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 *ІП-15 Химич А. М.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Соколовський В.В.* _____
(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

Повторити поки НЕ isSorted()

Split()

Merge()

Метод isSorted():

Поки i<numbers:

Якщо numbers[i]>numbers[i+1]

Повернути false

Інакше

Повернути True

I+=1

Метод Split():

Зчитати вхідну послідовність в numbers[int]

CurrentSequence[int]

Series[][]

Повторити для і від 0 до length(numbers)

Якщо numbers[i-1]<numbers[i]

CurrentSequence.append(numbers[i])

I+=1

Інакше

Series.append(CurrentSequence)

CurrentSequence.clear()

Відкрити файл B та файл C для запису

Повторити для і від 0 до length(series)

Якщо і парне :

До файлу B **записати** series[i]

Інакше

До файлу C **записати** series[i]

Метод Merge()

Зчитати з файлу B seriesB[int]

Зчитати з файлу C seriesC[int]

Numbers[int]

serieNumber:=0

Повторити поки (serieNumber<length(seriesB) AND serieNumber< length(seriesC)

I:=0

J:=0

Повторити поки (i < lenght(seriesB[serieNumber]) AND j < length(seriesC[serieNumber]))

Якщо seriesB[serieNumber][i] > seriesC[serieNumber][j]:

numbers.append(seriesC[serieNumber][j])

j += 1

інакше:

numbers.append(seriesB[serieNumber][i])

i += 1

Якщо (i >= len(seriesB[serieNumber]))

numbers.extend(seriesC[serieNumber][j:])

Інакше Якщо j >= len(seriesC[serieNumber]

numbers.extend(seriesB[serieNumber][i:])

serieNumber += 1

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код Базової версії мовою python

```
from datetime import datetime
FILEPATH_A = "C:/Users/user/Desktop/fileA.txt"
FILEPATH_B = "C:/Users/user/Desktop/file2.txt"
FILEPATH_C = "C:/Users/user/Desktop/file3.txt"


def writeSequence(flag, currentSequence,seriesB,seriesC):
   if flag:
      seriesB.extend(currentSequence)
   else:
      seriesC.extend(currentSequence)
   currentSequence.clear()


def readFile(FILEPATH):
   with open(FILEPATH) as file:
      data = file.read()
      data = data.strip()
   numbers = [int(num) for num in data.split(' ')]
   return numbers


def isSorted():
   numbers = readFile(FILEPATH_A)
   for i in range(len(numbers) - 1):
      if (numbers[i] > numbers[i + 1]):
         return False
   return True


def writeFile(series):
   seriesB=[]
   seriesC=[]
   for i in range(len(series)):
     if i %2 == 0:
        seriesC.append(" ".join(map(str, series[i])))
     else:
        seriesB.append(" ".join(map(str, series[i])))

   with open(FILEPATH_B,"w") as fileB:
      fileB.write(" ".join(map(str, seriesB)))
   with open(FILEPATH_C,"w") as fileC:
      fileC.write(" ".join(map(str, seriesC)))

def split():
   numbers=readFile(FILEPATH_A)
   currentSequence = []
   series = []
   currentSequence.append(numbers.pop(0))


   for number in numbers:
      if currentSequence[len(currentSequence) - 1] > number:
         series.append(currentSequence[:])
         currentSequence.clear()
```

```
            currentSequence.append(number)
        series.append(currentSequence[:])
        writeFile(series)


def readSeries(FILEPATH):
    numbers = readFile(FILEPATH)
    currentSequence = []
    series = []
    currentSequence.append(numbers.pop(0))

    for number in numbers:
        if currentSequence[len(currentSequence) - 1] > number:
            series.append(currentSequence[:])
            currentSequence.clear()

        currentSequence.append(number)
    series.append(currentSequence[:])
    return series


def merge():
    seriesB = readSeries(FILEPATH_B)
    seriesC = readSeries(FILEPATH_C)
    numbers = []
    serieNumber = 0
    while (serieNumber < len(seriesB) and serieNumber < len(seriesC)):

        i = j = 0
        while (i < len(seriesB[serieNumber]) and j < len(seriesC[serieNumber])):
            if seriesB[serieNumber][i] > seriesC[serieNumber][j]:
                numbers.append(seriesC[serieNumber][j])
                j += 1
            else:
                numbers.append(seriesB[serieNumber][i])
                i += 1
            if (i >= len(seriesB[serieNumber])):
                numbers.extend(seriesC[serieNumber][j:])
            elif j >= len(seriesC[serieNumber]):
                numbers.extend(seriesB[serieNumber][i:])
        serieNumber += 1


    if serieNumber > len(seriesB):
        for serieNumber in range(len(seriesC)):
            numbers.extend(seriesC[serieNumber][:])
            serieNumber += 1
    elif(serieNumber > len(seriesC)):
        for serieNumber in range(len(seriesB)):
            numbers.extend(seriesB[serieNumber][:])
            serieNumber += 1

    with open(FILEPATH_A, "w") as file:
        file.write(" ".join(map(str, numbers)))




while not isSorted():
    split()
    merge()
```

