**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**”





Київ 2022

Зміст

[**1**](#_heading=h.30j0zll) **Мета лабораторної роботи 3**

[**2**](#_heading=h.1fob9te) **Завдання 4**

[**3**](#_heading=h.3znysh7) **Виконання 6**

[3.1](#_heading=h.2et92p0) Псевдокод алгоритму 6

[3.2](#_heading=h.tyjcwt) Програмна реалізація алгоритму 6

[*3.2.1*](#_heading=h.3dy6vkm) *Вихідний код 6*

[**Висновок 7**](#_heading=h.1t3h5sf)

[**Критерії оцінювання 8**](#_heading=h.4d34og8)

# 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

* + 1. Псевдокод базового алгоритму

**MULTIWAYSORT.PY**

**infileName <- input('Enter the file name: ')**

**filesCount <- int(input('How many files to split into: '))**

**fileSize <- SIZE(infileName)**

**func.splitFile(infileName, filesCount, fileSize)**

**func.mergeFiles(filesCount)**

**func.clearGarbage(filesCount)**

**FUNC.PY**

**FUNCTION splitFile(infileName, filesCount, fileSize):**

**infile <- open(infileName, READ\_BIN)**

**outfileArr <- [open("b0.bin", WRITE\_BIN)]**

**currOutfileInd <- 0**

**outfilesOpen <- 1**

**prevNum <- b'\x00\x00\x00\x00'**

**FOR i = 0 to fileSize//4 do:**

**buffer <- infile.read(4)**

**IF buffer < prevNum:**

**currOutfileInd <- (currOutfileInd + 1) % filesCount**

**IF outfilesOpen < filesCount:**

**outfileArr.append(open(f"b{outfilesOpen}.bin", WRITE\_BIN))**

**outfilesOpen += 1**

**ENDIF**

**ENDIF**

**outfileArr[currOutfileInd].write(buffer)**

**prevNum <- buffer**

**ENDFOR**

**FOR i = 0 to outfilesOpen:**

**outfileArr[i].close()**

**ENDFOR**

**infile.close()**

**ENDFUNCTION**

**FUNCTION BtoC(filesCount):**

**infileArr <- []**

**outfileArr <- [open(f"c0.bin", WRITE\_BIN)]**

**outfilesOpen <- 1**

**currOutfileInd <- 0**

**lastInSeries <- b'\x00\x00\x00\x00'**

**waitingNumbers <- []**

**FOR i = 0 to filesCount:**

**infileArr.append(open(f"b{i}.bin", READ\_BIN))**

**waitingNumbers.append(infileArr[i].read(4))**

**ENDFOR**

**seriesCount <- 1**

**WHILE True:**

**min <- b'\xff\xff\xff\xff'**

**trueMin <- b'\xff\xff\xff\xff'**

**minInd <- 0**

**trueMinInd <- 0**

**FOR i = 0 to filesCount:**

**IF (waitingNumbers[i] >= lastInSeries) AND (waitingNumbers[i] <= min):**

**min <- waitingNumbers[i]**

**minInd <- i**

**ENDIF**

**IF waitingNumbers[i] <= trueMin:**

**trueMin <- waitingNumbers[i]**

**trueMinInd <- i**

**ENDIF**

**ENDFOR**

**IF min = b'\xff\xff\xff\xff':**

**IF trueMin = b'\xff\xff\xff\xff':**

**FOR i = 0 to len(infileArr):**

**infileArr[i].close()**

**ENDFOR**

**FOR i = 0 to len(outfileArr):**

**outfileArr[i].close()**

**ENDFOR**

**RETURN outfilesOpen, seriesCount**

**ELSEIF outfilesOpen < filesCount:**

**outfileArr.append(open(f"c{outfilesOpen}.bin", WRITE\_BIN))**

**outfilesOpen += 1**

**seriesCount += 1**

**currOutfileInd += 1**

**ELSE:**

**currOutfileInd <- (currOutfileInd + 1) % filesCount**

**seriesCount += 1**

**ENDIF**

**outfileArr[currOutfileInd].write(trueMin)**

**lastInSeries <- trueMin**

**waitingNumbers[trueMinInd] <- infileArr[trueMinInd].read(4)**

**IF not waitingNumbers[trueMinInd]:**

