

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-12 Волков Вадим Всеволодович 5.10.2022
(шифр, прізвище, ім'я, по батькові)

Перевірив Сопов Олексій Олександрович 6.10.2022
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ.....	9
2 ЗАВДАННЯ.....	9
3 ВИКОНАННЯ.....	9
3.1 Псевдокод алгоритму.....	9
3.2 Програмна реалізація алгоритму.....	9
3.2.1 Вихідний код.....	9
ВИСНОВОК.....	9
КРИТЕРІЇ ОЦІНЮВАННЯ.....	9

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

1. Псевдокод алгоритму

```
class Run:
    constructor(start, length, binFile):
        this.start = start
        this.length = length
        this.pos = 0
        this.binFile = binFile
        if length > 0 then
            this.hasMore = True
            this.value = binFile.getInt32(start)
        else
            this.hasMore = False
            this.value = 0
        end if
    end function
    next():
        this.pos += 1
        if this.pos < this.length then
            this.value = this.binFile.getInt32(this.start + this.pos)
        else
            this.hasMore = False
        end if
    end function
end class

sort(fileCount, runSize, visualize):
    binf = createAndOpenFile("numbers")

    # Створення файлів
    inputFiles = []
    inputFileRuns = []
    outputFile = createAndOpenFile("workingFile0")
    outputFileRuns = []
    for i=1 to fileCount do
        inputFiles.append(createAndOpenFile("workingFile"+i))
        inputFileRuns.append([])
    end for

    # Розрахунок розподілів
    distributions = [1 ... 1] (n = inputFiles.length)
    currentSize = inputFiles.length
    targetSize = ceiling(binf.size / runSize)
    while currentSize < targetSize do
        maxVal = -1
        maxI = 0
        for i=0 to distributions.length do
            if distributions[i] > maxVal then
                maxVal = distributions[i]
                maxI = i
            end
        end
        for i=0 to distributions.length do
            if i != maxI then
                distributions[i] += maxVal
                currentSize += maxVal
            end if
        end for
    end while
    totalRuns = currentSize

    # Розбиття вхідного файлу у відсортовані серії певної довжини
    writeFilePos = [0 ... 0] (n = inputFiles.length)
    writeToFiles = [0, 1 ... inputFiles.length]
    distr = distributions.copy()
    writeToFile = 0
    writeToFileIdx = 0
    readPos = 0
    runsWrittenSoFar = 0
    while readPos < binf.size do
```

```

        ints = binf.getInts32(readPos, runSize)
        ints.sort()
        inputFiles[writeToFile].setInts32(writeFilePos[writeToFile], ints)
        inputFileRuns[writeToFile].append(new Run(writeFilePos[writeToFile], ints.length,
inputFiles[writeToFile]))
        distr[writeToFileIdx] -= 1
        if distr[writeToFileIdx] < 1 then
            writeToFileIdx.remove(writeToFileIdx)
            distr.remove(writeToFileIdx)
            writeToFileIdx -= 1
        end if

        readPos += runSize
        writeFilePos[writeToFile] += runSize
        writeToFileIdx = (writeToFileIdx + 1) % writeToFileIdx.length
        writeToFile = writeToFileIdx[writeToFileIdx]
        runsWrittenSoFar += 1
    end while

```

Додання пустих серій

```

    while runsWrittenSoFar < totalRuns do
        inputFileRuns[writeToFile].append(new Run(writeFilePos[writeToFile], 0,
inputFiles[writeToFile]))
        distr[writeToFileIdx] -= 1

        if distr[writeToFileIdx] < 1 then
            writeToFileIdx.pop(writeToFileIdx)
            distr.pop(writeToFileIdx)
            writeToFileIdx -= 1
        end

        if writeToFileIdx.length > 0 then
            writeToFileIdx = (writeToFileIdx + 1) % writeToFileIdx.length
            writeToFile = writeToFileIdx[writeToFileIdx]
        end if
        runsWrittenSoFar += 1
    end while

```

Багатофазне злиття

```

keepGoing = True
outputPos = 0
while keepGoing do
    hasMore = True
    outputStart = outputPos
    while hasMore do
        minI = -1
        minVal = 9999999
        for i=0 to inputFiles.length do
            if inputFileRuns[i].length > 0 and inputFileRuns[i][0].hasMore then
                value = inputFileRuns[i][0].value
                if value < minVal then
                    minVal = value
                    minI = i
                end if
            end if
        end for
        if minI > -1 then
            outputFile.setInt32(outputPos, minVal)
            outputPos += 1
            inputFileRuns[minI][0].next()
            hasMore = True
        else
            hasMore = False
        end if
    end while

    outputLength = outputPos - outputStart
    outputFileRuns.append(new Run(outputStart, outputLength, outputFile))
    switchTo = -1
    switchToCount = 0
    for i=0 to inputFiles.length then
        if inputFileRuns[i].length > 0 then
            inputFileRuns[i].pop(0)
            if inputFileRuns[i].length == 0 then
                switchTo = i
                switchToCount += 1
            end if
        end if
    end for
    if switchToCount == inputFiles.length then
        keepGoing = False
    end if
end while

