**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-22 Присяжний А. О.*

**Перевірив**

(прізвище, ім'я, по батькові)

*Ахаладзе І.Е.*

Київ 2023

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. 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.

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

**int** index\_of\_file\_to\_write, ideal\_amount\_of\_series[amount\_of\_supporting\_files], amount\_of\_empty\_series[amount\_of\_supporting\_files], level;

**select\_supporting\_file\_to\_write\_series()** {

**int** max\_amount\_of\_series;

**if** (amount\_of\_empty\_series[index\_of\_file\_to\_write] < amount\_of\_empty\_series[index\_of\_file\_to\_write + 1]) **do**

index\_of\_file\_to\_write = index\_of\_file\_to\_write + 1;

**else** **do**

**if** (amount\_of\_empty\_series[index\_of\_file\_to\_write] == 0) **do**

level = level + 1;

max\_amount\_of\_series = ideal\_amount\_of\_series[0];

**for** i = 0 **to** amount\_of\_supporting\_files - 1 **do**

amount\_of\_empty\_series[i] = max\_ideal\_amount\_of\_series + ideal\_amount\_of\_series[i + 1] - ideal\_amount\_of\_series[i];

ideal\_amount\_of\_series[i] = max\_ideal\_amount\_of\_series + ideal\_amount\_of\_series[i + 1];

**end for**

**end if**

index\_of\_file\_to\_write = 0;

**end if**

amount\_of\_empty\_series[index\_of\_file\_to\_write] = amount\_of\_empty\_series[index\_of\_file\_to\_write] – 1;

}

**main()** {

**int** max\_amount\_of\_merged\_series, amount\_of\_active\_files, i, min, index\_of\_file\_with\_min\_element;

**fstream** file\_to\_sort;

**fstream** supporting\_files[amount\_of\_supporting\_files];

**int** supporting\_files\_indexes[amount\_of\_supporting\_files], active\_supporting\_files\_indexes[amount\_of\_supporting\_files-1];

**відкрити** file\_to\_sort на читання;

**for** i = 0 **to** amount\_of\_supporting\_files - 1 **do**

**відкрити** supporting\_files[i] на запис;

**end for**

**for** i = 0 **to** amount\_of\_supporting\_files - 1 **do**

ideal\_amount\_of\_series[i] = 1;

amount\_of\_empty\_series[i] = 0;

**записати** одну серію з file\_to\_sort в supporting\_files[i];

**end for**

level = 1;

index\_of\_file\_to\_write = 0;

**while** (не кінець file\_to\_sort) **do**

select\_supporting\_file\_to\_write\_series();

**if** (останній елемент supporting\_files[index\_of\_file\_to\_write] більший за перший елемент file\_to\_sort) **do**

**записати** одну серію з file\_to\_sort в supporting\_files[index\_of\_file\_to\_write];

**else**

**записати** одну серію з file\_to\_sort в supporting\_files[index\_of\_file\_to\_write];

**if** (file\_to\_sort не закінчився) **do**

**записати** одну серію з file\_to\_sort в supporting\_files[index\_of\_file\_to\_write];

**else**

amount\_of\_empty\_series[index\_of\_file\_to\_write] = amount\_of\_empty\_series[index\_of\_file\_to\_write] + 1;

**end if**

**end if**

**end while**

**for** i = 0 **to** amount\_of\_supporting\_files-1 **do**

supporting\_files\_indexes[i] = i;

**відкрити** supporting\_files[i] на читання;

**end for**

**відкрити** supporting\_files[amount\_of\_supporting\_files-1] на запис;

**do**

max\_amount\_of\_merged\_series = ideal\_amount\_of\_series[amount\_of\_supporting\_files - 2];

amount\_of\_empty\_series[amount\_of\_supporting\_files - 1] = 0;

**do**

amount\_of\_active\_files = 0

**for** i = 0 **to** amount\_of\_supporting\_files-1 **do**

**if** (amount\_of\_empty\_series[i] > 0) **do**

amount\_of\_empty\_series[i] = amount\_of\_empty\_series[i] – 1;

**else**

active\_supporting\_files\_indexes [amount\_of\_active\_files] = supporting\_files\_indexes[i];

amount\_of\_active\_files = amount\_of\_active\_files+1;

**end if**

**end for**

**if** (amount\_of\_active\_files == 0) **do**

amount\_of\_empty\_series[amount\_of\_supporting\_files-1] = amount\_of\_empty\_series[amount\_of\_supporting\_files-1]–1;

**else**

**do**

i = 0;

index\_of\_file\_with\_min\_element = 0;

min = перший елемент supporting\_files[active\_supporting\_files\_indexes[0]];

**while** (i < amount\_of\_active\_files) **do**

i = i + 1;

**int** current\_element = перший елемент supporting\_files[active\_supporting\_files\_indexes[i]];

**if** (current\_element < min) **do**

min = current\_element;

index\_of\_file\_with\_min\_element = i;

**end if**

**end while**

скопіювати елемент з supporting\_files[active\_supporting\_files\_indexes[index\_of\_file\_with\_min\_element]] в supporting\_files[supporting\_files\_indexes[amount\_of\_supporting\_files-1]];

