**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНТЕЛЕКТУАЛЬНИХ**

**ТЕХНОЛОГІЙ І ЗВ’ЯЗКУ**

**Звіт**

**з дисципліни Технологіх Розподілених Систем та Паралельних Обчислень**

**Лабораторна робота №3**

Виконав: студент 3 курсу, групи ІПЗ-4.04 спеціальності

121 Інженерія програмного забезпечення

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Бухта М.М.

Перевірив\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Развалінов В.Ю.

**Одеса  2024**

**Опис завдання:**

1. Розробити класи у відповідності до варіанту завдання.
2. Створити компаратори класу за різними можливими способами сортування екземплярів класу.
3. Створити колекцію екземплярів класу та виконати сортування елементів колекції з використанням алгоритмів звичайного сортування та сортування з паралельними обчисленнями, реалізованих у стандартних бібліотеках Java. Використовуйте як сортування за одним критерієм, так і за кількома одночасно. Наприклад, відсортувати готелі міста за категорією та за назвою міста розташування.
4. Збільшуючи обсяг масиву, що сортується, вимірювати час виконання програми. Побудувати графік залежності часу виконання програми від розміру масиву для методу звичайного сортування та сортування з паралельними обчисленнями. Порівняти результати та зробити висновки. **Код програми:**

**main.cpp**

#include "Student.hpp"

#include <algorithm>

#include <iostream>

#include <numeric>

#include <queue>

#include <random>

#include <thread>

template <typename Arr> void parallelSorting(Arr &array) {

auto threadsCount = std::thread::hardware\_concurrency();

auto arrSize = array.size();

auto chunkSize = arrSize / threadsCount;

auto arrBegin = std::begin(array);

std::queue<std::thread> threads;

for (std::int32\_t i{0}; i < threadsCount; ++i) {

auto start = i \* chunkSize;

auto end = i == threadsCount - 1 ? arrSize : start + chunkSize;

threads.emplace(std::thread([&arrBegin, start, end]() {

std::sort(arrBegin + start, arrBegin + end);

}));

}

// std::sort(arrBegin, arrBegin + arrSize);

for (std::int32\_t i{0}; i < threadsCount; ++i) {

threads.front().join();

threads.pop();

}

std::int32\_t step = chunkSize;

while (step < arrSize) {

for (std::int32\_t i = 0; i < arrSize - step; i += 2 \* step) {

std::int32\_t left = i;

std::int32\_t mid = i + step;

std::int32\_t right = std::min<std::int32\_t>(i + 2 \* step, arrSize);

std::inplace\_merge(array.begin() + left, array.begin() + mid,

array.begin() + right);

}

step \*= 2;

}

}

template <typename Arr> void commonSorting(Arr &array) {

std::sort(std::begin(array), std::end(array));

}

template <typename Arr, typename SortFunction>

void sortTest(Arr &array, SortFunction sortMethod) {

auto start = std::chrono::steady\_clock::now();

sortMethod(array);

auto end = std::chrono::steady\_clock::now();

std::cout << "Время выполнения: "

<< std::chrono::duration\_cast<std::chrono::milliseconds>(end -

start)

.count()

<< " миллисекунд" << std::endl;

if (std::is\_sorted(std::begin(array), std::end(array)))

std::cout << "Массив отсортирован";

else

std::cout << "Не отсортирован массив";

std::cout << std::endl;

}

int main(int argc, char \*\*argv) {

for (size\_t i = 1; i <= 10; ++i) {

size\_t sizeVector = i \* (size\_t)1e7;

std::vector<Student> arr(sizeVector);

std::for\_each(std::begin(arr), std::end(arr),

[i = 0.01](Student &elem) mutable {

elem.setAvarageGrade(i);

i += 0.01;

});

std::mt19937\_64 urng{121216};

std::shuffle(arr.begin(), arr.end(), urng);

std::cout << "Количество элементов: " << sizeVector << std::endl;

std::cout << "Сортировка с параллельными вычислениями:" << std::endl;

sortTest(arr, parallelSorting<std::vector<Student>>);

std::shuffle(arr.begin(), arr.end(), urng);

std::cout << "Сортировка в одном потоке:" << std::endl;

sortTest(arr, commonSorting<std::vector<Student>>);

std::cout << std::endl;

}

return 0;

