

**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)**

*ІП-14 Бабіч Денис*

(шифр, прізвище, ім'я, по батькові)

**Перевірів**

*Ахаладзе Ілля Елдарійович*

(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	9
3.2.1	Вихідний код	9
	<b>Висновок</b>	<b>14</b>
	<b>КРИТЕРІЙ ОЦІНЮВАННЯ</b>	<b>15</b>

## 1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні алгоритми зовнішнього сортування та способи їх модифікації, оцінити поріг їх ефективності.

## 2 ЗАВДАННЯ

### Варіант 2

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

```
const DEFAULT = 0
const MODIFICATION = 1
path = "Files\\A.txt"

if SelectApproach() == MODIFICATION then
    PreSort(path, GetSize(path))
endif

while (!IsFileSorted(path)) do
    ManageData(path)
    MergeData(path)
end while

function SelectApproach() returns void
do then
    if Int32.Parse(Console.ReadLine(), out int value) then
        if value > 1 or value < 0 then
            Console.WriteLine("Допустимі лише значення.")
            continue
        return value
    else
        Console.WriteLine("Введіть коректні дані.")
    endif
while (true)

function GetSize(path) returns void
fileSize = FileInfo(path).Length
int size
do then
    if Int32.Parse(Console.ReadLine(), out size) then
        if value > 1 or value < 0 then
            Console.WriteLine("Допустимі лише значення.")
            continue
        return size / sizeof(Int32)
    else
        Console.WriteLine("Введіть коректні дані.")
    endif
while (true)

function IsFileSorted(path) returns bool
sr = StreamReader(File.OpenRead(path))
previousNumber = 0
currentNumber = 0
TryGetNumber(sr, out previousNumber)
while (TryGetNumber(sr, out currentNumber)) do
    if (previousNumber > currentNumber) then
        return false
    endif
    previousNumber = currentNumber
end while
return true
```

```

function TryGetNumber(sr, out number) returns bool
    temp = ""

    while (sr.Peek() != ' ' and sr.Peek() != -1) do
        temp += (char)sr.Read()
    endwhile

    if(sr.Peek() != -1) then
        sr.Read()
    endif

    if(Int32.TryParse(temp, out number)) then
        return true
    else
        return false
    endif
endfunction

function ManageData(string path) returns void
    pathB = path.Replace("A.txt", "B.txt")
    pathC = path.Replace("A.txt", "C.txt")
    File.WriteAllText(pathB, string.Empty)
    File.WriteAllText(pathC, string.Empty)
    sr = StreamReader(File.OpenRead(path))
    sw1 = new StreamWriter(File.OpenWrite(pathB)),
    sw2 = new StreamWriter(File.OpenWrite(pathC))

    numberOfSeries = 0
    previousNumber = 0
    currentNumber = 0

    TryGetNumber(sr, out previousNumber)
    sw1.Write($"{previousNumber} ")

    while (TryGetNumber(sr, out currentNumber)) do
        if (previousNumber > currentNumber) then
            numberOfSeries += 1
        endif

        previousNumber = currentNumber

        if (numberOfSeries % 2 == 0) then
            sw1.Write($"{currentNumber} ")
        else
            sw2.Write($"{currentNumber} ")
        endif
    endwhile
endfunction

function MergeData(string path) returns void
    pathB = path.Replace("A.txt", "B.txt")
    pathC = path.Replace("A.txt", "C.txt")

    File.WriteAllText(path, string.Empty)

    sr1 = new StreamReader(File.OpenRead(pathB)),
    sr2 = new StreamReader(File.OpenRead(pathC))

    numbers = new List<int>()
    previousNumber, currentNumber
    lastAFile? = null
    lastBFile = null

```

```

TryGetNumber(sr1, out previousNumber)
numbers.Add(previousNumber)

while(TryGetNumber(sr1, out currentNumber)) do
    if (previousNumber <= currentNumber) then
        numbers.Add(currentNumber);
    else
        lastAFile = currentNumber
        break
    endif
    previousNumber = currentNumber
endwhile

TryGetNumber(sr2, out previousNumber)
numbers.Add(previousNumber)

while (TryGetNumber(sr2, out currentNumber)) do
    if (previousNumber <= currentNumber) then
        numbers.Add(currentNumber)
    else
        lastBFile = currentNumber
        break
    endif
    previousNumber = currentNumber
endwhile

Sort(numbers)

sw = StreamWriter(File.OpenWrite(path))
for (i = 0 to numbers.Count) do
    sw.Write($"{numbers[i]} ")
endfor

numbers.Clear()

while(true) do
    if(lastAFile != null) then
        previousNumber = lastAFile
        numbers.Add(previousNumber)
        lastAFile = null
    endif
    while (TryGetNumber(sr1, out currentNumber)) do
        if (previousNumber <= currentNumber) then
            numbers.Add(currentNumber)
        else
            lastAFile = currentNumber
            break
        endif
        previousNumber = currentNumber
    endwhile
    if (lastBFile != null) then
        previousNumber = lastBFile
        numbers.Add(previousNumber)
        lastBFile = null
    endif
    while (TryGetNumber(sr2, out currentNumber)) do
        if (previousNumber <= currentNumber) then
            numbers.Add(currentNumber)
        else
            lastBFile = currentNumber
            break
        endif
    endwhile
endwhile

