Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Российский химико-технологический университет имени Д.И. Менделеева»

Кафедра информационных компьютерных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 1

Выполнил студент группы КС-38 (Нергарян Геворг Гарегинович)

Ссылка на репозиторий: (https://github.com/MUCTR-IKT-CPP/Nergaryan\_KC-38\_Algos)

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: …………………….10.03.2023…………………………………………………………………..

Оглавление

[Описание задачи. 2](#_Toc63548272)

[Описание метода/модели. 2](#_Toc63548273)

[Выполнение задачи. 3](#_Toc63548274)

[Заключение. 8](#_Toc63548275)

# Описание задачи.

Вариант 1.

Необходимо реализовать метод быстрой сортировки.

Для реализованного метода сортировки необходимо провести серию тестов для всех значений N из списка (1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000), при этом:

* в каждом тесте необходимо по 20 раз генерировать вектор, состоящий из N элементов
* каждый элемент массива заполняется случайным числом с плавающей запятой от -1 до 1

На основании [статьи](https://neerc.ifmo.ru/wiki/index.php?title=%D0%91%D1%8B%D1%81%D1%82%D1%80%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0#.D0.A1.D0.BF.D0.BE.D1.81.D0.BE.D0.B1_.D0.BF.D0.BE.D1.81.D1.82.D1.80.D0.BE.D0.B8.D1.82.D1.8C_.D0.BC.D0.B0.D1.81.D1.81.D0.B8.D0.B2_.D1.81_.D0.BC.D0.B0.D0.BA.D1.81.D0.B8.D0.BC.D0.B0.D0.BB.D1.8C.D0.BD.D1.8B.D0.BC_.D0.BA.D0.BE.D0.BB.D0.B8.D1.87.D0.B5.D1.81.D1.82.D0.B2.D0.BE.D0.BC_.D1.81.D1.80.D0.B0.D0.B2.D0.BD.D0.B5.D0.BD.D0.B8.D0.B9_.D0.BF.D1.80.D0.B8_.D0.B2.D1.8B.D0.B1.D0.BE.D1.80.D0.B5_.D1.81.D1.80.D0.B5.D0.B4.D0.BD.D0.B5.D0.B3.D0.BE_.D1.8D.D0.BB.D0.B5.D0.BC.D0.B5.D0.BD.D1.82.D0.B0_.D0.B2_.D0.BA.D0.B0.D1.87.D0.B5.D1.81.D1.82.D0.B2.D0.B5_.D0.BE.D0.BF.D0.BE.D1.80.D0.BD.D0.BE.D0.B3.D0.BE) реализовать проверки негативных случаев и устроить на них серии тестов аналогичные второму пункту:

* Отсортированный массив
* Массив с одинаковыми элементами
* Массив с максимальным количеством сравнений при выборе среднего элемента в качестве опорного
* Массив с максимальным количеством сравнений при детерминированном выборе опорного элемента

При работе сортировки подсчитать количество вызовов рекурсивной функции, и высоту рекурсивного стека. Построить график худшего, лучшего, и среднего случая для каждой серии тестов.

Для каждой серии тестов построить график худшего случая.

Подобрать такую константу c, что бы график функции c \* n \* log(n) находился близко к графику худшего случая, если возможно построить такой график.

Проанализировать полученные графики и определить есть ли на них следы деградации метода относительно своей средней сложности.

# Описание метода/модели.

Метод быстрой сортировки (также известный как quicksort) - это алгоритм сортировки, который использует стратегию "разделяй и властвуй".

Он работает следующим образом:

1. Выбирается опорный элемент из массива (обычно это первый, последний или средний элемент).
2. Остальные элементы массива сравниваются с опорным элементом и разделяются на две группы: элементы, меньшие опорного, и элементы, большие опорного.
3. Рекурсивно повторяется для каждой из двух групп, пока каждый элемент не станет отдельным подмассивом.
4. Результаты объединяются, чтобы получить отсортированный массив.

Быстрая сортировка - один из самых эффективных алгоритмов сортировки. Он имеет среднее время выполнения O(n log n) и часто используется в программных библиотеках и приложениях. Однако в худшем случае время выполнения может быть O(n^2), если выбранный опорный элемент оказывается наименьшим или наибольшим элементом в массиве, что приводит к неэффективной работе алгоритма.

# Выполнение задачи.

Использовался язык C++. Время измеряли в микросекундах.

