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-13 Карамян Вартан Суренович* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов Олексій Олександрович* _____
(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

3.1 Псевдокод алгоритму

```
MultiWayMerge()
for i=0 to n_files:
    files_b [i] = open("B{i}.bin")
    files_c[i] = open("C{i}.bin)
end for
first_distribution(files_b)
flag = true
while (!is_sorted(path_a, files_b[0], files_c[0])):
    if flag:
        merge(files_b, files_c)
    else:
        merge(files_c, files_b)
    flag = !flag
end while
if size(path_a) == size(files_b[0]):
    copy(files_b[0], path_d)
else:
    copy(files_c[0], path_d)

is_sorted(init_file, file_b1, file_c1):
    return size(init_file) == size(file_b1) | | size(init_file) == size(file_c1)

are_all_empty(files):
    for file in files:
        if file.curr:
            return False
    end for
    return True
```

```
merge(input_files, output_files):
        j = 0
        seq = []
        while (!are_all_empty(input_files)):
                min_val = MAX_INT
                min_ind = -1
                for i = 0 to n_files:
                        if input_files[i].curr:
                                num = int(input_files[i].curr)
                                if !seq | | num >= seq[-1]:
                                        if num <= min_val:
                                                min_val = num
                                                min_ind = i
                end for
                if min_ind < 0:
                        for num in seq:
                                output_files[j].write(num)
                        end for
                        seq = []
                        j = (j+1) % n_files
                else:
                        seq.append(min_value)
                        next(input_files[min_ind])
                for num in seq:
                        output_files[j].write(num)
                end for

                if seq.size >= 2621440:
                        for num in seq:
                                output_files[j].write(num)
```

end for

seq = []

end while

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

main.py

```
from multi_way_merge import MultiWayMerge
import time


start_time = time.time()


print("256 = 1KB\n"
      "262144 = 1MB\n"
      "268435356 = 1ГВ")

n = int(input("Enter amount of numbers: "))
n_files = int(input("Enter amount of files: "))


sorter = MultiWayMerge("Files/A.bin", "Files/D.bin", n, n_files)
sorter.sort()

print(f"\ntime: {(time.time() - start_time)}s seconds ")
```

multi_way_merge.py

```
import os.path
import shutil
from random import randint

MAX_INT = 2147483647


class ReadFromFile:
    def __init__(self, path: str):
        self._file = open(path, "rb")
        self.curr = self._file.read(4)
        self.next = self._file.read(4)

    def __next__(self):
        self.curr = self.next
        self.next = self._file.read(4)

    def close(self):
        self._file.close()
```

```
class MultiWayMerge:
    def __init__(self, path_a: str, path_d: str, n: int, n_files: int = 2):
        self._path_a = path_a
        self._path_d = path_d
        self._n = n
        self._n_files = n_files

    def _create_file(self):
        with open(self._path_a, "wb") as a:
            for i in range(self._n):
                a.write(randint(1, MAX_INT).to_bytes(4, "big"))

    def sort(self):

        # generate init file A
        self._create_file()

        # string arrays with paths
        files_b_paths = []
        files_c_paths = []
        for i in range(self._n_files):
            files_b_paths.append(f"Files/B{i + 1}.bin")
            files_c_paths.append(f"Files/C{i + 1}.bin")

        # create empty C files
        for path in files_c_paths:
            with open(path, "wb") as file:
                pass

        self._first_distribution(files_b_paths)

        flag = True
        while not self._is_sorted(self._path_a, files_b_paths[0],
files_c_paths[0]):
            if flag:
                self._merge(files_b_paths, files_c_paths)
            else:
                self._merge(files_c_paths, files_b_paths)
            flag = not flag

        if flag:
            shutil.copy(files_b_paths[0], self._path_d)
        else:
            shutil.copy(files_c_paths[0], self._path_d)

    @staticmethod
    def _is_sorted(init_file: str, file_b1: str, file_c1: str) -> bool:
        return os.path.getsize(init_file) == os.path.getsize(file_b1) or \
               os.path.getsize(init_file) == os.path.getsize(file_c1)

    def _first_distribution(self, output_files: list):
        # open files
        a = ReadFromFile(self._path_a)

        # open array of B files
        files_B = []
        for path in output_files:
            files_B.append(open(path, "wb"))

        # number of file
        i = 0

        # distribution
```

```
        while a.curr:
            files_B[i].write(a.curr)
            if a.curr > a.next:
                i = (i + 1) % self._n_files
            next(a)

        # close files
        a.close()
        for file in files_B:
            file.close()

    def _merge(self, input_files_paths: list, output_files_paths: list):
        # open files
        input_files = []
        output_files = []

        for path in input_files_paths:
            input_files.append(ReadFromFile(path))

        for path in output_files_paths:
            output_files.append(open(path, "wb"))

        # main algorithm

        j = 0
        buff = []
        while not self._are_all_empty(input_files):
            min_val = MAX_INT
            min_ind = -1

            # find the smallest number among all input files
            for i in range(self._n_files):

                if input_files[i].curr:  # file is not empty
                    num = int.from_bytes(input_files[i].curr, "big")

                    if not buff or num >= buff[-1]:  # buff is empty yet or
                                                     # num is bigger than last
element in the series
                        if num <= min_val:
                            min_val = num
                            min_ind = i

            # end of the series, writing in a file and changing the file
            if min_ind < 0:
                for num in buff:
                    output_files[j].write(num.to_bytes(4, "big"))
                buff = []
                j = (j + 1) % self._n_files

            # add num to the series
            else:
                buff.append(min_val)
                next(input_files[min_ind])

            # buff is full, writing in a file
            if len(buff) >= 262144:
                for num in buff:
                    output_files[j].write(num.to_bytes(4, "big"))
                buff = []

        # writing the last series in output file
        for num in buff:
            output_files[j].write(num.to_bytes(4, "big"))
```

```
        # close files
        for file in input_files:
            file.close()

        for file in output_files:
            file.close()

    @staticmethod
    def _are_all_empty(input_files: list) -> bool:
        for file in input_files:
            if file.curr:
                return False
        return True
```

ВИСНОВОК

При виконанні даної лабораторної роботи я познайомився з алгоритмами зовнішнього сортування. Дослідив та реалізував програмно алгоритм збалансованого багатошляхового злиття.

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%.