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

Логинов Сергей Сергеевич

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

Оглавление

[Постановка задачи 3](#_Toc91460961)

[Методы решения 4](#_Toc91460962)

[Сортировка пузырьком 4](#_Toc91460963)

[Сортировка расческой 4](#_Toc91460964)

[Сортировка слиянием 5](#_Toc91460965)

[Поразрядная сортировка 6](#_Toc91460966)

[Руководство пользователя 9](#_Toc91460967)

[Описание программной реализации 10](#_Toc91460968)

[Сортировка пузырьком: 10](#_Toc91460969)

[Сортировка расческой 10](#_Toc91460970)

[Сортировка слиянием 11](#_Toc91460971)

[Поразрядная сортировка 12](#_Toc91460972)

[Подтверждение корректности 13](#_Toc91460973)

[Результаты экспериментов 14](#_Toc91460974)

[Сортировка пузырьком 15](#_Toc91460975)

[Сортировка расческой 17](#_Toc91460976)

[Сортировка слиянием: 20](#_Toc91460977)

[Поразрядная сортировка 22](#_Toc91460978)

[Заключение 25](#_Toc91460979)

# Постановка задачи

Целью лабораторной работы являлась реализовать на языке программирования Си следующие сортировки:

1. Сортировка пузырьком
2. Сортировка расчёской
3. Сортировка слиянием
4. Поразрядная сортировка

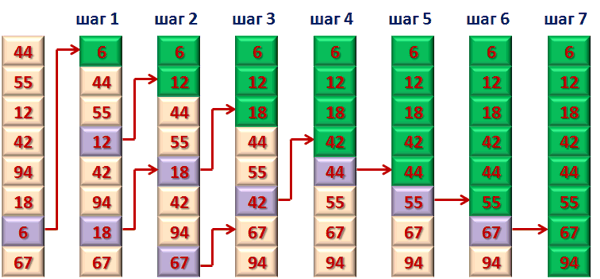
Необходимо было выполнить 3 задания к сортировкам:

1. Реализовать сортировки для типов данных по предложенной таблице.
2. Замерить число перестановок и количество сравнений выполненное при сортировке.
3. Написать отчет. Эксперименты поставить так, чтобы показать теоретическую сложность алгоритмов.

Сортировки необходимо реализовать для данных типа double. Нужно описать программную реализацию и алгоритмы работы данных сортировок. Необходимо подтвердить корректность реализации данных сортировок, а также провести эксперименты для подтверждения сложности алгоритма. Описать способы проведения экспериментов и сделать вывод по полученным результатам.

# Методы решения

## Сортировка пузырьком

 Идея метода состоит в том, чтобы попарно сравнивать соседние элементы. Каждый проход начинается с начала последовательности. Сравнивается первый элемент со вторым: если порядок между ними нарушен, то они меняются местами. Затем сравниваются второй с третьим, третий с четвертым и так далее до конца массива; элементы с неправильным порядком в паре меняются местами. В итоге, после первого прохода, максимальный (или минимальный, в зависимости от вида сортировки: по возрастанию/по убыванию) элемент будет находиться на последнем месте в массиве, он как бы “всплывет” наверх. Алгоритм выполняет n проходов по массиву. Сложность данного алгоритма сортировки равна O(n2).  


1)Иллюстрация работы алгоритма сортировки пузырьком

## Сортировка расческой

Сортировка расчёской улучшает сортировку пузырьком, и конкурирует с алгоритмами, подобными быстрой сортировке.

В сортировке пузырьком, когда сравниваются два элемента, промежуток (расстояние друг от друга) равен 1. Основная идея сортировки расчёской в том, что этот промежуток может быть гораздо больше, чем единица.

Тем самым первоначально берется большое расстояние между сравниваемыми элементами и по мере упорядочивания массива постепенно сужать это число до единицы.

Первоначальный разрыв между сравниваемыми элементами лучше брать с учётом специальной величины, называемой фактором уменьшения, оптимальное значение которой равно примерно 1,247  


Сложность сортировки:

|  |  |
| --- | --- |
| [**Худшее время**](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) | O(*n*2) |
| [**Лучшее время**](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C) | O(*n logn*) |

## Сортировка слиянием

В основе сортировки слиянием лежит принцип “разделяй и властвуй”. Список разделяется на равные или практически равные части, каждая из которых сортируется отдельно. После чего уже упорядоченные части сливаются воедино.

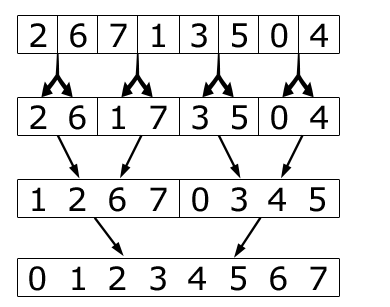
Для решения задачи сортировки эти три этапа выглядят так:

1. Сортируемый массив разбивается на две части примерно одинакового размера;
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;
3. Два упорядоченных массива половинного размера соединяются в один.

1.1. — 2.1. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

3.1. Соединение двух упорядоченных массивов в один.  
Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть мы имеем два уже отсортированных по возрастанию подмассива. Тогда:  
3.2. Слияние двух подмассивов в третий результирующий массив.  
На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1.  
3.3. «Прицепление» остатка.  
Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

Сложность алгоритма сортировки: O(nlogn). Необходимо О(n) памяти.



2)Иллюстрация работы сортировки слиянием

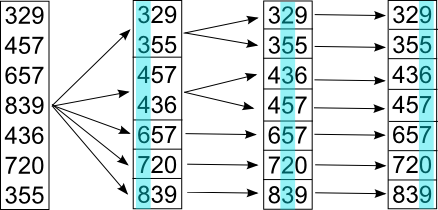
## Поразрядная сортировка

Исходно эта сортировка предназначена для сортировки целых чисел, записанных цифрами. Но так как в памяти компьютеров любая информация записывается целыми числами, алгоритм пригоден для сортировки любых объектов, запись которых можно поделить на «разряды», содержащие сравнимые значения. Например, так сортировать можно не только числа, записанные в виде набора цифр, но и строки, являющиеся набором символов, и вообще произвольные значения в памяти, представленные в виде набора байт.   
Сравнения производятся поразрядно:

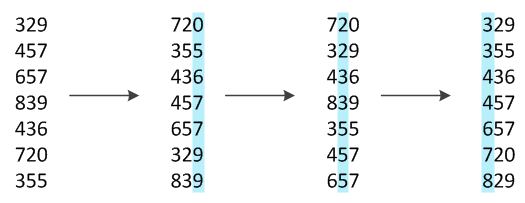
1. Сравниваются значения одного крайнего разряда, и элементы группируются по результатам этого сравнения;
2. сравниваются значения следующего разряда, соседнего, и элементы либо упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются в целом, но сохраняя относительный порядок, достигнутый при предыдущей сортировке.
3. Затем аналогично делается для следующего разряда, и так до конца.

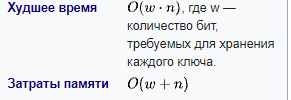
Существует два варианта этой сортировки, так как выравнивать записи можно в разную сторону относительно друг друга:

Для чисел они называются в терминах значимости разрядов числа, и получается так: можно выровнять записи чисел в сторону менее значащих цифр (по правой стороне, в сторону единиц, LSD) или более значащих цифр (по левой стороне, со стороны более значащих разрядов, MSD).



