

**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

<b>1</b>	<b>МЕТА ЛАБОРАТОРНОЇ РОБОТИ.....</b>	<b>3</b>
<b>2</b>	<b>ЗАВДАННЯ .....</b>	<b>4</b>
<b>3</b>	<b>ВИКОНАННЯ .....</b>	<b>6</b>
3.1	ПСЕВДОКОД АЛГОРИТМУ .....	6
3.2	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ .....	8
3.2.1	<i>Вихідний код.....</i>	8
	<b>ВИСНОВОК .....</b>	<b>13</b>
	<b>КРИТЕРІЇ ОЦІНЮВАННЯ .....</b>	<b>14</b>

## 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

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <ctime>
using namespace std;

constexpr auto NUM_OF_FILES = 4;

// Сортування збалансованим багатопотоковим злиттям.
int multiwayMerge(string, string);
// Створення масиву (вектору) назв файлів.
vector<string> createFileNames(string, int);
// 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>&, ifstream&);
// 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)
{
    ifstream file(path1);
    if (!file.is_open()) {
        return 1;
    }
    vector<string> pathsB = createFileNames("B", NUM_OF_FILES);
    vector<string> pathsC = createFileNames("C", NUM_OF_FILES);
    int num = splitInputFile(pathsB, file);
    file.close();
}
```



```

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, int num)
{
    vector<string> paths;
    for (int i = 0; i < num; ++i) {
        paths.push_back(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, ifstream& file)
{
    int k, m, i = 0, num = 0;
    file >> k;
    num++;
    ofstream
        B0(paths[0], ios::binary),
        B1(paths[1], ios::binary),
        B2(paths[2], ios::binary),
        B3(paths[3], ios::binary);
    B0.write((char*)&k, sizeof(int));
    while (!file.eof()) {
        file >> m;
        num++;
        if (k > m) {
            i++;
            if (i == NUM_OF_FILES) {
                i = 0;
            }
        }
        switch (i) {
            case 0:
                B0.write((char*)&m, sizeof(int));
                break;
            case 1:
                B1.write((char*)&m, sizeof(int));
                break;
            case 2:
                B2.write((char*)&m, sizeof(int));
                break;
            case 3:
                B3.write((char*)&m, sizeof(int));
        }
        k = m;
    }
    B0.close();
    B1.close();
    B2.close();
    B3.close();
    return num;
}

void mergeFiles(vector<string>& pathsB, vector<string>& pathsC)
{
    vector<int> nums;
    ifstream
        B0(pathsB[0], ios::binary),
        B1(pathsB[1], ios::binary),
        B2(pathsB[2], ios::binary),
        B3(pathsB[3], ios::binary);
    ofstream
        C0(pathsC[0], ios::binary),

```

```

    C1(pathsC[1], ios::binary),
    C2(pathsC[2], ios::binary),
    C3(pathsC[3], ios::binary);
int k, m, n;
for (int i = 0; i < NUM_OF_FILES; ++i) {
    k = INT_MAX;
    switch (i) {
    case 0:
        B0.read((char*)&k, sizeof(int));
        break;
    case 1:
        B1.read((char*)&k, sizeof(int));
        break;
    case 2:
        B2.read((char*)&k, sizeof(int));
        break;
    case 3:
        B3.read((char*)&k, sizeof(int));
    }
    nums.push_back(k);
}
n = findNumOfMin(nums);
k = nums[n];
int i = 0;
C0.write((char*)&k, sizeof(int));
while (true) {
    nums[n] = INT_MAX;
    switch (n) {
    case 0:
        B0.read((char*)&nums[n], sizeof(int));
        break;
    case 1:
        B1.read((char*)&nums[n], sizeof(int));
        break;
    case 2:
        B2.read((char*)&nums[n], sizeof(int));
        break;
    case 3:
        B3.read((char*)&nums[n], sizeof(int));
    }
    n = findNumOfMin(nums);
    m = nums[n];
    if (m == INT_MAX) {
        break;
    }
    if (k > m) {
        i++;
        if (i == NUM_OF_FILES) {
            i = 0;
        }
    }
    switch (i) {
    case 0:
        C0.write((char*)&m, sizeof(int));
        break;
    case 1:

```

```

        C1.write((char*)&m, sizeof(int));
        break;
    case 2:
        C2.write((char*)&m, sizeof(int));
        break;
    case 3:
        C3.write((char*)&m, sizeof(int));
    }
    k = m;
}
B0.close();
B1.close();
B2.close();
B3.close();
C0.close();
C1.close();
C2.close();
C3.close();
}

void convertBinToText(string pathBin, string pathTxt)
{
    ifstream bin(pathBin, ios::binary);
    bin.seekg(0, ios::end);
    int lenth = bin.tellg(), i = 0;
    ofstream txt(pathTxt);
    bin.seekg(0, ios::beg);
    while (i * sizeof(int) < lenth) {
        int k;
        bin.read((char*)&k, sizeof(int));
        txt << k << " ";
        i++;
    }
    txt.close();
    bin.close();
}

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++.

## 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%.