**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)

*Титаренко Данило Олегович*

**Перевірив**

(прізвище, ім'я, по батькові)

*Сопов О. О.*

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

void first\_distribution(string filename, filemanager& manager)

{

fstream f;

f.open(filename, ios::in | ios::binary);

int counter = manager.get\_out();

fibonacci fib\_vector(counter);

fib\_vector.make\_order\_vector();

fib\_vector.make\_dist\_mass();

int current = 0;

int next = 0;

int current\_file = 0;

int length = 0;

streampos start\_pos = manager.output[current\_file].file\_object.tellg();

streampos end\_pos;

manager.output[current\_file].real\_series++;

manager.write(current\_file, 0);

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

int\* diff = new int[counter];

diff[0] = 0;

**for** i = 1 **to** counter **do**

diff[i] = 1;

**end for**

bool flag = 0;

**while** (true) **do**

**while** (current <= next) **do**

manager.write(current\_file, next);

length++;

current = next;

**if** (!f.eof())

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

**end if**

**else**

flag = !flag;

break;

**end else**

**end while**

**if** (flag == 1) break;

**end if**

end\_pos = manager.output[current\_file].file\_object.tellg();

manager.output[current\_file].file\_object.seekg(start\_pos);

manager.write(current\_file, length);

manager.output[current\_file].file\_object.seekg(end\_pos);

**if** ((current\_file < counter - 1) && ((diff[current\_file] + 1) == diff[current\_file + 1]))

current\_file++;

diff[current\_file]--;

manager.output[current\_file].real\_series++;

**end if**

**else**

**if** (!diff[counter - 1])

fib\_vector.make\_order\_vector();

fib\_vector.make\_dist\_mass();

**for** i = 0 **to** counter **do**

diff[i] = fib\_vector.mass[i] - manager.output[i].real\_series;

**end for**

**end if**

current\_file = 0;

diff[current\_file]--;

manager.output[current\_file].real\_series++;

**end else**

start\_pos = manager.output[current\_file].file\_object.tellg();

manager.write(current\_file, 0);

length = 0;

manager.write(current\_file, next);

length++;

current = next;

**if** (!f.eof())

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

**end if**

**else**

break;

**end else**

**end while**

end\_pos = manager.output[current\_file].file\_object.tellg();

manager.output[current\_file].file\_object.seekg(start\_pos);

manager.write(current\_file, length);

manager.output[current\_file].file\_object.seekg(end\_pos);

**for** i = 0 **to** counter **do**

manager.output[i].empty\_series = diff[i];

**end for**

f.close();

}

**void merge**(filemanager& manager)

{

multimap <int, int> buffer;

int count = manager.get\_in();

int\* length\_of\_series = new int[count];

int sum\_of\_series = 2;

int current\_length;

**while** (sum\_of\_series > 1) **do**

{

current\_length = 0;

**for** i = 0 to count **do**

{

**if** (manager.input[i].empty\_series)

length\_of\_series[i] = 0;

manager.input[i].empty\_series--;

**end if**

**else**

{

**if** (!manager.input[i].real\_series)

{

manager.index\_swap(i, 0);

**if** (manager.input[i].empty\_series)

i--;

continue;

**end if**

**end if**

length\_of\_series[i] = manager.read(i);

manager.input[i].real\_series--;

buffer.insert(pair <int, int>(manager.read(i), i));

**end else**

current\_length += length\_of\_series[i]--;

**end for**

manager.write(0, current\_length);

manager.output[0].real\_series++;

auto buf = buffer.begin();

**while** (!buffer.empty())

{

buf = buffer.begin();

manager.write(0, buf->first);

**if** (length\_of\_series[buf->second]--)

buffer.insert(pair <int, int>(manager.read(buf->second), buf->second));

**end if**

buffer.erase(buf);

**end while**

sum\_of\_series = 0;

**for** i = 0 to count **do**

sum\_of\_series += manager.input[i].real\_series + manager.input[i].empty\_series;

**end for**

sum\_of\_series += manager.output[0].real\_series + manager.output[0].empty\_series;

