**­Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра автоматизованих систем обробки інформації**

**і управління**

**Звіт**

з лабораторної роботи № 6 з дисципліни

«Алгоритми та структури даних 2. Структури даних»

„ **Проектування і аналіз алгоритмів внутрішнього сортування**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-01 Галько Міла Вячеславівна*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.М.*

Київ 2021

# Мета лабораторної роботи

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

## Завдання:

Згідно варіанту 6 – «Природне (адаптивне) злиття, розробити та записати алгоритм зовнішнього сортування за допомогою псевдокоду.

Виконати програмну реалізацію алгоритму на мові С++ 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 методу за рахунок використання оперативної пам’яті і відсортувати випадковим чином згенерований масив цілих чисел, що зберігається у файлі розміром не менше 1Гб. Необхідно попередньо впорядкувати серії елементів довжиною, що займає не менше 100Мб оперативної пам’яті.

Зробити узагальнений висновок з лабораторної роботи, у якому порівняти основний та модифікований методи виконання.

# Псевдокод алгоритму

Сортування файлів:

MergeSorting(vector<int> &nums, int l, int r):

int m = (l + r) / 2

if (l < r):

MergeSorting(nums, l, m)

MergeSorting(nums, m + 1, r)

end if

int currentL = l

int currentR = m + 1

for (int i = l; i <= r; i++):

if (currentL <= m && (currentR > r || (nums[currentL] < nums[currentR]))):

Vec.push\_back(nums[currentL])

currentL++

end if

else:

Vec.push\_back(nums[currentR])

currentR++

end else

end for

int counter = 0

for (int i = l; i <= r; i++) :

nums[i] = subVec[counter]

counter++

end for

end MergeSorting

Злиття файлів:

bool newFirst = true

bool newSec = true

bool stopFirst = false

bool stopSec = false

while (true):

if (newFirst):

if (first.read((char\*)&currentFirst, 4)): newFirst = false

else:

stopFirst = true

newFirst = false

end else

end if

if (newSec):

if (sec.read((char\*)&currentSec, 4)): newSec = false

else:

stopSec = true

newSec = false

end else

end if

if (stopFirst && stopSec): break

else:

if (!stopFirst && (stopSec || currentFirst < currentSec)):

sub.write((char\*)&currentFirst, 4)

newFirst = true

end if

else:

sub.write((char\*)&currentSec, 4)

