**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-36 Перминова П.А.

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Дата сдачи:

**Оглавление**

[Описание задачи](#_ogxk3hnt0eu6) **3**

[Описание метода/модели](#_k3f55hfgbc2p) **4**

[Выполнение задачи](#_45uzn5swogwx) **5**

[Заключение](#_irj83k67q3up) **6**

# Описание задачи

1. Необходимо реализовать метод быстрой сортировки.
2. 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

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

1. При работе сортировки подсчитать количество вызовов рекурсивной функции, и высоту рекурсивного стека. Построить график худшего, лучшего, и среднего случая для каждой серии тестов.
2. Для каждой серии тестов построить график худшего случая.
3. Подобрать такую константу c, чтобы график функции c \* n \* log(n) находился близко к графику худшего случая, если возможно построить такой график.
4. Проанализировать полученные графики и определить есть ли на них следы деградации метода относительно своей средней сложности.

# Описание метода/модели

Быстрая сортировка (англ. quicksort) — алгоритм сортировки, разработанный английским информатиком Тони Хоаром, является одним из самых быстрых известных универсальных алгоритмов сортировки массивов, но из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

QuickSort - это существенно улучшенный вариант алгоритма сортировки с помощью прямого обмена (его варианты известны как «Пузырьковая сортировка» и «Шейкерная сортировка»), известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы.

Общая идея алгоритма состоит в следующем:

1. Выбрать 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.
2. Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».
3. Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее.

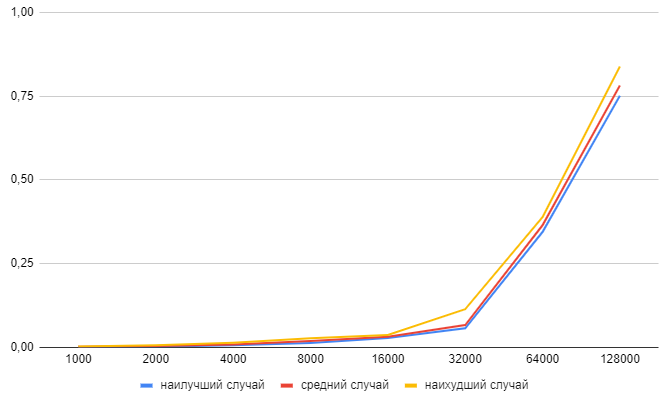
# Выполнение задачи

Программа алгоритма написана на языке Python.

Далее представлены полученные в ходе выполнения программы данные о скорости ее выполнения: количество элементов, минимальное, среднее и максимальное затраченное время на сортировку. При работе сортировки также были подсчитаны количество вызовов рекурсивной функции.

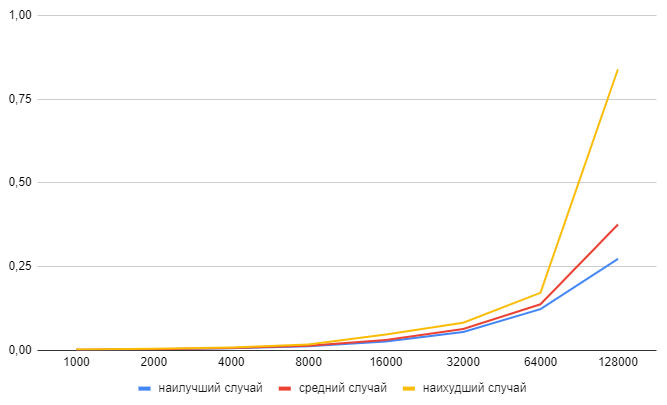
|  | min | average | max | recursion |
| --- | --- | --- | --- | --- |
| 1000 | 0,000993013 | 0,001484406 | 0,0022614 | 26778 |
| 2000 | 0,002000093 | 0,004066384 | 0,006396532 | 80224 |
| 4000 | 0,005993843 | 0,008242846 | 0,013342381 | 187120 |
| 8000 | 0,01328826 | 0,018629301 | 0,027470827 | 400382 |
| 16000 | 0,027551174 | 0,031752074 | 0,036988258 | 827036 |
| 32000 | 0,057166576 | 0,066479003 | 0,113755226 | 1679774 |
| 64000 | 0,344644785 | 0,364630616 | 0,388974428 | 3386422 |
| 128000 | 0,751099825 | 0,781695664 | 0,838529825 | 6799888 |

Также построен график худшего, лучшего, и среднего случая для каждой серии тестов.