**end while**

}

**void sort\_function**(filemanager& manager, polyphase& sorting, bool show\_output, clock\_t& start\_dist, clock\_t& end\_dist, clock\_t& start\_merge, clock\_t& end\_merge)

{

start\_dist = clock();

sorting.first\_distribution(input\_name, manager);

end\_dist = clock();

int max = manager.output[0].empty\_series;

**for** i = 1 **to** < amount\_of\_files – 1 **do**

**if** (max > manager.output[i].empty\_series)

max = manager.output[i].empty\_series;

**end if**

**end for**

**if** (max != 0)

**for** i = 1 **to** < amount\_of\_files – 1 **do**

manager.output[i].empty\_series -= max;

**end for**

manager.input[0].empty\_series += max;

**end if**

manager.fileswap();

start\_merge = clock();

sorting.merge(manager);

manager.fileswap();

}

## Програмна реалізація алгоритму

### Вихідний код

Файл Polyphase\_Sort.cpp

#include "Polyphase\_merge.h"

#include <ctime>

#define input\_name "input.txt"

#define output\_name "output.txt"

#define debug\_file "debug\_info.txt"

const int amount\_of\_numbers = 3000000;

const int amount\_of\_files = 5;

using namespace std;

/\*Показати вихідні файли \*/

void show\_output\_files(polyphase& sorting)

{

for (int i = 0; i < amount\_of\_files - 1; i++)

{

string file\_name = "file\_output\_" + to\_string(i) + ".txt";

cout << file\_name << ": ";

sorting.show\_binary\_file\_length(file\_name);

}

}

/\* Функція сортування \*/

void sort\_function(filemanager& manager, polyphase& sorting, bool show\_output, clock\_t& start\_dist, clock\_t& end\_dist, clock\_t& start\_merge, clock\_t& end\_merge)

{

start\_dist = clock();

sorting.first\_distribution(input\_name, manager);

end\_dist = clock();

if (show\_output)

{

cout << "Файлы містять (перший розподіл): " << endl;

show\_output\_files(sorting);

}

/\* Працюємо з порожніми серіями \*/

int max = manager.output[0].empty\_series;

for (int i = 1; i < amount\_of\_files - 1; i++)

{

if (max > manager.output[i].empty\_series)

{

max = manager.output[i].empty\_series; //Забираємо максимальну к-ть порожніх серій

}

}

if (max != 0) //Якщо 0 - то продовжуємо, інакше пропускаємо

{

for (int i = 0; i < amount\_of\_files - 1; i++)

{

manager.output[i].empty\_series -= max;

}

manager.input[0].empty\_series += max;

}

manager.fileswap();

cout << endl << "Сортування... ";

start\_merge = clock();

sorting.merge(manager);

end\_merge = clock();

cout << "Виконано" << endl;

manager.fileswap();

}

/\* Інфа \*/

void view\_debug\_info(polyphase& sorting)

{

cout << "Файл " << input\_name << " містить: " << endl;

int quantity = sorting.show\_binary\_file\_eof(input\_name);

cout << "К-ть перевірених значень:" << quantity << endl << endl;

cout << "Перевірка серій: " << endl;

quantity = sorting.amount\_of\_series(input\_name);

cout << "К-ть серій: " << quantity << endl << endl;

}

int main()

{

setlocale(LC\_ALL, "Russian");

filemanager manager(1, amount\_of\_files - 1);

polyphase sorting;

clock\_t start\_generate, end\_generate, start\_dist, end\_dist, start\_merge, end\_merge;

start\_generate = clock();

sorting.generate(input\_name, amount\_of\_numbers);

end\_generate = clock();

sort\_function(manager, sorting, 0, start\_dist, end\_dist, start\_merge, end\_merge);

/\* Запис результату сортування у файл\*/

fstream f;

f.open(output\_name, ios::out);

int length = manager.read(0);

for (int i = 0; i < length; i++)

{

f << " " << manager.read(0);

}

f.close();

cout << endl;

/\* Вивід на экран \*/

sorting.check\_sort(output\_name);

float gen\_t = ((double)end\_generate - start\_generate) / ((double)CLOCKS\_PER\_SEC);

float merge\_t = ((double)end\_merge - start\_merge) / ((double)CLOCKS\_PER\_SEC);

cout << "К-ть елементів: " << amount\_of\_numbers << endl;

cout << "К-ть файлів: " << amount\_of\_files << endl;

cout << "Час генерації: " << gen\_t << " сек." << endl;

cout << "Час сортування: " << merge\_t << " сек." << endl << endl;

