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 № 2

Выполнил студент группы КС-38 (Казанцев Леонард Антонович)

Ссылка на репозиторий: (https://github.com/SimpleMaking/Algorithms-6-sem-/tree/main/Lab\_2(сортировка\_part2))

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Дата сдачи: (17.02.2023)

Оглавление

[Описание задачи. 2](#_Toc127270410)

[Описание метода/модели. 3](#_Toc127270411)

[Выполнение задачи. 4](#_Toc127270412)

[Заключение. 4](#_Toc127270413)

# Описание задачи.

В данной лабораторной работе необходимо было реализовать метод быстрой сортировки. Эта сортировка считается одной из самых быстрых.

Помимо реализации еще нужно было провести тесты работы алгоритма 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 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.

По окончанию всех тестов необходимо нанести все точки, полученные в результате замеров времени на график где на ось абсцисс(Х) нанести N, а на ось ординат(Y) нанести значения времени на сортировку. По полученным точкам построить график лучшего (минимальное количество вызовов рекурсивной функции для каждого N), худшего (максимальное минимальное количество вызовов рекурсивной функции для каждого N) и среднего (среднее минимальное количество вызовов рекурсивной функции для каждого N) случая.

В качестве дополнительного задания, необходимо построить график худшего случая, и график c \* n \* log(n), где n \* log(n) соответствует асимптотической сложности рассматриваемого метода сортировки, подобрав такое значение C, что бы график асимптотической сложности находился близко к графику худшего случая, если возможно построить такой график.

# Описание метода/модели.

Быстрая сортировка

**Быстрая сортировка** (англ. *quick sort*, сортировка Хоара) — один из самых известных и широко используемых алгоритмов сортировки. Среднее время работы O(n\*log(n)), что является асимптотически оптимальным временем работы для алгоритма, основанного на сравнении. Хотя время работы алгоритма для массива из n элементов в худшем случае может составить , на практике этот алгоритм является одним из самых быстрых.

# Выполнение задачи.

Для решения поставленной задачи были использованы языки программирования С++ и Python. Первый использовался для получения максимального, минимального и среднего количества вызовов рекурсивной функции для каждого из 8 проведенных тестов. В каждом тесте производилось 20 попыток.

Алгоритм:

Функция алгоритма quick\_sort принимает первым аргументом значение для левой части отрезка, вторым аргументом – для правой части.

В самой функции:

* Выбирается из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.
* Сравнивается все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на два непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные и большие».
* Для отрезков «меньших» и «больших» значений выполняется рекурсивно та же последовательность операций, если длина отрезка больше единицы.

Код функции:

int quick\_sort(int S, int P)

{

int k1 = S, k2 = P;

int recursion\_count = 0;

double mid = ::data[(k1 + k2) / 2];

while (k1 <= k2)

{

while (::data[k1] < mid)

++k1;

while (::data[k2] > mid)

--k2;

if (k1 <= k2)

{

swap(::data[k1], ::data[k2]);

++k1;

--k2;

}

}

if (S < k2)

{

recursion\_count += quick\_sort(S, k2);

recursion\_count += 1;

}

if (P > k1)

{

recursion\_count += quick\_sort(k1, P);

recursion\_count += 1;

}

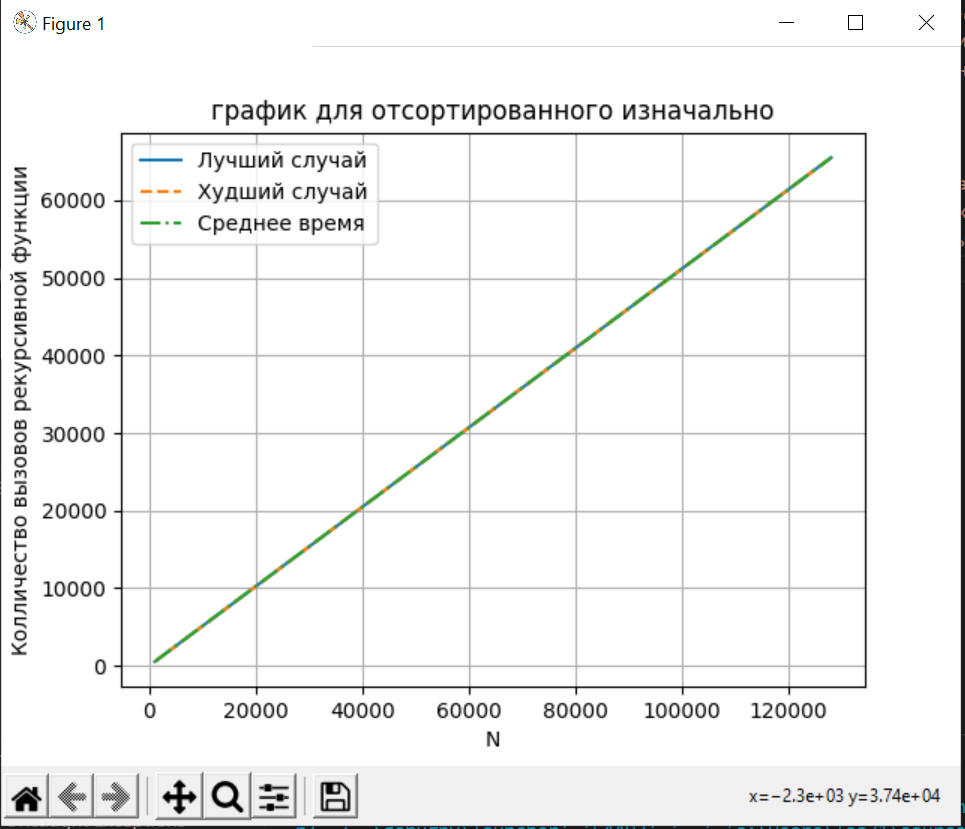
return recursion\_count;