newSec = true

end else

end else

end while

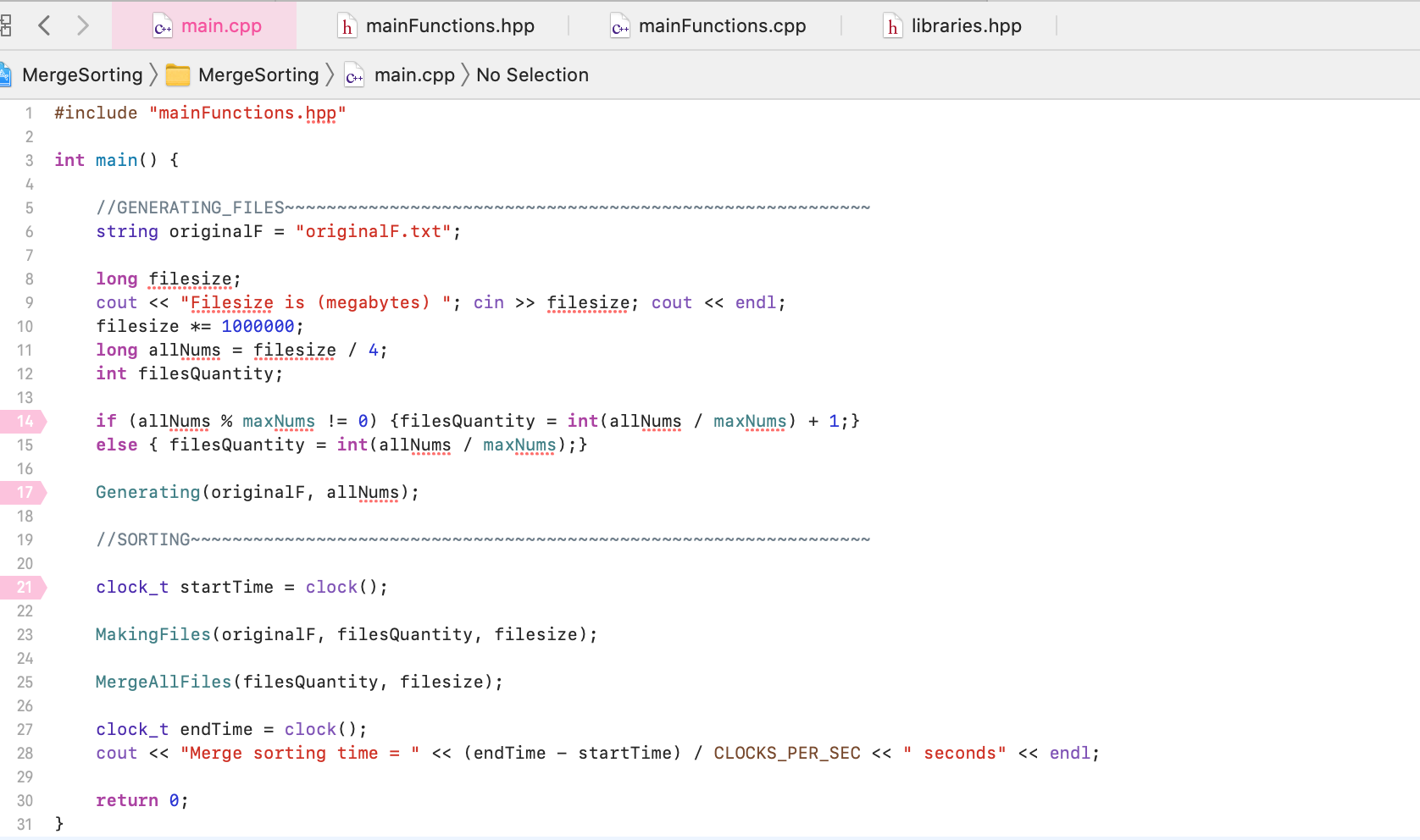
