**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-30 Лобачев Дмитрий Сергеевич

Ссылка на репозиторий: https://github.com/MUCTR-IKT-CPP/DSLobachev\_30/blob/main/Algorithms/Laba2/Laba2.cpp

Приняли: Пысин Максим Дмитриевич

Краснов Дмитрий Олегович

Лобанов Алексей Владимирович

Крашенинников Роман Сергеевич

Дата сдачи: 06.03.2023

**Оглавление**

[Описание задачи. 3](#_Toc129215044)

[Описание метода/модели. 4](#_Toc129215045)

[Выполнение задачи. 5](#_Toc129215046)

[Заключение. 24](#_Toc129215047)

# Описание задачи.

Необходимо реализовать метод быстрой сортировки.

Для реализованного метода сортировки необходимо провести серию тестов для всех значений N из списка (100, 200, 400, 800, 1600, 3200 ), 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

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, чтобы 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.

Быстрая сортировка является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена (Пузырьковая или шейкерная сортировки) 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 (таким образом улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов).

В связи с этим, быстрая сортировка является одним из самых быстрых алгоритмов сортировки массивов.

Общая идея алгоритма состоит в следующем:

* 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.
* 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».
* 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, так как упрощает алгоритм разделения.

# Выполнение задачи.

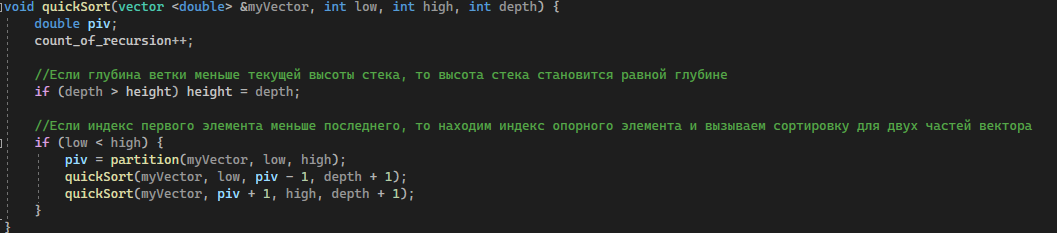
Для реализации данного метода сортировки использовался язык программирования C++.

**Алгоритм быстрой сортировки**

1. Функция quickSort принимает на вход вектор myVector элементов, индексы первого и последнего элементов вектора low и high, а также высоту данной ветки стека depth.

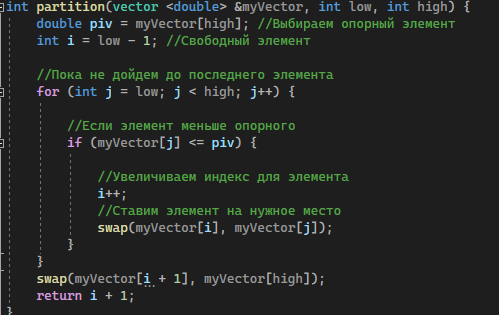
При заходе в функцию, мы увеличиваем значения счетчика количества рекурсий count\_of\_recursion и высоты стека height.

Затем происходит проверка, является ли индекс первого элемент меньше последнего, и если да, то с помощью функции partition находим индекс опорного элемента. После этого вызываются две рекурсивные функции сортировки для двух частей вектора (до опорного элемента и после него).



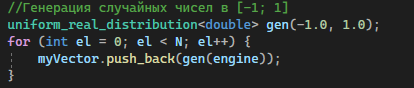
1. Функция partition принимает на вход вектор элементов myVector, индексы первого и последнего элементов low и high.

Выбирается значение опорного элемента piv. Затем идет перестановка элементов относительно опорного элемента. После итерации свободный элемент заменяется элементом с индексом i+1. Функция возвращает индекс опорного элемента.



**Начальные значения**

1. В качестве начальных значений были заданы:
   * Количество тестов M = 20;
   * Начальное количество элементов N = 100
2. Генерация вектора элементов



В качестве вектора из одинаковых элементов, был создан вектор из нулей.