```

```

        previousNumber = currentNumber
    endwhile
    if (numbers.Count != 0)
        numbers.Sort()
        for (i = 0 to numbers.Count) do
            sw.Write($"{numbers[i]} ")
        endfor
    else
        break
    endif
endwhile

```

## 3.2 Програмна реалізація алгоритму

### 3.2.1 Вихідний код

```

4  namespace ExternalSorting
5  {
6      public class ExternalSorting
7      {
8          private const int DEFAULT = 0;
9          private const int MODIFICATION = 1;
10
11         private static void Main()
12         {
13             Console.InputEncoding = System.Text.Encoding.Unicode;
14             Console.OutputEncoding = System.Text.Encoding.Unicode;
15
16             string path = "Files\\A.txt";
17
18             if (SelectApproach() == MODIFICATION)
19                 PreSort(path, GetSize(path));
20
21             while (!IsFileSorted(path))
22             {
23                 ManageData(path);
24                 MergeData(path);
25             }
26
27             Console.WriteLine("Файл відсортовано!");
28         }
29
30         public static int SelectApproach()
31         {
32             do
33             {
34                 Console.Write($"Оберіть спосіб сортування. Звичайне зовнішнє
сортування ({DEFAULT}), або зовнішнє сортування з модифікацією ({MODIFICATION}): ");
35
36                 if (Int32.TryParse(Console.ReadLine(), out int value))
37                 {
38                     if (value > 1 || value < 0)
39                     {
40                         Console.WriteLine($"Допустимі значення лише {DEFAULT} або
{MODIFICATION}!");
41                         continue;
42                     }
43
44                     return value;
45                 }
46                 else
47                     Console.WriteLine("Введіть коректні дані!");

```

```

48         } while (true);
49     }
50
51     public static int GetSize(string path)
52     {
53         long fileSize = new FileInfo(path).Length;
54         int size;
55
56         do
57         {
58             Console.Write("Скільки байтів файлу Ви бажаєте попередньо
59 впорядкувати: ");
60
61             if (Int32.TryParse(Console.ReadLine(), out size))
62             {
63                 if (size > fileSize)
64                 {
65                     Console.WriteLine("Розмір послідовності має бути меншим за
66 розмір файлу!");
67                     continue;
68                 }
69                 return size /= sizeof(Int32);
70             }
71             else
72                 Console.WriteLine("Введіть коректні дані.");
73         } while (true);
74     }
75
76     public static bool TryGetNumber(StreamReader sr, out int number)
77     {
78         string temp = String.Empty;
79
80         while (sr.Peek() != ' ' && sr.Peek() != -1)
81             temp += (char)sr.Read();
82
83         if (sr.Peek() != -1)
84             sr.Read();
85
86         if (Int32.TryParse(temp, out number))
87             return true;
88         else
89             return false;
90     }
91
92     public static void ManageData(string path)
93     {
94         string pathB = path.Replace("A.txt", "B.txt");
95         string pathC = path.Replace("A.txt", "C.txt");
96
97         File.WriteAllText(pathB, string.Empty);
98         File.WriteAllText(pathC, string.Empty);
99
100         using (StreamReader sr = new StreamReader(File.OpenRead(path)))
101         {
102             using (StreamWriter sw1 = new StreamWriter(File.OpenWrite(pathB)), sw2
103 = new StreamWriter(File.OpenWrite(pathC)))
104             {
105                 int numberOfSeries = 0;
106                 int previousNumber, currentNumber;
107
108                 TryGetNumber(sr, out previousNumber);
109                 sw1.Write($"{previousNumber} ");