}

Файл Polyphase\_merge.h

#pragma once

#pragma once

#include "Filemanager.h"

#include "Fibonacci.h"

#include <string>

#include <ctime>

#include <iostream>

#include <fstream>

#include <map>

using namespace std;

class polyphase

{

public:

vector <int> position\_series; //Вектор, який зберігає позиції "обривів" серій

polyphase() {};

void generate(string filename, int amount); //Генератор файлу

int show\_binary\_file\_eof(string filename); //Вывод бінарного файла (до кінця файлу)

int show\_binary\_file\_length(string filename); //Вивід бінарного файлу (на основі довжини серій)

void show\_txt\_file(string filename); //Вивід текстового файлу

int check\_sort(string filename); //Перевірка сортування

void first\_distribution(string filename, filemanager& manager); //Перше розділення

void merge(filemanager& manager); //Сортування

bool eof(string filename); //Перевірка на порожність

int amount\_of\_series(string filename); //Повернення к-ті серій

};

bool polyphase::eof(string filename)

{

fstream f;

f.open(filename, ios::in | ios::binary | ios::ate);

bool eof = !f.tellg();

f.close();

return eof;

}

void polyphase::merge(filemanager& manager)

{

multimap <int, int> buffer; //контейнер: <значення, номер файлу>

int count = manager.get\_in(); //К-ть файлів на вході

int\* length\_of\_series = new int[count]; //Масив довжин серій

int sum\_of\_series = 2; //Загальна сума серій

int current\_length; //Поточна довжина

while (sum\_of\_series > 1)

{

current\_length = 0;

for (int i = 0; i < count; i++)

{

if (manager.input[i].empty\_series) //Якщо є порожні серії

{

length\_of\_series[i] = 0;

manager.input[i].empty\_series--;

}

else

{

if (!manager.input[i].real\_series) //Якщо нема справжніх серій

{

manager.index\_swap(i, 0);

if (manager.input[i].empty\_series) //Дивимся наявність порожніх серій в новому файлі

{

i--;

continue; //Починаємо заново, ігноруючи подальші операції

}

}

length\_of\_series[i] = manager.read(i);

manager.input[i].real\_series--;

buffer.insert(pair <int, int>(manager.read(i), i));

}

current\_length += length\_of\_series[i]--;

}

manager.write(0, current\_length);

manager.output[0].real\_series++;

auto buf = buffer.begin();

while (!buffer.empty())

{

buf = buffer.begin();

manager.write(0, buf->first);

if (length\_of\_series[buf->second]--)

{

buffer.insert(pair <int, int>(manager.read(buf->second), buf->second));

}

buffer.erase(buf);

}

/\* Рахуємо суму серій \*/

sum\_of\_series = 0;

for (int i = 0; i < count; i++)

{

sum\_of\_series += manager.input[i].real\_series + manager.input[i].empty\_series;

}

sum\_of\_series += manager.output[0].real\_series + manager.output[0].empty\_series;

}

}

void polyphase::first\_distribution(string filename, filemanager& manager)

{

fstream f; //Відкриваємо сгенерований файл

f.open(filename, ios::in | ios::binary);

int counter = manager.get\_out(); //Забираємо число вихідних файлів

fibonacci fib\_vector(counter); //Робимо ряд фібоначчі порядку counter

fib\_vector.make\_order\_vector(); //Робимо початковий ряд фібоначчі

fib\_vector.make\_dist\_mass();

int current = 0; //Поточна позиція

int next = 0; //позиція після поточної

int current\_file = 0; //Поточний файл

int length = 0; //Довжина серії

streampos start\_pos = manager.output[current\_file].file\_object.tellg(); //Початкова позиція(тут 0)

streampos end\_pos; //Кінцева позиція

manager.output[current\_file].real\_series++; //Збільшуємо серію на 1 (т.к. начинаем)

manager.write(current\_file, 0); //Резервуємо місце під запис серій

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

int\* diff = new int[counter]; //Різниця між поточним та розподіленням фібоначчі

diff[0] = 0;

for (int i = 1; i < counter; i++)

{

diff[i] = 1; //[0 1 1 1 1 1 1 1 ...]

