**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)

*Логвиненко В.О. ІП-12*

**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

Open file\_1 and file\_2

Open output\_file to write

If file\_position not 0:

Move\_pointer to spec. position in file\_2

End if

Save current position in file\_1\_pos

Save spec. start position in file\_2\_pos

Read num from file\_1 and file\_2 to file\_1\_curr and file\_2\_curr

If not file\_1\_curr or not file\_2\_curr:

Print(“Array is sorted”)

Return 1

End if

Else:

If file\_1\_curr and file\_2\_curr are not in empty seq:

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

File\_1\_prev, File\_2\_prev = File\_1\_curr,File\_2\_curr

End if

*while* file\_1\_curr *and* file\_2\_curr are not reached the end:  
 *while* file\_1\_curr *and* file\_2\_curr not empty   
 *and continues* file\_1 and file\_2:

*if* file\_1\_curr < file\_2\_curr:  
 write file\_1\_curr to output\_file  
 Save curr pointer position of file\_1 in file\_1\_pos  
 Save curr pointer element of file\_1\_curr in file\_1\_prev  
 Read num from file\_1 to file\_1\_curr

end if  
 *else*:  
 write file\_2\_curr to output\_file  
 Save curr pointer position of file\_2 in file\_2\_pos  
 Save curr pointer element of file\_2\_curr in file\_1\_prev  
 Read num from file\_2 to file\_2\_curr

end else  
 *if* file\_1\_curr is end of file:  
 Move file pointer in file\_2 to file\_2\_pos  
 Addit rest of seq. from file\_2 to ouput  
 clear file\_1  
 save pointer position of file\_2 to position  
 x.close()  
 y.close()  
 f.close()  
 merge\_new(output\_file,file\_2,file\_1,position)  
 *return* 1

end if  
 *elif not* file\_2\_curr:  
 *if* file\_2\_curr is end of file:  
 Move file pointer in file\_1 to file\_1\_pos  
 Addit rest of seq. from file\_1 to ouput  
 clear file\_2  
 save pointer position of file\_1 to position  
 x.close()  
 y.close()  
 f.close()  
 merge\_new(output\_file,file\_1, file\_2,position)  
 *return* 1

end elif

*elif* file\_1\_curr reached empty seq.:  
 Move file pointer in file\_2 to file\_2\_pos  
 Addit rest of seq. from file\_2 to ouput  
 clear file\_1  
 save pointer position of file\_2 to position  
 Read number from file\_2 to file\_2\_curr

*break*

end elif

*elif* file\_2\_curr reached empty seq.:  
 Move file pointer in file\_1 to file\_1\_pos  
 Addit rest of seq. from file\_1 to ouput  
 clear file\_2  
 save pointer position of file\_1 to position  
 Read number from file\_1 to file\_1\_curr

*break*

end elif

*else*:  
 Convert tape\_1\_curr and tape\_2\_curr to int  
 *if* file\_1\_curr < file\_1\_prev:  
 Move file pointer in file\_2 to file\_2\_pos  
 Addit rest of seq. from file\_2 to ouput  
 save pointer position of file\_2 to position

Read number from file\_2 to file\_2\_curr

Save current element of file\_2\_curr to file\_2\_prev  
 *if not* file\_2\_curr:  
 Clear file\_2  
 x.close()  
 y.close()  
 f.close()  
 merge\_new(output\_file,file\_1,file\_2,

file\_1\_pos)  
 *return* 1

end if  
 *elif* file\_2\_curr reach empty seq.:  
 Move file pointer in file\_1 to file\_1\_pos  
 Addit rest of seq. from file\_1 to ouput  
 save pointer position of file\_1 to position

Read number from file\_1 to file\_1\_curr

*Break*

*End elif  
 else*:  
 Convert file\_2\_curr to int  
 file\_2\_prev = file\_2\_curr

*End else*

End if  
 *elif* file\_2\_curr < file\_2\_prev:  
 Move file pointer in file\_1 to file\_1\_pos  
 Addit rest of seq. from file\_1 to ouput  
 save pointer position of file\_1 to position  
 Read number from file\_1 to file\_1\_curr  
 Save current element of file\_1\_curr to file\_1\_prev