```

```

110
111         while (TryGetNumber(sr, out currentNumber))
112         {
113             if (previousNumber > currentNumber)
114                 ++numberOfSeries;
115
116             previousNumber = currentNumber;
117
118             if (numberOfSeries % 2 == 0)
119                 sw1.Write($"{currentNumber} ");
120             else
121                 sw2.Write($"{currentNumber} ");
122         }
123     }
124 }
125
126
127 public static void MergeData(string path)
128 {
129     string pathB = path.Replace("A.txt", "B.txt");
130     string pathC = path.Replace("A.txt", "C.txt");
131
132     File.WriteAllText(path, string.Empty);
133
134     using (StreamReader sr1 = new StreamReader(File.OpenRead(pathB)), sr2 =
new StreamReader(File.OpenRead(pathC)))
135     {
136         List<int> numbers = new List<int>();
137
138         int previousNumber, currentNumber;
139         int? lastAFile = null, lastBFile = null;
140
141         TryGetNumber(sr1, out previousNumber);
142         numbers.Add(previousNumber);
143
144         while (TryGetNumber(sr1, out currentNumber))
145         {
146             if (previousNumber <= currentNumber)
147                 numbers.Add(currentNumber);
148             else
149             {
150                 lastAFile = currentNumber;
151                 break;
152             }
153
154             previousNumber = currentNumber;
155         }
156
157         TryGetNumber(sr2, out previousNumber);
158         numbers.Add(previousNumber);
159
160         while (TryGetNumber(sr2, out currentNumber))
161         {
162             if (previousNumber <= currentNumber)
163                 numbers.Add(currentNumber);
164             else
165             {
166                 lastBFile = currentNumber;
167                 break;
168             }
169
170             previousNumber = currentNumber;
171         }
172
173         numbers.Sort();

```

```

174
175 using (StreamWriter sw = new StreamWriter(File.OpenWrite(path)))
176 {
177     for (int i = 0, length = numbers.Count; i < length; ++i)
178         sw.Write($"{numbers[i]} ");
179
180     numbers.Clear();
181
182     while (true)
183     {
184         if (lastAFile != null)
185         {
186             previousNumber = lastAFile.Value;
187             numbers.Add(previousNumber);
188             lastAFile = null;
189         }
190
191         while (TryGetNumber(sr1, out currentNumber))
192         {
193             if (previousNumber <= currentNumber)
194                 numbers.Add(currentNumber);
195             else
196             {
197                 lastAFile = currentNumber;
198                 break;
199             }
200
201             previousNumber = currentNumber;
202         }
203
204         if (lastBFile != null)
205         {
206             previousNumber = lastBFile.Value;
207             numbers.Add(previousNumber);
208             lastBFile = null;
209         }
210
211         while (TryGetNumber(sr2, out currentNumber))
212         {
213             if (previousNumber <= currentNumber)
214                 numbers.Add(currentNumber);
215             else
216             {
217                 lastBFile = currentNumber;
218                 break;
219             }
220
221             previousNumber = currentNumber;
222         }
223
224         if (numbers.Count != 0)
225         {
226             numbers.Sort();
227
228             for (int i = 0, length = numbers.Count; i < length; ++i)
229                 sw.Write($"{numbers[i]} ");
230
231             numbers.Clear();
232         }
233         else
234             break;
235     }
236 }
237 }
238 }

```

```

239
240 public static bool IsFileSorted(string path)
241 {
242     using (StreamReader sr = new StreamReader(File.OpenRead(path)))
243     {
244         int previousNumber, currentNumber;
245
246         TryGetNumber(sr, out previousNumber);
247
248         while (TryGetNumber(sr, out currentNumber))
249         {
250             if (previousNumber > currentNumber)
251                 return false;
252
253             previousNumber = currentNumber;
254         }
255     }
256
257     return true;
258 }
259
260 public static void PreSort(string path, int size)
261 {
262     string tempPath = path.Replace("A.txt", "Temp.txt");
263
264     using (StreamReader sr = new StreamReader(File.OpenRead(path)))
265     {
266         using (StreamWriter sw = new StreamWriter(File.OpenWrite(tempPath)))
267         {
268             List<int> sequence = new List<int>(size);
269
270             int counter = 0;
271             int currentNumber;
272
273             while (true)
274             {
275                 while (counter < size)
276                 {
277                     if (TryGetNumber(sr, out currentNumber))
278                         sequence.Add(currentNumber);
279
280                     ++counter;
281                 }
282
283                 sequence.Sort();
284
285                 for (int i = 0, length = sequence.Count; i < length; ++i)
286                     sw.Write($"{sequence[i]} ");
287
288                 sequence.Clear();
289
290                 counter = 0;
291
292                 if (sr.Peek() == -1)
293                     break;
294             }
295         }
296     }
297
298     File.Delete(path);
299     File.Move(tempPath, path);
300 }
301
302}

```

## ВИСНОВОК

При виконанні даної лабораторної роботи дослідив алгоритм сортування природне (адаптивне) злиття. Переконався у його працездатності на завчасно підготованих текстових файлах. При перевірці виконання роботи файл розміром 10 мб був відсортований приблизно за 13 с, а файл розміром 1 гб було відсортовано за 2.5 хв, враховуючи те, що початковий файл було попередньо відсортовано серіями розмірами по 500 мб.

## 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%.