**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)

*Піонтківський Віталій*

**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** |
| 20 | Багатофазне сортування |

# Виконання

## Псевдокод алгоритму

START

FUNCTION merge(tape\_1\_path: str, tape\_2\_path: str, output\_path: str, tape\_start\_position=0):

OPEN(tape\_1\_path, "r+") as a:

OPEN(tape\_2\_path, "r+") as b:

OPEN(output\_path, "w") as o:

IF tape\_start\_position: b.seek(tape\_start\_position, 0)

tape\_1\_pos, tape\_2\_pos = a.tell(), tape\_start\_position

tape\_1\_curr, tape\_2\_curr = PolyphaseSort.read\_number(a), PolyphaseSort.read\_number(b)

END IF

IF NOT tape\_1\_curr or NOT tape\_2\_curr:

print("Source is sorted, file: "+ tape\_1\_path)

RETURN 1

ELSE:

IF tape\_1\_curr != "N" and tape\_2\_curr != "N":

tape\_1\_curr, tape\_2\_curr = int(tape\_1\_curr), int(tape\_2\_curr)

tape\_1\_prev, tape\_2\_prev = tape\_1\_curr, tape\_2\_curr

END IF

WHILE tape\_1\_curr and tape\_2\_curr:

WHILE tape\_1\_curr != "N" and tape\_2\_curr != "N" \

and tape\_1\_curr >= tape\_1\_prev and tape\_2\_curr >= tape\_2\_prev:

IF tape\_1\_curr < tape\_2\_curr:

o.write(str(tape\_1\_curr) + " ")

tape\_1\_pos = a.tell()

tape\_1\_prev = tape\_1\_curr

tape\_1\_curr = PolyphaseSort.read\_number(a)

ELSE:

o.write(str(tape\_2\_curr) + " ")

tape\_2\_pos = b.tell()

tape\_2\_prev = tape\_2\_curr

tape\_2\_curr = PolyphaseSort.read\_number(b)

IF NOT tape\_1\_curr:

PolyphaseSort.adjust\_pointer(b, tape\_2\_pos)

PolyphaseSort.copy\_rest(b, o)

a.truncate(0)

position = b.tell()

a.close()

b.close()

o.close()

PolyphaseSort.merge(output\_path, tape\_2\_path, tape\_1\_path, position)

RETURN 1

ELSE IF NOT tape\_2\_curr:

PolyphaseSort.adjust\_pointer(a, tape\_1\_pos)

PolyphaseSort.copy\_rest(a, o)

b.truncate(0)

position = a.tell()

a.close()

b.close()

o.close()

PolyphaseSort.merge(output\_path, tape\_1\_path, tape\_2\_path, position)

RETURN 1

ELSE IF tape\_1\_curr == "N":

PolyphaseSort.adjust\_pointer(b, tape\_2\_pos)

PolyphaseSort.copy\_rest(b, o)

tape\_2\_pos = b.tell()

tape\_2\_curr = PolyphaseSort.read\_number(b)

break

ELSE IF tape\_2\_curr == "N":

PolyphaseSort.adjust\_pointer(a, tape\_1\_pos)

PolyphaseSort.copy\_rest(a, o)

tape\_1\_pos = a.tell()

tape\_1\_curr = PolyphaseSort.read\_number(a)

break

ELSE:

tape\_1\_curr, tape\_2\_curr = int(tape\_1\_curr), int(tape\_2\_curr)

IF tape\_1\_curr < tape\_1\_prev:

PolyphaseSort.adjust\_pointer(b, tape\_2\_pos)

PolyphaseSort.copy\_rest(b, o)

tape\_2\_pos = b.tell()

tape\_2\_curr = PolyphaseSort.read\_number(b)

tape\_1\_prev = tape\_1\_curr

IF NOT tape\_2\_curr:

b.truncate(0)

a.close()

b.close()

o.close()

PolyphaseSort.merge(output\_path, tape\_1\_path, tape\_2\_path, tape\_1\_pos)

RETURN 1

ELSE IF tape\_2\_curr == "N":

PolyphaseSort.adjust\_pointer(a, tape\_1\_pos)

PolyphaseSort.copy\_rest(a, o)

tape\_1\_pos = a.tell()

tape\_1\_curr = PolyphaseSort.read\_number(a)

break

ELSE:

tape\_2\_curr = int(tape\_2\_curr)

tape\_2\_prev = tape\_2\_curr

END IF

ELSE IF tape\_2\_curr < tape\_2\_prev:

PolyphaseSort.adjust\_pointer(a, tape\_1\_pos)

