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**”

Виконала *ІП-12 Кушнір Ганна Вікторівна* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов Олексій Олександрович* _____
(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**
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

3 ВИКОНАННЯ

3.1 Псевдокод алгоритму

```
MultiwayMerge(start_file, m)

B ← [ file(“B1”), file(“B2”), …, file(“Bm”) ]
C ← [ file(“C1”), file(“C2”), …, file(“Cm”) ]
num ← SplitInputFile(B, start_file, m)
i ← 0
while true
      do    if i mod 2 = 0
                  then  MergeAndSplitFiles(B, C, m)
                        if C[0].size() = num * Int.size()
                              then  flag ← 2
                                    break
                        end if
                  else  MergeAndSplitFiles(C, B, m)
                        if B[0].size() = num * Int.size()
                              then  flag ← 1
                                    break
                        end if
            end if
            i ← i + 1
end while
if flag = 1
      then  end_file ← B[0]
      else  end_file ← C[0]
end if
for i from 0 to m
      do    B[i].deleteFile()
            C[i].deleteFile()
end for
return end_file


SplitInputFile(B, start_file, NUM_OF_FILES)

i ← 0
num ← 0
k ← start_file.readInt()
num ← num + 1
B[0].writeInt(k)
while not start_file.eof()
      do    m ← start_file.readInt()
            num ← num + 1
            if k > m
                  then  i ← i + 1
                        if i = NUM_OF_FILES
                              then  i ← 0
                        end if
            end if
            B[i].writeInt(m)
```

```
            k ← m
end while
return num
```

MergeAndSplitFiles(B, C, NUM_OF_FILES)

```
nums ← []
for i from 0 to NUM_OF_FILES
      do    if not B[i].eof()
                  then  k ← B[i].readInt()
                  else  k ← ∞
            end if
            nums[i] ← k
end for
n ← FindNumOfMin(nums)
k ← nums[n]
i ← 0
C[0].writeInt(k)
while true
      do    if not B[i].eof()
                  then  nums[n] ← B[i].readInt()
                  else  nums[n] ← ∞
            end if
            n ← FindNumOfMin(nums)
            m ← nums[n]
            if m = ∞
                  then  break
            end if
            if k > m
                  then  i ← i + 1
                        if i = NUM_OF_FILES
                              then  i ← 0
                        end if
            end if
            C[i].writeInt(m)
            k ← m
end while
return
```

FindNumOfMin(nums)

```
j ← 0
for i from 1 to nums.size()
      do    if nums[i] < nums[j]
                  then  j ← i
            end if
end for
return j
```

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
#pragma warning(disable : 4996)
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include "stdio.h"
#include <ctime>
using namespace std;

constexpr auto NUM_OF_FILES = 16;

// Сортування збалансованим багатошляховим злиттям.
int multiwayMerge(string, string);
// Створення масиву (вектору) назв файлів.
vector<string> createFileNames(string);
// Пошук номеру найменшого елемента послідовності.
int findNumOfMin(vector<int>&);
// Почергове злиття файлів B1, B2, ..., Bm у файли C1, C2, ..., Cm і навпаки,
// поки у B1 або C1 не утвориться одна серія.
int alternateMergingOfFiles(vector<string>&, vector<string>&, int);
// Розподіл серій вхідного файлу по m допоміжних файлах (B1, B2, ..., Bm).
int splitInputFile(vector<string>&, FILE*);
// Злиття файлів B1, B2, ..., Bm у файли C1, C2, ..., Cm.
void mergeFiles(vector<string>&, vector<string>&);
// Перетворення бінарного файлу у текстовий.
void convertBinToText(string, string);
// Видалення допоміжних файлів.
void deleteFiles(vector<string>&);

int main()
{
    string path1 = "start_file.txt";
    string path2 = "end_file.txt";
    clock_t start = clock();
    if (multiwayMerge(path1, path2)) {
        return 1;
    }
    double duration = (clock() - start) / CLOCKS_PER_SEC;
    cout << "Duration: " << duration << " seconds.\n";
    return 0;
}

int multiwayMerge(string path1, string path2)
{
    FILE* file;
    file = fopen(path1.c_str(), "rt");
    if (file == NULL) {
        return 1;
    }
    vector<string> pathsB = createFileNames("B");
    vector<string> pathsC = createFileNames("C");
```