**waitingNumbers[trueMinInd] <- b'\xff\xff\xff\xff'**

**ENDIF**

**ELSE:**

**outfileArr[currOutfileInd].write(min)**

**lastInSeries <- min**

**waitingNumbers[minInd] <- infileArr[minInd].read(4)**

**IF not waitingNumbers[minInd]:**

**waitingNumbers[minInd] <- b'\xff\xff\xff\xff'**

**ENDIF**

**ENDIF**

**ENDWHILE**

**ENDFUNCTION**

**FUNCTION CtoB(filesCount):**

**infileArr <- []**

**outfileArr <- [open(f"b0.bin", WRITE\_BIN)]**

**outfilesOpen <- 1**

**currOutfileInd <- 0**

**lastInSeries <- b'\x00\x00\x00\x00'**

**waitingNumbers <- []**

**FOR i = 0 to filesCount:**

**infileArr.append(open(f"c{i}.bin", READ\_BIN))**

**waitingNumbers.append(infileArr[i].read(4))**

**ENDFOR**

**seriesCount <- 1**

**WHILE True:**

**min <- b'\xff\xff\xff\xff'**

**trueMin <- b'\xff\xff\xff\xff'**

**minInd <- 0**

**trueMinInd <- 0**

**FOR i = 0 to filesCount:**

**IF (waitingNumbers[i] >= lastInSeries) AND (waitingNumbers[i] <= min):**

**min <- waitingNumbers[i]**

**minInd <- i**

**ENDIF**

**IF waitingNumbers[i] <= trueMin:**

**trueMin <- waitingNumbers[i]**

**trueMinInd <- i**

**ENDIF**

**ENDFOR**

**IF min = b'\xff\xff\xff\xff':**

**IF trueMin = b'\xff\xff\xff\xff':**

**FOR i = 0 to len(infileArr):**

**infileArr[i].close()**

**ENDFOR**

**FOR i = 0 to len(outfileArr):**

**outfileArr[i].close()**

**ENDFOR**

**RETURN outfilesOpen, seriesCount**

**ELSEIF outfilesOpen < filesCount:**

**outfileArr.append(open(f"b{outfilesOpen}.bin", WRITE\_BIN))**

**outfilesOpen += 1**

**seriesCount += 1**

**currOutfileInd += 1**

**ELSE:**

**currOutfileInd <- (currOutfileInd + 1) % filesCount**

**seriesCount += 1**

**ENDIF**

**outfileArr[currOutfileInd].write(trueMin)**

**lastInSeries <- trueMin**

**waitingNumbers[trueMinInd] <- infileArr[trueMinInd].read(4)**

**IF not waitingNumbers[trueMinInd]:**

**waitingNumbers[trueMinInd] <- b'\xff\xff\xff\xff'**

**ENDIF**

**ELSE:**

**outfileArr[currOutfileInd].write(min)**

**lastInSeries <- min**

**waitingNumbers[minInd] <- infileArr[minInd].read(4)**

**IF not waitingNumbers[minInd]:**

**waitingNumbers[minInd] <- b'\xff\xff\xff\xff'**

**ENDIF**

**ENDIF**

**ENDWHILE**

**ENDFUNCTION**

**FUNCTION mergeFiles(filesCount):**

**filesFilled <- filesCount**

**WHILE filesFilled > 1:**

**filesFilled, seriesCount <- BtoC(filesFilled)**

**stoppedAt <- 'c0.bin'**

**IF filesFilled > 1:**

**filesFilled, seriesCount <- CtoB(filesFilled)**

**stoppedAt <- 'b0.bin'**

**ENDIF**

**ENDWHILE**

**RENAME(stoppedAt, 'sorted.bin')**

**ENDFUNCTION**

**FUNCTION clearGarbage(filesCount):**

**FOR i = 0 to filesCount:**

**DELETE(f'b{i}.bin')**

**DELETE(f'c{i}.bin')**

**ENDFOR**

**ENDFUNCTION**

* + 1. Псевдокод модифікованого алгоритму