```

```

        end if
    end if
end for

if switchToCount == inputFiles.length then
    keepGoing = False
else
    if switchTo > -1 then
        temp = outputFile
        outputFile = inputFiles[switchTo]
        inputFiles[switchTo] = temp
        temp = outputFileRuns
        outputFileRuns = inputFileRuns[switchTo]
        inputFileRuns[switchTo] = temp
        outputPos = 0
    end if
end if
end while

# Завершення виконання

copy(outputFile, binf)
deleteAll(inputFiles)
delete(outputFile)
end function

```

2. Програмна реалізація алгоритму

1. Вихідний код

BinFile.py

```

import os
class BinFile:
    def __init__(self, name, mode="r+b"):
        try:
            myFile = open(name, mode)
            self.myFile = myFile
            self.name = name
            myFile.seek(0,2)
            self.size = myFile.tell() // 4
            self.ok = True
            self.closed = False
        except IOError:
            self.ok = False

    def getInt32(self, pos):
        self.myFile.seek(pos*4)
        arr = self.myFile.read(4)
        return ((arr[3]*256+arr[2])*256+arr[1])*256+arr[0]

    def getInts32(self, pos, amount):
        self.myFile.seek(pos*4)
        arr = self.myFile.read(amount*4)
        outputs = []
        for i in range(0, len(arr), 4):
            outputs.append(((arr[i+3]*256+arr[i+2])*256+arr[i+1])*256+arr[i])
        return outputs

    def setInt32(self, pos, value):
        self.myFile.seek(pos*4)
        arr = []
        for i in range(4):
            arr.append(value % 256)
            value //= 256
        self.myFile.write(bytearray(arr))
        if(pos+1 > self.size):
            self.size = pos+1

    def setInts32(self, pos, values):
        self.myFile.seek(pos*4)
        arr = []
        for j in range(len(values)):

```



```
        value = values[j]
        for i in range(4):
            arr.append(value % 256)
            value //= 256
        self.myFile.write(bytearray(arr))

    def close(self):
        if(not self.closed):
            self.myFile.close()
            self.closed = True

    def delete(self):
        if(not self.closed):
            self.myFile.close()
            self.closed = True
        os.remove(self.name)
```