-Иллюстрация поразрядной сортировки (MSD)

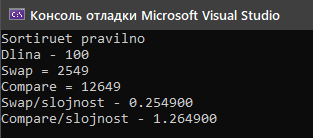
  
-Иллюстрация поразрядной сортировки (LSD)

Сложность поразрядной сортировки:   


# Руководство пользователя

Каждая выполненная сортировка выводит в консоль длину сортируемого массива, корректность выполнения сортировки, количество перестановок и сравнений, а также количество перестановок и сравнений деленную на теоретическую сложность алгоритма.  
 Формат вывода:

1. Правильность сортировки
2. Длина массива
3. Количество перестановок
4. Количество сравнений
5. Количество перестановок, деленное на теоретическую сложность алгоритма
6. Количество сравнений, деленное на теоретическую сложность алгоритма



# Описание программной реализации

## Сортировка пузырьком:

Void massiv (double\*m, double\* cm)-На вход принимает 2 массива с плавающей запятой и заполняет их "рандомными" данными.  
 int com (const void\* z, const void\* p)- функция принимает на вход два указателя на сравниваемые числа любого типа данных. Затем возвращает их взаимное отношение в виде больше, равно, меньше. Функция является вспомогательной для qsort() из стандартной библиотеки Си.  
 int main()-Функция, которая выполняется при запуске программы, в себе объявляет переменные необходимые для работы программы, вызывает функцию заполенения массива, выполняет сортировку массива mas, выполняет сортировку массива copymas с помощью функции qsort, сравнивает массивы mas и copymas и выводит результат сравнения в консоль.

## Сортировка расческой

Void massiv (double\*m, double\* cm)-На вход принимает 2 массива с плавающей запятой и заполняет их "рандомными" данными.  
int com (const void\* z, const void\* p)- функция принимает на вход два указателя на сравниваемые числа любого типа данных. Затем возвращает их взаимное отношение в виде больше, равно, меньше. Функция является вспомогательной для qsort() из стандартной библиотеки Си.  
int main()-Функция, которая выполняется при запуске программы, в себе объявляет переменные необходимые для работы программы, вызывает функцию заполенения массива, выполняет сортировку массива mas, выполняет сортировку массива copymas с помощью функции qsort, сравнивает массивы mas и copymas и выводит результат сравнения в консоль.

## 

## Сортировка слиянием

Void massiv (double\*m, double\* cm)-На вход принимает 2 массива с плавающей запятой и заполняет их "рандомными" данными.

int com (const void\* z, const void\* p)- функция принимает на вход два указателя на сравниваемые числа любого типа данных. Затем возвращает их взаимное отношение в виде больше, равно, меньше. Функция является вспомогательной для qsort() из стандартной библиотеки Си.

void merge(double\* a, int levo, int center, int pravo) - На вход принимает 4 значения: значение a-указатель на сортируемый массив, levo, pravo, int целочисленного типа, функция выполняет сортировку массива согласно полученным данным. Возвращает указатель на отсортированную часть массива.

void mergeSort(double\* dmass, double\* vtoroy, int l, int r)- На вход принимает 4 значения dmass , vtoroy указатель на значение числа с плавающей запятой, l, r целочисленного типа, выполняет проверку значений массива с рекурсивным вызовом самой себя и вызовом функции сортировки массива.

void main() - Функция, которая выполняется при запуске программы, в себе объявляет переменные необходимые для работы программы, вызывает функцию заполенения массива, выполняет сортировку массива mas, выполняет сортировку массива mas\_copy с помощью функции qsort, сравнивает массивы mas и copymas и выводит результат сравнения в консоль.

## Поразрядная сортировка

Void massiv(double\* m, double\* cm1, double\* cm2)- На вход принимает 3 массива с плавающей запятой и заполняет их "рандомными" данными

int com (const void\* z, const void\* p)- функция принимает на вход два указателя на сравниваемые числа любого типа данных. Затем возвращает их взаимное отношение в виде больше, равно, меньше. Функция является вспомогательной для qsort() из стандартной библиотеки Си.

Int\* createCounters(double\* data, int N) - на вход принимает указатель на первый элемент массива данных и длину этого массива. Функция считает количество встречающихся значений каждого байта, перенося значения в массив размером 256\*sizeof(ulonglong)\*sizeof(int). Функция возвращает указатель на полученный массив.

void radixPass(short Offset, int N, ulonglong\* sourse, ulonglong\* dest, int\* count) – функция принимает позицию, начиная с которой нужно вставлять число в выходной массив, длину сортируемого массива, указатели на 1 элемент исходного массива, на 1 элемент выходного массива, указатель на начало позиций сортировки рязряда из массива, который создан функцией createCounters() . Функция выполняет поразрядную сортировку .

void signedRadixSort(double\* data, double\* sorted\_data, int N)- функция принимает на вход указатель на 1 элемент входного массива, указатель на 1 элемент выходного массива, их длину. Функция сортирует входной массив с помощью Void radixPass.

void main() - Функция, которая выполняется при запуске программы, в себе объявляет переменные необходимые для работы программы, вызывает функцию заполенения массива, выполняет сортировку массива mas, выполняет сортировку массива mas\_copy с помощью функции qsort, сравнивает массивы mas и copymas и выводит результат сравнения в консоль.

# Подтверждение корректности

Для подтверждения корректности в программах использовалась сортировка из стандартной библиотеки <stdlib.h> qsort. Сперва сортируется массив с помощью сортировки, реализованной в программе. Затем сортируется оригинальный массив с помощью qsort. После сортировок начинается сравнение каждого элемента массивов. Если элементы не совпадают, то сравнение обрывается и возвращается значение 0. Если же все работает правильно и массивы идентичны, то возвращается значение 1. В командной строке выводится надпись “Sortiruet nepravilno” при неправильной сортировке, а при правильной выводит “Sortiruet pravilno” соответственно.  


-Пример из сортировки пузырьком

Функция cmp() является вспомогательной для функции qsort(). Функция cmp() устанавливает взаимное отношение двух элементов.

# 

# Результаты экспериментов

N- количество элементов в массиве  
Сложность вычислялась по указанной сложности для каждой сортировки.

Отношение количества присвоений к сложности и количества сравнений к сложности должны быть равны или сходиться к некоторому числу.

## Сортировка пузырьком

Сложность данной сортировки = O(n2)

На графике 1 показано изменение отношения количества перестановок к сложности при росте количества элементов в массиве. Значения сходятся приблизительно к ~0.25. Так как диапазон значений с увеличением количества элементов сходится к некоторому числу, то это является подтверждением указанной сложности для данной сортировки.

На графике 2 показано изменение отношения количества сравнений к сложности при росте количества элементов в массиве. Значения сходятся приблизительно к ~1.25. Так как диапазон значений с увеличением количества элементов сходится к некоторому числу, то это является подтверждением указанной сложности для данной сортировки.

На графике 3 показана зависимость количества перестановок от количества элементов в массиве.  
 На графике 4 показана зависимость количества сравнений от количества элементов в массиве

