Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное   
учреждение высшего образования

Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**«Сортировки»**

**Выполнил**:

студент группы 3821Б1ПМ2

Гурьев Г.А

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**Оглавление**

[Постановка задачи 3](#__RefHeading___Toc2373_4239795057)

[Методы решения 4](#__RefHeading___Toc2375_4239795057)

[Руководство пользователя 7](#__RefHeading___Toc2377_4239795057)

[Описание программной реализации 8](#__RefHeading___Toc2379_4239795057)

[Подтверждение корректности 10](#__RefHeading___Toc2381_4239795057)

[Результаты экспериментов 11](#__RefHeading___Toc2383_4239795057)

[Заключение 23](#__RefHeading___Toc2385_4239795057)

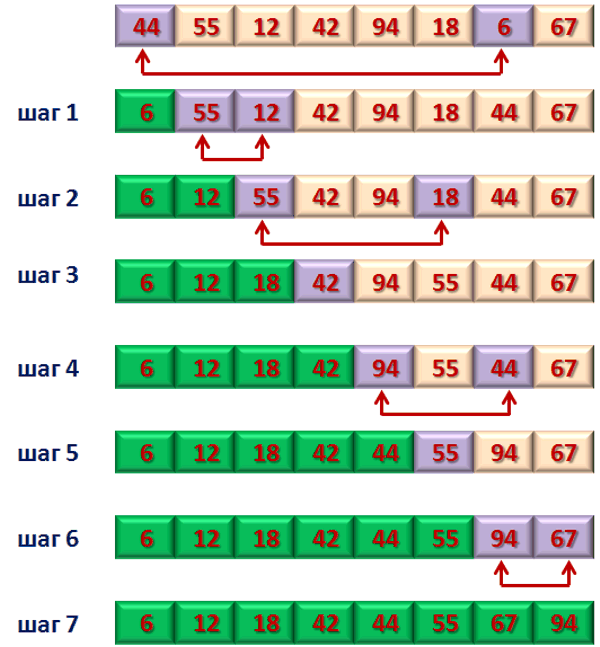
# Постановка задачи

Целью лабораторной работы является написание сортировки выбором, сортировки слиянием, а также Быстрой и Поразрядной сортировок на языке программирования СИ с использованием типа данных floatU+002e

# Методы решения

## Сортировка выбором

Алгоритм сортировки выбором довольно прост. Находится минимальное значение в массиве, оно встает на место первого неотсортированного элемента массива. Далее это повторяется для неотсортированной части массива. Сложность алгоритма составляет O(n2)

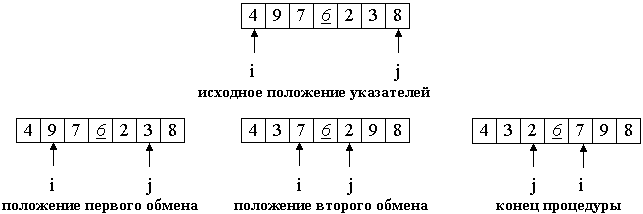


работа сортировки выбором

## Быстрая сортировка

Алгоритм сортировки заключается в том , чтобы выбрать опорный элемент из массива и разбить массив на две части относительно этого элемента. Далее слева от элемента располагаются элементы, которые меньше самого опорного элемента, а те элементы , что больше перемещаются влево от опорного элемента. Далее рекурсивно применяем это к двум двум оставшимся подмассивам. Сложность такого алгоритма составляет

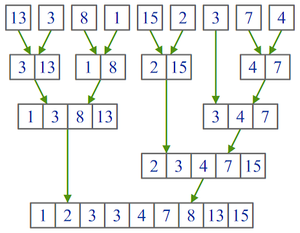
O(n log(n))



Как работает быстрая сортировка

## Сортировка слиянием

Алгоритм Сортировки слиянием заключается в разбиении массива на части и сортировке их по-отдельности.Сначала массив разбивается на два примерно равных подмассива. Затем эти два подмассива также рекурсивно разбиваются на примерно равные части, пока у нас не останутся массивы размером в один элемент. Далее все эти подмассивы массивы объединяются в отсортированные результирующие массивы.Сложность алгоритма составляет O(n\*log(n)).

сортировка слиянием

## Поразрядная сортировка

При поразрядной сортировке используется устойчивый алгоритм сортировки подсчетом. Для каждого разряда, начиная с наименьшего применяется сортировка подсчетом. Сложность алгоритма линейная и составляет O(w\*n),где w- количество бит, требуемых для хранения ключей.

