**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-12 Басараб О. А.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О.О.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc115541724)

[2 Завдання 4](#_Toc115541725)

[3 Виконання 6](#_Toc115541726)

[3.1 Псевдокод алгоритму 6](#_Toc115541727)

[3.2 Програмна реалізація алгоритму 9](#_Toc115541728)

[3.2.1 Вихідний код 9](#_Toc115541729)

[3.3 Тестування програмної реалізації 12](#_Toc115541730)

[Висновок 16](#_Toc115541736)

[Критерії оцінювання 17](#_Toc115541737)

# 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

Варіант 2

Природне (адаптивне) злиття

## Псевдокод алгоритму

**procedure** sort()

**while** notisSorted() **do**

split()

merge()

**end while**

**end procedure**

**function** isSorted()

inA = open(pathA)

isSorted = true

curr = inA.read()

**while** not inA.eof() **do**

next = inA.read()

**if** not inA.eof() **then**

**if** next < curr **then**

isSorted = false

break

**end if**

curr = next

**end if**

**end while**

inA.close()

**return** isSorted

**end function**

**procedure** split()

inA = open(pathA)

outB = open(pathB)

outC = open(pathC)

writeToB = true

curr = inA.read()

**while** not inA.eof() **do**

**if** writeToB == true **then**

outB.write(curr)

**else**

outC.write(curr)

**end if**

next = inA.read()

**if** not inA.eof() **then**

**if** next < curr **then**

writeToB = not WriteToB

**end if**

curr = next

**end if**

**end while**

curr = -inf

outB.write(curr)

outC.write(curr)

inA.close()

outB.close()

outC.close()

**end procedure**

**procedure** merge()

outA = open(pathA)

inB = open(pathB)

inC = open(pathC)

currB = inB.read()

currC = inC.read()

nextB = inB.read()

nextC = inC.read()

**repeat**

**if** currB <= nextB and currC <= nextC **then**

**if** currB <= currC **then**

outA.write(currB)

currB = nextB

nextB = inB.read()

**else**

outA.write(currC)

currC = nextC

nextC = inC.read()

**end if**

**else**

**if** currB > nextB and currC <= nextC **then**

**while** currC <= nextC and not inC.eof() **do**

**if** currB <= currC **then**

outA.write(currB)

currB = nextB

nextB = inB.read()

**while** currC <= nextC and not inC.eof() **do**

outA.write(currC)

currC = nextC

nextC = inC.read()

**end while**

**if** not inC.eof() **then**

outA.write(currC)

currC = nextC

nextC = inC.read()

**end if**

break

**else**

outA.write(currC)

currC = nextC

nextC = inC.read()

**end if**

**end while**

**else**

**if** currB <= nextB and currC > nextC **then**

**while** currB <= nextB and not inB.eof() **do**

**if** currC <= currB **then**

outA.write(currC)

currC = nextC

nextC = inC.read()

**while** currB <= nextB and not inB.eof() **do**

outA.write(currB)

currB = nextB

nextB = inB.read()

**end while**

**if** not inB.eof() **then**

outA.write(currB)

currB = nextB

nextB = inB.read()

**end if**

break

**else**

outA.write(currB)

currB = nextB

nextB = inB.read()

**end if**

**end while**

**else**

**if** currB <= currC **then**

outA.write(currB)

outA.write(currC)

**else**

outA.write(currC)

outA.write(currB)

**end if**

currB = nextB

currC = nextC

nextB = inB.read()

nextC = inC.read()

**end if**

**end if**

**end if**

**until** not inB.eof() and not inC.eof()

**if** not inB.eof() **then**

**while** not inB.eof() **do**

outA.write(currB)

currB = nextB

nextB = inB.read()

**end while**

**end if**

**if** not inC.eof() **then**

**while** not inC.eof() **do**

outA.write(currC)

currC = nextC

nextC = inC.read()

**end while**

**end if**

outA.close()

inB.close()

inC.close()

**end procedure**

## Програмна реалізація алгоритму

### Вихідний код

**ExternalSortEngine.h**

#pragma once

const int MAX\_PATH\_SIZE{ 50 };

class ExternalSortEngine

{

private:

char mPathA[MAX\_PATH\_SIZE], mPathB[MAX\_PATH\_SIZE], mPathC[MAX\_PATH\_SIZE];

// Розбиття файлу A на файли B і C, в які будуть послідовно записані "серії" з вихідного файлу;

// "Серія" -- неспадна послідовність чисел з вихідного файлу

void split();

// Злиття відповідних серій з файлів B і C в файл A

void merge();

public:

ExternalSortEngine(const char\* pathA, const char\* pathB, const char\* pathC);

// Сортування набору чисел з файлу A в неспадному порядку

void sort();

// Генерація набору випадкових aLength випадкових чисел, що будуть записані до файлу A

void generateRandomA(long long aLength);

// Перевірка на відсортованість в неспадному порядку чисел файлу A

bool isSorted();

void print10(const char\* filePath);