}

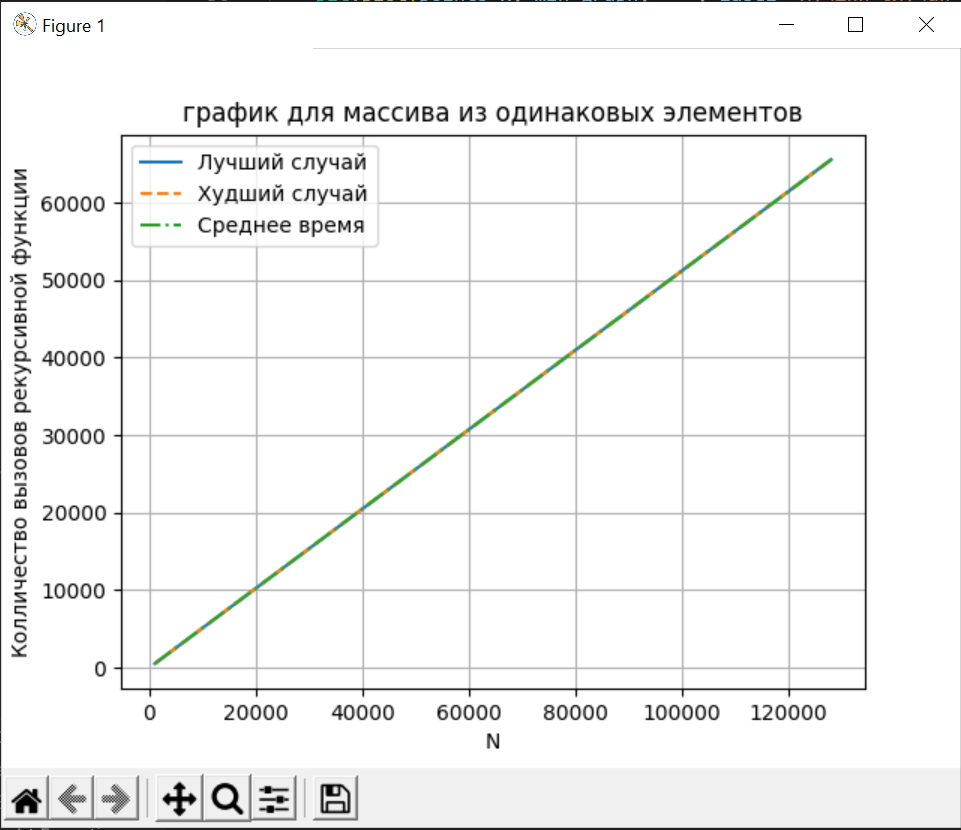
Также были проведены тесты для разных массивов:

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

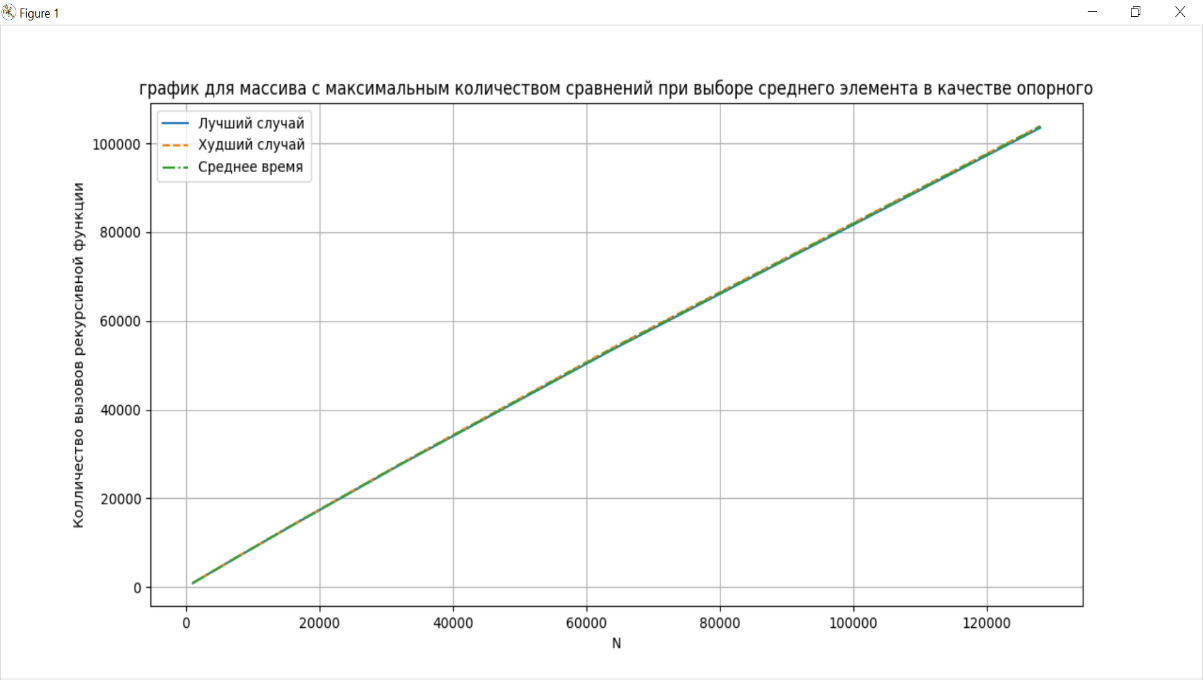
Вывод графиков для мин. макс. и среднего значений всех тестов осуществлялся через скрипт graphics.py:



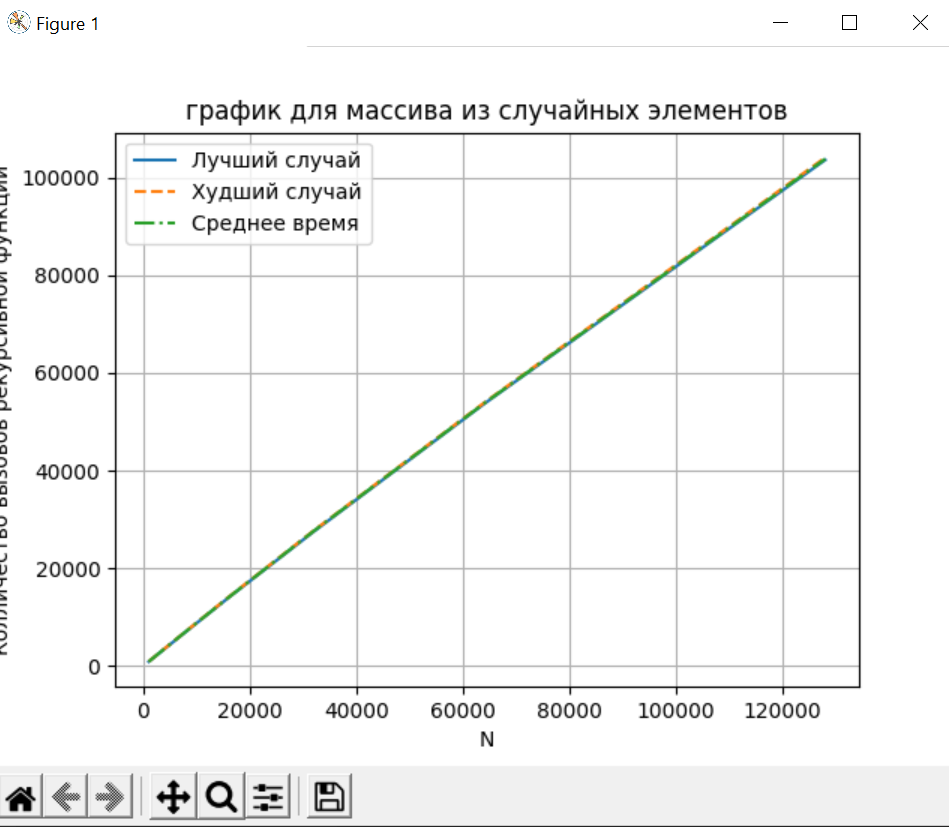
Массив, отсортированный изначально



Массив из одинаковых элементов



Массив с макс. количеством сравнений при выборе среднего элемента в качестве опорного



Массив из случайных элементов

Из графиков видно, что для массива из случайных элементов различие для трех функций наиболее заметно. Чем больше элементов в массиве, тем больше различие между функциями. У первых двух массивов логично все функции идентичны, у третьего – едва заметные различия к большему числу элементов.

# Заключение.

Алгоритм является одним из самых эффективных по времени и по памяти, так как в его реализации используется рекурсивные вызовы. Также он довольно прост в реализации. Деградации метода не заметил, чем больше элементов, тем больше вызовов рекурсивной функции. Использование данного алгоритма сортировки в его рекурсивной реализации дает преимущество по времени работы по сравнению с другими алгоритмами.