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-15 Мєшков Андрій Ігорович* _____
(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

Варіант 18

Природне (адаптивне) злиття

3.1 Псевдокод алгоритму

```
procedure Sort()
      while not IsSorted(path_a) do
            Distribute()
            Merge()
      end while
end procedure Sort()

procedure IsSorted(path_a)
      a = open(path_a, mode=”rb”)
      curr = a.read()
      next = a.read()
      while next do
            if curr > next then
                  a.close()
                  return false
            end if
            curr = next
            next = a.read()
      end while
      a.close()
      return true
end procedure IsSorted()

procedure Distribute(path_a, path_b, path_c)
      a = open(path_a, mode=”RB”)
      b = open(path_b, mode=”WB”)
      c = open(path_c, mode=”WB”)

      i = True
      curr = a.read()
      next = a.read()
      while curr do
            if i then
                  b.write(curr)
            else
                  c.write(curr)
```

```
            end if
            if curr > next then
                  i = NOT i
            end if
            curr = next
            next = a.read()
      end while
      a.close()
      b.close()
      c.close()
end procedure Distribute()

procedure Merge(path_a, path_b, path_c)
      a = open(path_a, mode="WB")
      b = open(path_b, mode="RB")
      c = open(path_c, mode="RB")
      curr_b = b.read()
      curr_c = c.read()
      next_b = b.read()
      next_c = c.read()
      while curr_b and curr_c do
            if curr_b <= next_b and curr_c <= next_c then
                  if curr_b <= curr_c then
                        a.write(curr_b)
                        curr_b = next_b
                        next_b = b.read()
                  else
                        a.write(current_c)
                        curr_c = next_c
                        next_c = c.read()
                  end if
            else if curr_b >= next_b and curr_c <= next_c then
                  while curr_c <= next_c do
                        if curr_b <= curr_c then
                              a.write(curr_b)
                              curr_b = next_b
                              next_b = b.read()
                              while curr_c <= next_c do
                                    a.write(curr_c)
                                    curr_c = next_c
                                    next_c = c.read()
                              end while
```

```
                    a.write(curr_c)
                    curr_c = next_c
                    next_c = c.read()
                    break
              else
                    a.write(curr_c)
                    curr_c = next_c
                    next_c = c.read()
              end if
        end while
else if curr_c >= next_c and curr_b <= next_b then
        while curr_b <= next_b do
              if curr_c <= curr_b then
                    a.write(curr_c)
                    curr_c = next_c
                    next_c = c.read()
                    while curr_b <= next_b do
                          a.write(curr_b)
                          curr_b = next_b
                          next_b = b.read()
                    end while
                    a.write(curr_b)
                    curr_b = next_b
                    next_b = b.read()
                    break
              else
                    a.write(curr_b)
                    curr_b = next_b
                    next_b = b.read()
              end if
else
        if curr_c <= curr_b then
              a.write(curr_c)
              a.write(curr_b)
        else
              a.write(curr_b)
              a.write(curr_c)
        end if
        curr_c = next_c
        curr_b = next_b
        next_c = c.read()
        next_b = b.read()
end if
```

```
        if not curr_b and curr_c then
            while curr_c do
                a.wriet(curr_c)
                curr_c = next_c
                next_c = c.read()
            end while
        else if not curr_c and b_curr THEN
            while curr_b do
                a.write(curr_b)
                curr_b = next_b
                next_b = b.read()
            end while
        end if
        a.close()
        b.close()
        c.close()
    end procedure Merge()
```

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
from random import randint
import shutil


class Reader:
    """Клас, що представляє бінарний файл - два покажчики читання: поточний і наступний"""

    def __init__(self, path: str):
        self.path = path
        self.f = open(path, "rb")
        self.curr = self.f.read(16)
        self.next = self.f.read(16)

    def close(self):
        self.f.close()

    def __iter__(self):
        return self

    def __next__(self):
        """Повертає поточний покажчик і переміщує покажчики вперед на 1"""
        tmp = self.curr
        self.curr = self.next
        self.next = self.f.read(16)
        return tmp


class Sorter:
    def __init__(self, path_a: str, path_b: str, path_c: str):
        """Файл B і C очищено, файл A – ні"""
        self.path_a = path_a
```

```
        self.path_b = path_b
        self.path_c = path_c
        self.clear(path_b)
        self.clear(path_c)

    def generate_file(self, n: int, max_n: int):
        with open(self.path_a, "wb") as a:
            for i in range(n):
                a.write(randint(1, max_n).to_bytes(16, byteorder="big"))

    def copy_file(self, path: str):
        shutil.copy(path, self.path_a)

    def sort(self):
        while not self.is_sorted():
            self.distribute()
            self.merge()

    def __str__(self):
        a, b, c = "A: " + self.read(self.path_a), "B: " +
self.read(self.path_b), "C: " + self.read(self.path_c)
        return "\n".join([x for x in [a, b, c] if x])

    def distribute(self):
        a = Reader(self.path_a)
        b = open(self.path_b, "wb")
        c = open(self.path_c, "wb")

        i = True
        while a.curr:
            b.write(a.curr) if i else c.write(a.curr)
            if a.curr > a.next:
                i = not i
            next(a)

        a.close(), b.close(), c.close()

    def merge(self):
        """
        Злиття відповідних серій.
            Якщо обидва дані числа не в кінці серії, ми знаходимо менше та
дописуємо його, далі посуваємо
            покажчик - таким чином зливаючи ці дві серії.
            Тобто якщо число файлу B в кінці серії, то ми зливаємо актуальну
серію файла B і залишки
            серії файла C. Якщо число із файла C менше, ми його дописуємо, а як
тільки меншим
            виявиться останнє число серії із файла B, то ми це число дописуємо,
і всі інші
            числа із серії із файла C так само дописуємо аж до кінця серії.
Після цієї операції обидва
            покажчика будуть на початку нової серії.
            Аналогічно робимо якщо число файлу C в кінці серії.
            Якщо ж обидва числа в кінці серії, дописуємо їх по зростанню і
зсуваємо покажчик на крок уперед,
            в результаті обидва покажчики на початку серії.
            Якщо покажчик в кінці файлу, дописуємо залишок іншого файлу.
        """
        a = open(self.path_a, "wb")
        b = Reader(self.path_b)
        c = Reader(self.path_c)
        while b.curr and c.curr:
            if b.curr <= b.next and c.curr <= c.next:
                if b.curr <= c.curr:
```