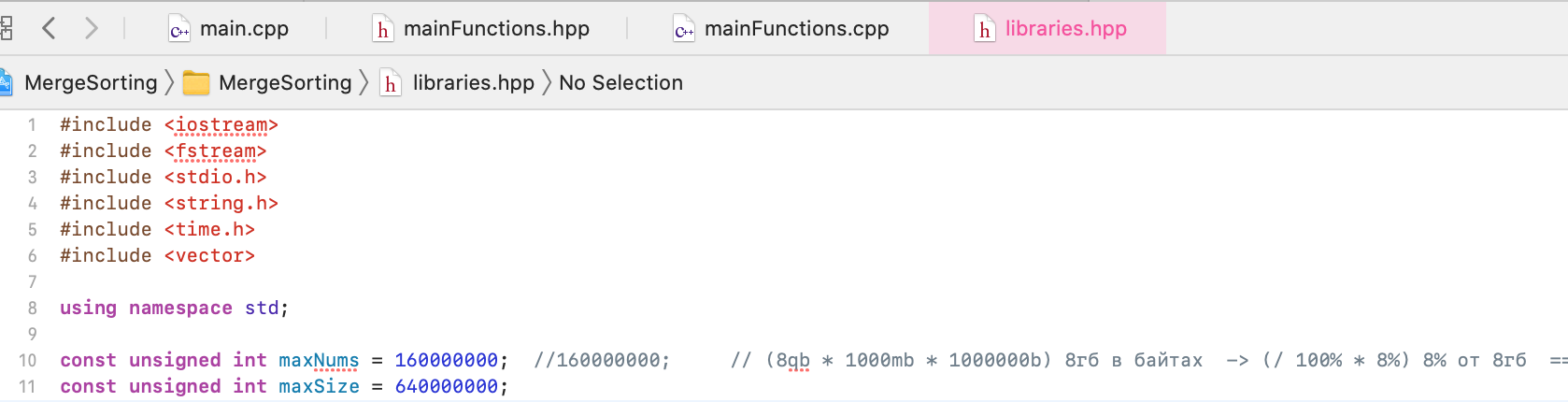
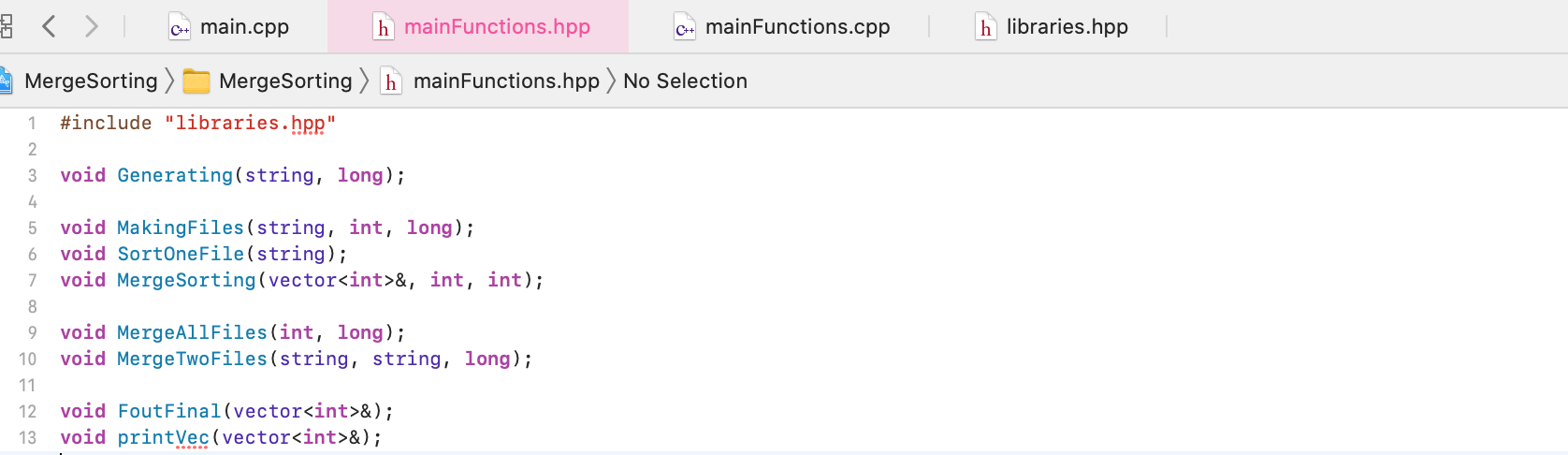
\_fs::filesystem::remove(second)