}

bool flag = 0; //Прапор виходу

while (true)

{

/\* запись серии \*/

while (current <= next)

{

manager.write(current\_file, next); //Пишемо в поточний файл обраний елемент

length++; //Збільшуємо довжину серії

current = next;

if (!f.eof())

{

//f >> next; //Якщо файл не скінчився беремо наступний

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

}

else

{

flag = !flag; //Виходимо з циклів

break;

}

}

if (flag == 1) break;

/\* Змінюємо файли \*/

end\_pos = manager.output[current\_file].file\_object.tellg(); //Запамятовуємо поточну(кінццеву позицію в файлі)

manager.output[current\_file].file\_object.seekg(start\_pos); //Переходимо на початок файлу

manager.write(current\_file, length); //Записуємо довжину замість 0

manager.output[current\_file].file\_object.seekg(end\_pos); //Повертаємося на збережену позицію

/\* Для розподілу \*/

if ((current\_file < counter - 1) && ((diff[current\_file] + 1) == diff[current\_file + 1]))

{

current\_file++;

diff[current\_file]--;

manager.output[current\_file].real\_series++;

}

else

{

if (!diff[counter - 1])

{

fib\_vector.make\_order\_vector();

fib\_vector.make\_dist\_mass();

for (int i = 0; i < counter; i++)

{

diff[i] = fib\_vector.mass[i] - manager.output[i].real\_series;

}

}

current\_file = 0;

diff[current\_file]--;

manager.output[current\_file].real\_series++;

}

/\* Зберігаємо місце для запису довжини серій \*/

start\_pos = manager.output[current\_file].file\_object.tellg(); //Робимо стартову позицію поточною

manager.write(current\_file, 0); //Резервуємо місце під довжину серій

length = 0; //Обнуляем длину серии

manager.write(current\_file, next); //Пишемо елемент з наступної серії

length++;

current = next;

if (!f.eof()) //Якщо файл не закінчився зчитуємо наступний елемент

{

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

}

else

{

break;

}

}

end\_pos = manager.output[current\_file].file\_object.tellg();

manager.output[current\_file].file\_object.seekg(start\_pos);

manager.write(current\_file, length);

manager.output[current\_file].file\_object.seekg(end\_pos);

for (int i = 0; i < counter; i++)

{

manager.output[i].empty\_series = diff[i]; //Пишемо к-ть порожніх серій в масив

}

f.close();

}

int polyphase::show\_binary\_file\_length(string filename)

{

if (eof(filename))

{

cout << "[пустой]" << endl;

}

else

{

fstream f;

f.open(filename, ios::in | ios::binary);

int temp;

int length;

while (!f.eof())

{

f.read((char\*)&length, sizeof(int));

if (length)

{

cout << "[len: " << length << "] ";

while (length)

{

int temp;

f.read((char\*)&temp, sizeof(int));

cout << temp << " ";

length--;

}

}

}

cout << endl;

f.close();

}

return 0;

}

void polyphase::generate(string filename, int amount)

{

fstream f;

srand(time(NULL));

f.open(filename, ios::out | ios::binary);

for (int i = 0; i < amount - 1; i++)

{

int value = rand() % (i+1);

f.write((char\*)&value, sizeof(int));

}

f.close();

}

int polyphase::check\_sort(string filename)

{

bool flag = 1;

int counter = 1;

int temp1, temp2;

fstream f;

f.open(filename, ios::in);

f >> temp1;

while (!f.eof())

{

f >> temp2;

counter++;

if (temp1 > temp2)

{

cout << "Помилка сортування: " << counter << " позиція";

flag = 0;

}

temp1 = temp2;

}

if (flag = true)

{

cout << "Сортування правильне: " << counter << " Елементів було відсортовано";

}

cout << endl;

return flag;

}

int polyphase::show\_binary\_file\_eof(string filename)

{

fstream f;

f.open(filename, ios::in | ios::binary);

int temp;

int quantity = 0;

while (!f.eof())