1. Перед каждым тестом обнуляются значения высоты стека и количества рекурсий (depth, height и count\_of\_recursion). Задаются начальные значения индексов начального и конечного элементов сортировки (low = 0, high = myVector.size() - 1).

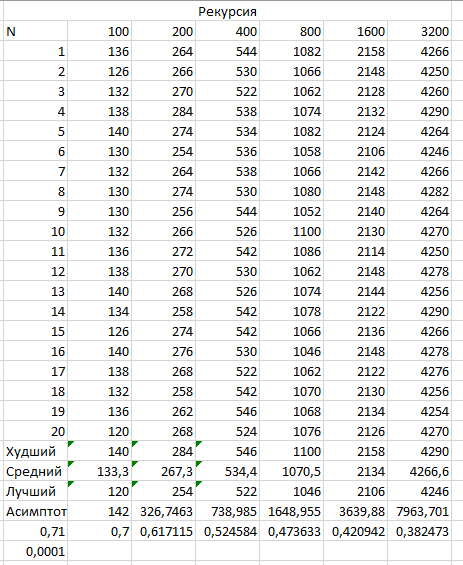
**Запись в файл**

1. С помощью функции fileWriter полученные результаты записываем в файл: затраченное на сортировку время в миллисекундах, количество рекурсий и максимальную высоту стека.

**Обычный вектор**

1. Вектор элементов был получен с помощью генерации случайных чисел от -1 до 1. Было проведено 20 тестов для различных размеров вектора (100, 200, 400, 800, 1600, 3200).
2. Количество рекурсий в зависимости от длины вектора.

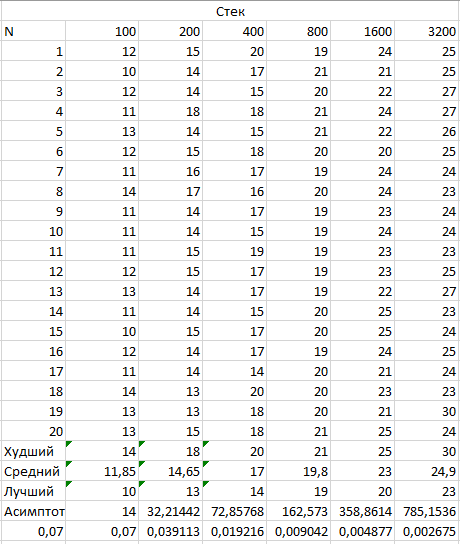
* Полученные результаты



* График худшего, среднего и лучшего случаев для количества рекурсий в зависимости от длины вектора.
* Был построен график асимптотической сложности с помощью формулы c\*O(N\*LOG(N)). Для данного худшего случая c = 0,71 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.

1. Высота стека в зависимости от длины вектора.

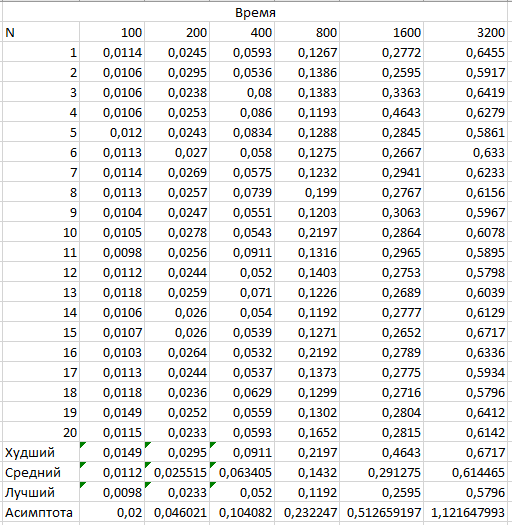
* Полученные результаты



* График худшего, среднего и лучшего случаев для высоты стека в зависимости от длины вектора.
* Был построен график асимптотической сложности с помощью формулы c\*O(N\*LOG(N)). Для данного худшего случая c = 0,07 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.

1. Время сортировки в зависимости от длины вектора.

* Полученные результаты



* График худшего, среднего и лучшего случаев времени сортировки в зависимости от длины вектора.
* Был построен график асимптотической сложности с помощью формулы c\*O(N\*LOG(N)). Для данного худшего случая c = 0,0001 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.