**MULTIWAYSORTBETTER.PY**

**infileName <- input('Enter the file name: ')**

**filesCount <- int(input('How many files to split into: '))**

**fileSize <- os.stat(infileName).st\_size**

**numsPerFile <- fileSize // 4 // filesCount**

**rest <- fileSize // 4 % numsPerFile**

**func.splitFile(infileName, filesCount, numsPerFile)**

**FOR i = 0 to filesCount - 1:**

**func.sortFile(f"b{i}.bin", numsPerFile, i)**

**ENDFOR**

**func.sortFile(f"b{filesCount - 1}.bin", numsPerFile, i, rest)**

**func.mergeFiles(filesCount, fileSize)**

**func.clearGarbage(filesCount)**

**FUNC.PY**

**FUNCTION splitFile(infileName, filesCount, numsPerFile):**

**infile <- open(infileName, READ\_BIN)**

**FOR i = 0 to filesCount - 1:**

**currFile <- open(f"b{i}.bin", WRITE\_BIN)**

**buffer <- infile.read(4\*numsPerFile)**

**currFile.write(buffer)**

**currFile.close()**

**ENDFOR**

**currFile <- open(f"b{filesCount - 1}.bin", WRITE\_BIN)**

**buffer <- infile.read()**

**currFile.write(buffer)**

**currFile.close()**

**infile.close()**

**ENDFUNCTION**

**FUNCTION sortFile(currFileName, numsPerFile, fileInd, rest <- 0):**

**currFile <- open(currFileName, READ\_BIN)**

**unsortedArr <- list(int32)**

**unsortedArr.fromfile(currFile, numsPerFile + rest)**

**li <- unsortedArr.tolist()**

**li.sort()**

**arr <- list(int32)**

**arr.fromlist(li)**

**currFile <- open(currFileName, WRITE\_BIN)**

**arr.tofile(currFile)**

**currFile.close()**

**ENDFUNCTION**

**FUNCTION mergeFiles(filesCount, totalSize):**

**infileArr <- []**

**outfile <- open(f"sorted.bin", WRITE\_BIN)**

**waitingNumbers <- []**

**FOR i = 0 to filesCount:**

**infileArr.append(open(f"b{i}.bin", READ\_BIN))**

**waitingNumbers.append(infileArr[i].read(4))**

**ENDFOR**

**FOR loop = 0 to totalSize // 4:**

**min <- waitingNumbers[0]**

**minInd <- 0**

**FOR i = 1 to filesCount:**

**IF waitingNumbers[i] <= min:**

**min <- waitingNumbers[i]**

**minInd <- i**

**ENDIF**

**ENDFOR**

**IF min = b'\xff\xff\xff\xff':**

**FOR i = 0 to len(infileArr):**

**infileArr[i].close()**

**ENDFOR**

**filled <- SIZE('sorted.bin')**

**leftToFill <- totalSize - filled**

**outfile.write(b'\xff'\*leftToFill)**

**outfile.close()**

**break**

**ELSE:**

**outfile.write(min)**

**waitingNumbers[minInd] <- infileArr[minInd].read(4)**

**IF not waitingNumbers[minInd]:**

**waitingNumbers[minInd] <- b'\xff\xff\xff\xff'**

**ENDIF**

**ENDIF**

**ENDFOR**

**ENDFUNCTION**

**FUNCTION clearGarbage(filesCount):**

**for i in range(filesCount):**

**DELETE(f'b{i}.bin')**

## Програмна реалізація алгоритму

### Вихідний код базового алгоритму

“MultiwaySort.py”

import func

import os

import time

infileName = input('Enter the file name: ')

filesCount = int(input('How many files to split into: '))

if not os.path.isfile(infileName):

print("File doesn't exist, please create the file with FileGenerator.py:")

else:

fileSize = os.stat(infileName).st\_size

print(f"\nSorting the contents of {infileName}.")

startTime = time.time()

func.splitFile(infileName, filesCount, fileSize)

func.mergeFiles(filesCount)

endTime = time.time()

print(f"\nTotal time to sort the file: {round((endTime - startTime) \* 1000)/1000} seconds.")

print(f"Total file size: {round((fileSize//1024) \* 1000)/1000}KB.")

print(f"Algorithm speed: {round(((fileSize/1073741824) / ((endTime - startTime) / 600)) \* 100000)/100000}GB/10min")

func.clearGarbage(filesCount)

“func.py”

import time

import os

def splitFile(infileName, filesCount, fileSize):