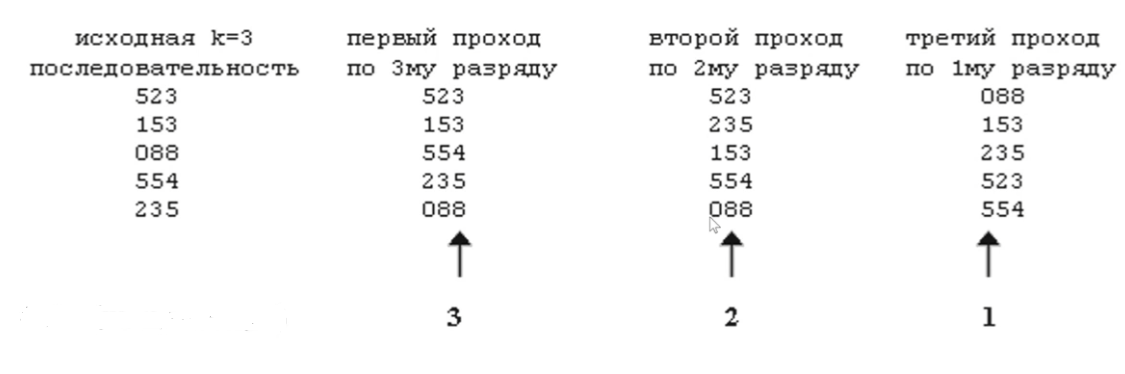


Рисунок 1 - иллюстрация работы сортировки

# Руководство пользователя

Каждая из сортировок выводит сложность алгоритма, количество сравнений, количество перестановок и количество элементов. Вывод осуществляется в формате “Сложность алгоритма; количество сравнений ;количество перестановок; количество элементов”.

# Описание программной реализации

**Сортировка выбором**

Функция void(float \*Array,int size) принимает на вход:

float \*Array – сортируемый массив.

Int size – размер сортируемого массива

**Быстрая сортировка**

void quickSort(float \*array, int first, int last) принимает на вход:

float \*array – сортируемый массив

int first – индекс первого элемента, с которого начинаем сортировать массив

int last – индекс последнего элемента, по который надо сортировать

**Поразрядная сортировка**

Функция CreateCounter(float\* data, int n) принимает на вход:

float\* data — указатель на сортируемый массив

int n – длина массива

Функция radixPass(short Offs, int n, float\* Array, float\* dest, int\* count)

принимает на вход:

short Offs – индекс элемента массива, с которого числа вставляются в сортируемый массив

int n – длина массива

float\* Array – указатель на сортируемый массив

float\* dest – указатель на выходной массив

int\* count – указатель на позиции, с которых начинается сортировка разрядов, созданных CreateCounter.

Функция float radixSort(float \*Array, float \*sorted\_arr,int n) получает на вход:

float \*Array – сортируемый массив

float \*sorted\_arr – массив, в который функция положит отсортированный массив.

Int n – длина массива

**Сортировка слиянием**

Функция float\* mergesort (float\* arr, float\* buff, unsigned int strt, unsigned int end) принимает на вход:   
float\* arr – указатель на сортируемый массив

float\* buff – указатель на массив-буфер

unsigned int strt – индекс первого элемента, с которого надо начинать сортировать массив

unsigned int end – индекс последнего элемента,по который надо сортировать массив

на выход функция передает указатель на отсортированный массив arr2.

# Подтверждение корректности

Для подтверждения корректности сортировки я создаю копию основного массива, которая сортируется с помошью стандартной функции Си qsort(). После этого сравниваются массив отсортированный qsort() и массив отсортированный самой функцией. Которую необходимо проверить на корректность сортировки.

# Результаты экспериментов

n обозначает количество элементов в массиве.

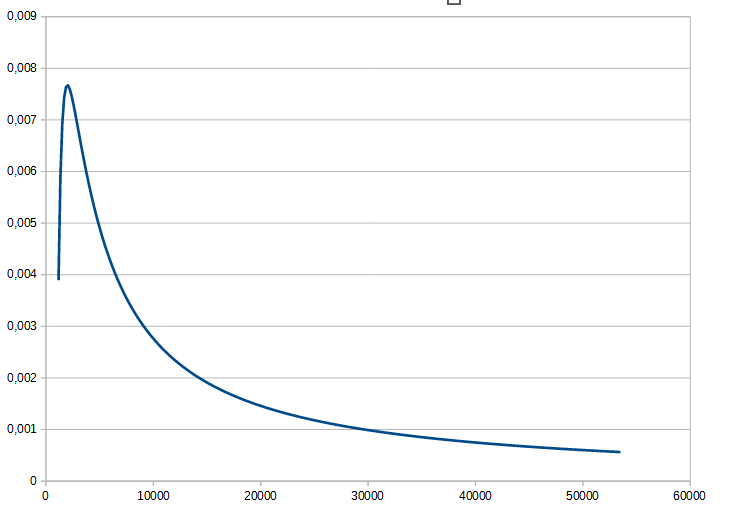
Сложность вычислялась по сложности указанной для каждой сортировки. Нотация “о большое” не учитывает константу, поэтому отношение количества присвоений к сложности и сравнений к сложности , должны быть равны или должны сходиться к некоторому числу.

