститися в оперативну пам'ять, використовуючи зовнішню пам'ять для зберігання і обробки даних.

У першій частині роботи було реалізовано базовий алгоритм сортування, який включав генерацію випадкових чисел, збереження їх у файл, читання з файлу та сортування з використанням алгоритму злиття. Оцінка часу виконання показала, що алгоритм здатний ефективно сортувати масиви середнього розміру (до кількох гігабайт) за порівняно короткий час.

У другій частині роботи була виконана модифікація програми для сортування великих файлів, розмір яких перевищує доступну оперативну пам'ять. Зокрема, застосовувалися техніки збереження частин масиву в проміжних файлах та їх подальше злиття, що дозволило значно зменшити час сортування великих обсягів даних. Було досягнуто високої ефективності, при цьому час сортування залишався в межах прийнятних для реальних умов.

Порівняння результатів базової та модифікованої версії програми показало, що застосування зовнішнього сортування значно збільшує масштабованість алгоритму і дозволяє працювати з файлами, обсяг яких суттєво перевищує розмір оперативної пам'яті комп'ютера.

Загалом, ця лабораторна робота дозволила краще зрозуміти принципи зовнішнього сортування та важливість ефективної роботи з великими даними, що є критично важливим при обробці великих обсягів інформації в реальних умовах.