{

int temp;

f.read((char\*)&temp, sizeof(int));

cout << temp << " ";

quantity++;

}

cout << endl;

f.close();

return quantity;

}

void polyphase::show\_txt\_file(string filename)

{

fstream f;

f.open(filename);

int temp;

while (!f.eof())

{

f >> temp;

cout << temp << " ";

}

cout << endl;

f.close();

}

int polyphase::amount\_of\_series(string filename)

{

fstream f;

f.open(filename, ios::in | ios::binary);

int quantity = 1;

int current = 0;

int next = 0;

position\_series.push\_back(f.tellg());

cout << "Текущая позиция: " << 1 + f.tellg() << " :: ";

f.read((char\*)&current, sizeof(int));

cout << "Серия №" << quantity << " начинается со значения: " << current << endl;

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

while (!f.eof())

{

/\*если всё верно\*/

if (next > current)

{

f.read((char\*)&current, sizeof(int));

}

else

{

quantity++;

cout << "Текущая позиция: " << f.tellg() / sizeof(int) << " :: ";

position\_series.push\_back(f.tellg());

cout << "Серия №" << quantity << " начинается со значения: " << next << endl;

f.read((char\*)&current, sizeof(int));

}

if (current > next)

{

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

}

else

{

quantity++;

cout << "Текущая позиция: " << f.tellg() / sizeof(int) << " :: ";

position\_series.push\_back(f.tellg());

cout << "Серия №" << quantity << " начинается со значения: " << current << endl;

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

}

}

f.close();

return quantity;

}

Файл FileManager.h

#pragma once

#pragma once

#include <vector>

#include <string>

#include <fstream>

#include <iostream>

using namespace std;

/\*Filemanager для бінарного файлу\*/

class filemanager

{

private:

struct file\_definition //Файл через структуру

{

fstream file\_object; //Файл

string filename; //Імя

int real\_series; //К-ть справжніх серій

int empty\_series; //К-ть порожніх серій

file\_definition(string \_filename); //Початкова ініціалізація

void swap(file\_definition& rhs); //Свій свап

};

public:

vector <file\_definition> input, output; //Заводимо вектор пар обєкт-імя

int in\_count, out\_count; //лічильник для векторів

filemanager(int in\_count, int out\_count); //Конструктор

void fileswap();

void index\_swap(int index\_in, int index\_out);

int read(int index); //Прочитати элемент з вектора

void write(int index, int value); //Записати элемент до вектора

bool eof(int index); //Перевірка на кінець файлу

~filemanager(); //Деструктор

int get\_in(); //Повернути in лічильник

int get\_out(); //Повернути out лічильник

};

Файл FileManager.срр

#include "Filemanager.h"

filemanager::filemanager(int \_in\_count, int \_out\_count)

{

in\_count = \_in\_count;

out\_count = \_out\_count;

input.reserve(in\_count); /\* Резервуємо місце у векторах \*/

output.reserve(out\_count);

for (long long i = 0; i < in\_count; i++)

{

string file\_name = "file\_input\_" + to\_string(i) + ".txt";

input.push\_back(file\_definition(file\_name));

}

for (long long i = 0; i < out\_count; i++)

{

string file\_name = "file\_output\_" + to\_string(i) + ".txt";

output.push\_back(file\_definition(file\_name));

}

}

void filemanager::fileswap()

{

for (int i = 0; i < in\_count; i++)

{

input[i].file\_object.close();

input[i].file\_object.open(input[i].filename, ios::out | ios::binary);

}

for (int i = 0; i < out\_count; i++)

{

output[i].file\_object.close();

output[i].file\_object.open(output[i].filename, ios::in | ios::binary);

}

input.swap(output);

//oh, hi - свапаем счетчики

swap(in\_count, out\_count);

}

void filemanager::index\_swap(int index\_in, int index\_out)

{

input[index\_in].file\_object.close();

input[index\_in].file\_object.open(input[index\_in].filename, ios::out | ios::binary);

output[index\_out].file\_object.close();

output[index\_out].file\_object.open(output[index\_out].filename, ios::in | ios::binary);

input[index\_in].swap(output[index\_out]);