PolyphaseSort.copy\_rest(a, o)

tape\_1\_pos = a.tell()

tape\_1\_curr = PolyphaseSort.read\_number(a)

tape\_2\_prev = tape\_2\_curr

IF NOT tape\_1\_curr:

a.truncate(0)

a.close()

b.close()

o.close()

PolyphaseSort.merge(output\_path, tape\_2\_path, tape\_1\_path, tape\_2\_pos)

RETURN 1

ELSE IF tape\_1\_curr == "N":

PolyphaseSort.adjust\_pointer(b, tape\_2\_pos)

PolyphaseSort.copy\_rest(b, o)

tape\_2\_pos = b.tell()

tape\_2\_curr = PolyphaseSort.read\_number(b)

break

ELSE:

tape\_1\_curr = int(tape\_1\_curr)

tape\_1\_prev = tape\_1\_curr

END IF

END IF

END IF

END WHILE

WHILE tape\_1\_curr == "N" or tape\_2\_curr == "N":

IF tape\_1\_curr == "N" and tape\_2\_curr:

PolyphaseSort.adjust\_pointer(b, tape\_2\_pos)

PolyphaseSort.copy\_rest(b, o)

tape\_1\_pos = a.tell()

tape\_2\_pos = b.tell()

tape\_1\_curr = PolyphaseSort.read\_number(a)

tape\_2\_curr = PolyphaseSort.read\_number(b)

ELSE IF tape\_2\_curr == "N" and tape\_1\_curr:

PolyphaseSort.adjust\_pointer(a, tape\_1\_pos)

PolyphaseSort.copy\_rest(a, o)

tape\_1\_pos = a.tell()

tape\_2\_pos = b.tell()

tape\_1\_curr = PolyphaseSort.read\_number(a)

tape\_2\_curr = PolyphaseSort.read\_number(b)

ELSE IF NOT tape\_2\_curr:

b.truncate(0)

a.close()

b.close()

o.close()

PolyphaseSort.merge(output\_path, tape\_1\_path, tape\_2\_path, tape\_1\_pos)

RETURN 1

ELSE IF NOT tape\_1\_curr:

a.truncate(0)

a.close()

b.close()

o.close()

PolyphaseSort.merge(output\_path, tape\_2\_path, tape\_1\_path, tape\_2\_pos)

RETURN 1

END IF

END WHILE

END WHILE

IF NOT tape\_1\_curr:

a.truncate(0)

a.close()

b.close()

o.close()

PolyphaseSort.merge(output\_path, tape\_2\_path, tape\_1\_path, tape\_2\_pos)

RETURN 1

IF NOT tape\_2\_curr:

b.truncate(0)

a.close()

b.close()

o.close()

PolyphaseSort.merge(output\_path, tape\_1\_path, tape\_2\_path, tape\_1\_pos)

RETURN 1

END IF

END IF

END

## Програмна реалізація алгоритму

### Вихідний код

from typing import TextIO

class PolyphaseSort:

    #зчитування одного числа або псевдо послідовності з файлу

    @staticmethod

    def read\_number(file: TextIO):

        r = file.read(1)

        curr = file.read(1)

        end\_symbols = [" ", ""]

        while curr not in end\_symbols:

            r += curr

            curr = file.read(1)

        return r.strip()

    #зміна позицію покажчика читання файла

    @staticmethod

    def adjust\_pointer(file: TextIO, position: int):

        file.seek(position, 0)

    #функція, яка розбиває початковий файл на два нових

    @staticmethod

    def initial\_distribution(input\_path: str, tape\_1\_path: str, tape\_2\_path: str):

        flag = 0

        with open(input\_path, "r") as i:

            with open(tape\_1\_path, "w") as a:

                with open(tape\_2\_path, "w") as b:

                    current\_str = PolyphaseSort.read\_number(i)

                    current = int(current\_str)

                    # previous = current

                    flag = 0

                    while current\_str!="":

                        array =[]

                        n = 0

                        while current\_str!="" and n<=17500000:

                            current = int(current\_str)

                            array.append(current)

                            n+=1

                            current\_str = PolyphaseSort.read\_number(i)

                        array.sort()

                        if not flag:

                            a.write(" ".join(str(i) for i in array))

                            a.write(" ")

                            flag = 1

                            print("Written to the first")

                        else:

                            b.write(" ".join(str(i) for i in array))

                            b.write(" ")

                            flag = 0

                            print("Written to the second")