Sorter.py

```
#!/bin/env python
from BinFile import BinFile
import random
import math
import time
class Run:
    def __init__(self, start, length, binFile):
        self.start = start
        self.length = length
        self.pos = 0
        self.binFile = binFile
        if length > 0:
            self.hasMore = True
            self.value = binFile.getInt32(start)
        else:
            self.hasMore = False
            self.value = 0

    def next(self):
        self.pos += 1
        if(self.pos < self.length):
            self.value = self.binFile.getInt32(self.start + self.pos)
        else:
            self.hasMore = False

def sort(fileCount, runSize, visualize):
    startTime = time.time()
    binf = BinFile("numbers", "r+b")

    # Створення файлів
    inputFiles = []
    inputFileRuns = []
    outputFile = BinFile("workingFile0", "w+b")
    outputFileRuns = []
    for i in range(1, fileCount):
        inputFiles.append(BinFile("workingFile"+str(i), "w+b"))
        inputFileRuns.append([])
    print("Created "+str(fileCount)+" temporary files.\n")

    # Розрахунок розподілу
    distributions = [1] * len(inputFiles)
    currentSize = len(inputFiles)
    targetSize = math.ceil(binf.size / runSize)
    while currentSize < targetSize:
        print(distributions)
        maxVal = -1
        maxI = 0
        for i in range(len(distributions)): # Знайти найбільше
            if(distributions[i] > maxVal):
                maxVal = distributions[i]
                maxI = i
        for i in range(len(distributions)): # Додати до всіх інших
            if(i != maxI):
                distributions[i] += maxVal
                currentSize += maxVal

    print("-"*60)
    print("Розподіл: "+str(distributions))
    print("Всього серій: "+str(currentSize))
    print("Справжніх серій: "+str(targetSize))
    print("Пустих серій: "+str(currentSize - targetSize)+"\n")
    totalRuns = currentSize

    # Розбиття вхідного файлу в серії та розкладання їх по файлам
    writeFilePos = [0] * len(inputFiles)
    writeToFiles = [x for x in range(len(inputFiles))]
    distr = [distributions[x] for x in range(len(inputFiles))]
    writeToFile = 0
    writeToFileIdx = 0
    readPos = 0
    runsWrittenSoFar = 0
    while readPos < binf.size:
        ints = binf.getInts32(readPos, runSize)
        ints.sort()
        inputFiles[writeToFile].setInts32(writeFilePos[writeToFile], ints)
        inputFileRuns[writeToFile].append(Run(writeFilePos[writeToFile], len(ints),
inputFiles[writeToFile]))
```

```

        if(visualize):
            print("запис "+str(len(ints))+" значень в файл
"+inputFiles[writeToFile].name+" "+str(distr))
            distr[writeToFileIdx] -= 1
            if(distr[writeToFileIdx] < 1):
                writeToFileIdx.pop(writeToFileIdx)
                distr.pop(writeToFileIdx)
                writeToFileIdx -= 1
            readPos += runSize
            writeToFilePos[writeToFile] += runSize
            writeToFileIdx = (writeToFileIdx + 1) % len(writeToFileFiles)
            writeToFile = writeToFileFiles[writeToFileIdx]
            runsWrittenSoFar += 1
        while runsWrittenSoFar < totalRuns:
            inputFileRuns[writeToFile].append(Run(writeToFilePos[writeToFile], 0,
inputFiles[writeToFile]))
            if(visualize):
                print("запис пустої серії в файл "+inputFiles[writeToFile].name+"
"+str(distr))
            distr[writeToFileIdx] -= 1
            if(distr[writeToFileIdx] < 1):
                writeToFileFiles.pop(writeToFileIdx)
                distr.pop(writeToFileIdx)
                writeToFileIdx -= 1
            if(len(writeToFileFiles) > 0): # Запобігти вилітам від %0 та читання поза межами
                writeToFileIdx = (writeToFileIdx + 1) % len(writeToFileFiles)
                writeToFile = writeToFileFiles[writeToFileIdx]
            runsWrittenSoFar += 1

# Злиття
keepGoing = True
outputPos = 0
while keepGoing:
    hasMore = True
    outputStart = outputPos
    while hasMore:
        minI = -1
        minVal = 9999999
        for i in range(len(inputFiles)):
            if(len(inputFileRuns[i]) > 0 and inputFileRuns[i][0].hasMore):
                value = inputFileRuns[i][0].value
                #print("Перевіряємо значення: "+str(value))
                if(value < minVal):
                    minVal = value
                    minI = i

        if(minI > -1):
            #print("Мінімум: "+str(minVal))
            outputFile.setInt32(outputPos, minVal)
            outputPos += 1
            inputFileRuns[minI][0].next()
            hasMore = True

        else:
            hasMore = False
    #print("Немає нічого")
    outputLength = outputPos - outputStart
    outputFileRuns.append(Run(outputStart, outputLength, outputFile))
    switchTo = -1
    switchToCount = 0
    for i in range(len(inputFiles)):
        if(len(inputFileRuns[i]) > 0):
            inputFileRuns[i].pop(0)
            if(len(inputFileRuns[i]) == 0):
                switchTo = i
                switchToCount += 1
    if(switchToCount == len(inputFiles)): # Всі файли пусті - вийти
        keepGoing = False
    elif(switchTo > -1): # Один з файлів пустий - переключитись на нього
        if(visualize):
            print("В файлі "+inputFiles[switchTo].name+" скінчилися серії.
Робимо його вихідним")
        outputFile, inputFiles[switchTo] = inputFiles[switchTo], outputFile
        outputFileRuns, inputFileRuns[switchTo] = inputFileRuns[switchTo],
outputFileRuns
        outputPos = 0
    if(visualize):
        print("Розподіл серій:")
        for i in range(len(inputFileRuns)):
            print(" "+inputFiles[i].name+"> "+("="*len(inputFileRuns[i])))
            print(" "+outputFile.name+"< "+("="*len(outputFileRuns)))
        for i in range(outputFileRuns[0].length):

```

```

        binf.setInt32(i, outputFile.getInt32(i))
    for inf in inputFiles:
        inf.delete()
    outputFile.delete()
    print("На виконання пішло "+str(time.time() - startTime)+" секунд")

def main():
    fileCount = int(input("Кількість файлів: "))
    runSize = int(input("Кількість чисел в серії: "))
    visualize = (input("Показувати сортування? (т/н) ") == "т")
    sort(fileCount, runSize, visualize)

if __name__ == "__main__":
    main()

