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: *ІП-13, Музичук Віталій Андрійович* _____
(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**
17	Пряме злиття

3 ВИКОНАННЯ

3.1 Псевдокод алгоритму

Псевдокод алгоритму сортування:

1. ПОЧАТОК
2. sequence_size = 1
3. quantity_of_number = size / 4
4. ПОКИ sequence_size < quantity_of_number
   4.1. **split**(file_name, sequence_size)
   4.2. **merge**(file_name, sequence_size, quantity_of_numbers)
5. Вивести дані файлу на екран
6. КІНЕЦЬ

Псевдокод процедури split(file_name, sequence_size**):**

1. ПОЧАТОК
2. Відкрити файл з ім’ям file_name для читання
3. Створити два допоміжних файла з назвами “B.bin” і “C.bin” та відкрити їх для запису
4. value = 0
5. flag = true
6. Читати ПОКИ не дійдемо до кінця файлу file_name
   6.1. ЯКЩО flag = true, ТО
      6.1.1. Записати зчитане число з файлу file_name у перший файл “B.bin”
      6.1.2. value = value + 1
      6.1.3. ЯКЩО value >= sequence_size, ТО
         6.1.3.1. flag = false
         6.1.3.2. value = 0
   6.2. ІНАКШЕ
      6.2.1. Записати зчитане число з файлу file_name у дригий файл “C.bin”

6.2.2. value = value + 1

6.2.3. ЯКЩО value >= sequence_size, ТО

6.2.3.1. flag = true

6.2.3.2. value = 0

7. Закрити усі файли
8. КІНЕЦЬ

Псевдокод процедури **merge**(file_name, sequence_size, quantity_of_numbers):

1. ПОЧАТОК
2. Створити файл з ім’ям file_name та відкрити його для запису
3. Відкрити два допоміжних файла з назвами “B.bin” і “C.bin”
4. ПОКИ правда

4.1. Зчитати з другого файлу «C.bin» число у змінну first_number

4.2. ЯКЩО first_number є кінцем файлу, ТО

4.2.1. Завершити цикл

4.3. Зчитати з першого файлу «B.bin» число у змінну second_number

4.4. valueA = 0; valueB = 0

4.5. ПОКИ valueA не дорівнює sequence_size ТА valueB не дорівнює sequence_size

4.5.1. ЯКЩО first_number < second_number, ТО

4.5.1.1. Записати у вихідний файл file_name число first_number

4.5.1.2. valueA = valueA + 1

4.5.1.3. ЯКЩО valueA не дорівнює sequence_size, ТО

4.5.1.3.1. Зчитати наступне число з першого файлу “B.bin” у змінну first_number

4.5.2. ІНАКШЕ

4.5.2.1. Записати у вихідний файл file_name число second_number

4.5.2.2. valueB = valueB + 1

4.5.2.3. ЯКЩО valueB не дорівнює sequence_size, ТО

4.5.2.3.1. Зчитати наступне число з другого файлу “C.bin” у змінну second_number

4.6. ЯКЩО valueA > valueB, ТО

4.6.1. Записати у вихідний файл число second_number

4.6.2. valueB = valueB + 1

4.6.3. ПОКИ valueB < sequence_size ТА не дійшли до кінця другого файлу “C.bin”, ТО

4.6.3.1. Зчитати наступне число з другого файлу “C.bin” у змінну second_number

4.6.3.2. Записати у вихідний файл число second_number

4.6.3.3. valueB = valueB + 1

4.7. ІНАКШЕ ЯКЩО valueA < valueB, ТО

4.7.1. Записати у вихідний файл число first_number

4.7.2. valueA = valueA + 1

4.7.3. ПОКИ valueA < sequence_size ТА не дійшли до кінця першого файлу “B.bin”, ТО

