**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-15 Поліщук Валерій*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2022

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.

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

***Псевдокод алгоритму***

toTake = 1;

**Поки** (true) do

Split()

Merge()

**Якщо** (parts == 2)

**То**

break

**Все** **Якщо**

**Все** **Поки**

**Split()**

Відкрити файл file.bin для читання

Відкрити файли b.bin, c.bin для запису

parts = 1

counter = 0

flag = true

length = file.Length;

position = 0

**Поки** (position != length)

**Якщо** (counter == toTake)

**То**

flag = !flag

counter = 0

parts++

**Все** **Якщо**

element = br.ReadByte

position++

**Якщо** (flag)

**То**

b.Write(element)

**Інакше**

c.Write(element)

**Все** **Якщо**

counter++

**Все** **Поки**

Закрити файл b

Закрити файл c

Закрити файл file

**Merge()**

Відкрити файл file.bin для запису

Відкрити файли b.bin, c.bin для читання

counterB = toTake, counterC = toTake

elementB = 0, elementC = 0;

pickedB = false, pickedC = false, endB = false, endC = false

lengthB = B.Length

lengthC = C.Length

positionB = 0

positionC = 0

**Поки** (!endB || !endC)

**Якщо** (counterB == 0 && counterC == 0)

**То**

counterB = toTake

counterC = toTake

**Все** **Якщо**

**Якщо** (positionB != lengthB)

**То**

**Якщо** (counterB > 0 && !pickedB)

**То**

elementB = readerB.ReadByte

positionB += 1

pickedB = true

**Все** **Якщо**

**Інакше**

endB = true

**Все** **Якщо**

**Якщо** (positionC != lengthC)

**То**

**Якщо** (counterC > 0 && !pickedC)

**То**

elementC = readerC.ReadByte

positionC += 1

pickedC = true

**Все** **Якщо**

**Інакше**

endC = true;

**Все** **Якщо**

**Якщо** (pickedB)

**То**

**Якщо** (pickedC)

**То**

**Якщо** (elementB < elementC)

**То**

bw.Write(elementB)

counterB--

pickedB = false

**Інакше**

bw.Write(elementC)

counterC--

pickedC = false

**Все** **Якщо**

**Інакше**

bw.Write(elementB)

counterB--

pickedB = false

**Все** **Якщо**

**Інакше**

**Якщо** (pickedC)

**То**

bw.Write(elementC)

counterC--

pickedC = false

**Все** **Якщо**

**Все** **Якщо**

**Все** **Поки**

toTake \*= 2

Закрити файл b

Закрити файл c

Закрити файл file

***Псевдокод модифікованого алгоритму***

Prep()

**Поки** (true) do

SplitOptimized()

MergeOptimized()

**Якщо** (parts == 2)

**То**

break

**Все** **Якщо**

**Все** **Поки**

**Prep()**

Відкрити файл file.bin для читання

Відкрити файл prep.bin для запису

**Виконувати**

mas = br.ReadBytes(bufferSizeForPrep);

Array.Sort(mas);

prep.Write(mas);

**Поки** (mas.Length == bufferSizeForPrep)

Закрити файл file.bin

Закрити файл prep.bin

toTake = bufferSizeForPrep

Видалити file.bin

Перейменувати prep.bin в file.bin

**SplitOptimized()**

Відкрити файл file.bin для читання

Відкрити файли b.bin, c.bin для запису

parts = 1

counter = 0

flag = true

**Виконувати**

mas = file.ReadBytes(bufferSizeForSplit);

**Повторити** для i від 1 до mas.Length

**Якщо** (counter == toTake)

**То**

flag = !flag

counter = 0

parts++

**Все** **Якщо**

**Якщо** (flag)

**То**

b.Write(mas[i])

**Інакше**

c.Write(mas[i])

**Все** **Якщо**

counter++

**Все** **Повторити**

**Поки** (mas.Length == bufferSizeForSplit)

Закрити файл b

Закрити файл c

Закрити файл file

**MergeOptimized()**

Відкрити файл file.bin для запису

Відкрити файли b.bin, c.bin для читання

counterB = toTake, counterC = toTake

elementB = 0, elementC = 0;

**Виконувати**

masB = B.ReadBytes(bufferSizeForMerge)

masC = C.ReadBytes(bufferSizeForMerge)

positionForWrite = 0, position = 0, position = 0

pickedB = false, pickedC = false, endB = false, endC = false

positionForWrite = 0;

