**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)

ІП-з21 Єфремов Юрій Володимирович

**Перевірив**

(прізвище, ім'я, по батькові)

Головченко М.М.

Київ 2023

Зміст

[1 Мета лабораторної роботи 3](#_Toc109342184)

[2 ЗаВдання 4](#_Toc109342185)

[3 Виконання](#_Toc109342186) 5

[3.1 Псевдокод алгоритму 6](#_Toc109342187)

[3.2 Програмна реалізація алгоритму 6](#_Toc109342188)

[3.2.1 Вихідний код 6](#_Toc109342189)

[Висновок 7](#_Toc109342190)

# Мета лабораторної роботи

Мета роботи – вивчити основні алгоритми зовнішнього сортування та способи їх модифікації, оцінити поріг їх ефективності.

# Завдання

Згідно варіанту 25(пряме злиття), 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. Достатньо штучно обмежити доступну ОП, для уникнення багатогодинних сортувань (наприклад використовуючи віртуальну машину).

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.

# Виконання

## Псевдокод алгоритму

**function generate\_random\_array(size):**

**return array of size random integers**

**function write\_array\_to\_file(file\_path, array):**

**open file\_path in binary write mode**

**write array elements to the file**

**close the file**

**function read\_array\_from\_file(file\_path, array\_typecode):**

**open file\_path in binary read mode**

**read file content into an array of specified typecode**

**close the file**

**return the array**

**function sort\_array\_in\_memory(array):**

**sort the array in memory**

**function sort\_large\_array\_in\_chunks(file\_path, chunk\_size):**

**open the file\_path in binary read mode**

**create a temporary file temp\_sorted\_file**

**while not end of file:**

**read a chunk of size chunk\_size from the file**

**sort the chunk in memory**

**append the sorted chunk to temp\_sorted\_file**

**close both files**

**replace the original file with temp\_sorted\_file**

**function main():**

**# Part 1: Sort small array in memory**

**small\_array\_size = 10,000,000**

**small\_array = generate\_random\_array(small\_array\_size)**

**sort\_array\_in\_memory(small\_array)**

**write\_array\_to\_file("small\_array.bin", small\_array)**

**# Part 2: Sort large array in chunks**

**large\_array\_size = 1,000,000,000 # 1 GB**

**large\_array = generate\_random\_array(large\_array\_size)**

**write\_array\_to\_file("large\_array.bin", large\_array)**

**sort\_large\_array\_in\_chunks("large\_array.bin", chunk\_size=100,000,000)**

## Програмна реалізація алгоритму

### Вихідний код

Import random  
import os  
import struct  
import array  
import time  
  
  
def generate\_random\_array(size):  
 return [random.randint(1, 1000000) for \_ in range(size)]  
  
  
def write\_array\_to\_file(file\_path, array):  
 with open(file\_path, 'wb') as file:  
 array.tofile(file)  
  
  
def read\_array\_from\_file(file\_path, array\_typecode):  
 with open(file\_path, 'rb') as file:  
 file\_size = os.path.getsize(file\_path)  
 element\_size = struct.calcsize(array\_typecode)  
 array\_size = file\_size // element\_size  
 file\_content = array.array(array\_typecode)  
 file\_content.fromfile(file, array\_size)  
 return file\_content  
  
  
def sort\_array\_in\_memory(array):  
 array.sort()  
  
  
def sort\_large\_array\_in\_chunks(file\_path, chunk\_size):  
 temp\_file\_path = 'temp\_sorted\_file.bin'  
 with open(file\_path, 'rb') as file:  
 try:  
 while True:  
 chunk = array.array('q')  
 chunk.fromfile(file, chunk\_size)  
 if not chunk:  
 break  
 chunk = array.array('q', sorted(chunk))  
 with open(temp\_file\_path, 'ab') as temp\_file:  
 chunk.tofile(temp\_file)  
 except EOFError as err:  
 print(err)  
 os.replace(temp\_file\_path, file\_path)  
  
  
def main():  
 # Part 1: Sort small array in memory  
 small\_array\_size = 10000000  
 small\_array = generate\_random\_array(small\_array\_size)  
 sort\_array\_in\_memory(small\_array)  
  
 # Write and read small array to/from file  
 file\_path\_small = 'small\_array.bin'  
 write\_array\_to\_file(file\_path\_small, array.array('q', small\_array))  
 print('finished small array')  
 small\_array\_from\_file = read\_array\_from\_file(file\_path\_small, 'q')  
  
 # Part 2: Sort large array in chunks  
 # large\_array\_size = 1000000000 # 1 GB  
 large\_array\_size = 100000000 # 1 GB  
 large\_array = generate\_random\_array(large\_array\_size)  
  
 # Write large array to file  
 file\_path\_large = 'large\_array.bin'  
 start\_time = time.time()  
 write\_array\_to\_file(file\_path\_large, array.array('q', large\_array))  
 end\_time = time.time()  
 print(end\_time - start\_time)  
  
 # Sort large array in chunks  
 start\_time = time.time()  
 sort\_large\_array\_in\_chunks(file\_path\_large, chunk\_size=10000000) # 100MB chunks  
 end\_time = time.time()  
  
 print(f'Sorting time for large array: {end\_time - start\_time} seconds')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

Висновок

Дана лабораторна робота спрямована на дослідження ефективності сортування великих обсягів даних, які не вміщуються повністю в оперативну пам'ять. Завдання включало генерацію великого масиву випадкових цілих чисел, збереження його в файл, та подальше сортування цього масиву за допомогою розбиття на частини для ефективності.

У процесі роботи були використані наступні кроки:

1. \*\*Генерація масиву:\*\*

- Створення випадкового масиву заданого розміру.

2. \*\*Збереження в файл:\*\*

- Запис масиву в бінарний файл для подальшої обробки.

3. \*\*Сортування в пам'яті:\*\*

- Використання вбудованих засобів для сортування масиву в пам'яті.

4. \*\*Сортування великого масиву в чанках:\*\*

- Читання та сортування масиву частинами для ефективності при роботі з великим обсягом даних.

В результаті виконання лабораторної роботи було досягнуто мети щодо швидкості сортування великого масиву даних, що не вміщується в оперативну пам'ять. Використання частин (chunks) для сортування дозволяє оптимізувати роботу з великими обсягами даних та зменшити час сортування.

Узагальнюючи, важливо розуміти, що оптимізація обробки великих обсягів даних часто вимагає креативних підходів та використання алгоритмів, придатних для роботи з обмеженим обсягом оперативної пам'яті.