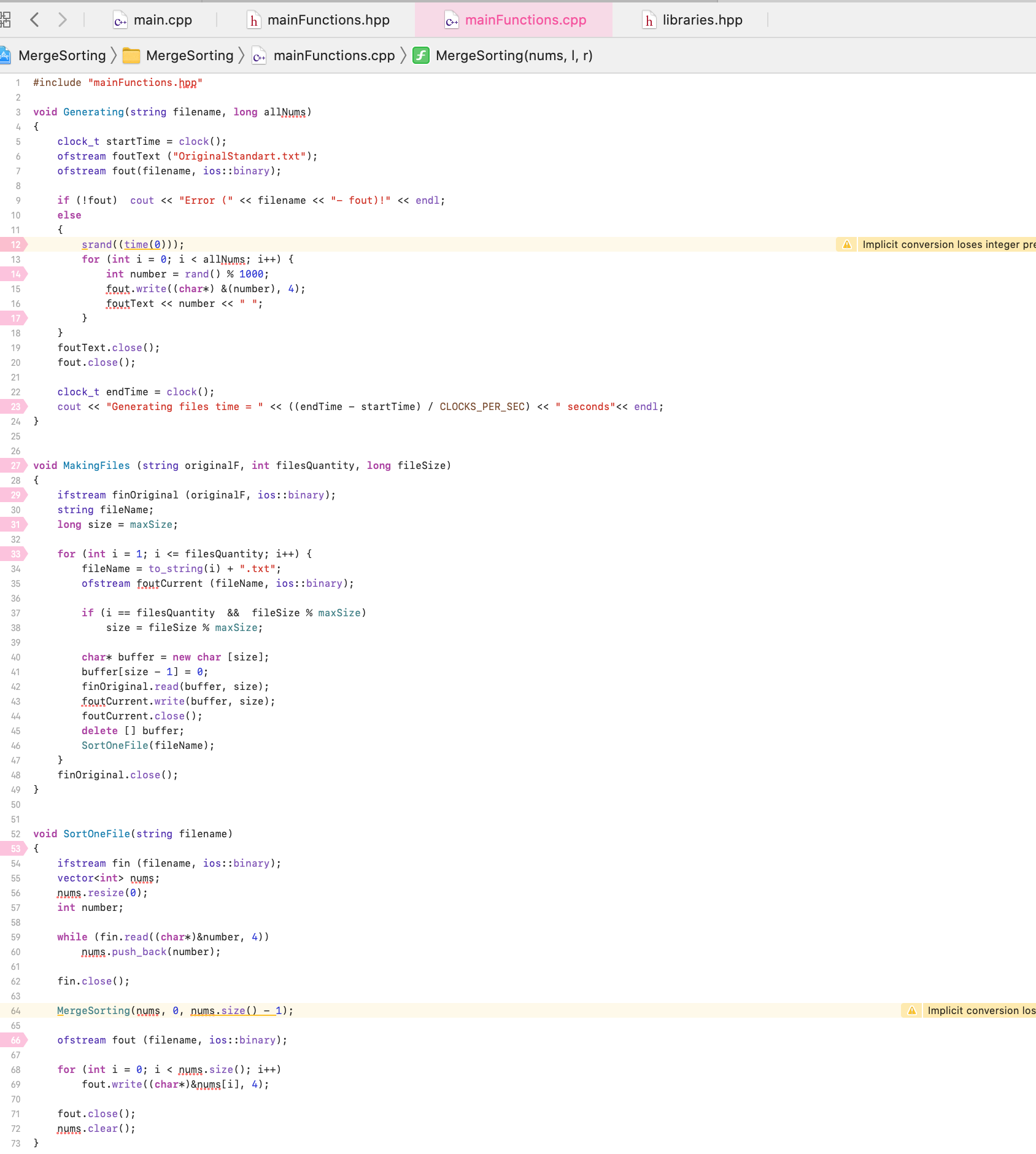
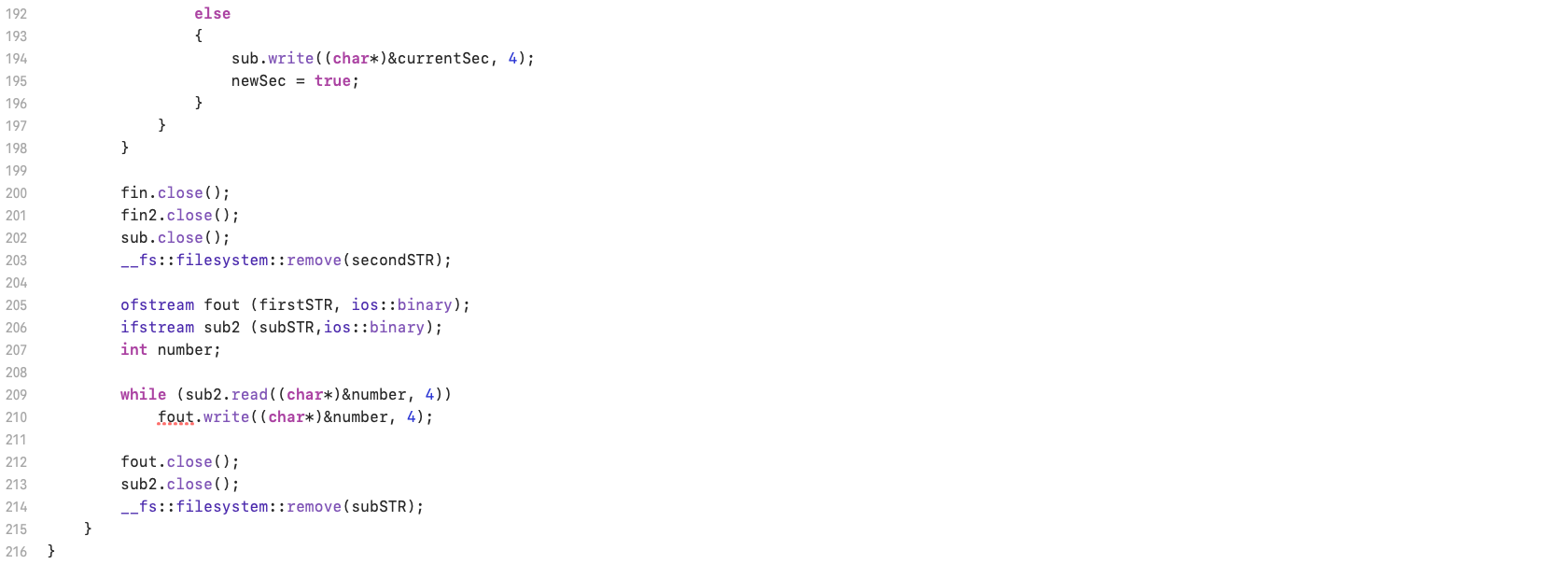