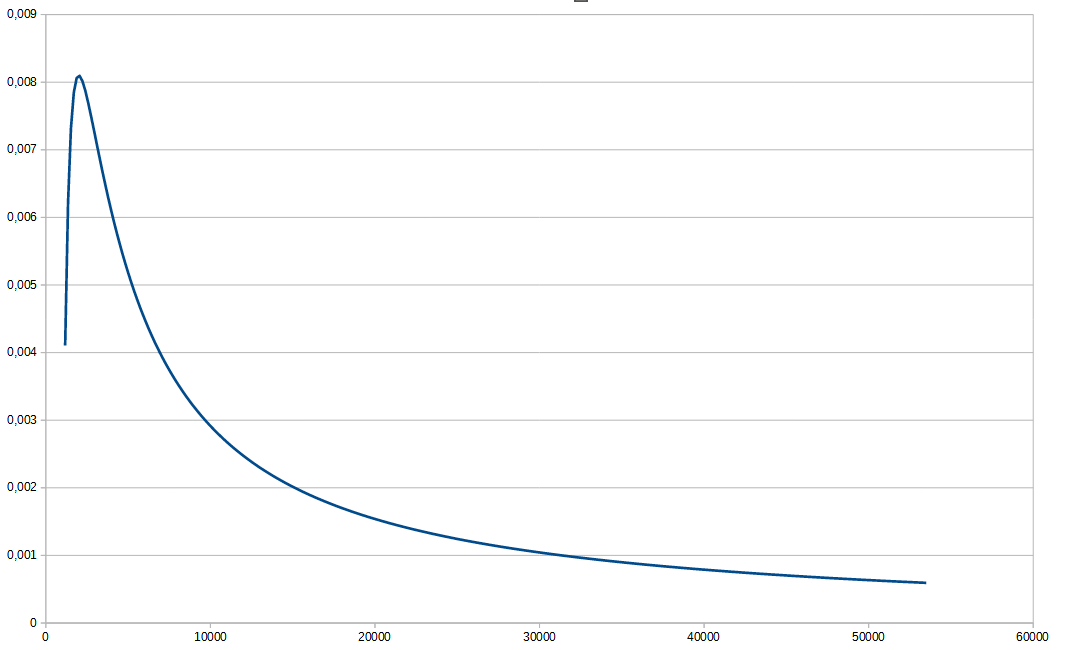
Перестановка элементов в сортировках осуществлялась в три присвоения.

## Сортировка выбором

Сложностью данной сортировки является O(n2).

Далее привожу несколько графиков с результатами эксперементов.