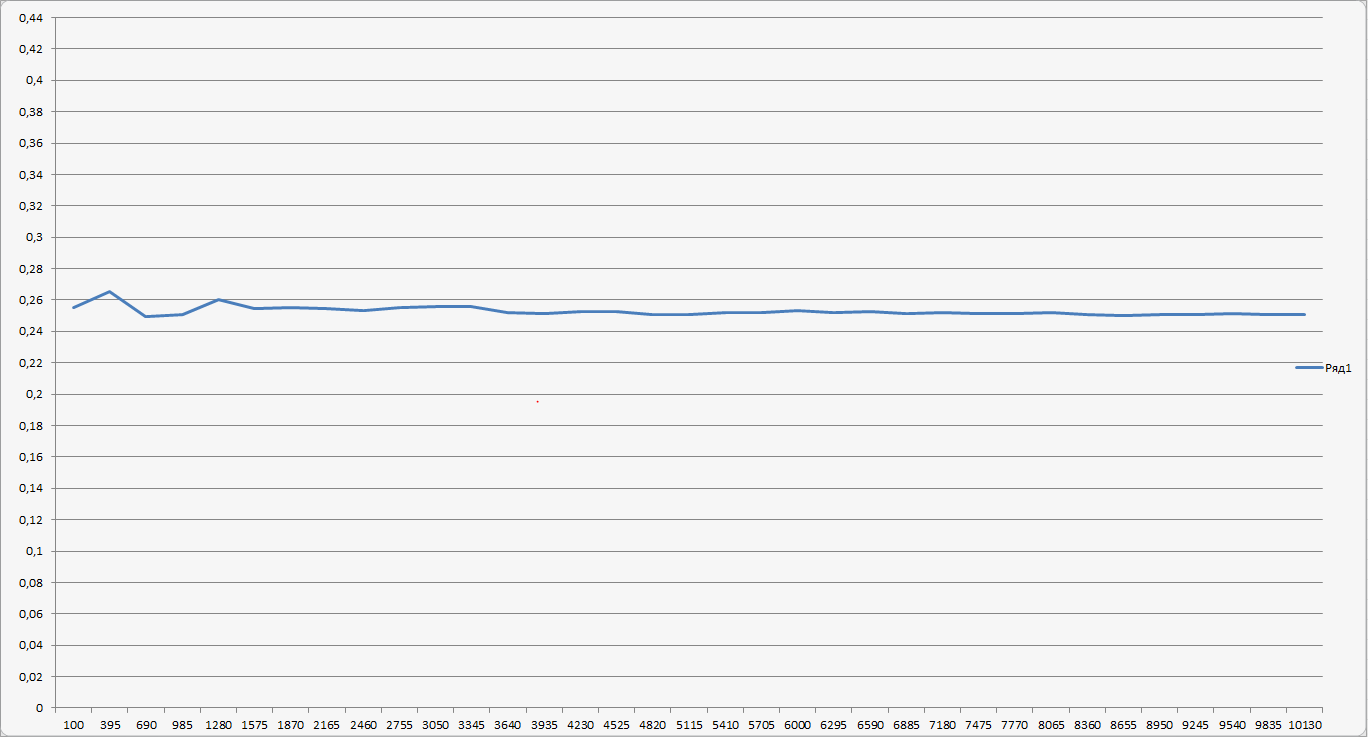


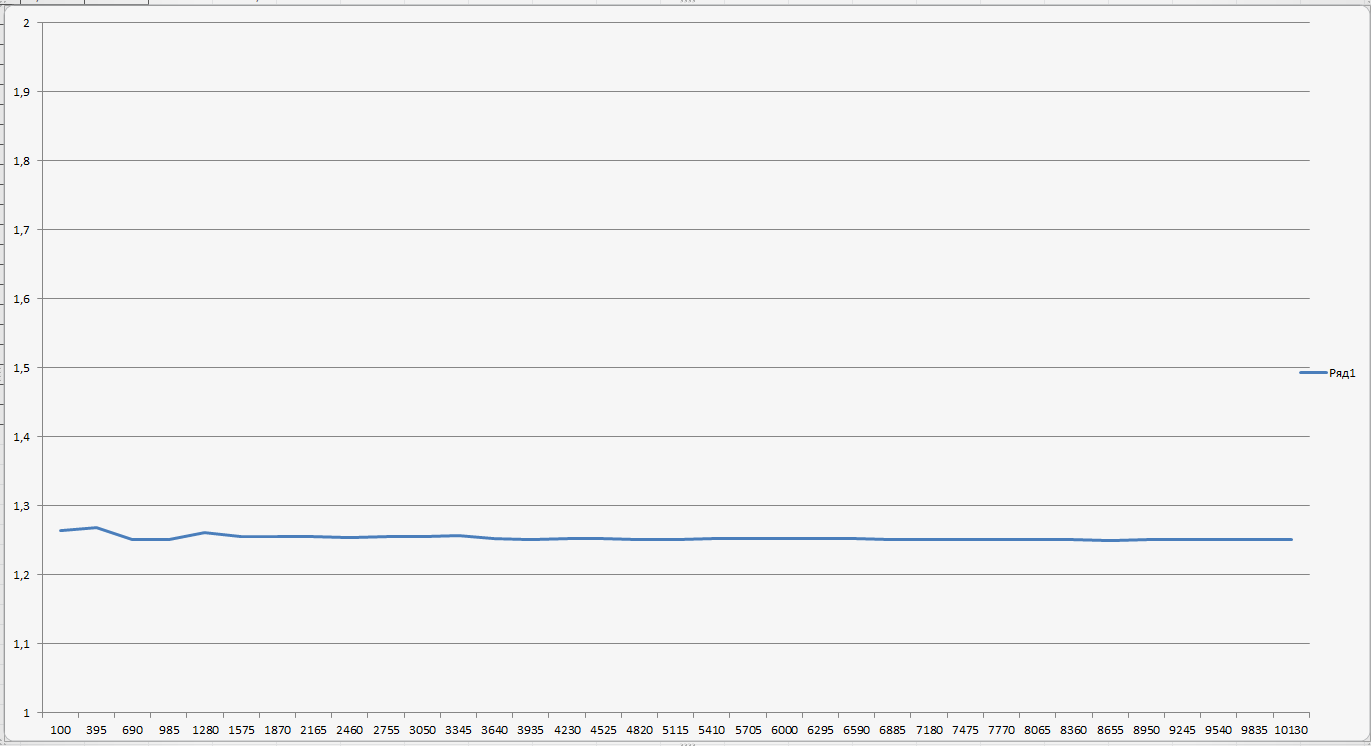
График 1-Перестановка/сложность

График 2-сравнения/сложность

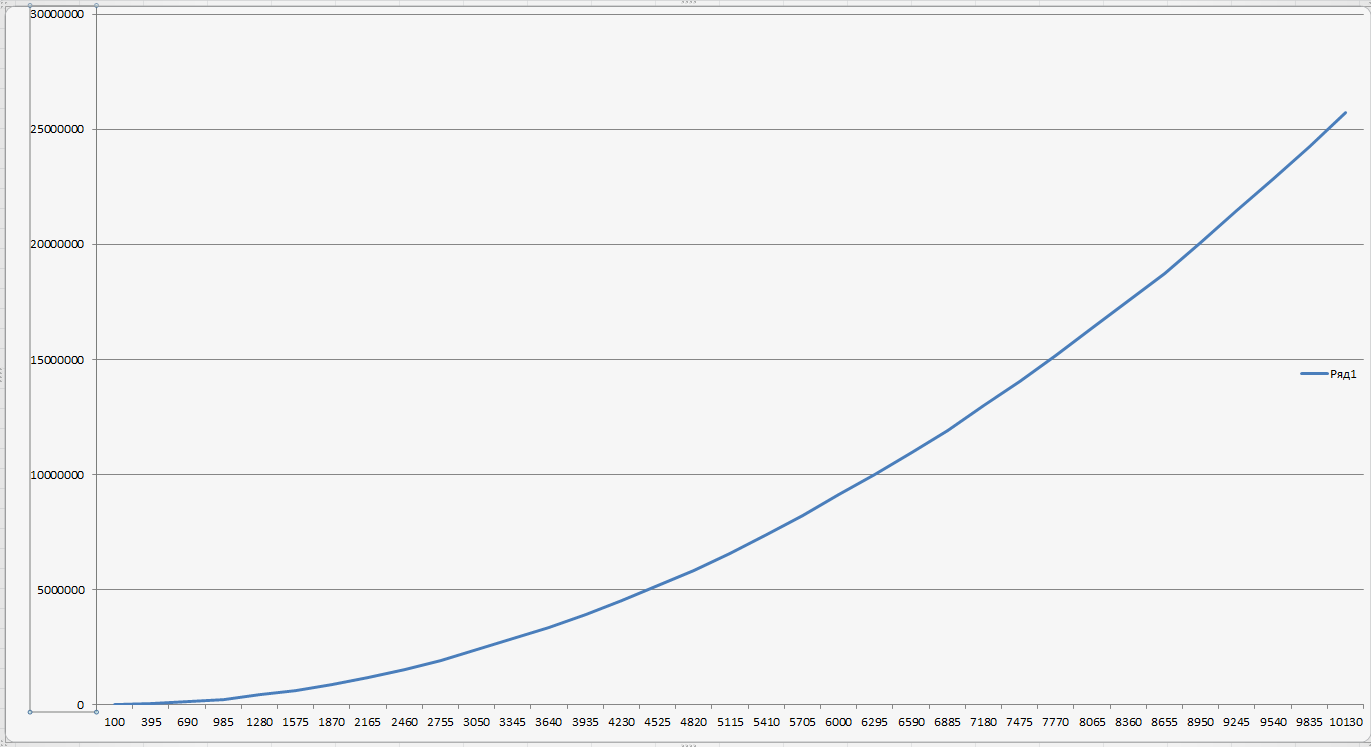


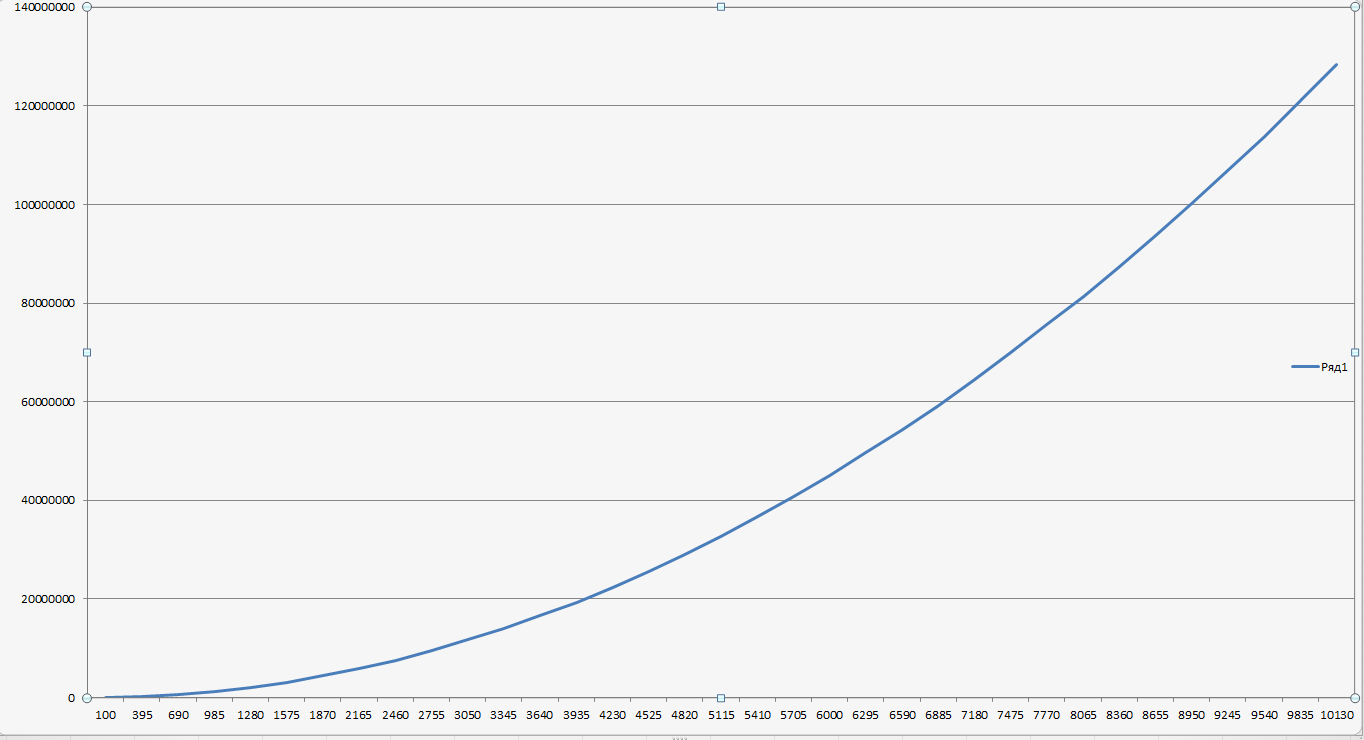
График 3- зависимость перестановок от количества элементов в массиве 

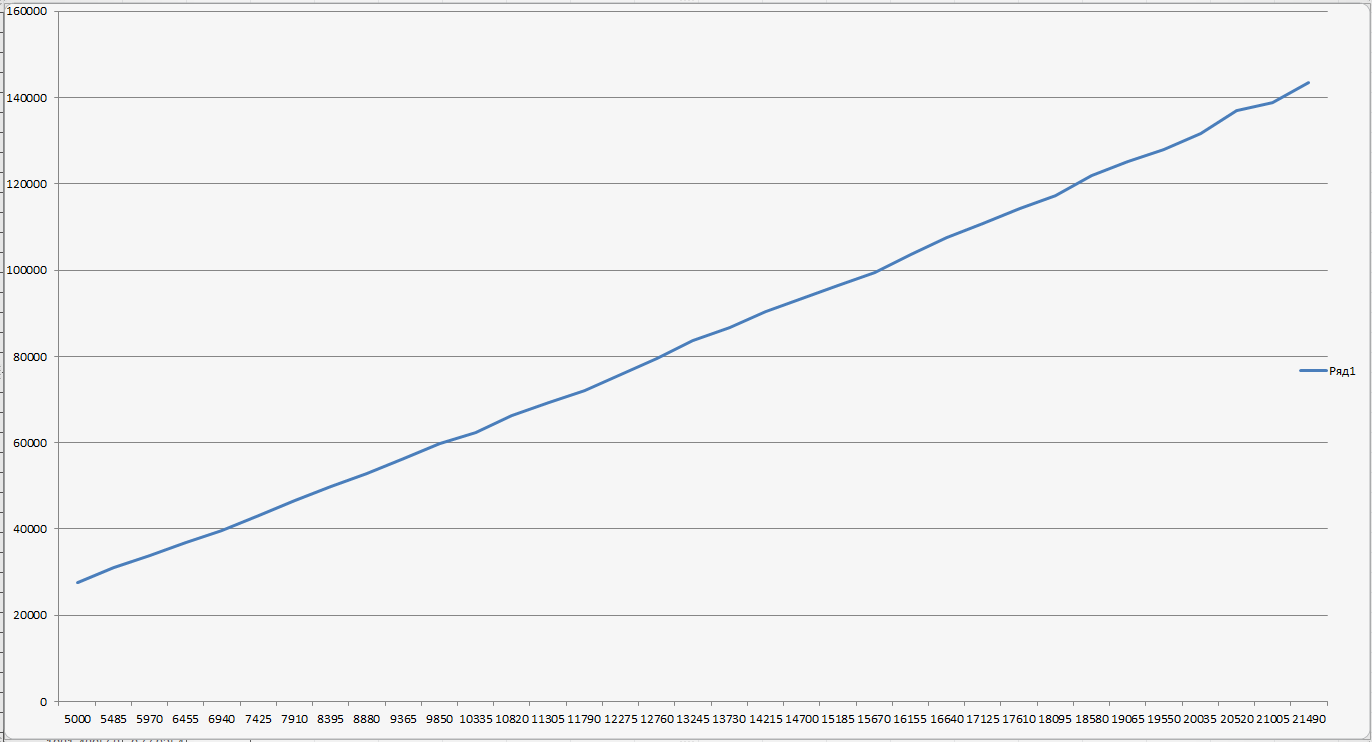
График 4- зависимость сравнений от количества элементов в массиве

## Сортировка расческой

**Сложность сортировки:**

Для того, чтобы доказать сложности данной сортировки построим графики зависимости количества сравнений и перестановок от длины массива N.

Если разделить каждое полученное значение на n \* log(n), то получится график сходящийся к константе. Для сравнений эта константа равна ~3,8. Для присвоений же эта константа равна ~0,67. Это доказывает теоретическую сложность сортировки.