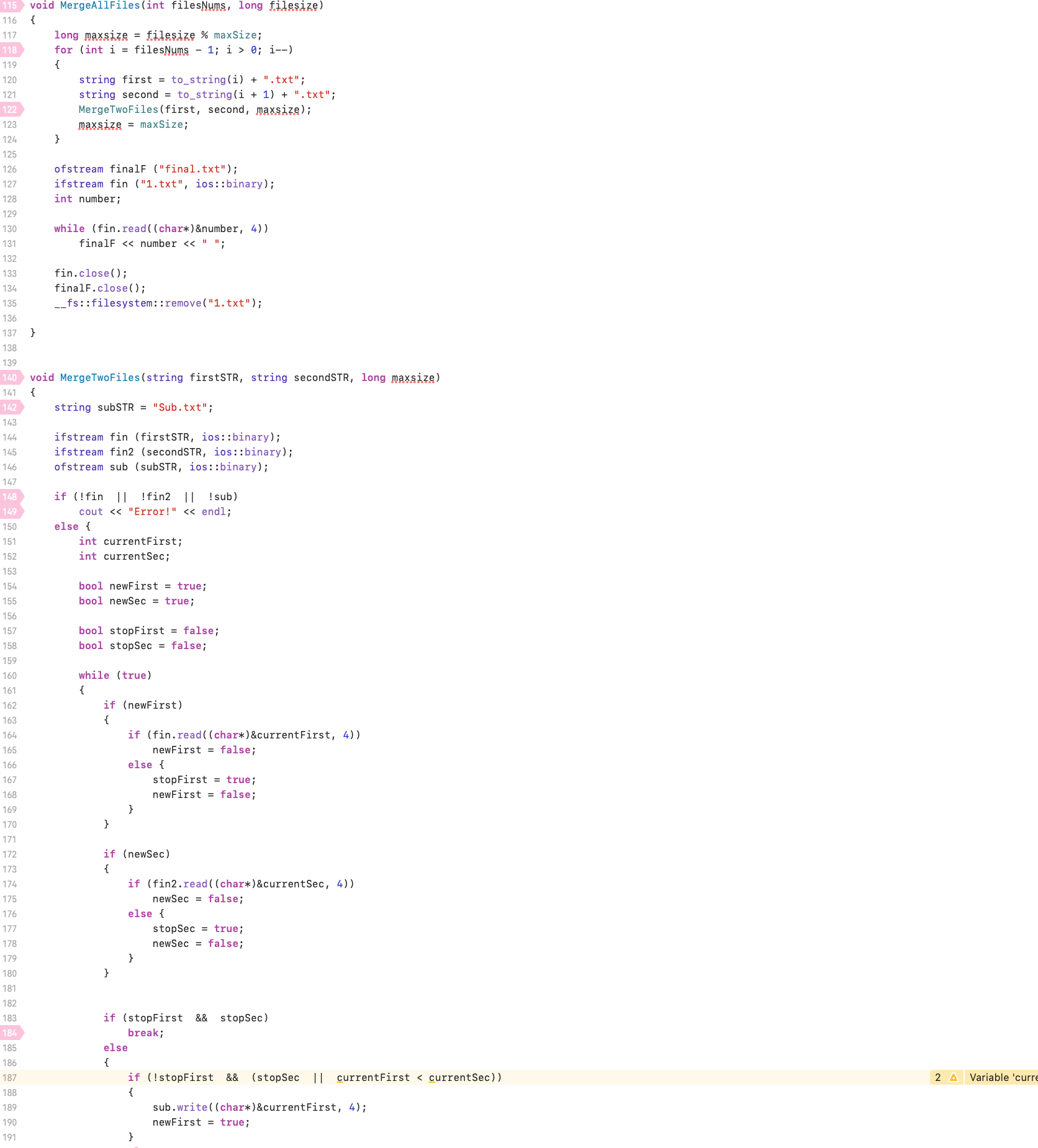
while (sub.read((char\*)&number, 4)): fout.write((char\*)&number, 4)