}

int filemanager::read(int index)

{

int temp;

input[index].file\_object.read((char \*)&temp, sizeof(int));

return temp;

}

void filemanager::write(int index, int value)

{

output[index].file\_object.write((char\*)&value, sizeof(int)); // Пишемо масив у файл

}

bool filemanager::eof(int index)

{

return input[index].file\_object.eof();

}

int filemanager::get\_in()

{

return in\_count;

}

int filemanager::get\_out()

{

return out\_count;

}

filemanager::~filemanager()

{

for (int i = 0; i < in\_count; i++)

{

input[i].file\_object.close(); //Закриваємо файли

}

for (int i = 0; i < out\_count; i++)

{

output[i].file\_object.close();

}

}

/\* Ініціалізуємо структуру\*/

filemanager::file\_definition::file\_definition(string \_filename)

{

filename = \_filename;

real\_series = 0;

empty\_series = 0;

file\_object = fstream(filename, ios::out | ios::binary);

}

void filemanager::file\_definition::swap(file\_definition &rhs)

{

file\_object.swap(rhs.file\_object);

filename.swap(rhs.filename);

std::swap(real\_series, rhs.real\_series);

std::swap(empty\_series, rhs.empty\_series);

}

Файл Fibonacci.h

#pragma once

#pragma once

#include <vector>

#include <iostream>

using namespace std;

/\* Клас Фібоначчі\*/

class fibonacci

{

private:

vector <int> fib\_series; //Вектор для генерації ряду фібоначчі будь-якого порядк

int current\_pos; //Поточна позиція для підрахунку

int index\_end; //індекс-позиція вектора

public:

int\* mass;

int order; //Порядок для ряду Фібоначчі

fibonacci(int \_order); //Конструктор

void show\_fib\_vector(); //Вивести фібоначчі

void show\_dist\_mass(); //Вивести основний масив

void make\_order\_vector(); //Зробити послідовність фібоначчі заданого порядку

void make\_dist\_mass(); //Зробити розподільчий масив(на основі order-vector)

};

Файл Fibonacci.cpp

#include "Fibonacci.h"

fibonacci::fibonacci(int \_order)

{

current\_pos = 0;

order = \_order;

index\_end = order - 1; //Знаходимо останню позицію у векторі

for (int i = 0; i < order; i++)

{

fib\_series.push\_back(0); //всі елементи 0 окрім одного

}

fib\_series[index\_end] = 1; //останній =1

}

void fibonacci::show\_fib\_vector()

{

for (int i = 0; i < fib\_series.size(); i++)

{

cout << fib\_series[i] << " ";

}

}

void fibonacci::show\_dist\_mass()

{

for (int i = 0; i < order; i++)

{

cout << mass[i] << " ";

}

}

/\* Робимо ряд Фібоначчі у векторі

- Добавляємо у вектор числа\*/

void fibonacci::make\_order\_vector()

{

int temp1 = 0;

int temp2 = 0;

for (int i = 0; i < fib\_series.size(); i++)

{

temp1 = temp1 + fib\_series[i];

if (i > order)

{

temp2 = temp2 + fib\_series[i - order];

}

}

fib\_series.push\_back(temp1 - temp2);

}

/\* Робимо масив який буде потрібен для сортування

(сумуємо Фібоначчі в потрібному

порядку)\*/

void fibonacci::make\_dist\_mass()

{

mass = new int[order];

int finish = current\_pos + order;

for (int i = 0; i < order; i++)

{

int temp = 0;

for (int j = current\_pos; j < finish; j++)

{

temp = temp + fib\_series[j];

}

current\_pos++;

mass[i] = temp;

}

current\_pos = finish - order + 1;

}

Висновок

При виконанні даної лабораторної було ознайомлено з різними алгоритмами зовнішнього сортування, був створений псевдокод, а також програмна реалізація на мові С++. Особливістю моєї роботи була розробка багатофазного сортування. В ході розробки було досліджено відмінності від інших алгоритмів зовнішнього сортування, а також виявлено, що програма сортує файл розміром 12МВ(3 000 000 чисел) приблизно за 4 хв, що є непоганим результатом

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%.