}

**Student.hpp**

#ifndef \_\_LAB\_3\_MULTYTHREADING\_STUDENT\_HPP\_\_

#define \_\_LAB\_3\_MULTYTHREADING\_STUDENT\_HPP\_\_

#include "common.hpp"

class Student {

public:

Student() = default;

Student(common::FullName fullName, float avarageGrade);

bool operator==(const Student &other) const;

bool operator!=(const Student &other) const;

bool operator>(const Student &other) const;

bool operator<(const Student &other) const;

common::FullName fullName() const;

void setFullName(common::FullName &val);

void setFullName(common::FullName &&val);

float avarageGrade() const noexcept;

void setAvarageGrade(float val) noexcept;

private:

common::FullName \_fullName;

float \_avarageGrade;

};

#endif // \_\_LAB\_3\_MULTYTHREADING\_STUDENT\_HPP\_\_;

**Student.cpp**

#include "Student.hpp"

Student::Student(common::FullName fullName, float avarageGrade)

: \_fullName{fullName}, \_avarageGrade{avarageGrade} {}

bool Student::operator==(const Student &other) const {

return this->\_avarageGrade == other.\_avarageGrade &&

this->\_fullName == other.\_fullName;

}

bool Student::operator!=(const Student &other) const {

return !operator==(other);

}

bool Student::operator>(const Student &other) const {

return this->\_fullName > other.\_fullName ||

(this->\_fullName == other.\_fullName &&

this->\_avarageGrade > other.\_avarageGrade);

}

bool Student::operator<(const Student &other) const {

return this->\_fullName < other.\_fullName ||

(this->\_fullName == other.\_fullName &&

this->\_avarageGrade < other.\_avarageGrade);

}

common::FullName Student::fullName() const { return \_fullName; }

void Student::setFullName(common::FullName &val) { \_fullName = val; }

void Student::setFullName(common::FullName &&val) {

\_fullName = std::move(val);

}

float Student::avarageGrade() const noexcept { return \_avarageGrade; }

void Student::setAvarageGrade(float val) noexcept { \_avarageGrade = val; }

**common.hpp**

#ifndef \_\_LAB\_3\_MULTYTHREADING\_COMMON\_HPP\_\_

#define \_\_LAB\_3\_MULTYTHREADING\_COMMON\_HPP\_\_

#include <string>

namespace common {

struct FullName {

std::string name;

std::string surname;

std::string middleName;

public:

bool operator==(const FullName &other) const;

bool operator!=(const FullName &other) const;

bool operator>(const FullName &other) const;

bool operator<(const FullName &other) const;

};

} // namespace common

#endif // \_\_LAB\_3\_MULTYTHREADING\_COMMON\_HPP\_\_;

**common.cpp**

#include "common.hpp"

namespace common {

bool FullName::operator==(const FullName &other) const {

return this->name == other.name && this->surname == other.surname &&

this->middleName == other.middleName;

}

bool FullName::operator!=(const FullName &other) const {

return !operator==(other);

}

bool FullName::operator>(const FullName &other) const {

return this->surname > other.surname && this->name > other.name &&

this->middleName > other.middleName;

}

bool FullName::operator<(const FullName &other) const {

return this->surname < other.surname && this->name < other.name &&

this->middleName < other.middleName;

}

} // namespace common

**Результат виконання програми:**

Рисунок 1 – результат обчислень (на більше не вистачає ОЗУ)