4.7.3.1. Зчитати наступне число з першого файлу “B.bin” у змінну first_number

4.7.3.2. Записати у вихідний файл число first_number

4.7.3.3. valueA = valueA + 1

5. ПОКИ не дійдемо до кінця першого файлу «B.bin», ТО

5.1. Записувати у вхиідний файл file_name число first_number

6. Закрити всі файли

7. КІНЕЦЬ

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

Базовий алгоритм

```
#include "classes.h"

extern int buff[BUFF_SIZE];

Sort::Sort(string name) : file_name(name), sequence_size(1) {
	get_size();
}

long long Sort::get_size() {
	ifstream file(FOLDER_NAME + file_name, ios::binary);
	file.seekg(0, ios::end);
	file_size = file.tellg();
	file.close();
	file_size /= 4;
	return file_size;
}

void Sort::test() {
	bool flag = true;
	ifstream file(FOLDER_NAME + COPY_FILE, ios::binary);
	int previous_num, next_num;
	if (file_size <= BUFF_SIZE) {
		file.read((char*)&buff, sizeof(int) * file_size);
		for (int i = 0; i < file_size - 1; i++) {
			previous_num = buff[i];
			next_num = buff[i+1];
			if (previous_num > next_num)
				flag = false;
		}
	}
	else {
		int count = file_size / BUFF_SIZE;
		int remainder = file_size % BUFF_SIZE;
		for (int i = 0; i < count; i++) {
			file.read((char*)&buff, sizeof(int) * BUFF_SIZE);
			for (int j = 0; j < BUFF_SIZE - 1; j++) {
				previous_num = buff[j];
				next_num = buff[j + 1];
				if (previous_num > next_num)
					flag = false;
			}
		}
		file.read((char*)&buff, 4 * remainder);
		for (int i = 0; i < remainder - 1; i++) {
			previous_num = buff[i];
			next_num = buff[i + 1];
			if (previous_num > next_num)
				flag = false;
		}
	}

	if (flag)
		cout << "Тест пройдено успішно, файл відсортований" << endl;
	else
		cout << "Тест не пройдено :(" << endl;
}
```

```
BasicSort::BasicSort(string name) : Sort(name) {}

void BasicSort::copy_file() {
	ifstream file(FOLDER_NAME + file_name, ios::binary);
	ofstream copy(FOLDER_NAME + COPY_FILE, ios::binary);

	file.read((char*)&buff, file_size * 4);
	copy.write((char*)&buff, file_size * 4);

	file.close();
	copy.close();
}

void BasicSort::split() {
	ifstream file_out(FOLDER_NAME + COPY_FILE, ios::binary);
	ofstream A(FOLDER_NAME + FIRST_FILE, ios::binary);
	ofstream B(FOLDER_NAME + SECOND_FILE, ios::binary);

	int number;
	bool flag = true;
	long long value = 0;

	while (file_out.read((char*)&number, sizeof(number))) {
		if (flag) {
			A.write((char*)&number, sizeof(number));
			value++;
			if (value >= sequence_size) {
				flag = false;
				value = 0;
			}
		}
		else {
			B.write((char*)&number, sizeof(number));
			value++;
			if (value >= sequence_size) {
				flag = true;
				value = 0;
			}
		}
	}
	A.close();
	B.close();
	file_out.close();
}

void BasicSort::basic_sort() {
	copy_file();
	for (; sequence_size < file_size; sequence_size *= 2) {
		split();
		merge();
	}
	test();
}

void BasicSort::merge() {
	ofstream file_out(FOLDER_NAME + COPY_FILE, ios::binary);
	ifstream A(FOLDER_NAME + FIRST_FILE, ios::binary);
	ifstream B(FOLDER_NAME + SECOND_FILE, ios::binary);
	int first_number; int second_number;

	while (true) {
		B.read((char*)&second_number, sizeof(int));
		if (B.eof())
			break;
		A.read((char*)&first_number, sizeof(int));
```

```
		long long valueA = 0, valueB = 0;

		while (valueA != sequence_size && valueB != sequence_size) {
			if (first_number < second_number) {
				file_out.write((char*)&first_number, sizeof(int));
				valueA++;
				if (valueA != sequence_size)
					A.read((char*)&first_number, sizeof(int));
			}
			else {
				file_out.write((char*)&second_number, sizeof(int));
				valueB++;
				if (valueB != sequence_size)
					B.read((char*)&second_number, sizeof(int));
			}
		}
		if (valueA > valueB) {
			file_out.write((char*)&second_number, sizeof(int));
			valueB++;
			while (valueB < sequence_size) {
				B.read((char*)&second_number, sizeof(int));
				if (B.eof())
					break;
				file_out.write((char*)&second_number, sizeof(int));
				valueB++;
			}

		}
		else if (valueA < valueB) {
			file_out.write((char*)&first_number, sizeof(int));
			valueA++;
			while (valueA < sequence_size) {
				A.read((char*)&first_number, sizeof(int));
				if (A.eof())
					break;
				file_out.write((char*)&first_number, sizeof(int));
				valueA++;
			}
		}
	}

	while (A.read((char*)&first_number, sizeof(int))) {
		file_out.write((char*)&first_number, sizeof(int));
	}