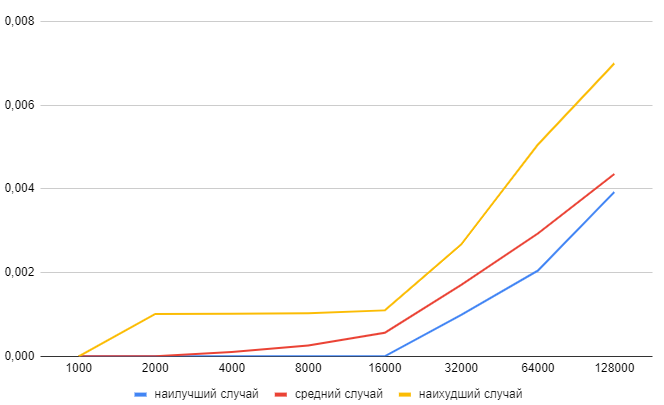
На представленных ниже данных можно наблюдать работу быстрой сортировки на заранее отсортированном массиве с использованием библиотечной функции Python sorted().

|  | min | average | max | recursion |
| --- | --- | --- | --- | --- |
| 1000 | 0,000999212 | 0,001573515 | 0,002485991 | 26764 |
| 2000 | 0,002991438 | 0,003356433 | 0,004639864 | 80124 |
| 4000 | 0,005927324 | 0,006831205 | 0,008538485 | 186866 |
| 8000 | 0,0125315192 | 0,013853276 | 0,01721096 | 400118 |
| 16000 | 0,026157856 | 0,030572331 | 0,047008276 | 826740 |
| 32000 | 0,054726362 | 0,06358906 | 0,082612276 | 1680320 |
| 64000 | 0,122895718 | 0,137383604 | 0,172170639 | 3386914 |
| 128000 | 0,272862911 | 0,375710559 | 0,838680029 | 6799638 |



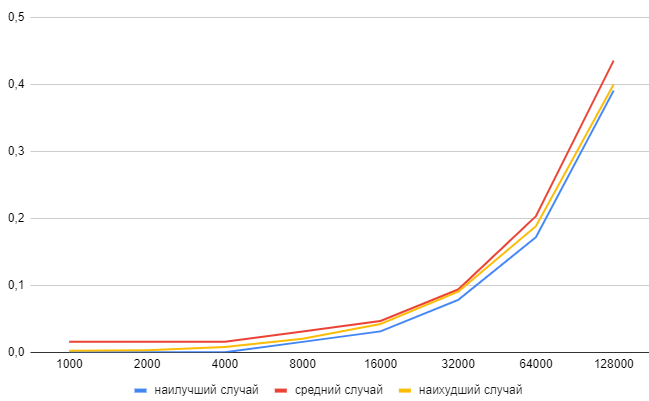
Далее представлены данные по массиву, который содержит в себе одинаковые элементы. На этих данных мы можем заметить, что количество рекурсивных вызовов функции возрастает в арифметической прогрессии. Это происходит из-за того, что количество элементов массива возрастает всегда вдвое, но сами элементы остаются неизменными.

|  | min | average | max | recursion |
| --- | --- | --- | --- | --- |
| 1000 | 0 | 0 | 0 | 60 |
| 2000 | 0 | 0 | 0,00101258102 | 120 |
| 4000 | 0 | 0,0001018526 | 0,00101503642 | 180 |
| 8000 | 0 | 0,000260782 | 0,00102663 | 240 |
| 16000 | 0 | 0,000561106 | 0,001097441 | 300 |
| 32000 | 0,000992537 | 0,001706525 | 0,002676487 | 360 |
| 64000 | 0,002048016 | 0,002934049 | 0,0050532822 | 420 |
| 128000 | 0,003921509 | 0,004355163 | 0,00699544 | 480 |

График, построенный на полученных данных:

Далее представлены данные, полученные при выборе среднего элемента в качестве опорного.

|  | min | average | max | recursion |
| --- | --- | --- | --- | --- |
| 1000 | 0 | 0,01562666893 | 0,002343916893 | 26610 |
| 2000 | 0 | 0,01562738419 | 0,003125286102 | 79980 |
| 4000 | 0 | 0,0156276226 | 0,007813036442 | 186852 |
| 8000 | 0,01562523842 | 0,03125286102 | 0,02031399012 | 400050 |
| 16000 | 0,03125214577 | 0,0468788147 | 0,04219071865 | 826544 |
| 32000 | 0,07813048363 | 0,09376525879 | 0,09054057598 | 1679402 |
| 64000 | 0,1718890667 | 0,2031404972 | 0,1883493662 | 3385974 |
| 128000 | 0,3906531334 | 0,4355163574 | 0,3997617722 | 6799694 |



# Заключение

В ходе выполнения лабораторной работы были сделаны следующие выводы: если в качестве опорного элемента использовать первый элемент массива, то время выполнения сортировки в среднем составляет O(n logn),а в наихудшем случае время работы составляет O(n2). Но при выборе среднего элемента массива в качестве опорного элемента можно наблюдать ускорение работы сортировки, так как время работы алгоритма становится, как для наилучшего случая, O(n logn).