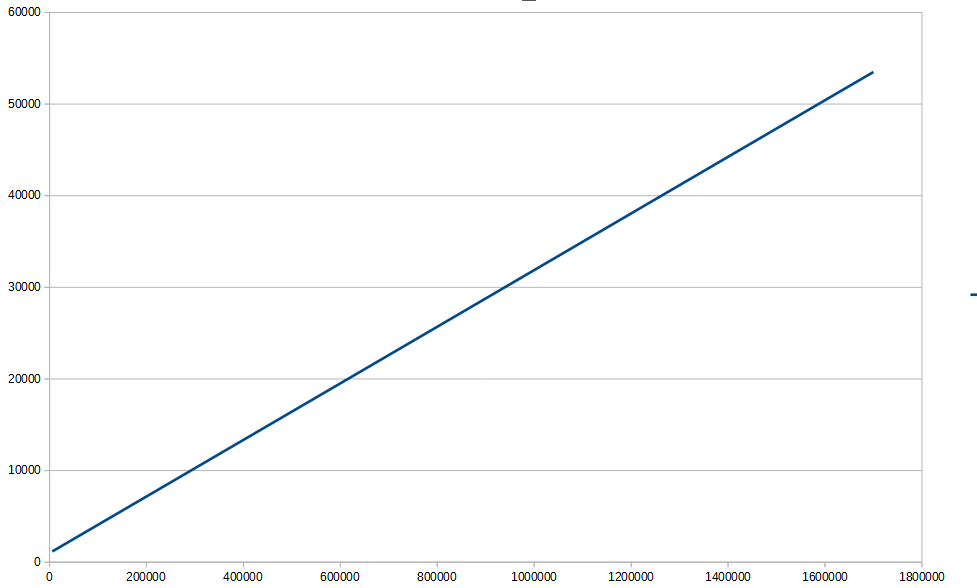
сравнения/сложность

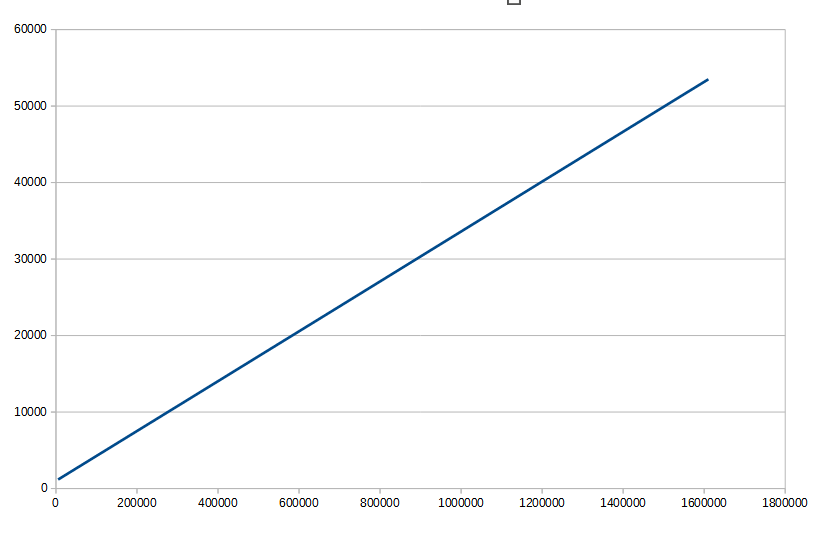


перестановки/сложность

Из двух графиков выше видно, что количество перестановок относительно сложности и количество сравнений относительно сложности сходятся к нулю.

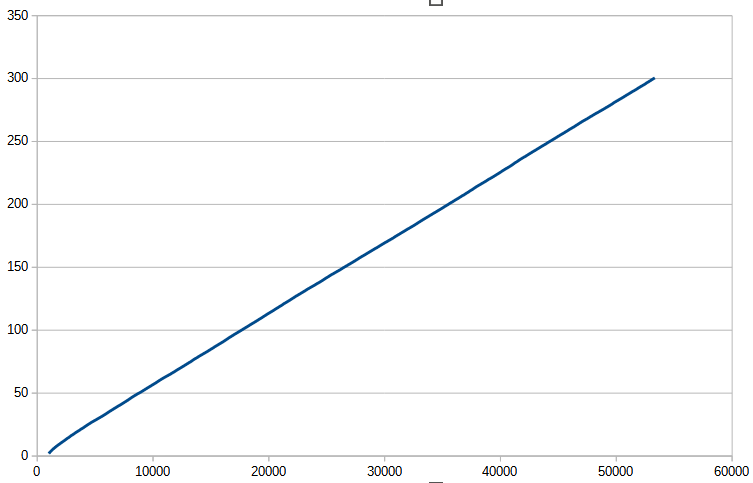
Количество сравнений и перестановок растет линейно, исходя из следующих двух графиков:



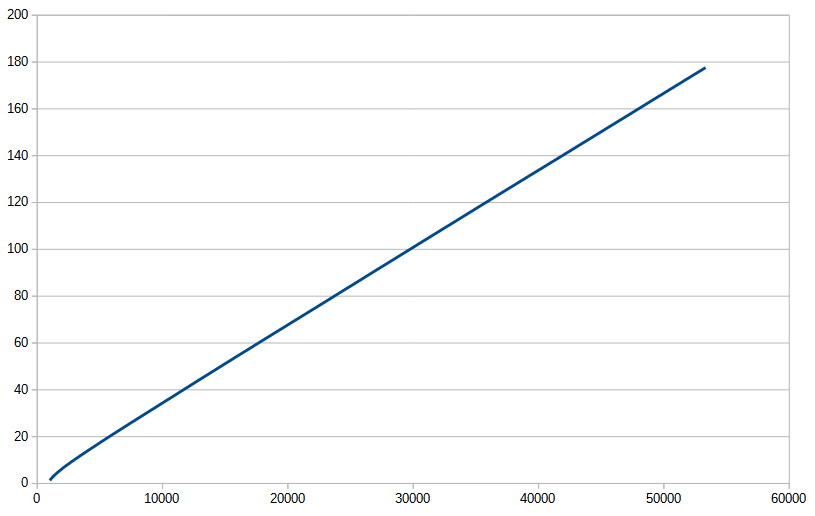


## Быстрая Сортировка

Сложностью данной сортировки является O(n\*log(n)).