	file_out.close();
	A.close();
	B.close();
}

ModifiedSort::ModifiedSort(string name) : Sort(name), num_of_chunks(0) {
	this->sequence_size = BUFF_SIZE;
}

void ModifiedSort::modified_sort() {
	pre_sort();

	Timer tim;

	for (; sequence_size < file_size; sequence_size *= 2) {
		mod_split();
		merge();
	}

	cout << "Зовнішнє сортування заняло " << tim.elapsed() << endl;
```

```
	test();
}

void ModifiedSort::pre_sort() {
	ifstream file(FOLDER_NAME + file_name, ios::binary);
	ofstream copy(FOLDER_NAME + COPY_FILE, ios::binary);

	num_of_chunks = file_size / BUFF_SIZE;

	Timer t;

	for (int i = 0; i < num_of_chunks; i++) {
		file.read((char*)&buff, BUFF_SIZE * 4);
		sort(buff, buff + BUFF_SIZE);
		copy.write((char*)&buff, BUFF_SIZE * 4);
	}

	cout << "Внутрішнє сортування заняло " << t.elapsed() << endl;

	file.close();
	copy.close();
}

void ModifiedSort::mod_split() {
	ifstream file_out(FOLDER_NAME + COPY_FILE, ios::binary);
	ofstream B(FOLDER_NAME + FIRST_FILE, ios::binary);
	ofstream C(FOLDER_NAME + SECOND_FILE, ios::binary);

	bool flag = true;
	long long value = 0;

	for (int i = 0; i < num_of_chunks; i++) {
		if (flag) {
			file_out.read((char*)&buff, sizeof(buff));
			B.write((char*)&buff, sizeof(buff));
			value += BUFF_SIZE;
			if (value >= sequence_size) {
				flag = false;
				value = 0;
			}
		}
		else {
			file_out.read((char*)&buff, sizeof(buff));
			C.write((char*)&buff, sizeof(buff));
			value += BUFF_SIZE;
			if (value >= sequence_size) {
				flag = true;
				value = 0;
			}
		}
	}
	B.close();
	C.close();
	file_out.close();
}

void ModifiedSort::merge() {
	ofstream clear(FOLDER_NAME + COPY_FILE, ios::binary);
	clear.close();
	ifstream A(FOLDER_NAME + FIRST_FILE, ios::binary);
	ifstream B(FOLDER_NAME + SECOND_FILE, ios::binary);
	int first_number; int second_number;

	while (true) {
		B.read((char*)&second_number, sizeof(int));
```

```
		if (B.eof())
			break;
		A.read((char*)&first_number, sizeof(int));
		long long valueA = 0, valueB = 0;
		int index = 0;

		while (valueA != sequence_size && valueB != sequence_size) {
			if (first_number < second_number) {
				flush(index);
				buff[index] = first_number;
				valueA++; index++;
				if (valueA != sequence_size)
					A.read((char*)&first_number, sizeof(int));
			}
			else {
				flush(index);
				buff[index] = second_number;
				valueB++; index++;
				if (valueB != sequence_size)
					B.read((char*)&second_number, sizeof(int));
			}
		}
		if (valueA > valueB) {
			flush(index);
			buff[index] = second_number;
			valueB++; index++;
			flush(index);
			while (valueB < sequence_size) {
				B.read((char*)&second_number, sizeof(int));
				if (B.eof())
					break;
				flush(index);
				buff[index] = second_number;
				valueB++; index++;
			}

		}
		else if (valueA < valueB) {
			flush(index);
			buff[index] = first_number;
			valueA++; index++;
			flush(index);
			while (valueA < sequence_size) {
				A.read((char*)&first_number, sizeof(int));
				if (A.eof())
					break;
				flush(index);
				buff[index] = first_number;
				valueA++; index++;
			}
		}
	}

	A.close();
	B.close();
}

void ModifiedSort::flush(int& index) {
	if (index + 1 > sequence_size) {
		ofstream file_out(FOLDER_NAME + COPY_FILE, ios::binary | ios::app);
		file_out.seekp(0, ios::end);
		file_out.write((char*)&buff, sizeof(int) * BUFF_SIZE);
		file_out.close();
		index = 0;
	}
```

```
}
```

GitHub

ВИСНОВОК

При виконанні даної лабораторної роботи я детально ознайомився з проблемою відсортовування файлів великих розмірів, дізнався про можливі розв’язки цієї задачі та самостійно реалізував псевдокод та програмну реалізацію методу прямого злиття. Був проведений аналіз роботи алгоритму, який показав, що базовий алгоритм без модифікацій сортує файл розміром 10Мб в середньому за 2 хвилини. Це є доволі непродуктивно, тому базовий алгоритм не варто використовувати для реальних задач. Відповідно до цього я модифікував базовий алгоритм для того щоб зменшити час сортування. Для модифікації було використано технологію попереднього сортування, коли ми частково відсортовуємо файл за допомогою більш швидкого алгоритму внутрішнього сортування. Також в програмі використовується буфер, який накопичує в собі дані, і потім записує їх у файл. Дана методологія дозволяє зменшити кількість звернень до жорсткої пам’яті комп’ютера, що пришвидшує запис/зчитування.