3.2.2 Вихідний код модифікації мовою python

```
import mmap

MEMORY =16*1024*1024*1024
CHUNKS=32
CHUNK_SIZE = int(MEMORY / CHUNKS)
FILEPATH_A = "C:/Users/user/Desktop/fileA.txt"
FILEPATH_B = "C:/Users/user/Desktop/file2.txt"
FILEPATH_C = "C:/Users/user/Desktop/file3.txt"


def readChunk(FIlEPATH,chunkNumber,CHUNK_SIZE):
    chunkNumber= chunkNumber*CHUNK_SIZE
    with open(FIlEPATH,"r+b") as file:
        mappedFile =
mmap.mmap(file.fileno(),length=CHUNK_SIZE,access=mmap.ACCESS_READ,offset=chunkNu
mber)
        numbers=[]
        numbers = [int(num) for num in mappedFile.read().decode().split(' ')]
    return numbers

def writeChunk(FIlEPATH,number,numbers,CHUNK_SIZE):
    number=number*CHUNK_SIZE
    with open(FIlEPATH, "r+b") as file:
        mappedFile =
mmap.mmap(file.fileno(),length=CHUNK_SIZE,access=mmap.ACCESS_WRITE,offset=number
)
        mappedFile.write(" ".join(map(str, numbers)).encode())

def merge(FILEPATH_B,FILEPATH_C,chunkNumber,CHUNK_SIZE):
   seriesB = readChunk(FILEPATH_B,chunkNumber,CHUNK_SIZE)
   seriesC = readChunk(FILEPATH_C,chunkNumber,CHUNK_SIZE)
   numbers = []
   number = 0
   i = j = 0
   while (number < len(seriesB) and number < len(seriesC)):

      if seriesB[i] > seriesC[j]:
        numbers.append(seriesC[j])
        j += 1
      else:
        numbers.append(seriesB[i])
        i += 1
      number += 1
   if (i >= len(seriesB)):
       numbers.extend(seriesC[j:])
   elif j >= len(seriesC):
       numbers.extend(seriesB[i:])
   return numbers
```

```
def prepare(FILEPATH_A,FILEPATH_B,FILEPATH_C,CHUNK_SIZE,CHUNKS):
    chunkNumberWrite = 0
    chunkNumberRead = 0
    while chunkNumberRead <= CHUNKS:
        numbers = readChunk(FILEPATH_A, chunkNumberRead, CHUNK_SIZE)
        numbers.sort()
        writeChunk(FILEPATH_B, chunkNumberWrite, numbers, CHUNK_SIZE)

        chunkNumberRead += 1
        numbers = readChunk(FILEPATH_A, chunkNumberRead, CHUNK_SIZE)
        numbers.sort()
        writeChunk(FILEPATH_C, chunkNumberWrite, numbers, CHUNK_SIZE)

        chunkNumberRead += 1
        chunkNumberWrite += 1


def sort(FILEPATH_A,FILEPATH_B,FILEPATH_C,CHUNK_SIZE,CHUNKS):
    prepare(FILEPATH_A, FILEPATH_B, FILEPATH_C, CHUNK_SIZE, CHUNKS)
    while CHUNK_SIZE<=MEMORY:
        CHUNKS=int(CHUNKS/2)
        chunkNumberRead = 0
        while chunkNumberRead< CHUNKS:
            merge(FILEPATH_B,FILEPATH_C,chunkNumberRead,CHUNK_SIZE)
        CHUNK_SIZE*=2


        chunkNumberWrite = 0
        chunkNumberRead = 0
        while chunkNumberRead <= CHUNKS:
            numbers = readChunk(FILEPATH_A, chunkNumberRead, CHUNK_SIZE)
            writeChunk(FILEPATH_B, chunkNumberWrite, numbers, CHUNK_SIZE)
            chunkNumberRead += 1

            numbers = readChunk(FILEPATH_A, chunkNumberRead, CHUNK_SIZE)
            writeChunk(FILEPATH_C, chunkNumberWrite, numbers, CHUNK_SIZE)
            chunkNumberRead += 1
            chunkNumberWrite += 1
```

ВИСНОВОК

При виконанні даної лабораторної роботи було спроектовано та реалізовано алгоритм зовнішнього сортування методом адаптивного(природнього) злиття. При виконанні базової версії завдання було розроблено програмну специфікацію зо здатна відсортувати вхідний масив розміром 10Мб за час 1хв 20секунд, масив вхідних даних розміром 1гб вдалось відсортувати за 20хв. В подальшому алгоритм було модифіковано, а саме: зчитування частини файла за допомогою відображення в пам’ять, сортування даної частини файла вбудованим методом sort(), що є імплементацією Timsort(часова складність: Найкращий випадок-O(n), середній випадок-O(nlogn) та найгірший випадок O(nlogn)). Та після перелічених вище процедур – злиття отриманих послідовнонстей до одного файлу.

Результатом даної модифікації є програмна специфікація, що здатна відсортувати файл даних розміром 16Гб за 51 хвилину, що відповідає швидкості 3хв 10секунд на 1Гб даних

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%.