**if** (кінець серії supporting\_files[active\_supporting\_files\_indexes[index\_of\_file\_with\_min\_element]]) **do**

amount\_of\_active\_files = amount\_of\_active\_files – 1;active\_supporting\_files\_indexes[index\_of\_file\_with\_min\_element]] = supporting\_files\_indexes[amount\_of\_active\_files];

**end if**

**while** (amount\_of\_active\_files > 0)

max\_amount\_of\_merged\_series = max\_amount\_of\_merged\_series – 1;

**end if**

**while** (max\_amount\_of\_merged\_series > 0)

**закрити** supporting\_files[supporting\_files\_indexes[amount\_of\_supporting\_files-1]];

**відкрити** supporting\_files[supporting\_files\_indexes[amount\_of\_supporting\_files-1]] на читання;

**int** last\_supporting\_file\_index = supporting\_files\_indexes[amount\_of\_supporting\_files - 1];

**int** amount\_of\_empty\_series\_in\_last\_file = amount\_of\_empty\_series[amount\_of\_supporting\_files - 1];

max\_amount\_of\_merged\_series = ideal\_amount\_of\_series[amount\_of\_supporting\_files - 2];

**for** i = amount\_of\_supporting\_files - 1 **to** 1 **do**

supporting\_files\_indexes[i] = supporting\_files\_indexes[i - 1];

amount\_of\_empty\_series[i] = amount\_of\_empty\_series[i - 1];

ideal\_amount\_of\_series[i] = ideal\_amount\_of\_series[i - 1] - max\_amount\_of\_merged\_series;

**end for**

supporting\_files\_indexes[0] = last\_supporting\_file\_index;

amount\_of\_empty\_series[0] = amount\_of\_empty\_series\_in\_last\_file;

ideal\_amount\_of\_series[0] = max\_amount\_of\_merged\_series;

level = level – 1;

**while** (level > 0)

//відсортований файл зберігається у supporting\_files[supporting\_files\_indexes[0]]

}

## Програмна реалізація алгоритму

### Вихідний код

#include "stdafx.h"

#include <iostream>

#include <ctime>

#include <iomanip>

usingnamespacestd;

voidinsertionSort(int\*, int); // прототип функции сортировки вставками

intmain(intargc, char\* argv[])

{

…

    system("pause");

    return0;

}

voidinsertionSort(int\*arrayPtr, intlength) // сортировка вставками

{

    …

}

Висновок

При виконанні даної лабораторної роботи я реалізував алгоритм багатофазного сортування злиттям на мові програмування С++, а також зобразив його за допомогою псевдокоду. Звичайний алгоритм сортує файл розміром 20 Мб за 10 секунд, про що свідчить наступний скріншот виконання програми:

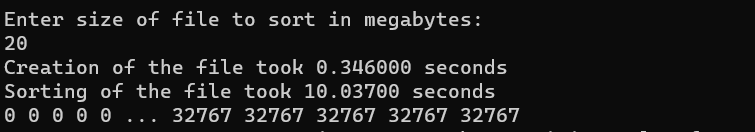


Рисунок 1 – Сортування файлу розміром 20 мегабайт звичайним алгоритмом багатофазного сортування

Щоб покращити швидкодію алгоритму я використав наступний підхід: для читання даних з початкового файлу я використовую технологію відображення файлу в оперативну пам’ять (File Mapping), розбиваю цей файл на частини розміром 10 Мб та сортую їх окремо за допомогою внутрішнього швидкого сортування (quick sort). File Mapping дозволяє багатократно пришвидшити операції читання і запису у файл, тому зменшується і загальний час роботи алгоритму сортування. Також, так як файл уже відсортований частинами, то у ньому значно менша кількість серій, ніж у невідсортованому, тому потрібно здійснювати менше операцій злиття під час сортування, що також пришвидшує роботу програми. У результаті модифікована версія алгоритму сортує файл на 20 Мб за 1 секунду, що у 10 разів швидше. Про це свідчить наступний скріншот:

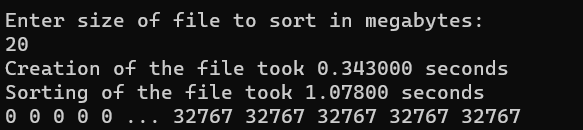


Рисунок 2 - Сортування файлу розміром 20 мегабайт модифікованим алгоритмом багатофазного сортування

Файл розміром 1 Гб модифікована версія алгоритму сортує за 3 хвилини 19 секунд (Рисунок 3).

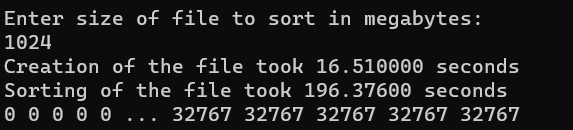


Рисунок 3 - Сортування файлу розміром 1 гігабайт модифікованим алгоритмом багатофазного сортування

Критерії оцінювання

У випадку здачі лабораторної роботи до 08.10.2022 включно максимальний бал дорівнює – 5. Після 08.10.2022 максимальний бал дорівнює – 4,5.

Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 15%;
* програмна реалізація алгоритму – 20%;
* програмна реалізація модифікацій – 20%;
* робота з git – 40%;
* висновок – 5%.