};

**ExternalSortEngine.cpp**

#include "ExternalSortEngine.h"

#include <iostream>

#include <fstream>

#include <random>

#include <ctime>

ExternalSortEngine::ExternalSortEngine(const char\* pathA, const char\* pathB, const char\* pathC) {

strcpy\_s(mPathA, pathA);

strcpy\_s(mPathB, pathB);

strcpy\_s(mPathC, pathC);

}

// Перевірка на відсортованість в неспадному порядку чисел файлу A

bool ExternalSortEngine::isSorted() {

std::fstream inA;

inA.open(mPathA, std::ios::binary | std::ios::in);

bool isSorted{ true };

int next{}, curr{};

inA.read((char\*)&curr, sizeof(curr));

while (!inA.eof()) {

inA.read((char\*)&next, sizeof(next));

if (!inA.eof()) {

if (next < curr) {

isSorted = false;

break;

}

curr = next;

}

}

inA.close();

return isSorted;

}

// Сортування набору чисел з файлу A в неспадному порядку

void ExternalSortEngine::sort() {

while (!isSorted()) {

this->split();

this->merge();

}

}

void ExternalSortEngine::print10(const char\* filePath) {

std::fstream fin;

fin.open(filePath, std::ios::binary | std::ios::in);

int curr{};

fin.read((char\*)&curr, sizeof(curr));

for (int i = 0; i < 10 && !fin.eof(); ++i) {

std::cout << curr << "\t";

fin.read((char\*)&curr, sizeof(curr));

}

std::cout << "\n";

fin.close();

}

// Розбиття файлу A на файли B і C, в які будуть послідовно записані "серії" з вихідного файлу;

// "Серія" -- неспадна послідовність чисел з вихідного файлу

void ExternalSortEngine::split() {

std::fstream inA, outB, outC;

inA.open(mPathA, std::ios::binary | std::ios::in);

outB.open(mPathB, std::ios::binary | std::ios::out | std::ios::trunc);

outC.open(mPathC, std::ios::binary | std::ios::out | std::ios::trunc);

bool writeToB{ true };

int next{}, curr{};

inA.read((char\*)&curr, sizeof(curr));

while (!inA.eof()) {

if (writeToB) {

outB.write((char\*)&curr, sizeof(curr));

}

else {

outC.write((char\*)&curr, sizeof(curr));

}

inA.read((char\*)&next, sizeof(next));

if (!inA.eof()) {

if (next < curr) {

writeToB = !writeToB;

}

curr = next;

}

}

curr = INT\_MIN;

outB.write((char\*)&curr, sizeof(curr));

outC.write((char\*)&curr, sizeof(curr));

inA.close();

outB.close();

outC.close();

}

// Злиття відповідних серій з файлів B і C в файл A

void ExternalSortEngine::merge() {

std::fstream outA, inB, inC;

outA.open(mPathA, std::ios::binary | std::ios::out | std::ios::trunc);

inB.open(mPathB, std::ios::binary | std::ios::in);

inC.open(mPathC, std::ios::binary | std::ios::in);

int currB{}, currC{}, nextB{}, nextC{};

inB.read((char\*)&currB, sizeof(currB));

inC.read((char\*)&currC, sizeof(currC));

inB.read((char\*)&nextB, sizeof(nextB));

inC.read((char\*)&nextC, sizeof(nextC));

do {

// Якщо поточні елементи файлів B і C не є останніми в своїх серіях,

// то записуємо менший з них до файлу A і зчитуємо наступний елемент відповідної серії

if ((currB <= nextB) && (currC <= nextC)) {

if (currB <= currC) {

outA.write((char\*)&currB, sizeof(currB));

currB = nextB;

inB.read((char\*)&nextB, sizeof(nextB));

}

else {

outA.write((char\*)&currC, sizeof(currC));

currC = nextC;

inC.read((char\*)&nextC, sizeof(nextC));

}

}

else {

// Опрацювання випадку, коли поточний елемент з файлу B є останнім в своїй серії,

// а поточний елемент з файлу C не є останнім в своїй серії

if ((currB > nextB) && (currC <= nextC)) {

while (currC <= nextC && !inC.eof()) {

if (currB <= currC) {

outA.write((char\*)&currB, sizeof(currB));

currB = nextB;

inB.read((char\*)&nextB, sizeof(nextB));

while (currC <= nextC && !inC.eof()) {

outA.write((char\*)&currC, sizeof(currC));

currC = nextC;

inC.read((char\*)&nextC, sizeof(nextC));

}

if (!inC.eof()) {

outA.write((char\*)&currC, sizeof(currC));

currC = nextC;

inC.read((char\*)&nextC, sizeof(nextC));

}

break;

}

else {

outA.write((char\*)&currC, sizeof(currC));

currC = nextC;

inC.read((char\*)&nextC, sizeof(nextC));

}

}

}

else {

// Опрацювання випадку, коли поточний елемент з файлу C є останнім в своїй серії,