Код:

#include <iostream>

#include <vector>

#include <algorithm>

#include <chrono>

#include <cassert>

#include <cmath>

using namespace std;

vector<double> quicksort(vector<double>& arr, int& kolvo\_recurs, int& glubina\_steka, int cur\_depth = 0) { //cur\_depth используется для отслеживания текущей глубины рекурсии.

kolvo\_recurs++;

if (arr.size() <= 1) { //если размер массива не больше 1, то он уже отсортирован и возвращается без изменений

return arr;

}

double pivot = arr[arr.size() / 2]; //опорный элемент (pivot) из середины массива

vector<double> left, middle, right; //три вектора для элементов, которые меньше, равны и больше опорного элемента

for (double x : arr) {

if (x < pivot) {

left.push\_back(x);

}

else if (x == pivot) {

middle.push\_back(x);

}

else {

right.push\_back(x);

}

}

int cur\_depth\_left = cur\_depth + 1;

vector<double> sorted\_left = quicksort(left, kolvo\_recurs, glubina\_steka, cur\_depth\_left);

glubina\_steka = max(glubina\_steka, cur\_depth\_left); //рекурсивно вызывает саму себя для двух подмассивов, созданных из элементов, которые меньше и больше опорного элемента

int cur\_depth\_right = cur\_depth + 1;

vector<double> sorted\_right = quicksort(right, kolvo\_recurs, glubina\_steka, cur\_depth\_right);

glubina\_steka = max(glubina\_steka, cur\_depth\_right);

vector<double> result;

result.reserve(sorted\_left.size() + middle.size() + sorted\_right.size());

result.insert(result.end(), sorted\_left.begin(), sorted\_left.end());

result.insert(result.end(), middle.begin(), middle.end());

result.insert(result.end(), sorted\_right.begin(), sorted\_right.end());

return result; // функция объединяет три отсортированных вектора (левый, средний и правый) в один и возвращает его

}

int main() {

setlocale(LC\_ALL, "Russian");

int min\_r, avg\_r, max\_r; // переменные для подсчета количества вызовов

vector<int> N\_values = { 1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000 };

for (int N : N\_values) {

int glubina\_steka = 0, kolvo\_recurs = 0, time = 0;

min\_r = 9999999, avg\_r = 0, max\_r = 0;

for (int i = 0; i < 20; i++) {

auto start = std::chrono::high\_resolution\_clock::now();//начало времени

glubina\_steka = 0, kolvo\_recurs = 0;

vector<double> arr(N);

generate(arr.begin(), arr.end(), []() { return ((double)rand() / RAND\_MAX) \* 2 - 1; }); //используется лямбда-функция без аргументов, которая генерирует случайное число в диапазоне от -1 до 1

vector<double> sorted\_arr = quicksort(arr, kolvo\_recurs, glubina\_steka); //отсортированные по возрастанию

bool is\_sorted = is\_sorted\_until(sorted\_arr.begin(), sorted\_arr.end()) == sorted\_arr.end(); //Проверяется, отсортирован ли "sorted\_arr" по возрастанию

if (!is\_sorted) {

cout << "Сортировка для N=" << N << " и итерации " << i << " была неверной!" << endl;

}

// Отсортированный массив

vector<double> sorted\_arr\_copy = sorted\_arr;// создается копия отсортированного массива "sorted\_arr"

sort(sorted\_arr\_copy.begin(), sorted\_arr\_copy.end()); //снова сортировка

bool is\_sorted\_copy = is\_sorted\_until(sorted\_arr\_copy.begin(), sorted\_arr\_copy.end()) == sorted\_arr\_copy.end();

if (!is\_sorted\_copy) { //проверка

cout << "Негативный тест: Сортировка отсортированного массива для N=" << N << " и итерации " << i << " была неверной!" << endl;

}

// Массив с одинаковыми элементами

vector<double> same\_arr(N, 1.0);

vector<double> same\_arr\_sorted = quicksort(same\_arr, kolvo\_recurs, glubina\_steka);

bool is\_same\_sorted = is\_sorted\_until(same\_arr\_sorted.begin(), same\_arr\_sorted.end()) == same\_arr\_sorted.end();

if (!is\_same\_sorted) {

cout << "Негативный тест: Сортировка массива с одинаковыми элементами для N=" << N << " и итерации " << i << " была неверной!" << endl;

}

// Массив с максимальным количеством сравнений при выборе среднего элемента в качестве опорного

vector<double> max\_comparison\_arr(N);

generate(max\_comparison\_arr.begin(), max\_comparison\_arr.end(), []() { return 1.0; }); //лямбда-функция без аргументов, которая всегда возвращает 1.0

vector<double> max\_comparison\_sorted\_arr = quicksort(max\_comparison\_arr, kolvo\_recurs, glubina\_steka); //сортировка по возрастанию

bool is\_max\_comparison\_sorted = is\_sorted\_until(max\_comparison\_sorted\_arr.begin(), max\_comparison\_sorted\_arr.end()) == max\_comparison\_sorted\_arr.end(); //Проверяется, отсортирован ли вектор по возрастанию

if (!is\_max\_comparison\_sorted) {

cout << "Негативный тест: Сортировка массива с максимальным количеством сравнений для N=" << N << " и итерации " << i << " была неверной!" << endl;