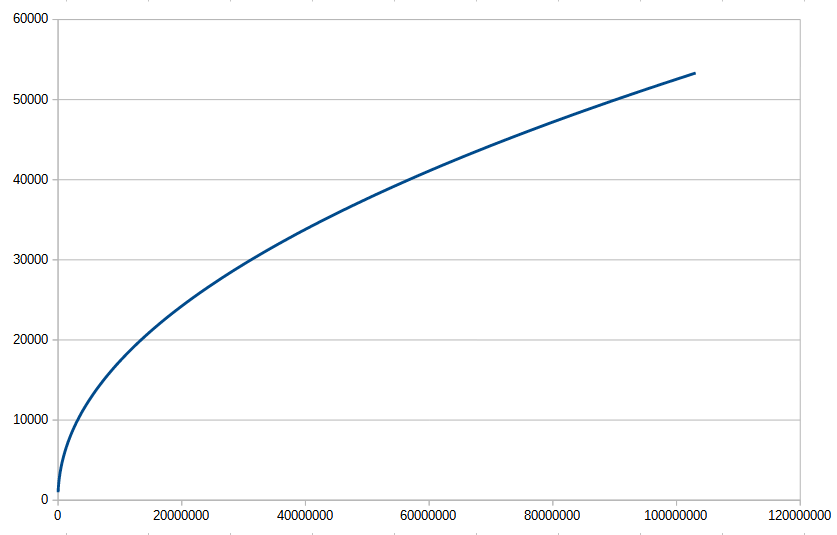


сравнения /сложность

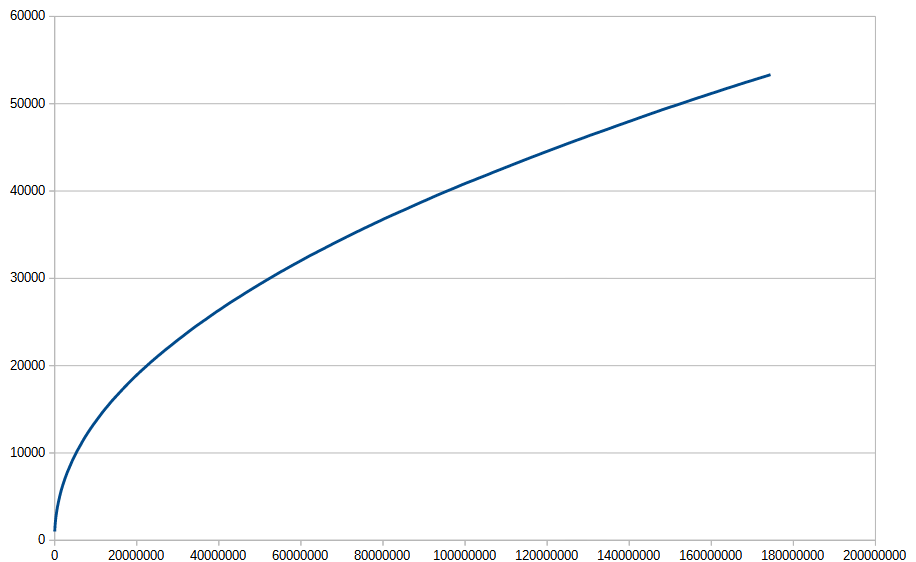


перестановки/сложность

Из двух графиков выше видно, что количество сравнений относительно сложности и перестановок относительно сложности растет линейно. Количество сравнений и количество перестановок возрастает по кривой:



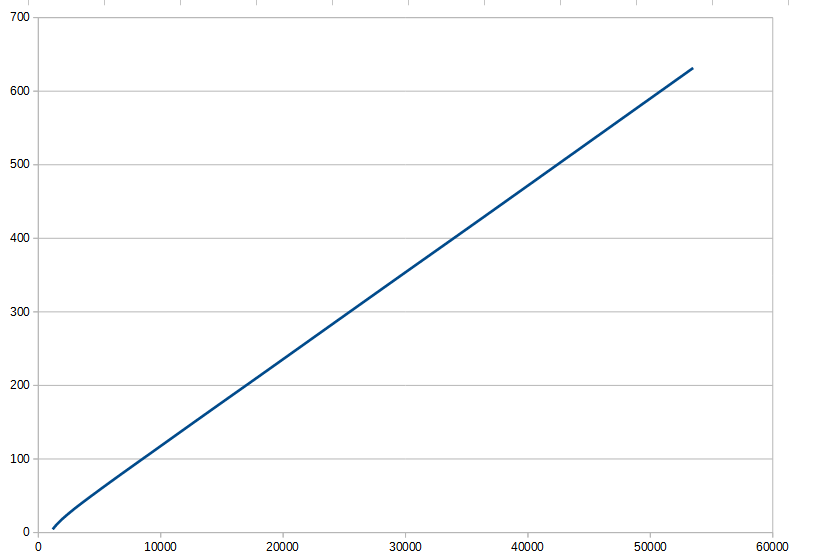
количество сравнений



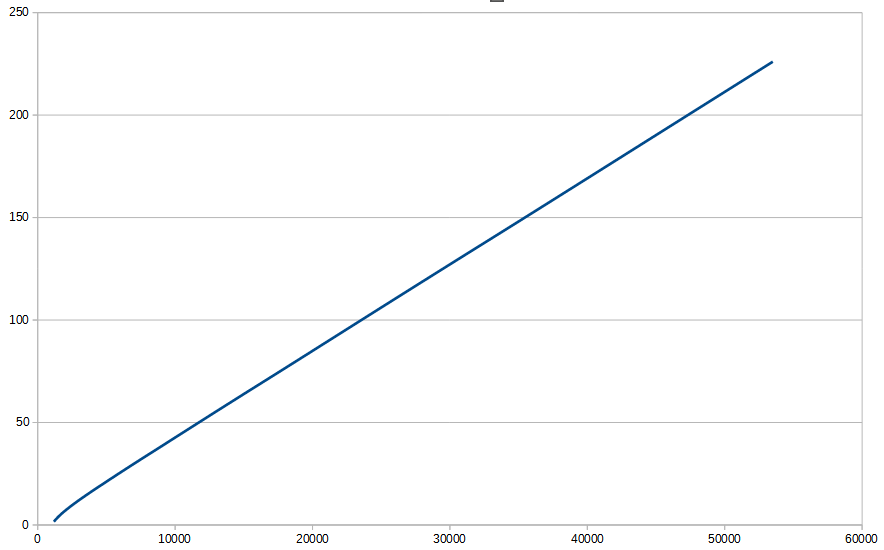
количество перестановок

## Сортировка слиянием

Сложностью данной сортировки является O(n\*log(n)). Количество сравнений относительно сложности и количество перестановок относительно сложности возрастает линейно , как видно из двух графиков снизу