**График 5 –** Зависимость перестановок от количества элементов в массиве

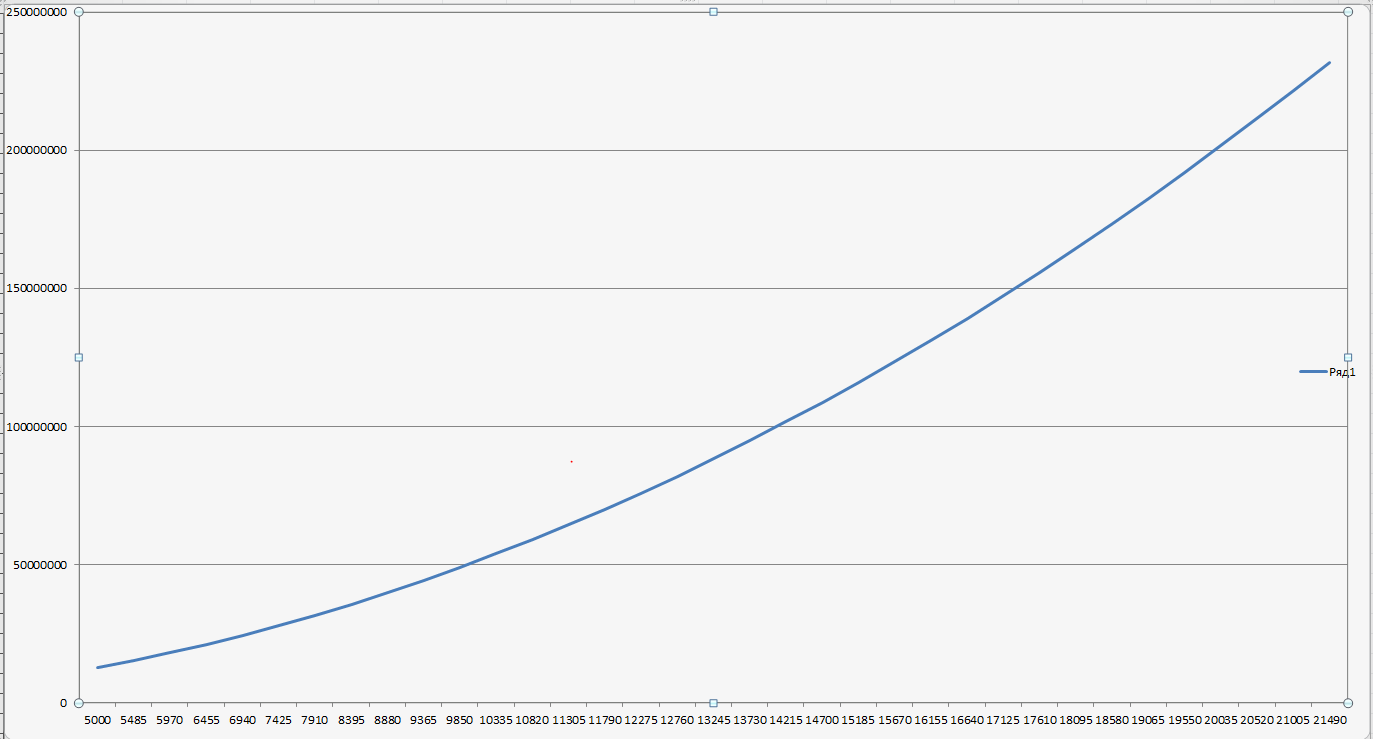


График 6-Зависимость сравнений от количества элементов в массиве

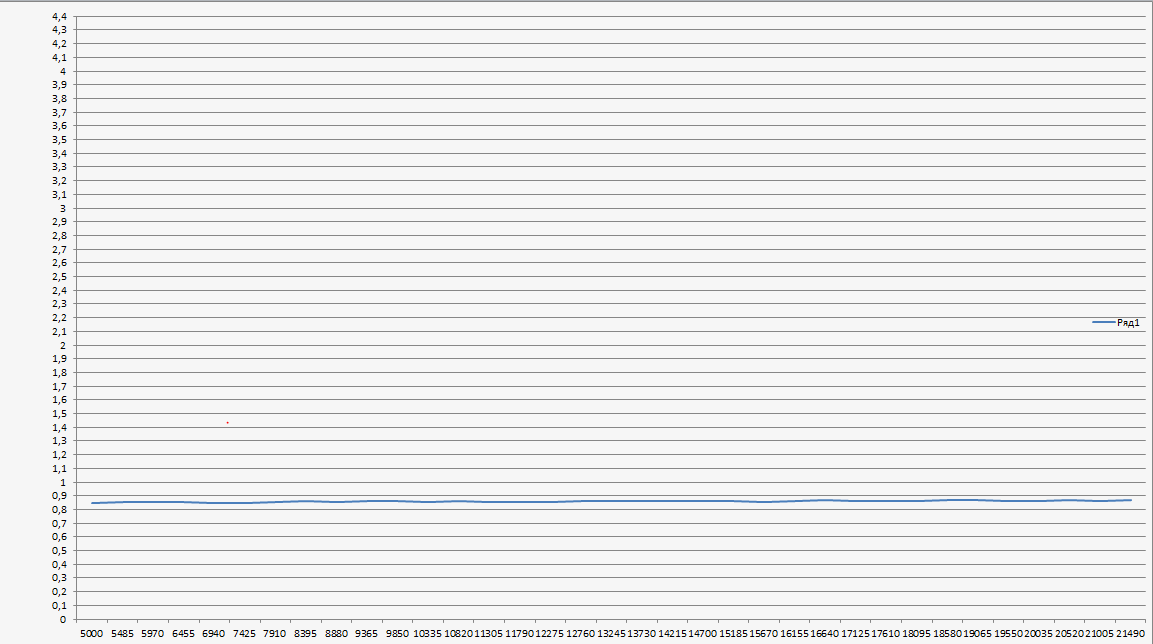


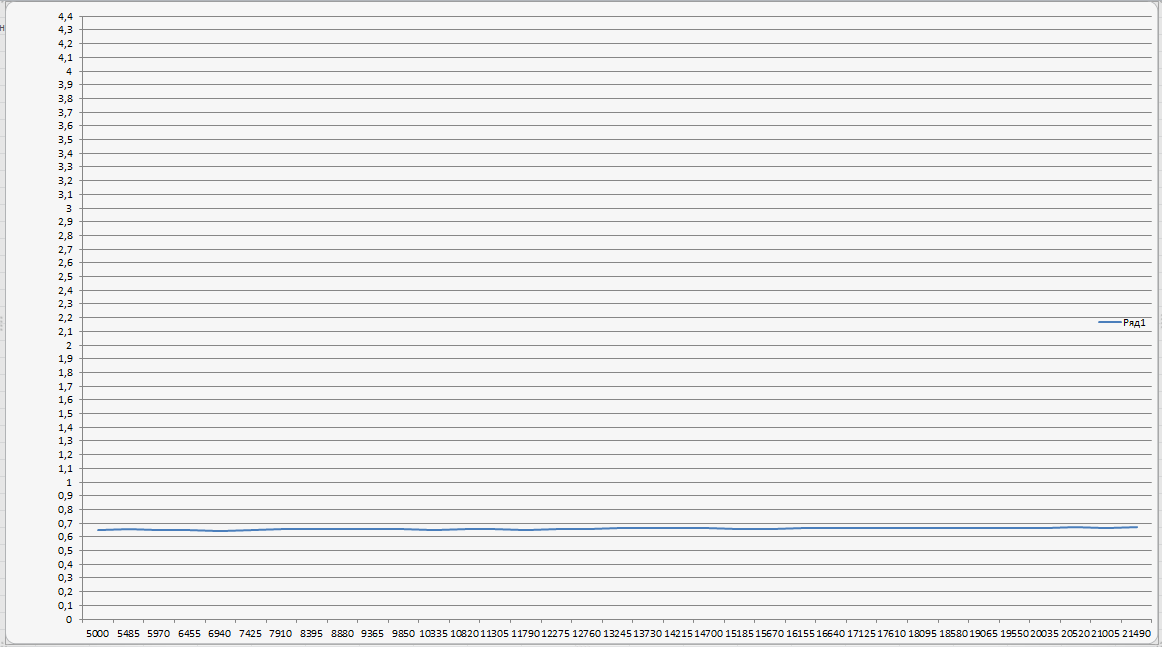
График 7-Зависимость отношения (сравнение/сложность)от количества элементов в массиве

График 8- зависимость отношения(перестановки/сложность)от количества элементов в массиве