*if not* file\_1\_curr:  
 Clear file\_1  
 x.close()  
 y.close()  
 f.close()  
 merge\_new(output\_file,file\_2,

file\_1,file\_2\_pos)  
 *return* 1

end if  
 *elif* file\_1\_curr reach empty seq.:  
 Move file pointer in file\_2 to file\_2\_pos  
 Addit rest of seq. from file\_2 to ouput  
 save pointer position of file\_2 to position  
 Read number from file\_2 to file\_2\_curr  
 Save current element of file\_2\_curr to

*break*

*end elif*

*else*:  
 Convert file\_1\_curr to int  
 file\_1\_prev = file\_1\_curr

*end else*

end elif

end else

end while

end while  
 *while* file\_1\_curr In empty seq or file\_2\_curr In empty seq  
 *if* file\_1\_curr In empty seq *and* file\_2\_curr not reached the end:  
 Move file pointer in file\_2 to file\_2\_pos  
 Addit rest of seq. from file\_2 to ouput

Save file\_1 and file\_2 position to file\_1\_pos,file\_2\_pos  
 Read number from file\_1 and file\_2 to tape\_1\_curr, tape\_2\_curr

End if

*elif* file\_2\_curr In empty seq *and* file\_1\_curr not  
 Move file pointer in file\_1 to file\_1\_pos  
 Addit rest of seq. from file\_1 to ouput

Save file\_1 and file\_2 position to file\_1\_pos,file\_2\_pos  
 Read number from file\_1 and file\_2 to tape\_1\_curr, tape\_2\_curr

End elif  
 *elif file\_2\_curr reached the end of file*:  
 cleare file\_2  
 x.close()  
 y.close()  
 f.close()  
 merge\_new(output\_file,file\_1,file\_2,file\_1\_pos)  
 *return* 1

End elif  
 *elif not* file\_1\_curr:  
 cleare file\_1  
 x.close()  
 y.close()  
 f.close()  
 merge\_new(output\_file,file\_2,file\_1,file\_2\_pos)  
 *return* 1

End elif

End while

End while   
*if file\_1\_curr reached the end*:  
 clear file\_1  
 x.close()  
 y.close()  
 f.close()  
 merge\_new(output\_file,file\_2,file\_1,file\_2\_pos)  
 *return* 1

End if  
*if file\_2\_curr reached the end*::  
 clear file\_2  
 x.close()  
 y.close()  
 f.close()  
 merge\_new(output\_file,file\_1,file\_2,file\_1\_pos)  
 *return* 1

End if

## Програмна реалізація алгоритму

### Вихідний код

Polyphase\_Sort.py

*from* typing *import* TextIO  
*import* time  
*class* Polyphase\_Sort:  
 @staticmethod  
 *def* read\_num(file:TextIO):  
 f = file.read(1)  
 current = file.read(1)  
 end\_symb = [" ",""]  
 *while* current *not in* end\_symb:  
 f += current  
 current = file.read(1)  
 *return* f.strip()  
  
 @staticmethod  
 *def* move\_pointer(file:TextIO,position: *int*):  
 file.seek(position,0)  
  
 @staticmethod  
 *def* modified\_distribution(input\_file:*str* , file\_1: *str*,file\_2:*str*, count:*int* ):  
 flag = 0  
 *with open*(input\_file, "r") *as* i:  
 *with open*(file\_1,"w") *as* x:  
 *with open*(file\_2,"w") *as* y:  
 value = 1  
 *while* value:  
 arr = []  
 counter = count  
 *while* value *and* counter:  
 value = Polyphase\_Sort.read\_num(i)  
 *if* value:  
 arr.append(*int*(value))  
 *else*:  
 *break* counter -= 1  
 *print*("Sorting process...")  
 start\_time = time.time()  
 arr.sort()  
 end\_time = time.time()  
 *print*("Sorting time: " + *str*(end\_time - start\_time) + " seconds")  
 *if* flag:  
 x.write(" ".join(*str*(i) *for* i *in* arr))  
 x.write(" ")  
 flag = 0  
 *else*:  
 y.write(" ".join(*str*(i) *for* i *in* arr))  
 y.write(" ")  
 flag = 1  
  