**Поки** (!endB || !endC)

**Якщо** (counterB == 0 && counterC == 0)

**То**

counterB = toTake

counterC = toTake

**Все** **Якщо**

**Якщо** (positionB != masB.length)

**То**

**Якщо** (counterB > 0 && !pickedB)

**То**

elementB = masB[positionB]

positionB += 1

pickedB = true

**Все** **Якщо**

**Інакше**

endB = true

**Все** **Якщо**

**Якщо** (positionC != masC.Length)

**То**

**Якщо** (counterC > 0 && !pickedC)

**То**

elementC = masC[positionC]

positionC += 1

pickedC = true

**Все** **Якщо**

**Інакше**

endC = true;

**Все** **Якщо**

**Якщо** (pickedB)

**То**

**Якщо** (pickedC)

**То**

**Якщо** (elementB < elementC)

**То**

masToWrite[positionForWrite] = elementB

positionForWrite++

counterB--

pickedB = false

**Інакше**

masToWrite[positionForWrite] = elementC

positionForWrite++

counterC--

pickedC = false

**Все** **Якщо**

**Інакше**

masToWrite[positionForWrite] = elementB

positionForWrite++

counterB--

pickedB = false

**Все** **Якщо**

**Інакше**

**Якщо** (pickedC)

**То**

masToWrite[positionForWrite] = elementC

positionForWrite++

counterC--

pickedC = false

**Все** **Якщо**

**Все** **Якщо**

**Все** **Поки**

file.Write(masToWrite);

**Поки** (masB.Length == bufferSizeForMerge || masC.Length == bufferSizeForMerge)

toTake \*= 2

Закрити файл b

Закрити файл c

Закрити файл file

## Програмна реалізація алгоритму

### Вихідний код

**Базовий алгоритм**

**public void DefaultStart()**

{

while (true)

{

Split();

Merge();

if (parts == 2)

{

break;

}

}

}

**private void Split()**

{

parts = 1;

BinaryReader br = new BinaryReader(File.OpenRead(PathA));

BinaryWriter writerB = new BinaryWriter(File.Create(PathB, 65536));

BinaryWriter writerC = new BinaryWriter(File.Create(PathC, 65536));

long counter = 0;

bool flag = true;

long length = br.BaseStream.Length;

long position = 0;

byte element;

while (position != length)

{

if (counter == toTake)

{

flag = !flag;

counter = 0;

parts++;

}

element = br.ReadByte();

position += 1;

if (flag)

{

writerB.Write(element);

}

else

{

writerC.Write(element);

}

counter++;

}

br.Close();

writerB.Close();

writerC.Close();

}

**private void Merge()**

{

BinaryReader readerB = new BinaryReader(File.OpenRead(PathB));

BinaryReader readerC = new BinaryReader(File.OpenRead(PathC));

BinaryWriter bw = new BinaryWriter(File.Create(PathA, 65536));

long counterB = toTake, counterC = toTake;

byte elementB = 0, elementC = 0;

bool pickedB = false, pickedC = false, endB = false, endC = false;

long lengthB = readerB.BaseStream.Length;

long lengthC = readerC.BaseStream.Length;

long positionB = 0;

long positionC = 0;

while (!endB || !endC)

{

if (counterB == 0 && counterC == 0)

{

counterB = toTake;

counterC = toTake;

}

if (positionB != lengthB)

{

if (counterB > 0 && !pickedB)

{

elementB = readerB.ReadByte();

positionB += 1;

pickedB = true;

}

}

else

{

endB = true;

}

if (positionC != lengthC)

{

if (counterC > 0 && !pickedC)

{

elementC = readerC.ReadByte();

positionC += 1;

pickedC = true;

}

}

else

{

endC = true;

}

if (pickedB)

{

if (pickedC)

{

if (elementB < elementC)

{

bw.Write(elementB);

counterB--;

pickedB = false;

}

else

{

bw.Write(elementC);

counterC--;

pickedC = false;

}

}

else

{

bw.Write(elementB);

counterB--;

pickedB = false;

}

}

else if (pickedC)

{

bw.Write(elementC);

counterC--;

pickedC = false;

}

}

toTake \*= 2;

bw.Close();

readerB.Close();

readerC.Close();

}

**Модифікований Алгоритм**

**public void** **OptimizedStart()**

{

Prep();

while (true)

{

SplitOptimized();

MergeOptimized();

if (parts == 2)

{

break;

}

}

}

**public void Prep()**

{

BinaryReader br = new BinaryReader(File.OpenRead(PathA));