                    # while current\_str:

                    #     current = int(current\_str)

                    #     if previous <= current:

                    #         if flag:

                    #             b.write(current\_str + " ")

                    #         else:

                    #             a.write(current\_str + " ")

                    #     else:

                    #         if flag:

                    #             flag = 0

                    #             a.write(current\_str + " ")

                    #         else:

                    #             flag = 1

                    #             b.write(current\_str + " ")

                    #     current\_str, previous = PolyphaseSort.read\_number(i), current

    #функція, що рахує кількість послідовностей у файлі

    @staticmethod

    def count\_sequences(file\_path: str):

        count = 0

        with open(file\_path, "r") as f:

            curr = PolyphaseSort.read\_number(f)

            prev = curr

            if curr:

                count = 1

            while curr:

                if curr == "N":

                    count += 1

                elif int(curr) < int(prev):

                    count += 1

                prev = curr

                curr = PolyphaseSort.read\_number(f)

        return count

    #повертає найближче більше число фібоначі та два попередніх

    @staticmethod

    def find\_closest\_fibonacci(number: int):

        fibonacci = [0, 1]

        index = 1

        while number > fibonacci[index]:

            index += 1

            fibonacci.append(fibonacci[index-2] + fibonacci[index-1])

        return [fibonacci[index-2], fibonacci[index-1], fibonacci[index]]

    #допоміжна функція для наступної

    @staticmethod

    def find\_min\_index(array: list[int]):

        if array[0] > array[1]:

            return [1, 0]

        else:

            return [0, 1]

    #функція, яка додає псевдопослідовності, щоб кількість послідовностей у файлах була числом фібоначі

    #причому кіскослі послідовностей мають бути числами фібоначі, що йдуть підряд

    @staticmethod

    def optimize\_to\_fibonacci():

        number\_seq = [PolyphaseSort.count\_sequences("0.txt"), PolyphaseSort.count\_sequences("1.txt")]

        if not number\_seq[0] or not number\_seq[1]:

            return False

        closest\_fibonacci = PolyphaseSort.find\_closest\_fibonacci(number\_seq[0] + number\_seq[1])

        print("Closest fibonacci num: " + str(closest\_fibonacci[2]))

        print("Sequences in first tape: "+ str(closest\_fibonacci[0]) + "\nSequences in second tape "+ str(closest\_fibonacci[1]))

        print("Number Of Sequences in Tapes: " + str(number\_seq))

        minimum = min(number\_seq[0], number\_seq[1])

        minimum\_index\_sorted = PolyphaseSort.find\_min\_index(number\_seq)

        print("Minimum number of Sequences in Tape: " + str(minimum))

        if closest\_fibonacci[0] < minimum:

            closest\_fibonacci = PolyphaseSort.find\_closest\_fibonacci(closest\_fibonacci[2]+1)

            print("New Fibonacci: " + str(closest\_fibonacci))

        with open(str(minimum\_index\_sorted[0]) + ".txt", "a") as a:

            with open(str(minimum\_index\_sorted[1]) + ".txt", "a") as b:

                while number\_seq[minimum\_index\_sorted[0]] < closest\_fibonacci[0]:

                    a.write("N ")

                    number\_seq[minimum\_index\_sorted[0]] += 1

                while number\_seq[minimum\_index\_sorted[1]] < closest\_fibonacci[1]:

                    b.write("N ")

                    number\_seq[minimum\_index\_sorted[1]] += 1

        return True

    #основна функція, багатофазне сортування

    @staticmethod

    def merge(tape\_1\_path: str, tape\_2\_path: str, output\_path: str, tape\_start\_position=0):

        print("New merge")

        with open(tape\_1\_path, "r+") as a:

            with open(tape\_2\_path, "r+") as b:

                with open(output\_path, "w") as o:

                    if tape\_start\_position: b.seek(tape\_start\_position, 0)

                    tape\_1\_pos, tape\_2\_pos = a.tell(), tape\_start\_position

                    tape\_1\_curr, tape\_2\_curr = PolyphaseSort.read\_number(a), PolyphaseSort.read\_number(b)

                    if not tape\_1\_curr or not tape\_2\_curr:

                        print("Source is sorted, file: "+ tape\_1\_path)

                        return 1

                    else:

                        if tape\_1\_curr != "N" and tape\_2\_curr != "N":

                            tape\_1\_curr, tape\_2\_curr = int(tape\_1\_curr), int(tape\_2\_curr)

                            tape\_1\_prev, tape\_2\_prev = tape\_1\_curr, tape\_2\_curr

                        while tape\_1\_curr and tape\_2\_curr:

                            while tape\_1\_curr != "N" and tape\_2\_curr != "N" \

                                    and tape\_1\_curr >= tape\_1\_prev and tape\_2\_curr >= tape\_2\_prev:

                                if tape\_1\_curr < tape\_2\_curr:

                                    o.write(str(tape\_1\_curr) + " ")