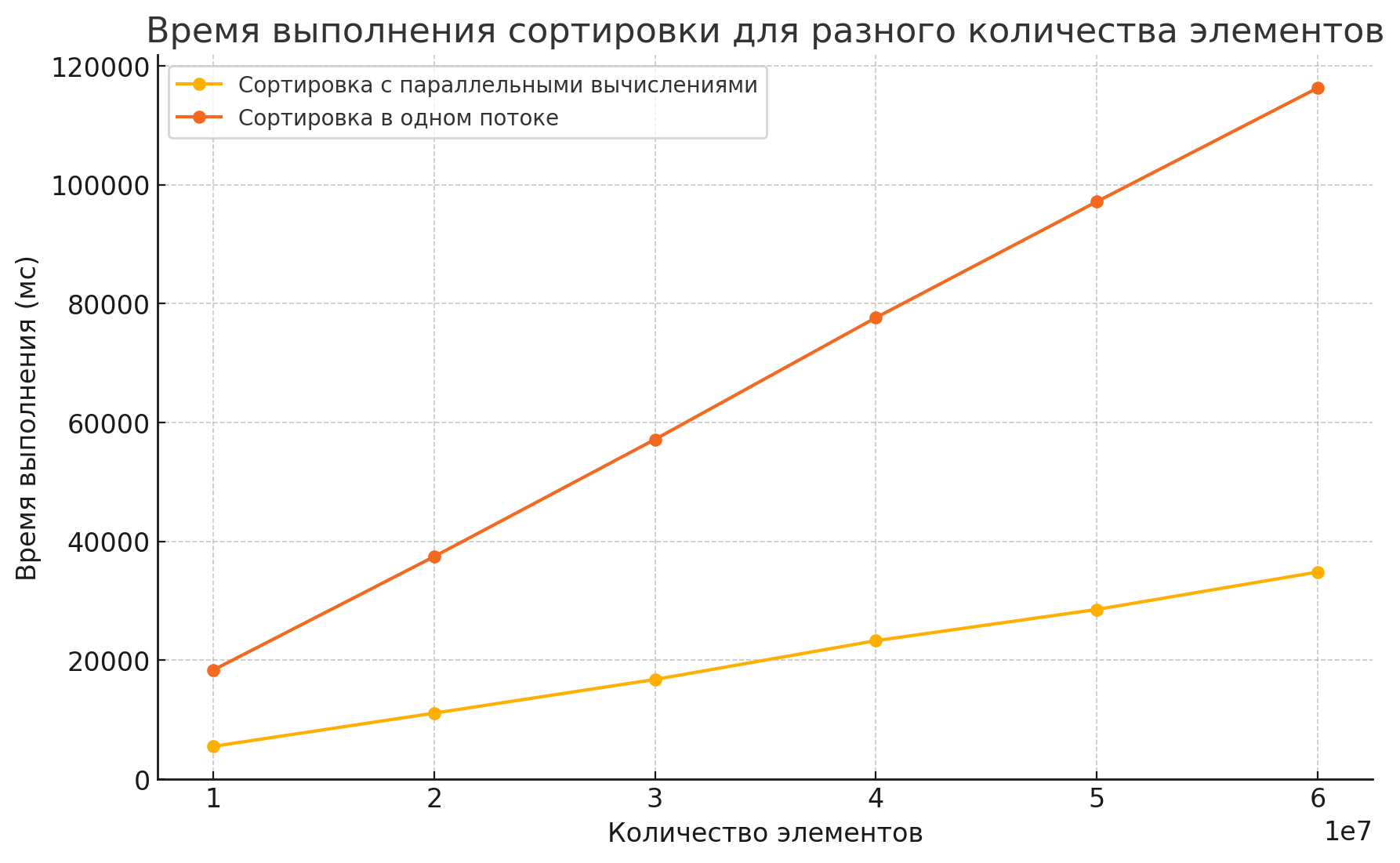
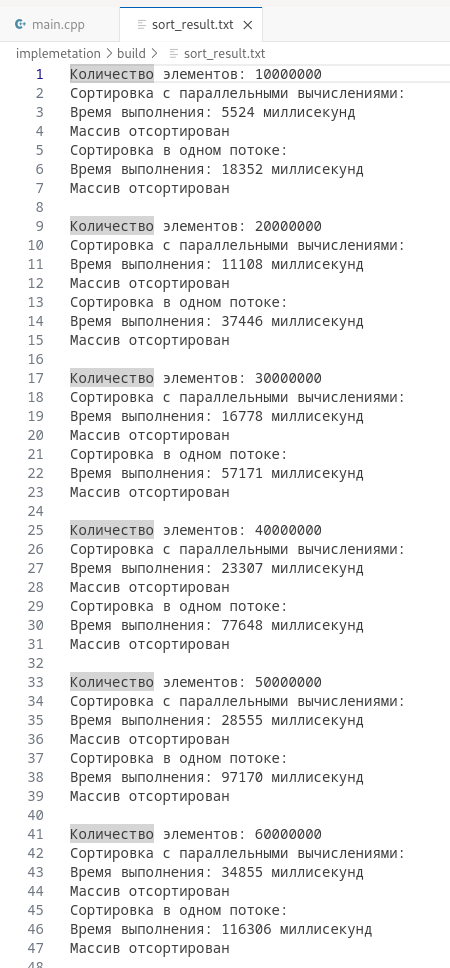


Рисунок 2 – графік обчислень кол елементів . Кол мс