```
    int num = splitInputFile(pathsB, file);
    fclose(file);

    int flag = alternateMergingOfFiles(pathsB, pathsC, num);

    if (flag == 1) {
        convertBinToText(pathsB[0], path2);
    }
    else {
        convertBinToText(pathsC[0], path2);
    }
    deleteFiles(pathsB);
    deleteFiles(pathsC);
    return 0;
}

vector<string> createFileNames(string name)
{
    vector<string> paths(NUM_OF_FILES);
    for (int i = 0; i < NUM_OF_FILES; ++i) {
        paths[i] = name + to_string(i) + ".dat";
    }
    return paths;
}

int findNumOfMin(vector<int>& nums)
{
    int j = 0;
    for (int i = 1; i < nums.size(); ++i) {
        if (nums[i] < nums[j]) {
            j = i;
        }
    }
    return j;
}

int alternateMergingOfFiles(vector<string>& pathsB, vector<string>& pathsC, int
num)
{
    int i = 0;
    while (true) {
        if (i % 2 == 0) {
            mergeFiles(pathsB, pathsC);
            ifstream C(pathsC[0], ios::binary);
            C.seekg(0, ios::end);
            if (num * sizeof(int) == C.tellg()) {
                return 2;
            }
            C.close();
        }
        else {
            mergeFiles(pathsC, pathsB);
            ifstream B(pathsB[0], ios::binary);
            B.seekg(0, ios::end);
            if (num * sizeof(int) == B.tellg()) {
                return 1;
```

```
            }
            B.close();
        }
        i++;
    }
}

int splitInputFile(vector<string>& paths, FILE* file)
{
    int k, m, i = 0, num = 0;
    fscanf(file, "%i", &k);
    num++;
    vector<FILE*> B(NUM_OF_FILES);
    for (int i = 0; i < NUM_OF_FILES; ++i) {
        B[i] = fopen(paths[i].c_str(), "wb");
    }
    fwrite(&k, sizeof(int), 1, B[0]);

    while (!feof(file)) {
        fscanf(file, "%i", &m);
        num++;
        if (k > m) {
            i++;
            if (i == NUM_OF_FILES) {
                i = 0;
            }
        }
        fwrite(&m, sizeof(int), 1, B[i]);
        k = m;
    }
    for (int i = 0; i < NUM_OF_FILES; ++i) {
        fclose(B[i]);
    }
    return num;
}

void mergeFiles(vector<string>& pathsB, vector<string>& pathsC)
{
    vector<int> nums;
    vector<FILE*> B(NUM_OF_FILES), C(NUM_OF_FILES);
    for (int i = 0; i < NUM_OF_FILES; ++i) {
        B[i] = fopen(pathsB[i].c_str(), "rb");
        C[i] = fopen(pathsC[i].c_str(), "wb");
    }
    int k, m, n;
    for (int i = 0; i < NUM_OF_FILES; ++i) {
        k = INT_MAX;
        fread(&k, sizeof(int), 1, B[i]);
        nums.push_back(k);
    }
    n = findNumOfMin(nums);
    k = nums[n];
    int i = 0;
    fwrite(&k, sizeof(int), 1, C[0]);
    while (true) {
        nums[n] = INT_MAX;
```

```
        fread(&nums[n], sizeof(int), 1, B[n]);
        n = findNumOfMin(nums);
        m = nums[n];
        if (m == INT_MAX) {
            break;
        }
        if (k > m) {
            i++;
            if (i == NUM_OF_FILES) {
                i = 0;
            }
        }
        fwrite(&m, sizeof(int), 1, C[i]);
        k = m;
    }
    for (int i = 0; i < NUM_OF_FILES; ++i) {
        fclose(B[i]);
        fclose(C[i]);
    }
}

void convertBinToText(string pathBin, string pathTxt)
{
    FILE* bin, *txt;
    bin = fopen(pathBin.c_str(), "rb");
    txt = fopen(pathTxt.c_str(), "wt");
    int i = 0, k;
    while (fread(&k, sizeof(int), 1, bin)) {
        fprintf(txt, "%i ", k);
        i++;
        k = false;
    }
    fclose(txt);
    fclose(bin);
}

void deleteFiles(vector<string>& paths)
{
    for (int i = 0; i < paths.size(); ++i) {
        remove(paths[i].c_str());
    }
}
```

ВИСНОВОК

При виконанні даної лабораторної роботи було вивчено основні алгоритми зовнішнього сортування та способи їх модифікації. Було написано псевдокод алгоритму зовнішнього сортування збалансованим багатошляховим злиттям та виконано його програмну реалізацію мовою програмування C++. Результуючу програму було протестовано на вхідних файлах розмірами 1 Мб та 10 Мб. Час, використаний на сортування даних файлів звичайним алгоритмом без модифікацій, становив відповідно 8 хвилин та 2 години.

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%.