}

//Массив с максимальным количеством сравнений при детерминированном выборе опорного элемента

vector<double> dete\_arr(N);

generate(dete\_arr.begin(), dete\_arr.end(), []() { return ((double)rand() / RAND\_MAX) \* 2 - 1; });

vector<double> dete\_sorted\_arr = quicksort(dete\_arr, kolvo\_recurs, glubina\_steka);

bool dete\_is\_sorted = is\_sorted\_until(dete\_sorted\_arr.begin(), dete\_sorted\_arr.end()) == dete\_sorted\_arr.end();

if (!dete\_is\_sorted) {

cout << "Негативный тест: Сортировка массива с детерминированным выбором опорного элемента для N=" << N << " была неверной!" << endl;

}

if (min\_r > kolvo\_recurs) { min\_r = kolvo\_recurs; }

if (max\_r < kolvo\_recurs) { max\_r = kolvo\_recurs; }

avg\_r += kolvo\_recurs;

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

auto duration = std::chrono::duration\_cast<std::chrono::microseconds>(end - start);

if (time < duration.count()) { time = duration.count(); }

}

cout << "Время худшего выполнения: " << time << " микросекунд" << endl;

cout << "Лучший случай:" << min\_r << " Средний случай:" << avg\_r/20 << " Худший случай:" << max\_r << endl;

cout << "Глубина стека для N=" << N << ":" << glubina\_steka << " Количество рекурсий для N=" << N << ":" << avg\_r << endl;

cout << "Тесты для N=" << N << " успешно пройдены!" << endl << endl << endl;

}

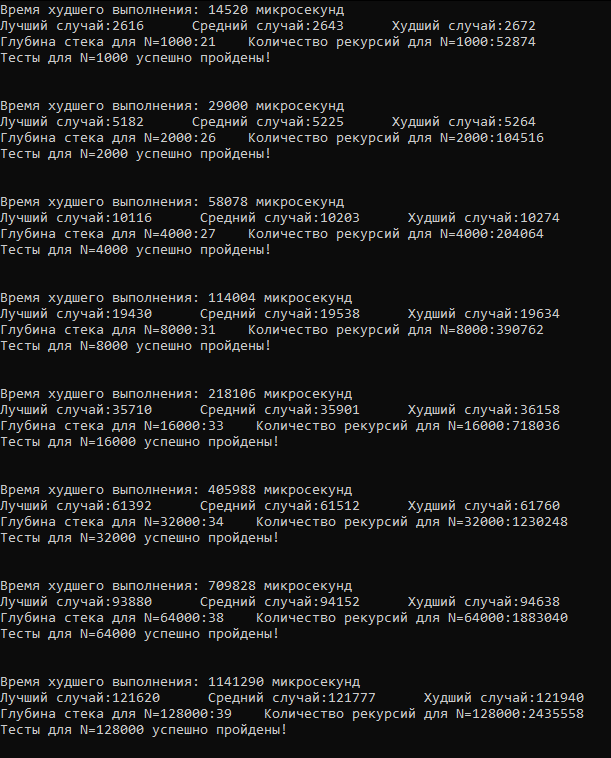
return 0;

}

Реализован метод быстрой сортировки для всех значений N. В каждом тесте генерируется 20 векторов, заполненных случайным числом с плавающей запятой от -1 до 1. Провел негативные тесты по тому же алгоритму.

Вывод:

Программа выводит худшее время теста для каждого N, минимальное, среднее и максимальное количество вызова рекурсии для теста, для всего N и высоту рекурсивного стека.



Графики:

График худшего, лучшего, и среднего случая для каждой серии тестов. Из-за незначительности отличий на графике хорошо видно только худший случай, но остальные за ней.



График худшего случая по времени для каждого теста. График функции c \* n \* log(n) и сама константа.



# Заключение.

Если данные, которые нужно отсортировать, уже упорядочены или имеют определенный порядок, то быстрая сортировка может замедлиться и работать со сложностью O(n^2), что значительно хуже, чем средняя сложность O(n log n). Это происходит потому, что в таких случаях первоначально выбранный опорный элемент может оказаться крайним элементом в массиве, и тогда каждая итерация сортировки будет уменьшать размер обрабатываемого массива всего на один элемент. Однако, если опорный элемент выбирается случайным образом, вероятность того, что массив уже отсортирован или почти отсортирован, очень мала, и быстрая сортировка будет работать со средней сложностью O(n log n).