                                    tape\_1\_pos = a.tell()

                                    tape\_1\_prev = tape\_1\_curr

                                    tape\_1\_curr = PolyphaseSort.read\_number(a)

                                else:

                                    o.write(str(tape\_2\_curr) + " ")

                                    tape\_2\_pos = b.tell()

                                    tape\_2\_prev = tape\_2\_curr

                                    tape\_2\_curr = PolyphaseSort.read\_number(b)

                                if not tape\_1\_curr:

                                    PolyphaseSort.adjust\_pointer(b, tape\_2\_pos)

                                    PolyphaseSort.copy\_rest(b, o)

                                    a.truncate(0)

                                    position = b.tell()

                                    a.close()

                                    b.close()

                                    o.close()

                                    PolyphaseSort.merge(output\_path, tape\_2\_path, tape\_1\_path, position)

                                    return 1

                                elif not tape\_2\_curr:

                                    PolyphaseSort.adjust\_pointer(a, tape\_1\_pos)

                                    PolyphaseSort.copy\_rest(a, o)

                                    b.truncate(0)

                                    position = a.tell()

                                    a.close()

                                    b.close()

                                    o.close()

                                    PolyphaseSort.merge(output\_path, tape\_1\_path, tape\_2\_path, position)

                                    return 1

                                elif tape\_1\_curr == "N":

                                    PolyphaseSort.adjust\_pointer(b, tape\_2\_pos)

                                    PolyphaseSort.copy\_rest(b, o)

                                    tape\_2\_pos = b.tell()

                                    tape\_2\_curr = PolyphaseSort.read\_number(b)

                                    break

                                elif tape\_2\_curr == "N":

                                    PolyphaseSort.adjust\_pointer(a, tape\_1\_pos)

                                    PolyphaseSort.copy\_rest(a, o)

                                    tape\_1\_pos = a.tell()

                                    tape\_1\_curr = PolyphaseSort.read\_number(a)

                                    break

                                else:

                                    tape\_1\_curr, tape\_2\_curr = int(tape\_1\_curr), int(tape\_2\_curr)

                                    if tape\_1\_curr < tape\_1\_prev:

                                        PolyphaseSort.adjust\_pointer(b, tape\_2\_pos)

                                        PolyphaseSort.copy\_rest(b, o)

                                        tape\_2\_pos = b.tell()

                                        tape\_2\_curr = PolyphaseSort.read\_number(b)

                                        tape\_1\_prev = tape\_1\_curr

                                        if not tape\_2\_curr:

                                            b.truncate(0)

                                            a.close()

                                            b.close()

                                            o.close()

                                            PolyphaseSort.merge(output\_path, tape\_1\_path, tape\_2\_path, tape\_1\_pos)

                                            return 1

                                        elif tape\_2\_curr == "N":

                                            PolyphaseSort.adjust\_pointer(a, tape\_1\_pos)

                                            PolyphaseSort.copy\_rest(a, o)

                                            tape\_1\_pos = a.tell()

                                            tape\_1\_curr = PolyphaseSort.read\_number(a)

                                            break

                                        else:

                                            tape\_2\_curr = int(tape\_2\_curr)

                                            tape\_2\_prev = tape\_2\_curr

                                    elif tape\_2\_curr < tape\_2\_prev:

                                        PolyphaseSort.adjust\_pointer(a, tape\_1\_pos)

                                        PolyphaseSort.copy\_rest(a, o)

                                        tape\_1\_pos = a.tell()

                                        tape\_1\_curr = PolyphaseSort.read\_number(a)

                                        tape\_2\_prev = tape\_2\_curr

                                        if not tape\_1\_curr:

                                            a.truncate(0)

                                            a.close()

                                            b.close()

                                            o.close()

                                            PolyphaseSort.merge(output\_path, tape\_2\_path, tape\_1\_path, tape\_2\_pos)

                                            return 1

                                        elif tape\_1\_curr == "N":

                                            PolyphaseSort.adjust\_pointer(b, tape\_2\_pos)

                                            PolyphaseSort.copy\_rest(b, o)

                                            tape\_2\_pos = b.tell()

                                            tape\_2\_curr = PolyphaseSort.read\_number(b)

                                            break

                                        else:

                                            tape\_1\_curr = int(tape\_1\_curr)

                                            tape\_1\_prev = tape\_1\_curr

                            while tape\_1\_curr == "N" or tape\_2\_curr == "N":

                                if tape\_1\_curr == "N" and tape\_2\_curr:

                                    PolyphaseSort.adjust\_pointer(b, tape\_2\_pos)

                                    PolyphaseSort.copy\_rest(b, o)