 @staticmethod  
 *def* simple\_distribution(input\_file: *str* , file\_1:*str*, file\_2 : *str*):  
 flag = 0  
 *with open*(input\_file,"r") *as* i:  
 *with open*(file\_1,"w") *as* x:  
 *with open*(file\_2,"w") *as* y:  
 current\_string = Polyphase\_Sort.read\_num(i)  
 current\_int = *int*(current\_string)  
 prev = current\_int  
 *while* current\_string:  
 current\_int = *int*(current\_string)  
 *if* prev <= current\_int:  
 *if* flag:  
 y.write(current\_string + " ")  
 *else*:  
 x.write(current\_string + " ")  
 *else*:  
 *if* flag:  
 flag = 0  
 x.write(current\_string + " ")  
 *else*:  
 flag = 1  
 y.write(current\_string + " ")  
 current\_string,prev = Polyphase\_Sort.read\_num(i),current\_int  
  
 @staticmethod  
 *def* count\_seq(file: *str*):  
 count = 0  
 *with open*(file,"r") *as* f:  
 curr = Polyphase\_Sort.read\_num(f)  
 prev = curr  
 *if* curr:  
 count = 1  
 *while* curr:  
 *if* curr == "|":  
 count += 1  
 *elif int*(curr) < *int*(prev):  
 count += 1  
 prev = curr  
 curr = Polyphase\_Sort.read\_num(f)  
 *return* count  
  
 @staticmethod  
 *def* closest\_fibonacci(number:*int*):  
 fib = [0,1]  
 index = 1  
 *while* number > fib[index]:  
 index += 1  
 fib.append(fib[index-2] + fib[index-1])  
 *return* [fib[index-2],fib[index-1],fib[index]]  
  
 @staticmethod  
 *def* min\_index(array: *list*[*int*]):  
 *if* array[0] > array [1]:  
 *return* [1,0]  
 *else*:  
 *return* [0,1]  
  