**Отсортированный вектор**

1. В качестве отсортированного вектора берется тот же вектор, который был создан ранее и и уже отсортирован. Было также проведено по 20 тестов для различных длин вектора.
2. Проанализировав получившиеся результаты, можно отметить, что количество рекурсий и максимальная высота стека полностью совпадает с сортировкой вектора с одинаковыми элементами. Из этого следует, что графики количества рекурсий и высоты стека в зависимости от количества элементов в векторе будут аналогичными, как и в векторе с одинаковыми элементами.

**Вектор с одинаковыми элементами**

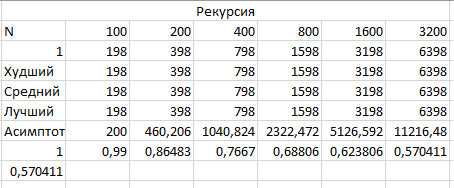
1. Вектор из нулей был получен с помощью начальной генерации вектора с N элементами. Было проведено по одному тесту для различных размеров вектора (100, 200, 400, 800, 1600, 3200).

В данном случае проведение нескольких тестов является несообразным, поскольку на высоту стека и количество рекурсий влияет рандомизация элементов вектора, а так как они все одинаковые, то значение на каждом тесте будет одно и то же. Поэтому было решено проводить по 1 тесту.

1. Количество рекурсий в зависимости от длины вектора.

* Полученные результаты

Из полученных результатов можно сказать, что количество рекурсий в зависимости от длины вектора N можно получить по формуле 2\*(N - 1).

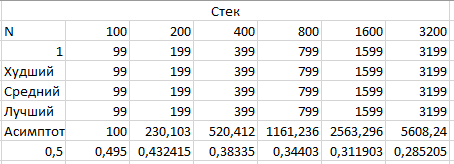


* График худшего, среднего и лучшего случаев для количества рекурсий в зависимости от длины вектора. Очевидно, что они будут полностью совпадать.
* Был построен график асимптотической сложности с помощью формулы c\*O(N\*LOG(N)). Для данного худшего случая c = 1 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.

1. Высота стека в зависимости от длины вектора.

* Полученные результаты

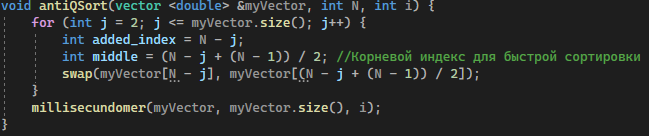
Из полученных результатов можно сказать, что высоту стека в зависимости от длины вектора N можно получить по формуле N - 1.



* График худшего, среднего и лучшего случаев для высоты стека в зависимости от длины вектора. Очевидно, что они будут полностью совпадать.
* Был построен график асимптотической сложности с помощью формулы c\*O(N\*LOG(N)). Для данного худшего случая c = 0,5 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.

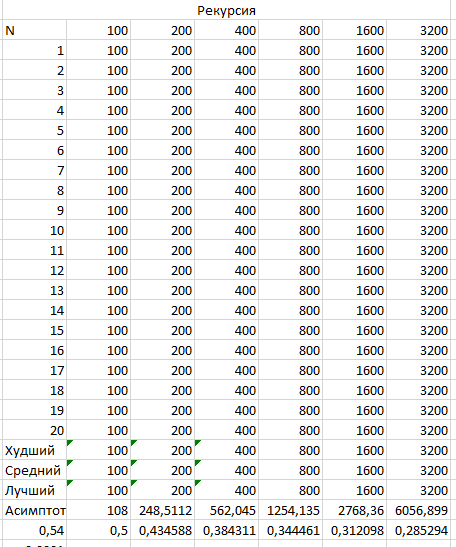
**Вектор с максимальным количеством сравнений при выборе среднего элемента в качестве опорного**

1. Для получения такого вектора было необходимо выполнить алгоритм, который преобразует наш ранее отсортированный вектор таким образом, чтобы при быстрой сортировке опорный элемент выбирался так, чтобы вектор делился на части по 1 и N-1 элементу.
2. Путем итерации элементы меняются местами так, чтобы на каждом шаге в качестве среднего будет выбираться самый крупный элемент.