                                    tape\_1\_pos = a.tell()

                                    tape\_2\_pos = b.tell()

                                    tape\_1\_curr = PolyphaseSort.read\_number(a)

                                    tape\_2\_curr = PolyphaseSort.read\_number(b)

                                elif tape\_2\_curr == "N" and tape\_1\_curr:

                                    PolyphaseSort.adjust\_pointer(a, tape\_1\_pos)

                                    PolyphaseSort.copy\_rest(a, o)

                                    tape\_1\_pos = a.tell()

                                    tape\_2\_pos = b.tell()

                                    tape\_1\_curr = PolyphaseSort.read\_number(a)

                                    tape\_2\_curr = PolyphaseSort.read\_number(b)

                                elif not tape\_2\_curr:

                                    b.truncate(0)

                                    a.close()

                                    b.close()

                                    o.close()

                                    PolyphaseSort.merge(output\_path, tape\_1\_path, tape\_2\_path, tape\_1\_pos)

                                    return 1

                                elif not tape\_1\_curr:

                                    a.truncate(0)

                                    a.close()

                                    b.close()

                                    o.close()

                                    PolyphaseSort.merge(output\_path, tape\_2\_path, tape\_1\_path, tape\_2\_pos)

                                    return 1

                        if not tape\_1\_curr:

                            a.truncate(0)

                            a.close()

                            b.close()

                            o.close()

                            PolyphaseSort.merge(output\_path, tape\_2\_path, tape\_1\_path, tape\_2\_pos)

                            return 1

                        if not tape\_2\_curr:

                            b.truncate(0)

                            a.close()

                            b.close()

                            o.close()

                            PolyphaseSort.merge(output\_path, tape\_1\_path, tape\_2\_path, tape\_1\_pos)

                            return 1

    #функція, яка переміщує остаток одного файлу з результуючий

    @staticmethod

    def copy\_rest(tape: TextIO, output: TextIO):

        print("Copy rest started")

        pos = tape.tell()

        curr = PolyphaseSort.read\_number(tape)

        prev = curr

        if curr == '':

            return

        if curr == "N":

            output.write(curr + " ")

        else:

            while curr and curr != "N" and int(curr) >= int(prev):

                #print(curr)

                output.write(curr + " ")

                prev = curr

                pos = tape.tell()

                curr = PolyphaseSort.read\_number(tape)

            PolyphaseSort.adjust\_pointer(tape, pos)

        print("Copy rest ended")

import time

import random

from typing import TextIO

from PolyphaseSort import \*

if \_\_name\_\_ == "\_\_main\_\_":

    #генерація файлу, який потрібно відсортувати

    # with open("source.txt", "w") as file:

    #     for i in range(0, 175000000\*2):

    #         file.write(str(random.randint(1, 100000)) + " ")

    #         if i%100000==0:

    #             print(i)

    #сортування

    start\_time = time.time()

    print("source.txt: ")

    PolyphaseSort.initial\_distribution("source.txt", "0.txt", "1.txt")

    if PolyphaseSort.optimize\_to\_fibonacci():

        PolyphaseSort.merge("0.txt", "1.txt", "2.txt")

    print("Sorting ended")

    end\_time = time.time()

    print("Total time: " + str(end\_time - start\_time) + " seconds")

    with open("time.txt", "w") as a:

        a.write("Total time: " + str(end\_time - start\_time) + " seconds")

Висновок

При виконанні даної лабораторної роботи було вивчено основні алгоритми зовнішнього сортування та способи їх модифікації, створено програму, що реалізує алгоритм багатофазного сортування. Через особливості роботи алгоритму було розроблено спеціальну функцію, що відслідковує кількість послідовностей (числа у послідовності вже відсортовані) у початкових файлах та додає до файлів псевдо послідовності, щоб кількість послідовностей кожному з файлів була числом Фібоначчі, причому ці числа мають іти підряд в ряду Фібоначчі. До основного алгоритму сортування додано обробку псевдо послідовностей.

Оптимізовано алгоритм зменшенням кількості операцій, що відкривають та взаємодіють з файлом. У початковій версії кількість цих операції прямопропорційно залежала від розмірів файлів, зараз файли відкриваються один раз, на початку роботи алгоритму.

Створена модифікація алгоритму для пришвидшення його роботи. Суттю модифікації є зміна дій зчитування та початкового розподілу даних. В модифікованій програмі дані зчитуються не по числу, а по 100 мегабайтів і сортуються в оперативній пам’яті. Що зменшує кількість послідовностей і кількість звернень до алгоритму сортування. В результаті робота алгоритму пришвидшилась в більш ніж в 10 разів.

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%.