## Сортировка слиянием:

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

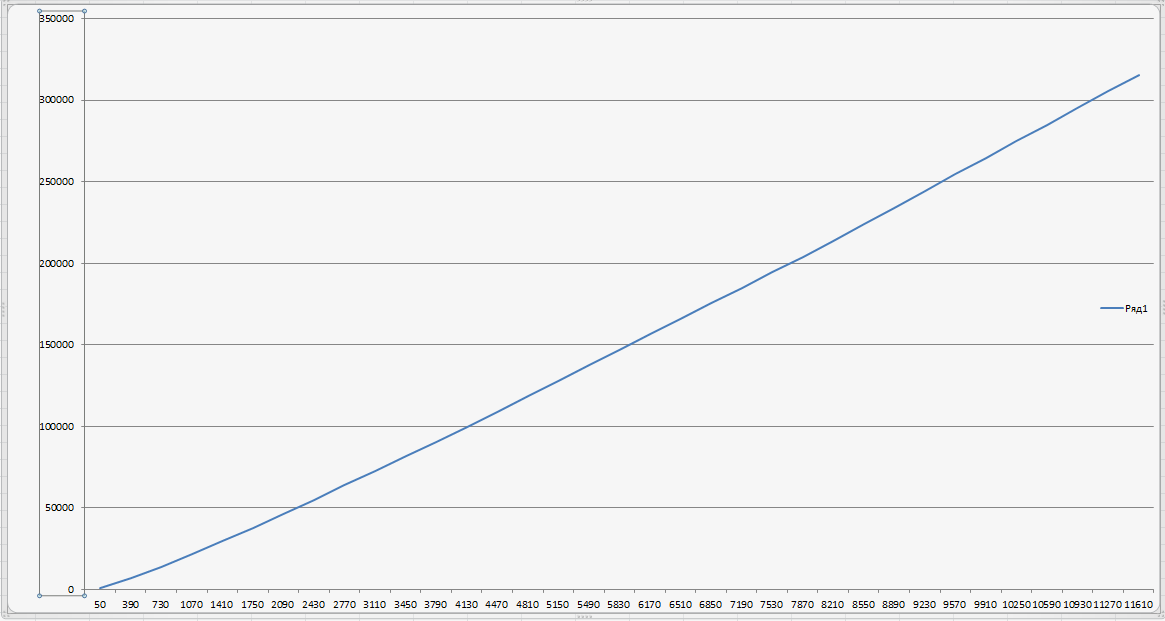
Чтобы доказать сложность данной сортировки построим график зависимости количества сравнений и присвоений от длины массива N. При делении полученных значений на предполагаемую сложность, то получатся графики, сходящиеся к константе. Для графика, на котором изображено отношение присвоений деленных на сложность, эта константа равна ~2,9. Для графика, на котором изображено отношение сравнений деленных на сложность, это константа равна ~5,5. В совокупность это доказывает предполагаемую сложность сортировки. 