"""splits the file into that amount of equal files"""

startTime = time.time()

infile = open(infileName, 'rb')

outfileArr = [open("b0.bin", 'wb')]

currOutfileInd = 0

outfilesOpen = 1

prevNum = b'\x00\x00\x00\x00'

for i in range(fileSize//4):

buffer = infile.read(4)

if buffer < prevNum:

currOutfileInd = (currOutfileInd + 1) % filesCount

if outfilesOpen < filesCount:

outfileArr.append(open(f"b{outfilesOpen}.bin", 'wb'))

outfilesOpen += 1

outfileArr[currOutfileInd].write(buffer)

prevNum = buffer

for i in range(outfilesOpen):

outfileArr[i].close()

infile.close()

endTime = time.time()

print(f"File split in {endTime - startTime} seconds.")

def BtoC(filesCount):

infileArr = []

outfileArr = [open(f"c0.bin", 'wb')]

outfilesOpen = 1

currOutfileInd = 0

lastInSeries = b'\x00\x00\x00\x00'

waitingNumbers = []

for i in range(filesCount):

infileArr.append(open(f"b{i}.bin", 'rb'))

waitingNumbers.append(infileArr[i].read(4))

seriesCount = 1

while True:

min = b'\xff\xff\xff\xff'

trueMin = b'\xff\xff\xff\xff'

minInd = 0

trueMinInd = 0

for i in range(0, filesCount):

if (waitingNumbers[i] >= lastInSeries) and (waitingNumbers[i] <= min):

min = waitingNumbers[i]

minInd = i

if waitingNumbers[i] <= trueMin:

trueMin = waitingNumbers[i]

trueMinInd = i

if min == b'\xff\xff\xff\xff':

if trueMin == b'\xff\xff\xff\xff':

for i in range(len(infileArr)):

infileArr[i].close()

for i in range(len(outfileArr)):

outfileArr[i].close()

return outfilesOpen, seriesCount

elif outfilesOpen < filesCount:

outfileArr.append(open(f"c{outfilesOpen}.bin", 'wb'))

outfilesOpen += 1

seriesCount += 1

currOutfileInd += 1

else:

currOutfileInd = (currOutfileInd + 1) % filesCount

seriesCount += 1

outfileArr[currOutfileInd].write(trueMin)

lastInSeries = trueMin

waitingNumbers[trueMinInd] = infileArr[trueMinInd].read(4)

if not waitingNumbers[trueMinInd]:

waitingNumbers[trueMinInd] = b'\xff\xff\xff\xff'

else:

outfileArr[currOutfileInd].write(min)

lastInSeries = min

waitingNumbers[minInd] = infileArr[minInd].read(4)

if not waitingNumbers[minInd]:

waitingNumbers[minInd] = b'\xff\xff\xff\xff'

def CtoB(filesCount):

infileArr = []

outfileArr = [open(f"b0.bin", 'wb')]

outfilesOpen = 1

currOutfileInd = 0

lastInSeries = b'\x00\x00\x00\x00'

waitingNumbers = []

for i in range(filesCount):

infileArr.append(open(f"c{i}.bin", 'rb'))

waitingNumbers.append(infileArr[i].read(4))

seriesCount = 1

while True:

min = b'\xff\xff\xff\xff'

trueMin = b'\xff\xff\xff\xff'

minInd = 0

trueMinInd = 0

for i in range(0, filesCount):

if (waitingNumbers[i] >= lastInSeries) and (waitingNumbers[i] <= min):

min = waitingNumbers[i]

minInd = i

if waitingNumbers[i] <= trueMin:

trueMin = waitingNumbers[i]

trueMinInd = i

if min == b'\xff\xff\xff\xff':

if trueMin == b'\xff\xff\xff\xff':

for i in range(len(infileArr)):

infileArr[i].close()

for i in range(len(outfileArr)):

outfileArr[i].close()

return outfilesOpen, seriesCount

elif outfilesOpen < filesCount:

outfileArr.append(open(f"b{outfilesOpen}.bin", 'wb'))

outfilesOpen += 1

seriesCount += 1

currOutfileInd += 1

else:

currOutfileInd = (currOutfileInd + 1) % filesCount

seriesCount += 1

outfileArr[currOutfileInd].write(trueMin)

lastInSeries = trueMin

waitingNumbers[trueMinInd] = infileArr[trueMinInd].read(4)