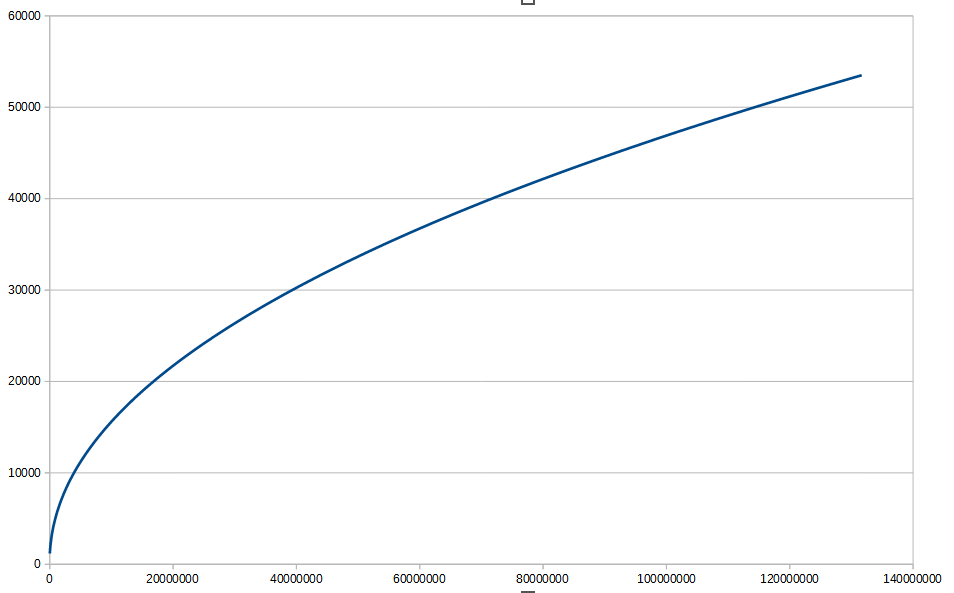


сравнения/сложность

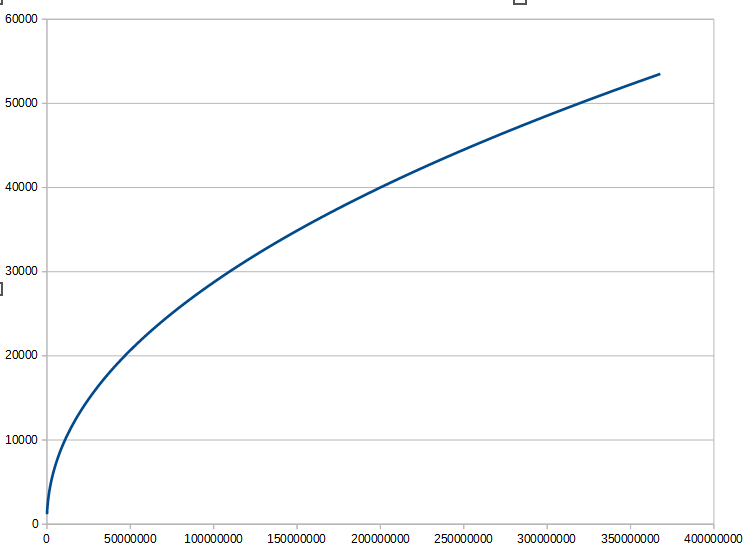


присвоения/сложность

## Количество сравнений и перестановок растет по кривой, это видно по двум графикам снизу.



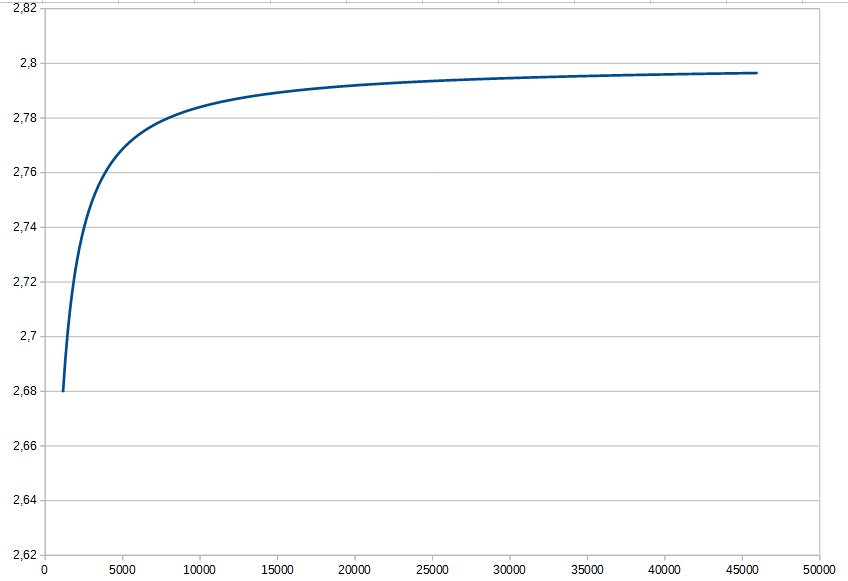
Количество сравнений



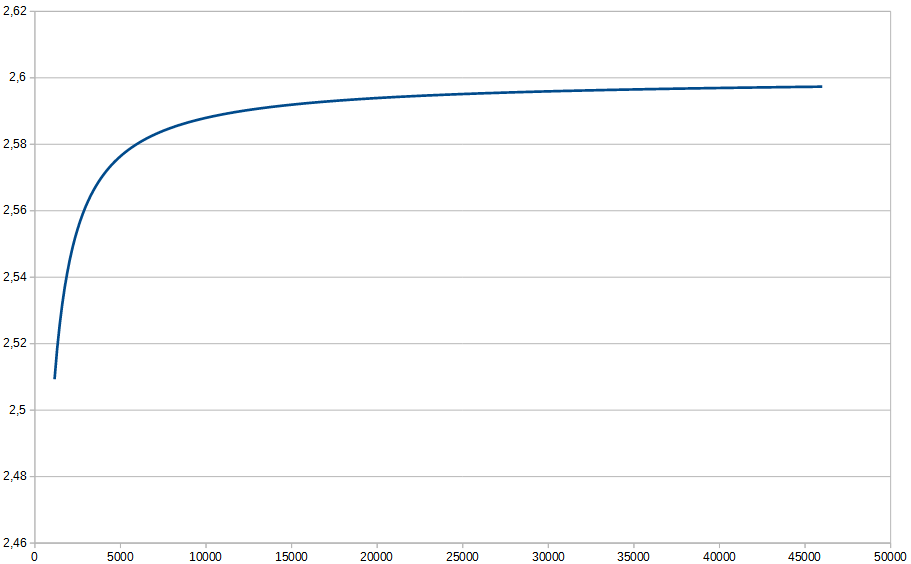
Количество присвоений

## Поразрядная сортировка

В данном случае сложность сортировки составляет O(4\*(n+256)+n). Значение количества сравнений относительно сложности и значение количества перестановок относительно сложности сходится к 2.8

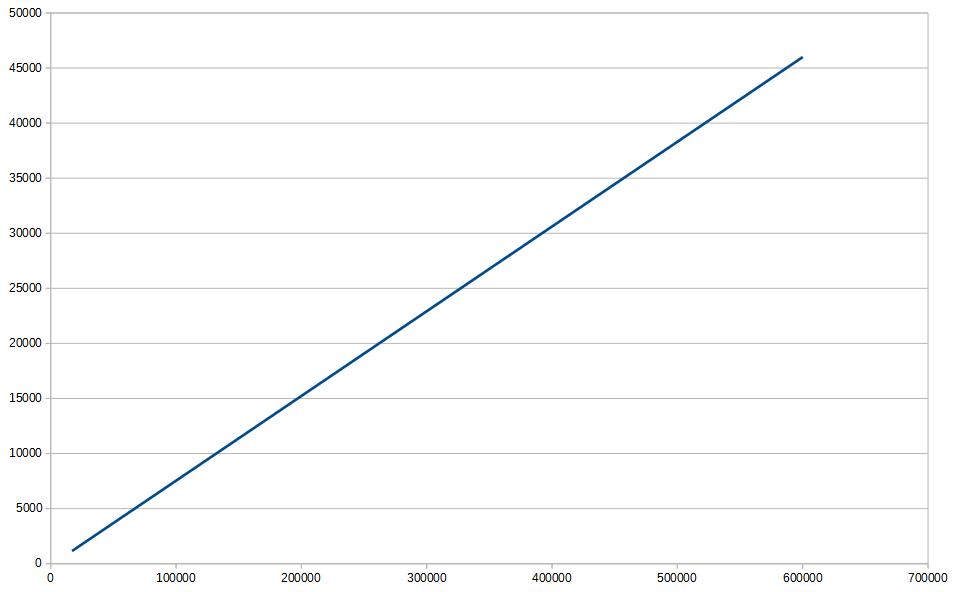


