**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)**

(шифр, прізвище, ім'я, по батькові)

ІП-11 Боровков І.І.

**Перевірив**

(прізвище, ім'я, по батькові)

Головченко М.М.

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc109342184)

[2 ЗаВдання 4](#_Toc109342185)

[3 Виконання 6](#_Toc109342186)

[3.1 Псевдокод алгоритму 6](#_Toc109342187)

[3.2 Програмна реалізація алгоритму 6](#_Toc109342188)

[3.2.1 Вихідний код 6](#_Toc109342189)

[Висновок 7](#_Toc109342190)

[Критерії оцінювання 8](#_Toc109342191)

# Мета лабораторної роботи

Мета роботи – вивчити основні алгоритми зовнішнього сортування та способи їх модифікації, оцінити поріг їх ефективності.

# 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 |

# Виконання

## Псевдокод алгоритму

**Main():**

**While(!**file1.empty() **and !**file2.empty()**) do**

**FormSeries();**

**MergeFiles();**

**end;**

**FormSeries():**

currEl **=** inputFile.read();

nextEl **=** inputFile.read();

isFirstFileSeries = **True;**

**while(!**inputFile.eof()**) do**

**if(**isFirstFileSeries**) do**

file1.write(currEl);

**end**

**else do**

file1.write(currEl);

**end;**

**if(**currEl **>** nextEl**) do**

isFirstFileSeries = !isFirstFileSeries;

**end;**

currEl = nextEl;

nextEl = inputFile.read();

**end;**

**if(**isFirstFileSeries**) do**

file1.write(currEl);

**end**

**else do**

file1.write(currEl);

**end;**

**MergeFiles():**

currEl1 **=** file1.read();

nextEl1 = file1.read();

currEl2 **=** file2.read();

nextEl2 = file2.read();

**while(!**file1.eof() **and !**file2.eof()**) do**

**if(**file1.eof()**) do**

currEl = file2.read();

inputFile.write(currEl);

**end;**

**elif(**file2.eof()**) do**

currEl = file1.read();

inputFile.write(currEl);

**end;**

**else do**

**if(**currEl1 > nextEl1 **and** currEl2 > nextEl2**) do**

**if(**currEl1 < currEl2**) do**

inputFile.write(currEl1);

inputFile.write(currEl2);

**end;**

**else**

inputFile.write(currEl2);

inputFile.write(currEl1);

**end;**

currEl1=nextEl1;

nextEl1=file1.read();

currEl2 = nextEl2;

nextEl2 = file2.read();

**end;**

**elif(**currEl1 <= nextEl1 **and** currEl2 > nextEl2**) do**

inputFile.write(currEl1);

currEl1 = nextEl1;

nextEl1=file1.read();

**end;**

**elif(**currEl1 > nextEl1 **and** currEl2 <= nextEl2**) do**

inputFile.write(currEl2);

currEl2 = nextEl2;

nextEl2=file2.read();

**end;**

**else do**

**if(**currEl1 < currEl2**) do**

inputFile.write(currEl1);

currEl1=nextEl1;

nextEl1 = file1.read();

**end;**

**else do**

inputFile.write(currEl2);

currEl2 = nextEl2;

nextEl2 = file2.read();

**end;**

**end;**

**end;**

**end;**

## Програмна реалізація алгоритму

### Вихідний код

Main.cpp

#include <iostream>  
#include <time.h>  
#include "Sorter.h"  
#include "ArrayGenerator.h"  
  
using namespace std;  
  
int main()  
{  
 srand(time(NULL));  
 char c;  
 cout << "Enter 1 to generate file \nEnter 2 to sort file \nEnter 3 to stop program \n";  
   
 bool isWorking = true;  
 while (isWorking)  
 {  
 cin >> c;  
 switch (c)  
 {  
 case '1':  
 cout << "Enter size of input file:\n";  
 int size;  
 cin >> size;  
 ArrayGenerator gen;  
 gen.GenerateArray("Input.txt", size);  
 cout << "File generated\n";  
 break;  
 case '2':  
 {  
 Sorter sorter;  
 sorter.SetInputFile("Input.txt");  
 sorter.SortFile();  
 cout << "File sorted\n";  
 break;  
 }  
 case '3':  
 isWorking = false;  
 default:  
 cout << "Wrong input";  
 }  
 }  
 return 0;  
}

ArrayGenerator.h

#pragma once  
  
class ArrayGenerator  
{  
public:  
 void GenerateArray(const char\* fileName, long long size);  
};

ArrayGenerator.cpp

#include "ArrayGenerator.h"  
#include <fstream>  
  
void ArrayGenerator::GenerateArray(const char\* fileName, long long size)  
{  
 std::ofstream file(fileName, std::ios::trunc);  
 for (long long i = 0; i < size; i++)  
 {  
 file << ' ' << rand() % size + 1;  
 }  
 file.close();  
}

Sorter.h

#pragma once  
#include <fstream>  
#include <string>  
*/\*\**  
 *\* Sort array in file*  
 *\*/*  
class Sorter  
{  
private:  
 std::string inputFileName;  
 std::fstream inputFile, tempFile1, tempFile2;  
   
public:  
 void SetInputFile(const char\* fileName);  
 void SortFile();  
 bool FormTempFiles();  
 void MergeTempFiles();  
};