if not waitingNumbers[trueMinInd]:

waitingNumbers[trueMinInd] = b'\xff\xff\xff\xff'

else:

outfileArr[currOutfileInd].write(min)

lastInSeries = min

waitingNumbers[minInd] = infileArr[minInd].read(4)

if not waitingNumbers[minInd]:

waitingNumbers[minInd] = b'\xff\xff\xff\xff'

def mergeFiles(filesCount):

filesFilled = filesCount

while filesFilled > 1:

startTime = time.time()

filesFilled, seriesCount = BtoC(filesFilled)

stoppedAt = 'c0.bin'

if filesFilled > 1:

filesFilled, seriesCount = CtoB(filesFilled)

stoppedAt = 'b0.bin'

endTime = time.time()

print(f"Merging numbers into {seriesCount} series was completed. Another {endTime - startTime} seconds elapsed.")

try:

os.remove('sorted.bin')

except:

pass

os.rename(stoppedAt, 'sorted.bin')

print(f"Sorting done. Check file 'sorted.bin' to see sorted contents of the initial file.")

def clearGarbage(filesCount):

for i in range(filesCount):

try:

os.remove(f'b{i}.bin')

except:

pass

try:

os.remove(f'c{i}.bin')

except:

pass

* + 1. Вихідний код модифікованого алгоритму

“MultiwaySortBetter.py”

import func

import os

import time

infileName = input('Enter the file name: ')

filesCount = int(input('How many files to split into: '))

if not os.path.isfile(infileName):

print("File doesn't exist, please create the file with FileGenerator.py:")

else:

fileSize = os.stat(infileName).st\_size

numsPerFile = fileSize // 4 // filesCount

rest = fileSize // 4 % numsPerFile

print(f"\nSorting the contents of {infileName} into 'c.bin'.")

startTime = time.time()

print(f"\nSplitting the file into {filesCount} files, {numsPerFile} numbers in each:")

func.splitFile(infileName, filesCount, numsPerFile)

for i in range(filesCount - 1):

print(f"\nSorting file number {i}")

func.sortFile(f"b{i}.bin", numsPerFile, i)

print(f"\nSorting file number {filesCount - 1}")

func.sortFile(f"b{filesCount - 1}.bin", numsPerFile, i, rest)

print("Merging the files")

func.mergeFiles(filesCount, fileSize)

endTime = time.time()

print(f"\nTotal time to sort the file: {round((endTime - startTime) \* 1000)/1000} seconds.")

print(f"Total file size: {round((fileSize//1024) \* 1000)/1000}KB.")

print(f"Algorithm speed: {round(((fileSize/1073741824) / ((endTime - startTime) / 600)) \* 100000)/100000}GB/10min")

func.clearGarbage(filesCount)

“func.py”

import time

import array

import os

import psutil

def splitFile(infileName, filesCount, numsPerFile):

"""splits the file into that amount of equal files"""

startTime = time.time()

infile = open(infileName, 'rb')

for i in range(filesCount - 1):

currFile = open(f"b{i}.bin", 'wb')

buffer = infile.read(4\*numsPerFile)

currFile.write(buffer)

currFile.close()

currFile = open(f"b{filesCount - 1}.bin", 'wb')

buffer = infile.read()

currFile.write(buffer)

currFile.close()

infile.close()

endTime = time.time()

print(f"File split in {endTime - startTime} seconds.")

def sortFile(currFileName, numsPerFile, fileInd, rest = 0):

"""sorts whatever is in the file"""

startTime = time.time()

currFile = open(currFileName, 'rb')

unsortedArr = array.array('L')

unsortedArr.fromfile(currFile, numsPerFile + rest)

unsortedArr.byteswap()

sortStartTime = time.time()

li = unsortedArr.tolist()

li.sort()

arr = array.array('L')

arr.fromlist(li)

arr.byteswap()

sortEndTime = time.time()

print(f"Sorting done in {sortEndTime - sortStartTime} seconds.")

currFile = open(currFileName, 'wb')

arr.tofile(currFile)

currFile.close()

endTime = time.time()

print(f"Contents of file {fileInd} updated in {endTime - startTime} seconds.")