// а поточний елемент з файлу B не є останнім в своїй серії

if ((currB <= nextB) && (currC > nextC)) {

while (currB <= nextB && !inB.eof()) {

if (currC <= currB) {

outA.write((char\*)&currC, sizeof(currC));

currC = nextC;

inC.read((char\*)&nextC, sizeof(nextC));

while (currB <= nextB && !inB.eof()) {

outA.write((char\*)&currB, sizeof(currB));

currB = nextB;

inB.read((char\*)&nextB, sizeof(nextB));

}

if (!inB.eof()) {

outA.write((char\*)&currB, sizeof(currB));

currB = nextB;

inB.read((char\*)&nextB, sizeof(nextB));

}

break;

}

else {

outA.write((char\*)&currB, sizeof(currB));

currB = nextB;

inB.read((char\*)&nextB, sizeof(nextB));

}

}

}

else {

// Якщо поточні елементи файлів A та B є останніми в своїх серіях,

// то спершу записуємо менший з них до файлу A, а після більший;

// далі зчитуємо наступні елементи обох файлів

if (currB <= currC) {

outA.write((char\*)&currB, sizeof(currB));

outA.write((char\*)&currC, sizeof(currC));

}

else {

outA.write((char\*)&currC, sizeof(currC));

outA.write((char\*)&currB, sizeof(currB));

}

currC = nextC;

currB = nextB;

inB.read((char\*)&nextB, sizeof(nextB));

inC.read((char\*)&nextC, sizeof(nextC));

}

}

}

} while (!inB.eof() && !inC.eof());

// Якщо в файлі B залишилися елементи, записуємо їх до файлу A

if (!inB.eof()) {

while (!inB.eof()) {

outA.write((char\*)&currB, sizeof(currB));

currB = nextB;

inB.read((char\*)&nextB, sizeof(nextB));

}

}

// Якщо в файлі C залишилися елементи, записуємо їх до файлу A

if (!inC.eof()) {

while (!inC.eof()) {

outA.write((char\*)&currC, sizeof(currC));

currC = nextC;

inC.read((char\*)&nextC, sizeof(nextC));

}

}

outA.close();

inB.close();

inC.close();

}

// Генерація набору випадкових aLength випадкових чисел, що будуть записані до файлу A

void ExternalSortEngine::generateRandomA(long long aLength) {

std::fstream fileA, fileA1;

std::random\_device rd;

std::default\_random\_engine defEngine(time(NULL));

std::uniform\_int\_distribution<int> dist(INT\_MIN + 1, INT\_MAX - 1);

fileA.open(mPathA, std::ios::binary | std::ios::out | std::ios::trunc);

for (int i{ 0 }; i < aLength; ++i) {

int curr = dist(defEngine);

fileA.write((char\*)&curr, sizeof(curr));

fileA1.write((char\*)&curr, sizeof(curr));

}

fileA.close();

fileA1.close();

}

**main.cpp**

#include <iostream>

#include "ExternalSortEngine.h"

#include <fstream>

#include <ctime>

int main()

{

const char\* pathA{ "a.dat" }, \* pathB{ "b.dat" }, \* pathC{ "c.dat" };

ExternalSortEngine se(pathA, pathB, pathC);

se.generateRandomA(10 \* 1024 \* 1024 / 4);

time\_t tStart = time(0);

se.sort();

time\_t tEnd = time(0);

if (se.isSorted()) {

std::cout << "Sorting is successful.\n";

}

std::cout << "Sorting time:\t" << (tEnd - tStart) << " secs\n";

return 0;

}

## Тестування програмної реалізації

Для тестування було створено файл розміром в 10 Мб за допомогою методу generateRandomA(10 \* 1024 \* 1024 / 4) та здійснено його сортування. Успішність сортування перевірено методом isSorted(), що виводить відповідне повідомлення. Також було виведено час сортування. Маємо:

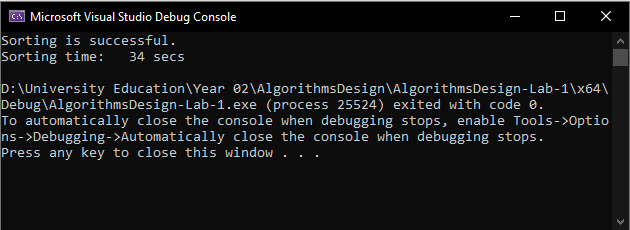


Рисунок 3.1 — Тестування програмної реалізації базового алгоритму

Висновок

При виконанні даної лабораторної роботи було спроектовано варіант реалізації алгоритму зовнішнього сортування «природне (адаптивне) злиття» у вигляді псевдокоду, була здійснена його реалізація на мові C++ та тестування на файлі розміру 10 Мб, що складається з випадково згенерованих чисел. Для сортування файлу розміром 10 Мб програмі знадобилося 34 секунди. Такий результат значно поступається алгоритмам внутрішнього сортування, що означає потребу в модифікації базового алгоритму.

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%.