 @staticmethod  
 *def* fibonacci\_optimization():  
 num\_seq = [Polyphase\_Sort.count\_seq("0.txt"),Polyphase\_Sort.count\_seq("1.txt")]  
 *if not* num\_seq[0] *or not* num\_seq[1]:  
 *return False* closest\_fibonacci = Polyphase\_Sort.closest\_fibonacci(num\_seq[0] + num\_seq[1])  
 *print*("Closest Fibonacci number: " + *str*(closest\_fibonacci))  
 *print*("Number of Sequences in files: " + *str*(num\_seq))  
 minimum = *min*(num\_seq[0],num\_seq[1])  
 minimum\_index = Polyphase\_Sort.min\_index(num\_seq)  
 *if* closest\_fibonacci[0] < minimum:  
 closest\_fibonacci = Polyphase\_Sort.closest\_fibonacci(closest\_fibonacci[2] + 1)  
 *print*("New Fibonacci numbers: " + *str*(closest\_fibonacci))  
 start\_time = time.time()  
 *with open*(*str*(minimum\_index[0]) + ".txt","a") *as* x:  
 *with open*(*str*(minimum\_index[1]) + ".txt","a") *as* y:  
 *while* num\_seq[minimum\_index[0]] < closest\_fibonacci[0]:  
 x.write("| ")  
 num\_seq[minimum\_index[0]] += 1  
 *while* num\_seq[minimum\_index[1]] < closest\_fibonacci[1]:  
 y.write("| ")  
 num\_seq[minimum\_index[1]] += 1  
 end\_time = time.time()  
 *print*("Adjusting to fibonacci time: " + *str*(end\_time - start\_time) + " seconds")  
 *return True* @staticmethod  
 *def* merge\_new(file\_1:*str* ,file\_2:*str*,output\_file:*str*, file\_position = 0):  
 *with open*(file\_1,"r+") *as* x:  
 *with open*(file\_2,"r+") *as* y:  
 *with open*(output\_file,"w") *as* f:  
 *if* file\_position:y.seek(file\_position,0)  
 file\_1\_pos,file\_2\_pos = x.tell(),file\_position  
 file\_1\_curr ,file\_2\_curr = Polyphase\_Sort.read\_num(x),Polyphase\_Sort.read\_num(y)  
 *if not* file\_1\_curr *or not* file\_2\_curr:  
 *print*("Array is sorted")  
 *return* 1  
 *else*:  
 *if* file\_1\_curr != "|" *and* file\_2\_curr != "|":  
 file\_1\_curr,file\_2\_curr = *int*(file\_1\_curr),*int*(file\_2\_curr)  
 file\_1\_prev,file\_2\_prev = file\_1\_curr,file\_2\_curr  
 *while* file\_1\_curr *and* file\_2\_curr:  
 *while* file\_1\_curr != "|" *and* file\_2\_curr != "|" \  
 *and* file\_1\_curr >= file\_1\_prev *and* file\_2\_curr >= file\_2\_prev:  
 *if* file\_1\_curr < file\_2\_curr:  
 f.write(*str*(file\_1\_curr) + " ")  
 file\_1\_pos = x.tell()  
 file\_1\_prev = file\_1\_curr  
 file\_1\_curr = Polyphase\_Sort.read\_num(x)  
 *else*:  
 f.write(*str*(file\_2\_curr) + " ")  
 file\_2\_pos = y.tell()  
 file\_2\_prev = file\_2\_curr  
 file\_2\_curr = Polyphase\_Sort.read\_num(y)  
 *if not* file\_1\_curr:  
 Polyphase\_Sort.move\_pointer(y,file\_2\_pos)  
 Polyphase\_Sort.addition(y,f)  
 x.truncate(0)  
 position = y.tell()  
 x.close()  
 y.close()  
 f.close()  
 Polyphase\_Sort.merge\_new(output\_file,file\_2,file\_1,position)  
 *return* 1  
 *elif not* file\_2\_curr:  
 Polyphase\_Sort.move\_pointer(x,file\_1\_pos)  
 Polyphase\_Sort.addition(x,f)  
 y.truncate(0)  
 position = x.tell()  
 x.close()  
 y.close()  
 f.close()  
 Polyphase\_Sort.merge\_new(output\_file,file\_1,file\_2,position)  
 *return* 1  
 *elif* file\_1\_curr == "|":  
 Polyphase\_Sort.move\_pointer(y,file\_2\_pos)  
 Polyphase\_Sort.addition(y,f)  
 file\_2\_pos = y.tell()  
 file\_2\_curr = Polyphase\_Sort.read\_num(y)  
 *break  
 elif* file\_2\_curr == "|":  
 Polyphase\_Sort.move\_pointer(x,file\_1\_pos)  
 Polyphase\_Sort.addition(x,f)  
 file\_1\_pos = x.tell()  
 file\_1\_curr = Polyphase\_Sort.read\_num(x)  
 *break  
 else*:  
 file\_1\_curr,file\_2\_curr = *int*(file\_1\_curr),*int*(file\_2\_curr)  
 *if* file\_1\_curr < file\_1\_prev:  
 Polyphase\_Sort.move\_pointer(y,file\_2\_pos)  
 Polyphase\_Sort.addition(y,f)  
 file\_2\_pos = y.tell()  
 file\_2\_curr = Polyphase\_Sort.read\_num(y)  
 file\_1\_prev = file\_1\_curr  
 *if not* file\_2\_curr:  
 y.truncate(0)  
 x.close()  
 y.close()  
 f.close()  
 Polyphase\_Sort.merge\_new(output\_file,file\_1,file\_2,file\_1\_pos)  
 *return* 1  
 *elif* file\_2\_curr == "|":  
 Polyphase\_Sort.move\_pointer(x,file\_1\_pos)  
 Polyphase\_Sort.addition(x,f)  
 file\_1\_pos = x.tell()  
 file\_1\_curr = Polyphase\_Sort.read\_num(x)  
 *break  
 else*:  
 file\_2\_curr = *int*(file\_2\_curr)  
 file\_2\_prev = file\_2\_curr  
 *elif* file\_2\_curr < file\_2\_prev:  
 Polyphase\_Sort.move\_pointer(x,file\_1\_pos)  
 Polyphase\_Sort.addition(x,f)  
 file\_1\_pos = x.tell()  
 file\_1\_curr = Polyphase\_Sort.read\_num(x)  
 file\_2\_prev = file\_2\_curr  
 *if not* file\_1\_curr:  
 x.truncate(0)  
 x.close()  
 y.close()  
 f.close()  
 Polyphase\_Sort.merge\_new(output\_file,file\_2,file\_1,file\_2\_pos)  
 *return* 1  
 *elif* file\_1\_curr == "|":  
 Polyphase\_Sort.move\_pointer(y,file\_2\_pos)  
 Polyphase\_Sort.addition(y,f)  
 file\_2\_pos = y.tell()  
 file\_2\_curr = Polyphase\_Sort.read\_num(y)  
 *break  
 else*:  
 file\_1\_curr = *int*(file\_1\_curr)  
 file\_1\_prev = file\_1\_curr  
 *while* file\_1\_curr == "|" *or* file\_2\_curr == "|":  
 *if* file\_1\_curr == "|" *and* file\_2\_curr:  
 Polyphase\_Sort.move\_pointer(y,file\_2\_pos)  
 Polyphase\_Sort.addition(y,f)  
 file\_1\_pos = x.tell()  
 file\_2\_pos = y.tell()  
 file\_1\_curr = Polyphase\_Sort.read\_num(x)  
 file\_2\_curr = Polyphase\_Sort.read\_num(y)  
 *elif* file\_2\_curr == "|" *and* file\_1\_curr:  
 Polyphase\_Sort.move\_pointer(x,file\_1\_pos)  
 Polyphase\_Sort.addition(x,f)  
 file\_1\_pos = x.tell()  
 file\_2\_pos = y.tell()  
 file\_1\_curr = Polyphase\_Sort.read\_num(x)  
 file\_2\_curr = Polyphase\_Sort.read\_num(y)  
 *elif not* file\_2\_curr:  
 y.truncate(0)  
 x.close()  
 y.close()  
 f.close()  
 Polyphase\_Sort.merge\_new(output\_file,file\_1,file\_2,file\_1\_pos)  
 *return* 1  
 *elif not* file\_1\_curr:  
 x.truncate(0)  
 x.close()  
 y.close()  
 f.close()  
 Polyphase\_Sort.merge\_new(output\_file,file\_2,file\_1,file\_2\_pos)  
 *return* 1  
 *if not* file\_1\_curr:  
 x.truncate(0)  
 x.close()  
 y.close()  
 f.close()  
 Polyphase\_Sort.merge\_new(output\_file,file\_2,file\_1,file\_2\_pos)  
 *return* 1  
 *if not* file\_2\_curr:  
 y.truncate(0)  
 x.close()  
 y.close()  
 f.close()  
 Polyphase\_Sort.merge\_new(output\_file,file\_1,file\_2,file\_1\_pos)  
 *return* 1  
  