def mergeFiles(filesCount, totalSize):

infileArr = []

outfile = open(f"sorted.bin", 'wb')

waitingNumbers = []

for i in range(filesCount):

infileArr.append(open(f"b{i}.bin", 'rb'))

waitingNumbers.append(infileArr[i].read(4))

for loop in range(totalSize // 4):

min = waitingNumbers[0]

minInd = 0

for i in range(1, filesCount):

if waitingNumbers[i] <= min:

min = waitingNumbers[i]

minInd = i

if min == b'\xff\xff\xff\xff':

for i in range(len(infileArr)):

infileArr[i].close()

filled = os.stat('sorted.bin').st\_size

leftToFill = totalSize - filled

outfile.write(b'\xff'\*leftToFill)

outfile.close()

break

else:

outfile.write(min)

waitingNumbers[minInd] = infileArr[minInd].read(4)

if not waitingNumbers[minInd]:

waitingNumbers[minInd] = b'\xff\xff\xff\xff'

def clearGarbage(filesCount):

for i in range(filesCount):

try:

os.remove(f'b{i}.bin')

except:

pass

Висновок

При виконанні даної лабораторної роботи було досліджено роботу алгоритму зовнішнього сортування, а саме збалансованого багатошляхового злиття. Було створено два алгоритми з використанням мови програмування Python: базовий та модифікований.

Базовий алгоритм передає основну сутність алгоритму багатошляхового злиття: перенесення чисел із великої кількості файлів у велику кількість файлів, що зменшує необхідну кількість злиттів.

Модифікований алгоритм поєднує багатошляхове злиття зі внутрішнім сортуванням. Також у модифікованому алгоритмі використані ефективніші методи зчитування інформації з файлів, а також зменшена кількість операцій.

Швидкість роботи алгоритму зросла в ~9 разів для вихідного файлу розміром 16МБ та ще більше для більших файлів завдяки модифікаціям. У результаті файл розміром 16ГБ було відсортовано за 3 год 6 хв, що означає середню швидкість для даної розмірності ~11.75хв/ГБ

Критерії оцінювання

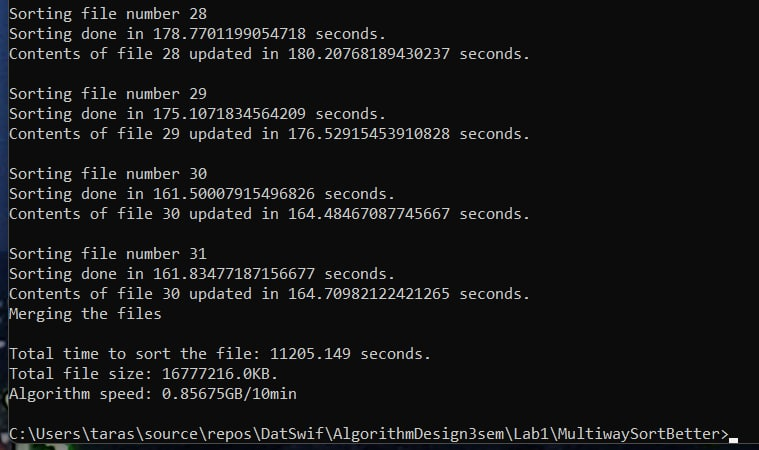
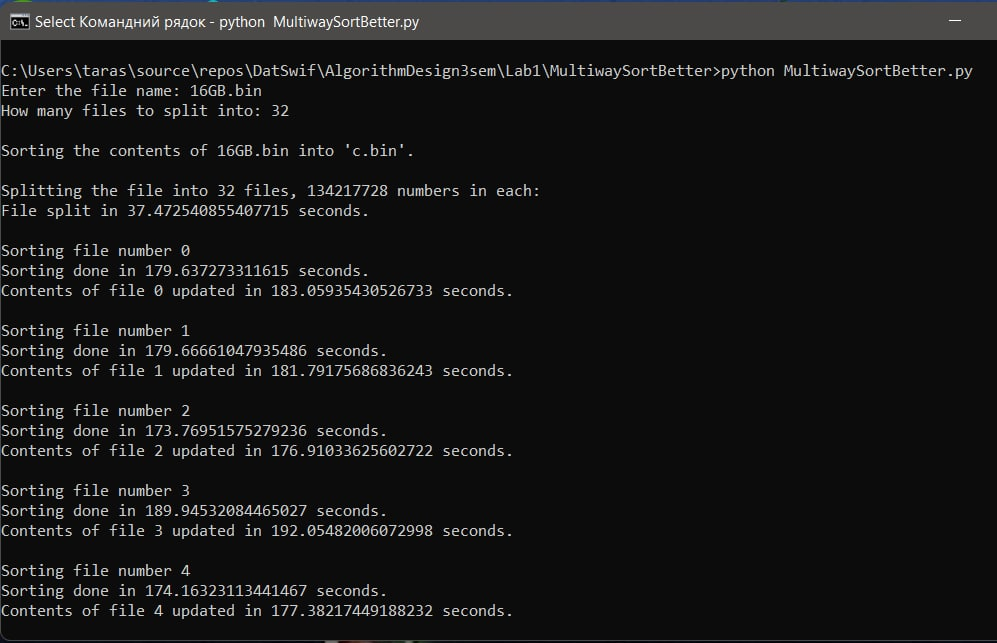
У випадку здачі лабораторної роботи до 09.10.2022 включно максимальний бал дорівнює – 5. Після 09.10.2022 максимальний бал дорівнює – 1.