```

2. Виконання

```

vadik@vadik:~> cd polymerge/
vadik@vadik:~/polymerge> ./GenRandom.py
Створити файл зі скількома випадковими числами?
1000
Готово
vadik@vadik:~/polymerge> ./FileViewer.py
Файл має 1000 чисел.
З якої позиції продивитись файл?
0
Скільки чисел продивитись?
10
===== Числа =====
0: 8058
1: 7427
2: 4341
3: 3087
4: 9282
5: 1197
6: 8002
7: 7559
8: 9311
9: 9900

```

```

vadik@vadik:~/polymerge> ./Sorter
bash: ./Sorter: Нет такого файла или каталога
vadik@vadik:~/polymerge> ./Sorter.py
Кількість файлів: 4
Кількість чисел в серії: 10
Показувати сортування? (т/н) т
Created 4 temporary files.

```

```

[1, 1, 1]
[1, 2, 2]
[3, 2, 4]
[7, 6, 4]
[7, 13, 11]
[20, 13, 24]

```

```

-----
Розподіл: [44, 37, 24]
Всього серій: 105
Справжніх серій: 100
Пустих серій: 5

```

```

запис 10 значень в файл workingFile1 [44, 37, 24]
запис 10 значень в файл workingFile2 [43, 37, 24]
запис 10 значень в файл workingFile3 [43, 36, 24]
запис 10 значень в файл workingFile1 [43, 36, 23]
запис 10 значень в файл workingFile2 [42, 36, 23]
запис 10 значень в файл workingFile3 [42, 35, 23]
запис 10 значень в файл workingFile1 [42, 35, 22]
запис 10 значень в файл workingFile2 [41, 35, 22]
запис 10 значень в файл workingFile3 [41, 34, 22]
запис 10 значень в файл workingFile1 [41, 34, 21]
запис 10 значень в файл workingFile2 [40, 34, 21]
запис 10 значень в файл workingFile3 [40, 33, 21]
запис 10 значень в файл workingFile1 [40, 33, 20]
запис 10 значень в файл workingFile2 [39, 33, 20]
запис 10 значень в файл workingFile3 [39, 32, 20]
запис 10 значень в файл workingFile1 [39, 32, 19]
запис 10 значень в файл workingFile2 [38, 32, 19]
запис 10 значень в файл workingFile3 [38, 31, 19]
запис 10 значень в файл workingFile1 [38, 31, 18]
запис 10 значень в файл workingFile2 [37, 31, 18]
запис 10 значень в файл workingFile3 [37, 30, 18]
запис 10 значень в файл workingFile1 [37, 30, 17]
запис 10 значень в файл workingFile2 [36, 30, 17]
запис 10 значень в файл workingFile3 [36, 29, 17]
запис 10 значень в файл workingFile1 [36, 29, 16]
запис 10 значень в файл workingFile2 [35, 29, 16]
запис 10 значень в файл workingFile3 [35, 28, 16]
запис 10 значень в файл workingFile1 [35, 28, 15]
запис 10 значень в файл workingFile2 [34, 28, 15]
запис 10 значень в файл workingFile3 [34, 27, 15]
запис 10 значень в файл workingFile1 [34, 27, 14]
запис 10 значень в файл workingFile2 [33, 27, 14]
запис 10 значень в файл workingFile3 [33, 26, 14]
запис 10 значень в файл workingFile1 [33, 26, 13]

```

[illegible]