 @staticmethod  
 *def* addition(file: TextIO, output:TextIO):  
 pos = file.tell()  
 curr = Polyphase\_Sort.read\_num(file)  
 prev = curr  
 *if* curr == '':  
 *return  
 if* curr == "|":  
 output.write(curr + " ")  
 *else*:  
 *while* curr *and* curr != "|" *and int*(curr) >= *int*(prev):  
 *print*(curr)  
 output.write(curr + " ")  
 prev = curr  
 pos = file.tell()  
 curr = Polyphase\_Sort.read\_num(file)  
 Polyphase\_Sort.move\_pointer(file,pos)

main\_pa.py

*import* math  
*import* time  
*import* random  
*from* typing *import* TextIO  
*from* Polyphase\_Sort *import* \*  
  
  
elements = 10000  
devision\_size = *int*(elements / 2)  
*with open*("sourse.txt","w") *as* file:  
 *for* i *in range*(0,elements):  
 file.write(*str*(random.randint(-10000,10000)) + " ")  
prog\_start\_time = time.time()  
*print*("File filled with random numbers")  
*# Polyphase\_Sort.modified\_distribution("sourse.txt" ,"0.txt","1.txt",devision\_size)*Polyphase\_Sort.simple\_distribution("sourse.txt", "0.txt", "1.txt")  
*if* Polyphase\_Sort.fibonacci\_optimization():  
 alg\_start\_time = time.time()  
 Polyphase\_Sort.merge\_new("0.txt","1.txt","2.txt")  
 alg\_end\_time = time.time()  
 *print*("Polyphase Sort ended")  
 *print*("Algoritm take time: " + *str*(alg\_end\_time - alg\_start\_time) + " seconds")  
prog\_end\_time = time.time()  
*print*("Total take time: " + *str*(prog\_end\_time - prog\_start\_time) + " seconds")

Висновок

При виконанні даної лабораторної роботи було розроблено та записано алгоритм Багатофазного Сортування.

Виконано програмну реалізацію алгоритма Багатофазного Сортування на мові програмування Python.

Модифіковано алгоритм Багатофазного Сортування за допомогою попереднього сортування послідовностей завдяки вбудованого сортування мови Python.

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%.