График 9 – Зависимость присвоений от длины массива

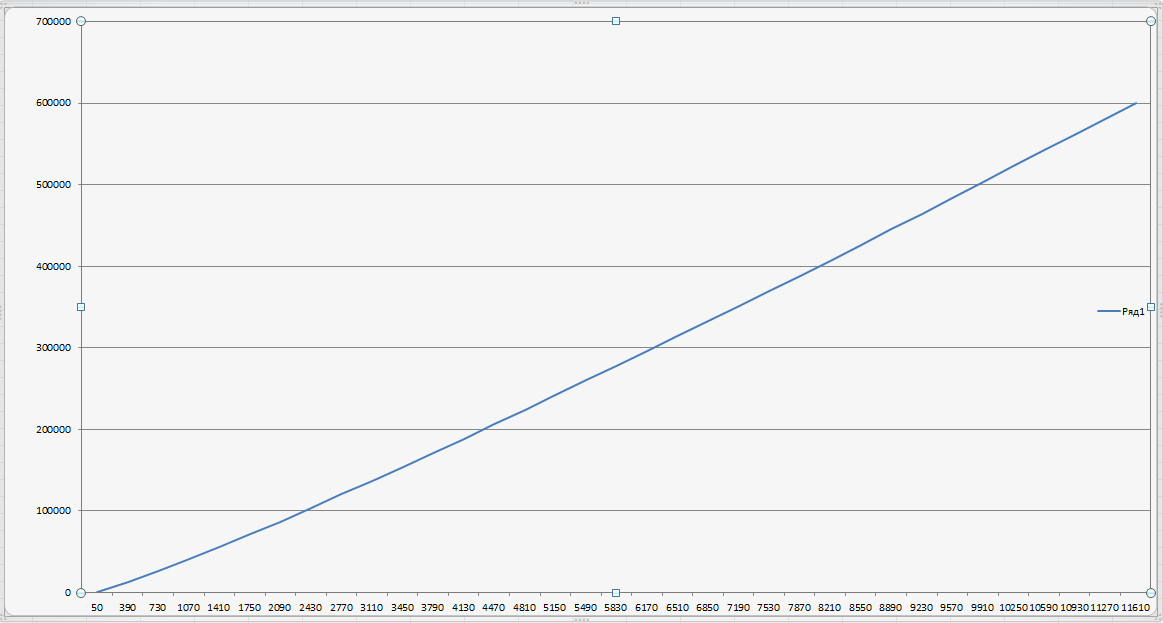


График 10- зависимость перестановок от количества элементов в массиве

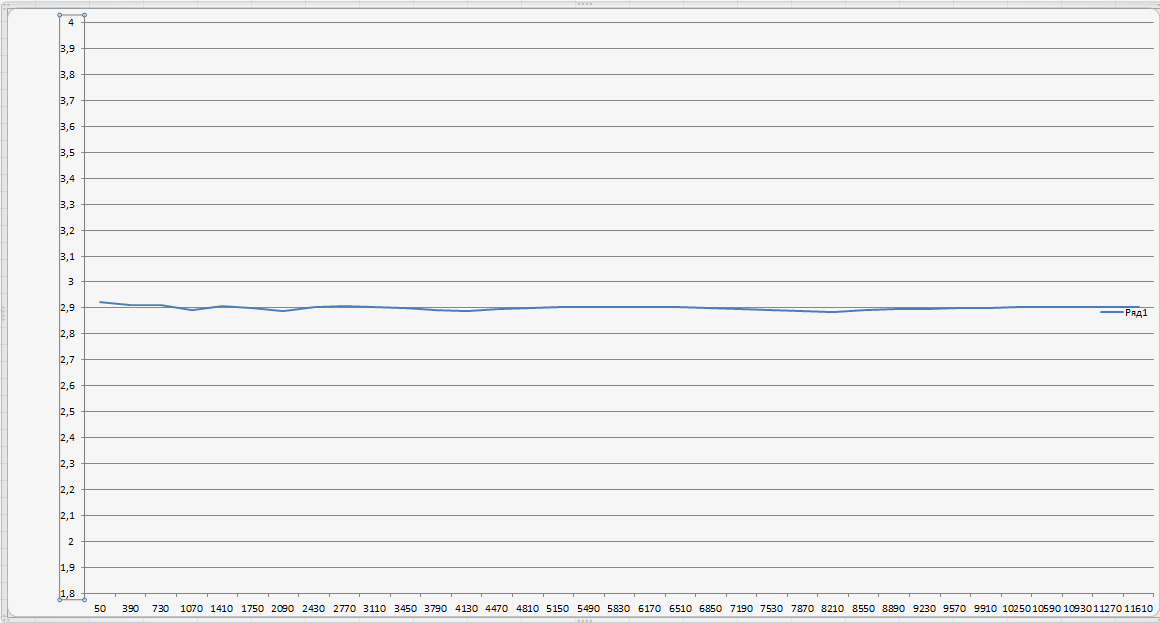


График 11- присвоение/сложность

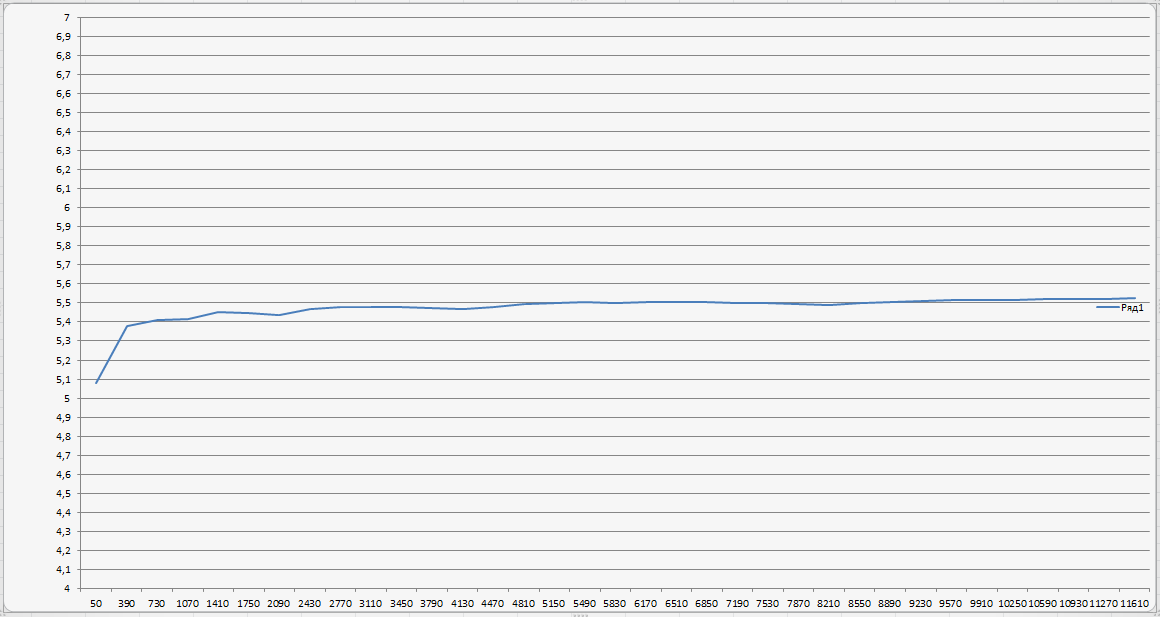


График 12- перестановки/сложность

## Поразрядная сортировка

Сложность данной сортировки: О(k\*(n+m)+n))

Чтобы доказать сложность данной сортировки построим график зависимости количества сравнений и присвоений от длины массива N. При делении полученных значений на предполагаемую сложность, то получатся графики, сходящиеся к константе. Для графика, на котором изображено отношение присвоений деленных на сложность, эта константа равна ~6,1. Для графика, на котором изображено отношение сравнений деленных на сложность, это константа равна ~3,24. В совокупность это доказывает предполагаемую сложность сортировки.

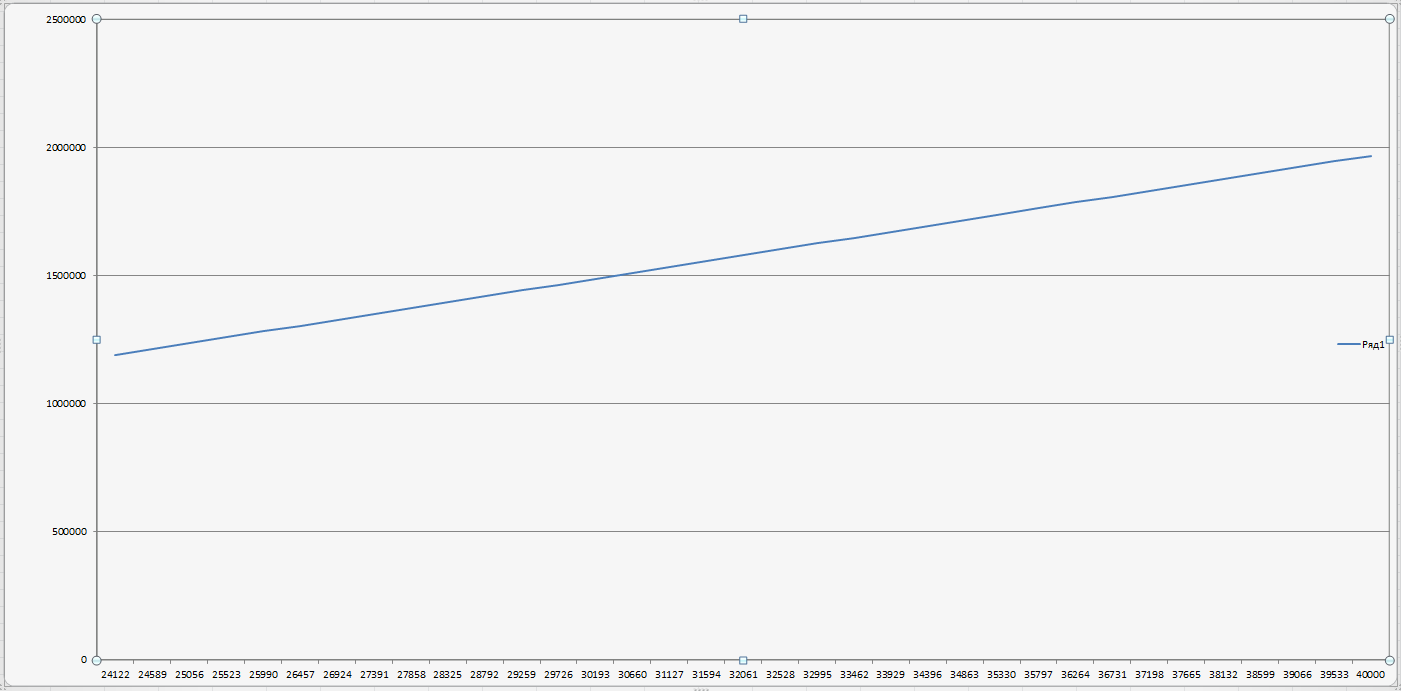


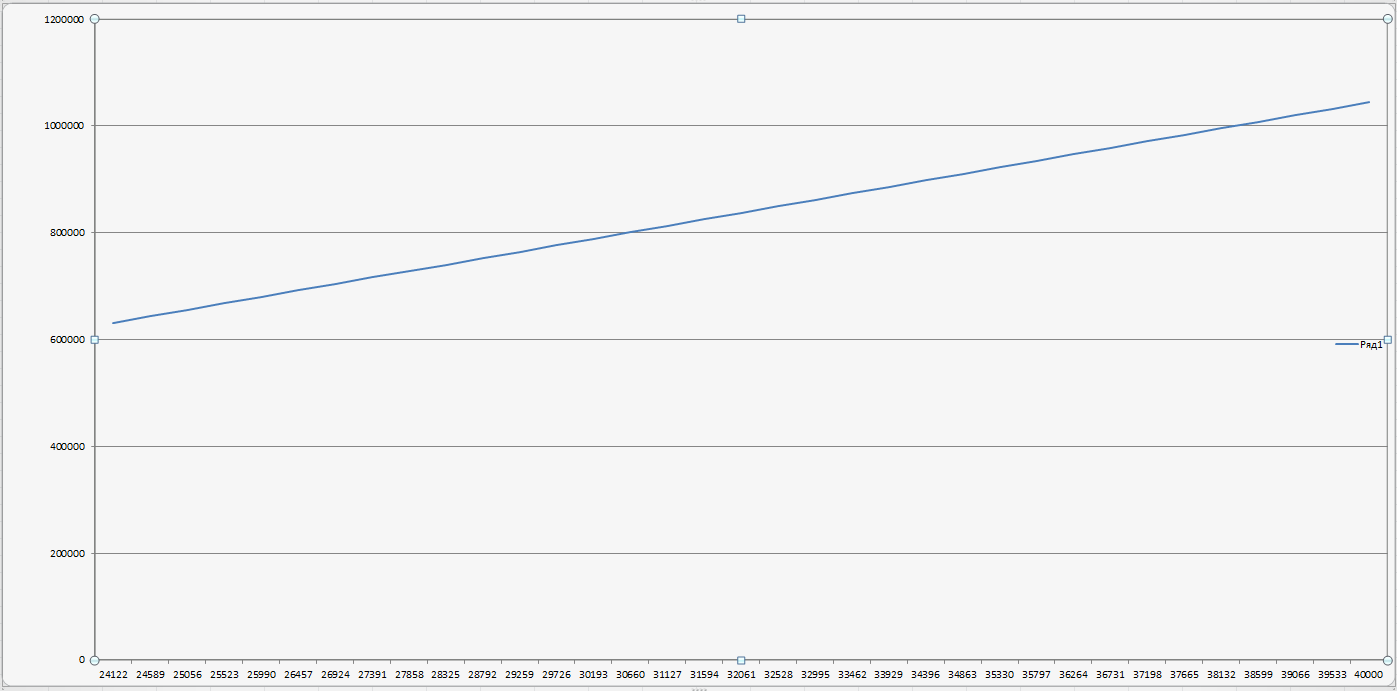
График 13- зависимость присвоений от количества элементов в массиве 