сравнения/сложность

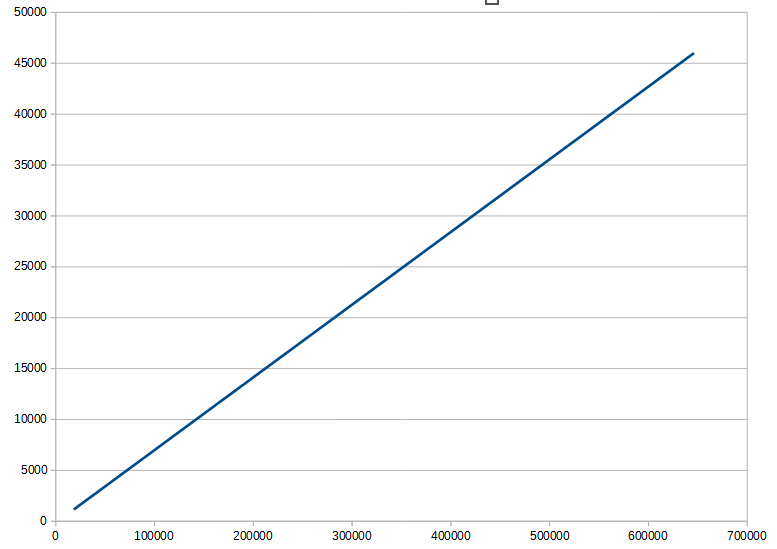


присвоения/сложность

Количество сравнений и количество присвоений растет линейно:

****

Количество сравнений



Количество присвоений

# Заключение

Были реализованы алгоритмы : Сортировка выбором, быстрая сортировка, сортировка слиянием и поразрядная сортировка. Все алгоритмы были реализованы на языке программирования Си. Данные алгоритмы были описаны, была проведена проверка корректности их работы, а также были проведены эксперименты, демонстрирующие рост количества сравнений и перестановок с ростом количества элементов.