Sorter.cpp

#include "Sorter.h"  
#include <iostream>  
  
*/\*\**  
 *\* \brief Set file that sorter should process*  
 *\* \param fileName name of file with unsorted array*  
 *\*/*  
void Sorter::SetInputFile(const char\* fileName)  
{  
 try  
 {  
 std::ifstream findFile(fileName);  
 if (!findFile)  
 throw("File doesn't exist.");  
 else  
 inputFileName = fileName;  
 findFile.close();  
 }  
 catch (const char\* msg)  
 {  
 std::cout << msg;  
 }  
}  
  
*/\*\**  
 *\* \brief Sort array in file*  
*\*/*  
void Sorter::SortFile()  
{  
 try  
 {  
 if (inputFileName.empty())  
 throw("File hasn't been chosen.");  
 while (FormTempFiles())  
 {  
 MergeTempFiles();  
 }  
 }  
 catch (const char\* msg)  
 {  
 std::cout << msg;  
 return;  
 }  
}  
  
*/\*\**  
 *\* \brief Function form temporary files of native algorithm*  
 *\* \return False if it is only one temp file (array is already sorted)*  
 *\*/*  
bool Sorter::FormTempFiles()  
{  
 long long currEl, nextEl;  
 bool fileSwitcher = true;  
  
 inputFile.open(inputFileName, std::ios::in);  
 tempFile1.open("Temp1.dat", std::ios::out | std::ios::trunc);  
 tempFile2.open("Temp2.dat", std::ios::out | std::ios::trunc);  
  
 inputFile >> currEl;  
 while (!inputFile.eof())  
 {  
 inputFile >> nextEl;  
  
 if (fileSwitcher)  
 tempFile1 << currEl << ' ';  
 else  
 tempFile2 << currEl << ' ';  
  
 if (nextEl < currEl)  
 fileSwitcher = !fileSwitcher;  
  
 currEl = nextEl;  
 }  
   
 *//finish tempFiles*  
if (fileSwitcher)  
 tempFile1 << currEl << ' ';   
 else  
 tempFile2 << currEl << ' ';  
  
 tempFile1.seekg(0, std::ios\_base::end);  
 tempFile2.seekg(0, std::ios\_base::end);  
 if (tempFile1.tellg() == 0 ||tempFile2.tellg() ==0)  
 {  
 return false;  
 }  
 *//Add garbage series to the end of temp files. They make merge algo process last series to the end*  
tempFile1 << nextEl - 1 << ' ' << nextEl - 1;  
   
 tempFile2 << nextEl - 1 << ' ' << nextEl - 1;  
   
 inputFile.close();  
 tempFile1.close();  
 tempFile2.close();  
  
 return true;  
}  
  
*/\*\**  
 *\* \brief Function merge formed temporary files with series*  
 *\*/*  
void Sorter::MergeTempFiles()  
{  
 inputFile.open(inputFileName, std::ios::out | std::ios::trunc);  
 tempFile1.open("Temp1.dat", std::ios::in);  
 tempFile2.open("Temp2.dat", std::ios::in);  
  
 long long currEl1, currEl2, nextEl1, nextEl2;  
   
 tempFile1 >> currEl1 >> nextEl1;  
 tempFile2 >> currEl2 >> nextEl2;  
  
 while (!(tempFile1.eof() && tempFile2.eof()))  
 {  
 *// Finish inputFile if one of the tempFile ended*  
if (tempFile1.eof())  
 {  
 inputFile << ' ' << currEl2;  
 currEl2 = nextEl2;  
 tempFile2 >> nextEl2;  
 continue;  
 }  
   
 if (tempFile2.eof())  
 {  
 inputFile << ' ' << currEl1;  
 currEl1 = nextEl1;  
 tempFile1 >> nextEl1;  
 continue;  
 }  
  
 *// Main logic*  
if (currEl1 > nextEl1 && currEl2 > nextEl2)  
 {  
 if (currEl1 < currEl2)  
 {  
 inputFile << ' ' << currEl1 << ' ' << currEl2;  
 }  
 else  
 {  
 inputFile << ' ' << currEl2 << ' ' << currEl1;  
 }  
 currEl1 = nextEl1;  
 tempFile1 >> nextEl1;  
 currEl2 = nextEl2;  
 tempFile2 >> nextEl2;  
 }  
 else  
 if (currEl1 <= nextEl1 && currEl2 > nextEl2)  
 {  
 inputFile << ' ' << currEl1;  
 currEl1 = nextEl1;  
 tempFile1 >> nextEl1;  
 }  
 else  
 if (currEl1 > nextEl1 && currEl2 <= nextEl2)  
 {  
 inputFile << ' ' << currEl2;  
 currEl2 = nextEl2;  
 tempFile2 >> nextEl2;  
 }  
 else  
 if (currEl1 <= nextEl1 && currEl2 <= nextEl2)  
 {  
 if (currEl1 < currEl2)  
 {  
 inputFile << ' ' << currEl1;  
 currEl1 = nextEl1;  
 tempFile1 >> nextEl1;  
 }  
 else  
 {  
 inputFile << ' ' << currEl2;  
 currEl2 = nextEl2;  
 tempFile2 >> nextEl2;  
 }  
 }  
 }  
   
 inputFile.close();  
 tempFile1.close();  
 tempFile2.close();  
}

Висновок

При виконанні даної лабораторної роботи я вивчив основні алгоритми зовнішнього сортування та способи їх модифікації та оцінив поріг їх ефективності. В ході роботи було розроблено та досліджено алгоритм природного сортування, логіка роботи якого заснована на пошуку відсортованих серій чисел та поступовому їх об’єднанні у більші.

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%.