\_\_fs::filesystem::remove(sub)

## 

# Програмна реалізація алгоритму

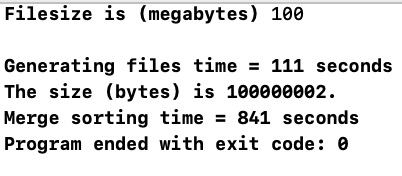
  

Висновок

При виконанні даної лабораторної роботи я вивчила та застосувала алгоритм «Природне злиття» для сортування випадково генерованої множини чисел. Також зробила модифікацію програми даного алгоритму для більш ефективного сортування даних розміром не менше 1 Гб.

В ході лабораторної роботи:

1. записала псевдокод алгоритму сортування чисел у файлі та злиття даних 2 файлів;
2. виконала програмну реалізацію (не модифіковану) для 100 Мб:

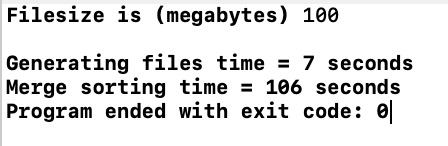
Маємо:

* час генерації = 1хв 51с;
* час сортування = 14хв 1с;
* робота з текстовими файлами.

1. виконала програмну модифіковану реалізацію:

Немодифікований метод базується на природньому злитті. Через випадкову генерацію кількість серій збільшується. Отже встановимо %8 від 8Гб (оперативна пам’ять) = 640Мб, та відсортуємо кожну частину файлу розміром 640Мб. Оскільки усі числа послідовності типу int, то максимальна кількість цифр яку ми можемо обробити і в подальшому помістити в бінарний файл == 640/4 = 160Мб. Після відсортування окремих частин маємо кількість серій еквівалентну кількості файлів, тобто: ПочатковоЗаданийРозмірФайлу / 640 (та +1 якщо є залишок при діленні); Далі слідує злиття кожних останніх двох серій, в результаті чого видаляється остання серія, а відсортована послідовність повністю записується у попередню. В ході даної операції не видаленим залишається лише файл “1.txt” (1-а серія). Її вміст копіюється у звичайний текстовий файл для читабельності результату.

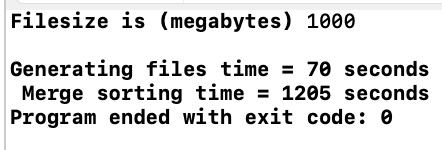
Результати для 100Мб:



Порівнюючи із немодифікованим методом зменшився:

* час генерації на 1хв 44с;
* час сортування на 12хв 15с;
* робота з бінарними файлами.

Результати для 1 Гб:



Маємо:

* час генерації = 1хв 10с,
* час сортування = 20хв 5с;

Бачимо, що час генерації файлів та послідовності бінарних файлів є більш швидкою порівнюючи час для 100Мб: 1хв 51с та 7с.

По усім параметрам модифікований метод пришвидшив час виконання алгоритму в 15,9 разів для генерації послідовності та формування файлів; та в 7,9 разів для злиття самих файлів.

Як висновок можна сказати, що для більш ефективних програм важливо розуміти які алгоритми правильніше використовувати в залежності від задачі та вміти їх комбінувати між собою.