[illegible]

```
workingFile2> =====
workingFile0> =====
workingFile3< =====
Розподіл серій:
workingFile1> =====
workingFile2> =====
workingFile0> =====
workingFile3< =====
Розподіл серій:
workingFile1> =====
workingFile2> =====
workingFile0> =====
workingFile3< =====
Розподіл серій:
workingFile1> =====
workingFile2> =====
workingFile0> =====
workingFile3< =====
Розподіл серій:
workingFile1> =====
workingFile2> ==
workingFile0> =====
workingFile3< =====
Розподіл серій:
workingFile1> =====
workingFile2> =
workingFile0> =====
workingFile3< =====
В файлі workingFile2 скінчилися серії. Робимо його вихідним
Розподіл серій:
workingFile1> =====
workingFile3> =====
workingFile0> =====
workingFile2< =====
Розподіл серій:
workingFile1> =====
workingFile3> =====
workingFile0> =====
workingFile2< =
Розподіл серій:
workingFile1> =====
workingFile3> =====
workingFile0> =====
workingFile2< ==
Розподіл серій:
workingFile1> =====
workingFile3> =====
workingFile0> =====
workingFile2< ==
Розподіл серій:
workingFile1> ==
workingFile3> =====
workingFile0> =====
workingFile2< =====
Розподіл серій:
workingFile1> ==
workingFile3> =====
workingFile0> =====
workingFile2< =====
Розподіл серій:
workingFile1> =
workingFile3> =====
workingFile0> =====
workingFile2< =====
В файлі workingFile1 скінчилися серії. Робимо його вихідним
Розподіл серій:
workingFile2> =====
workingFile3> =====
workingFile0> =====
workingFile1< =====
Розподіл серій:
workingFile2> =====
workingFile3> =====
workingFile0> ==
workingFile1< =
Розподіл серій:
workingFile2> =====
workingFile3> =====
workingFile0> ==
workingFile1< ==
Розподіл серій:
workingFile2> =====
workingFile3> ==
workingFile0> =
workingFile1< ==
Розподіл серій:
workingFile2> ==
workingFile3> =
workingFile0> ==
workingFile1> ==
workingFile0< =
В файлі workingFile0 скінчилися серії. Робимо його вихідним
Розподіл серій:
workingFile2> ==
workingFile3> ==
workingFile1> =====
workingFile0< =====
Розподіл серій:
workingFile2> ==
workingFile3> =
workingFile1> ==
workingFile0< =
В файлі workingFile3 скінчилися серії. Робимо його вихідним
```

```
Розподіл серій:
workingFile2> =
workingFile0> ==
workingFile1> ==
workingFile3<
```

В файлі workingFile2 скінчилися серії. Робимо його вихідним

```
Розподіл серій:
workingFile3> =
workingFile0> =
workingFile1> =
workingFile2<
```

```
Розподіл серій:
workingFile3>
workingFile0>
workingFile1>
workingFile2< =
```

На виконання пішло 0.024651527404785156 секунд

```
vadik@vadik:~/polymerge> ./FileViewer.py
```

Файл має 1000 чисел.

З якої позиції продивитись файл?

0

Скільки чисел продивитись?

100

==== Числа =====

```
0: 9
1: 29
2: 33
3: 43
4: 53
5: 66
6: 84
7: 101
8: 120
9: 140
10: 161
11: 178
12: 178
13: 208
14: 256
15: 257
16: 266
17: 267
18: 303
19: 311
20: 321
21: 321
22: 321
23: 338
24: 340
25: 351
26: 360
27: 360
28: 366
29: 367
30: 370
31: 400
32: 404
33: 428
34: 433
35: 437
36: 441
37: 447
38: 447
39: 448
40: 465
41: 485
42: 493
43: 519
44: 525
45: 552
46: 552
47: 568
48: 573
49: 573
50: 587
51: 597
52: 622
53: 624
54: 633
55: 635
56: 645
57: 648
58: 649
59: 667
60: 676
61: 678
62: 678
63: 683
64: 697
65: 701
66: 702
67: 706
68: 706
69: 721
70: 726
71: 736
72: 768
73: 775
74: 815
75: 817
76: 824
77: 827
78: 830
79: 835
80: 845
81: 877
82: 884
```


83: 891
84: 894
85: 896
86: 906
87: 916
88: 922
89: 937
90: 963
91: 971
92: 990
93: 1006
94: 1019
95: 1023
96: 1031
97: 1038
98: 1053
99: 1056

4. ВИСНОВОК

При виконанні лабораторної роботи було ознайомлено з різними алгоритмами зовнішнього сортування, був створений псевдокод, імплементація на мові програмування Python базового та модифікованого алгоритмів та була порівняна швидкість виконання обох алгоритмів. На сортування 10 МБ (2500000 чисел) базовим алгоритмом уходить 13 хвилин 41 секунда. Для підвищення швидкості роботи була створена модифікація алгоритму. Основна ідея модифікації заключається в розбитті вхідного файлу на секції певного розміру та сортування чисел усередині їх за допомогою внутрішнього сортування. На сортування 10 МБ (2500000 чисел) модифікованим алгоритмом з розміром секцій 100 чисел уходить 1 хвилина 45 секунд. На сортування 100 МБ (25000000 чисел) модифікованим алгоритмом з розміром секцій 10000 чисел уходить 13 хвилин 11 секунд. Модифікація дає сильний приріст швидкості використовуючи оперативну пам'ять. Навіть на старих системах зазвичай є хоч трохи вільної оперативної пам'яті, тому базову версію алгоритму практично ніхто не використовує.

5. 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%.