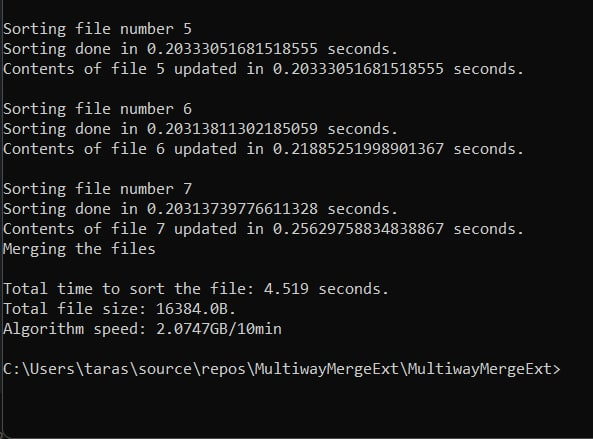
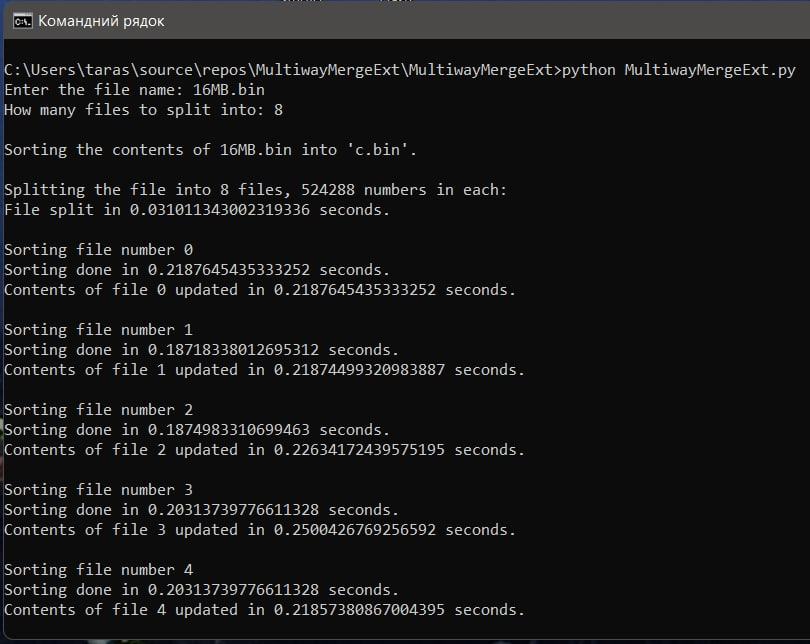
Критерії оцінювання у відсотках від максимального балу:

* псевдокод алгоритму – 15%;
* програмна реалізація алгоритму – 40%;
* програмна реалізація модифікацій – 40%;
* висновок – 5%.

Додаток - показ роботи програми

Розмір вихідного файлу 16ГБ, розділення на 32 файли:



Розмір вихідного файлу 16МБ, розділення на 8 файлів

Розмір вихідного файлу 16МБ, розділення на 8 файлів, базовий алгоритм