График 14- зависимость сравнений от количества элементов в массиве

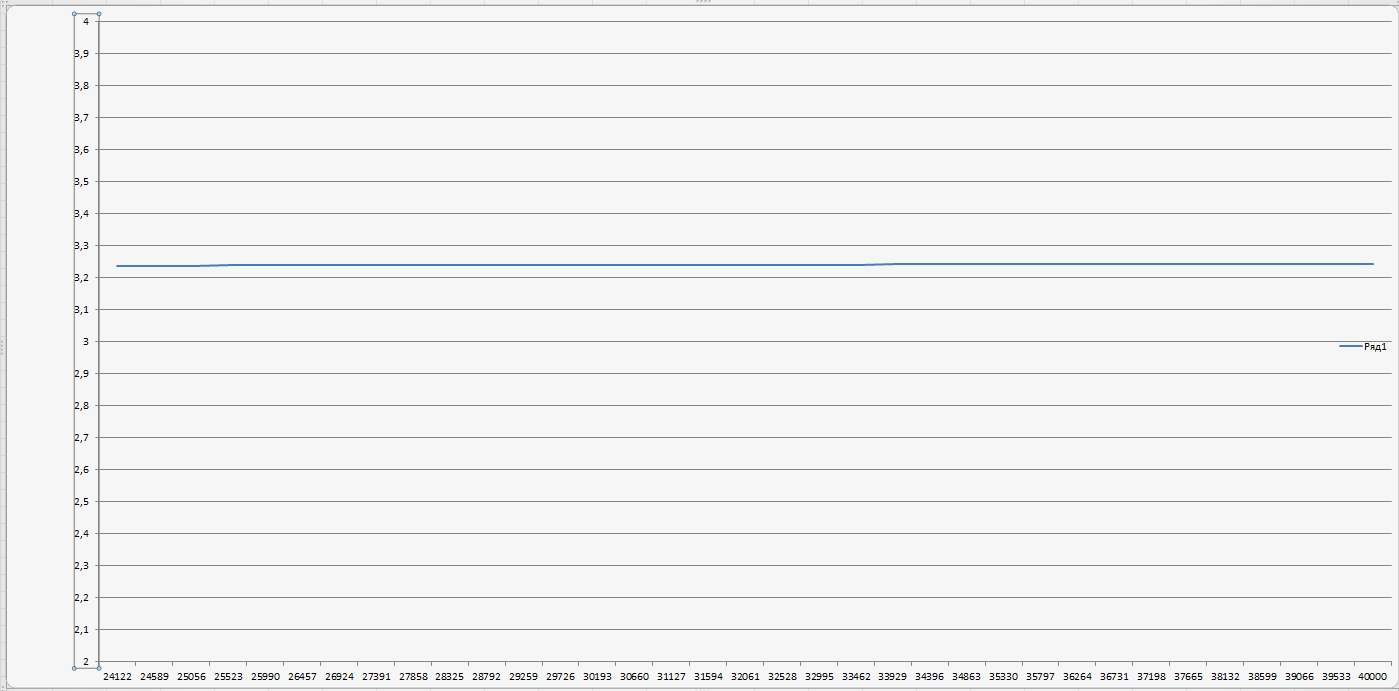


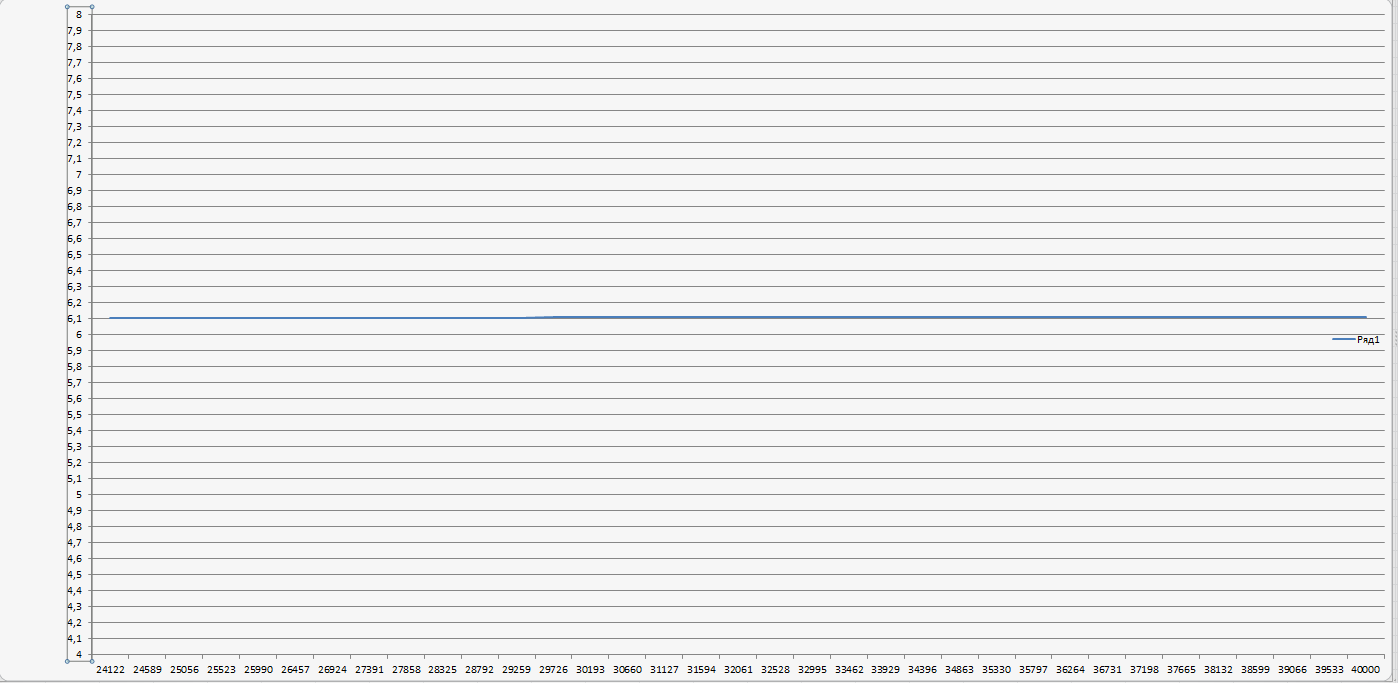
График 15- Зависимость сравнений от длины массива при делении на O(8\*(n+256)+n)

График 16- Зависимость присвоений от длины массива при делении на O(8\*(n+256)+n)

# Заключение

В ходе лабораторной работы были реализованы сортировка пузырьком, сортировка расчёской, сортировка слиянием и поразрядная сортировка на языке программирования Си для элементов типа double. Были описаны алгоритмы работы данных сортировок, проведена проверка корректности, приведены методы решения, описана программная реализация и проведены эксперименты для подтверждения их теоретической сложности.