BinaryWriter writerPrep = new BinaryWriter(File.Create(PathPrep, 65536));

byte[] mas;

do

{

mas = br.ReadBytes(bufferSizeForPrep);

Array.Sort(mas);

writerPrep.Write(mas);

}

while (mas.Length == bufferSizeForPrep);

br.Close();

writerPrep.Close();

toTake = bufferSizeForPrep;

File.Delete(PathA);

File.Move(PathPrep, PathA);

}

**private void SplitOptimized()**

{

parts = 1;

BinaryReader br = new BinaryReader(File.OpenRead(PathA));

BinaryWriter writerB = new BinaryWriter(File.Create(PathB, 65536)); //65536

BinaryWriter writerC = new BinaryWriter(File.Create(PathC, 65536));

byte[] mas;

long counter = 0;

bool flag = true;

do

{

mas = br.ReadBytes(bufferSizeForSplit);

for (int i = 0; i < mas.Length; i++)

{

if (counter == toTake)

{

flag = !flag;

counter = 0;

parts++;

}

if (flag)

{

writerB.Write(mas[i]);

}

else

{

writerC.Write(mas[i]);

}

counter++;

}

}

while (mas.Length == bufferSizeForSplit);

br.Close();

writerB.Close();

writerC.Close();

}

**private void MergeOptimized()**

{

BinaryReader readerB = new BinaryReader(File.OpenRead(PathB));

BinaryReader readerC = new BinaryReader(File.OpenRead(PathC));

BinaryWriter bw = new BinaryWriter(File.Create(PathA, 65536));

byte[] masB; byte[] masC;

long counterB = toTake, counterC = toTake;

byte elementB = 0, elementC = 0;

bool pickedB = false, pickedC = false, endB = false, endC = false;

long positionB;

long positionC;

long positionForWrite;

do

{

masB = readerB.ReadBytes(bufferSizeForMerge);

masC = readerC.ReadBytes(bufferSizeForMerge);

byte[] masToWrite = new byte[masB.Length + masC.Length];

positionForWrite = 0;

positionB = 0;

positionC = 0;

pickedB = false;

pickedC = false;

endB = false;

endC = false;

while (!endB || !endC)

{

if (counterB == 0 && counterC == 0)

{

counterB = toTake;

counterC = toTake;

}

if (positionB != masB.Length)

{

if (counterB > 0 && !pickedB)

{

elementB = masB[positionB];

positionB += 1;

pickedB = true;

}

}

else

{

endB = true;

}

if (positionC != masC.Length)

{

if (counterC > 0 && !pickedC)

{

elementC = masC[positionC];

positionC += 1;

pickedC = true;

}

}

else

{

endC = true;

}

if (pickedB)

{

if (pickedC)

{

if (elementB < elementC)

{

masToWrite[positionForWrite] = elementB;

positionForWrite++;

counterB--;

pickedB = false;

}

else

{

masToWrite[positionForWrite] = elementC;

positionForWrite++;

counterC--;

pickedC = false;

}

}

else

{

masToWrite[positionForWrite] = elementB;

positionForWrite++;

counterB--;

pickedB = false;

}

}

else if (pickedC)

{

masToWrite[positionForWrite] = elementC;

positionForWrite++;

counterC--;

pickedC = false;

}

}

bw.Write(masToWrite);

}

while (masB.Length == bufferSizeForMerge || masC.Length == bufferSizeForMerge);

toTake \*= 2;

bw.Close();

readerB.Close();

readerC.Close();

}

Висновок

Я виконав програмну реалізацію алгоритму зовнішнього сортування прямим злиттям на мові програмування С#.

Сортування випадково згенерованого файлу цілих чисел розміром 100 МБ цим алгоритмом тривало 1 хвилину 22 секунди.

Потім я здійснив модифікацію алгоритму : додав попереднє сортування та реалізував буфер для швидкого зчитування і запису у файл. Перша модифікація зменшила час сортування приблизно на 60 відсотків, а буфер зменшив час читання та запису у файли в 10 разів.

Після цього я запустив модифікований алгоритм на файлі розміром 100 МБ для перевірки його ефективності. Результат став набагато кращим – 9 секунд.

Нарешті, я перевірив алгоритм на файлі розміром двократного обсягу ОП мого комп’ютера (32 гб). Попередньо впорядковані серії елементів займали по 2 ГБ кожна. В результаті програма закінчила свою роботу через 82 хвилини 12 секунд.

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%.