1. Далее также было проведено 20 тестов для различных размеров вектора.
2. Количество рекурсий в зависимости от длины вектора.

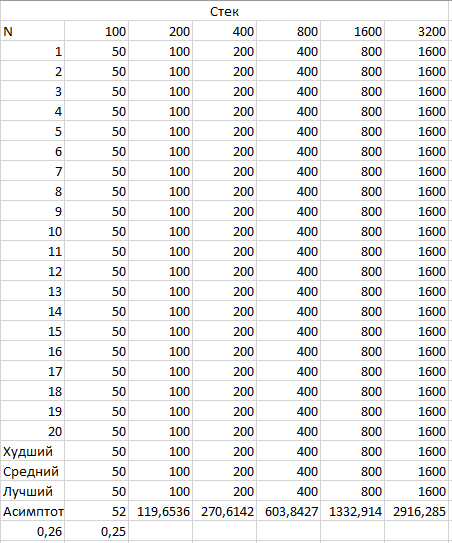
* Полученные результаты



* График худшего, среднего и лучшего случаев для количества рекурсий в зависимости от длины вектора.
* Был построен график асимптотической сложности с помощью формулы c\*O(N\*LOG(N)). Для данного худшего случая c = 0,54 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.

1. Высота стека в зависимости от длины вектора.

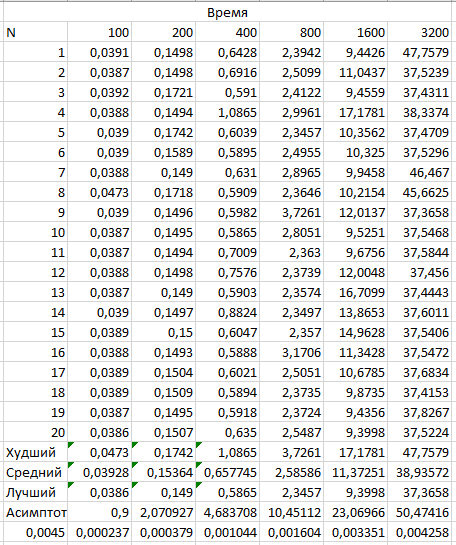
* Полученные результаты



* График худшего, среднего и лучшего случаев для высоты стека в зависимости от длины вектора.
* Был построен график асимптотической сложности с помощью формулы c\*O(N\*LOG(N)). Для данного худшего случая c = 0,26 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.

1. Время сортировки в зависимости от длины вектора.

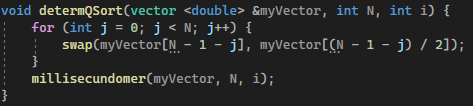
* Полученные результаты



* График худшего, среднего и лучшего случаев для времени сортировки в зависимости от длины вектора.
* Был построен график асимптотической сложности с помощью формулы c\*O(N\*LOG(N)). Для данного худшего случая c = 0,05 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.

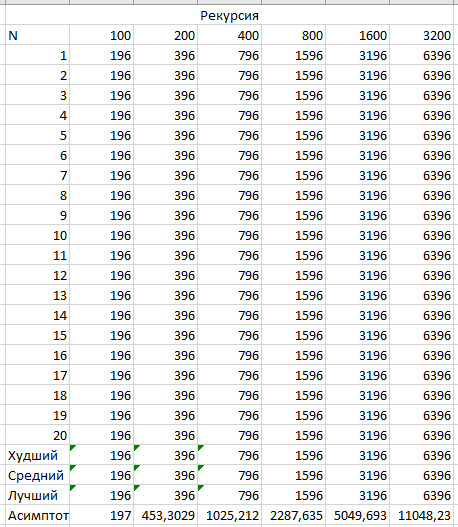
**Вектор с максимальным количеством сравнений при детерминированном выборе опорного элемента**

1. Для получения такого вектора было необходимо выполнить алгоритм, который преобразует наш ранее отсортированный вектор таким образом, чтобы при быстрой сортировке опорный элемент выбирался так, чтобы вектор делился на части по 1 и N-1 элементу.
2. Путем итерации элементы меняются местами так, чтобы на каждом шаге в качестве среднего будет выбираться самый крупный элемент.