```
                    a.write(next(b))
                else:
                    a.write(next(c))
            elif b.curr >= b.next and c.curr <= c.next:
                while c.curr <= c.next:
                    if b.curr <= c.curr:
                        a.write(next(b))
                        while c.curr <= c.next:
                            a.write(next(c))
                        a.write(next(c))
                        break
                    else:
                        a.write(next(c))
            elif c.curr >= c.next and b.curr <= b.next:
                while b.curr <= b.next:
                    if c.curr <= b.curr:
                        a.write(next(c))
                        while b.curr <= b.next:
                            a.write(next(b))
                        a.write(next(b))
                        break
                    else:
                        a.write(next(b))
            else:
                if c.curr <= b.curr:
                    a.write(c.curr)
                    a.write(b.curr)
                else:
                    a.write(b.curr)
                    a.write(c.curr)
                next(c), next(b)

        if not b.curr and c.curr:
            while c.curr:
                a.write(next(c))
        elif not c.curr and b.curr:
            while b.curr:
                a.write(next(b))

        a.close(), b.close(), c.close()

    def is_sorted(self):
        """Перевірити, чи файл A вже відсортовано"""
        a = Reader(self.path_a)
        while a.next:
            if a.curr > a.next:
                a.close()
                return False
            next(a)
        a.close()
        return True

    @staticmethod
    def read(path: str) -> str:
        """Прочитати перші 30 чисел із файлу, поставивши «|» в кінці кожної
серії"""
        s = ""
        f = Reader(path)

        for i in range(30):
            if not f.curr:
                break
            s += str(int.from_bytes(f.curr, byteorder="big")) + " "
            if f.curr > f.next:
```

```
                s += "| "
            next(f)

        f.close()
        return s

    @staticmethod
    def clear(path: str):
        with open(path, "wb"):
            pass
```

3.3 Тестування програмної реалізації

Для тестування створимо файл розміром в 16 Мб за допомогою методу generate_file (1024**2, 1000000) та здійснимо його сортування, виведемо перші 30 чисел файлу:

```
sorter = Sorter("a.bin", "b.bin", "c.bin")
sorter.generate_file(1024**2, 1000000)
sorter.sort()
print(sorter)
```

Отримали:

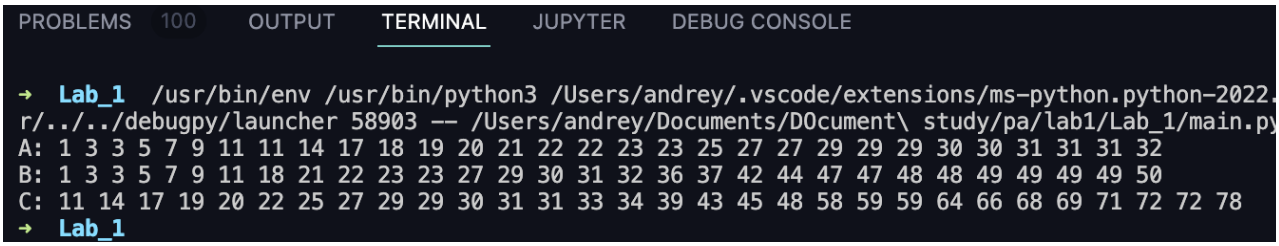

Рисунок 3.1 — тестування програмної реалізації

ВИСНОВОК

При виконанні даної лабораторної роботи було розроблено алгоритм реалізації зозвнішнього адаптивного злиття у вигляді псевдокоду, була здійснена його реалізація на Python та тестування на випадково згенерованому файлі розміру 16 Мб.

Алгоритм реалізовано у декілька процедур:

- Перевірка відсортованості головного файлу;
- Розподіл даних між двома файлами;
- Природне злиття.

Опис процедур:

Розподіл між файлами. Зчитуємо поступово файл A, дописуємо кожен елемент відповідно до файлу B чи C в залежності від прапора **i**, в кінці серії змінюємо прапор на протилежний.

Процедура злиття. Якщо обидва дані числа не в кінці серії, ми знаходимо менше та дописуємо його, далі посуваємо покажчик - таким чином зливаючи ці дві серії. Тобто якщо число файлу B в кінці серії, то ми зливаємо актуальну серію файлу B і залишки серії файлу C. Якщо число із файлу C менше, ми його дописуємо, а як тільки меншим виявиться останнє число серії із файлу B, то ми це число дописуємо, і всі інші числа із серії з файлу C так само дописуємо аж до кінця серії. Після цієї операції обидва покажчика будуть на початку нової серії. Аналогічно робимо якщо число файлу C в кінці серії. Якщо ж обидва числа в кінці серії, дописуємо їх по зростанню і зсуваємо покажчик на крок уперед, в результаті обидва покажчики на початку серії. Якщо покажчик в кінці файлу, дописуємо залишок іншого файлу.

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%.