1. Далее также было проведено 20 тестов для различных размеров вектора.
2. Количество рекурсий в зависимости от длины вектора.

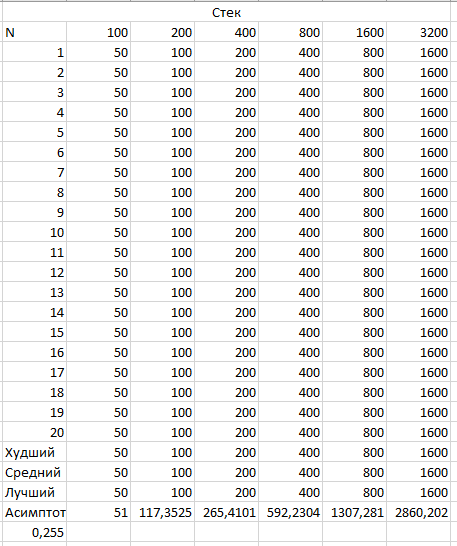
* Полученные результаты



* График худшего, среднего и лучшего случаев для количества рекурсий в зависимости от длины вектора.
* Был построен график асимптотической сложности с помощью формулы c\*O(N\*LOG(N)). Для данного худшего случая c = 0,99 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.

1. Высота стека в зависимости от длины вектора.

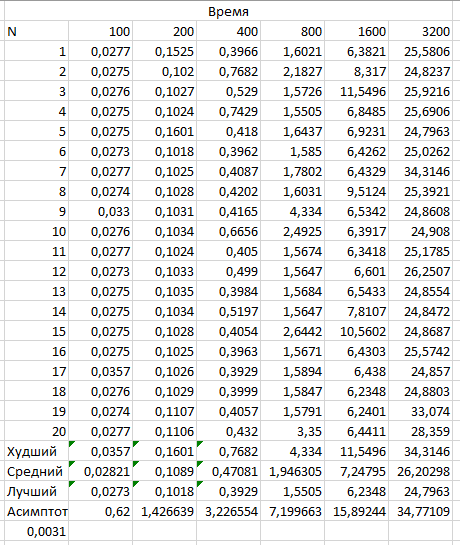
* Полученные результаты



* График худшего, среднего и лучшего случаев для высоты стека в зависимости от длины вектора.
* Был построен график асимптотической сложности с помощью формулы c\*O(N\*LOG(N)). Для данного худшего случая c = 0,255 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.

1. Время сортировки в зависимости от длины вектора.

* Полученные результаты



* График худшего, среднего и лучшего случаев для времени сортировки в зависимости от длины вектора.
* Был построен график асимптотической сложности с помощью формулы c\*O(N\*LOG(N)). Для данного худшего случая c = 0,0031 для того, чтобы асимптота была выше графика худшего случая при увеличении длины вектора N.

# Заключение

По результатам проделанной работы, можно сделать вывод, что метод быстрой сортировки действительно занимает меньше времени, чем другие методы сортировки. Однако скорость данного алгоритма зависит полностью от того, какие элементы находятся в массиве. В некоторых случаях выбор опорного элемента может происходить недостаточно эффективно.

Также одним из главных минусов данной сортировки можно отметить то, что данная сортировка выполняется рекурсивным методом, а это в свою очередь накладывает ограничение на максимальную длину сортируемого массив, так как при сортировке огромного количества элементов будет происходить переполнение стека. По данной причине в лабораторной работе были выполнены тесты на массивах с 100, 200, 400, 800, 1600, 3200 элементами.

К недостаткам так же можно добавить, что данная сортировка сильно деградирует по скорости в зависимости от генерации элементов в массиве, и что данная сортировка является неустойчивой, так как в процессе алгоритма одинаковые элементы могут перемешиваться.