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

Сулаймонов Н.Д.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**Содержание**

**1)Постановка задачи …...................................................................................4**

**2)Метод решения........................................................................................5**

**3)Руководство пользователя..................................................................8**

**4)Описание программной реализации...............................................11**

**5)Подтверждение корректности..........................................................15**

**6)Результаты экспериментов...............................................................16**

**7)Заключение..........................................................................................25**

**8)Приложение.........................................................................................26**

￼

￼

￼

￼

￼

￼

￼

￼

# Постановка задачи

В этой лабораторной работе главная задача является в реализации 4 сортировок массива (код написан на Си). Всего 4 сортировки: сортировка вставками, сортировка расческой, сортировка слиянием и поразрядная сортировка. Все эти сортировки написаны для упорядочивания элементов массива типа double. Каждый массив после выполнения сортировки должен быть упорядочен по возрастанию. Для каждой сортировки написать методы решений, руководство пользователя, описание программной реализации, подтверждение корректности, результаты экспериментов, приложение (основа кода сортировки) и заключение по всей лабораторной работе.

# Метод решения

**Сортировка вставками**

Сортировка вставками относится к простым сортировкам. Простые сортировки используют сравнения, просты в понимание и реализации. Перейдем к методу решения, мы делим массив на отсортированную и не отсортированную часть. При этом в начальный момент наша отсортированная последовательность является пустой. На каждом шаге алгоритма будет выбираться один из элементов входных данных и помещаться на нужную позицию уже в отсортированною часть последовательности до тех пор, пока набор входных данных не будет исчерпана также в любой момент времени в отсортированной части элементы удовлетворяют требованиям к выходным данным алгоритма.

Пример на иллюстрации:

На 1 шаге нашей сортировки 5(неотсортированный) элемент сравнивается с 4 элементом. Так как 4 больше 5 то они меняются местами. На 2 шаге сравниваются уже 4 и 3 элемент. Так как 3 больше 4 то они меняются местами.

На 3 шаге уже 3 элемент сравнивается со 2. Так как 2 элемент меньше 3 то перестановки не будет. Шаг завершен, элемент встал на свое место.

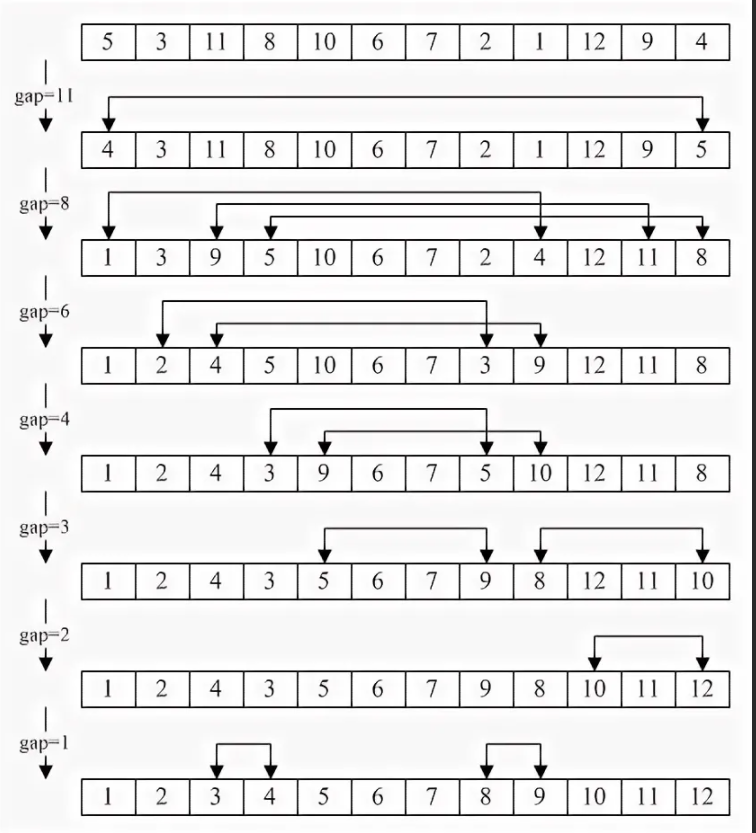


**Сортировка расческой**

Сортировка расческой относится к быстрым сортировкам. В сортировке расческой используется метод пузырьковой сортировки. В пузырьковой сортировке используется сравнение соседних элементов и продвижение их элементов на 1 позицию. В сортировке расчёской надо первоначально брать достаточно большое расстояние между сравниваемыми элементами и по мере упорядочивания массива сужать это расстояние до минимального. Можно скачать, что мы как бы причесываем массив, постепенно разглаживая на все более аккуратные подмножества. Все элементы двигаются сразу на несколько позиций в сторону своего места.

После этого вызывается сортировка пузырьком, но обычно для массива достаточно 1 прохода. Первоначальный разрыв между сравниваемыми элементами лучше брать с учетом специальной величины, называемой фактором уменьшения, ее значение равно примерно равно 1,247.

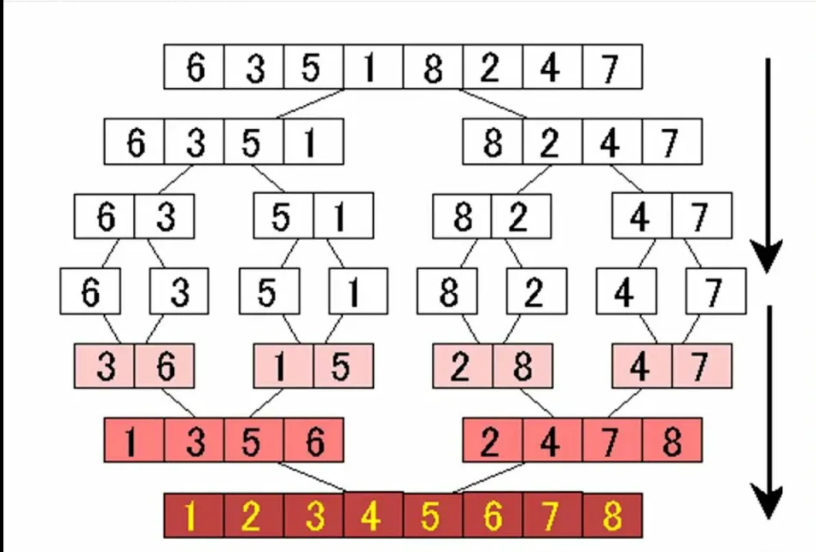
Пример на иллюстрации:



**Сортировка слиянием**

Сортировка слиянием относится к быстрым сортировкам. Основа этого метода является в объединение двух отсортированных массивов. Мы разбиваем наш массив на 2 части примерно одинакового размера и из каждой получившихся частей сортируется отдельно. Мы разбиваем наш массив до тех пор, пока размер массив не достигнет единицы.(Массив длины 1 является упорядоченным).Потом мы соединяем 2 упорядоченных массивов в 1.На каждом шаге мы берем меньший из 2-х первых элементов подмассивов и записываем его в результирующий массив. Счетчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1 и когда один из подмоссивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

Пример на иллюстрации:

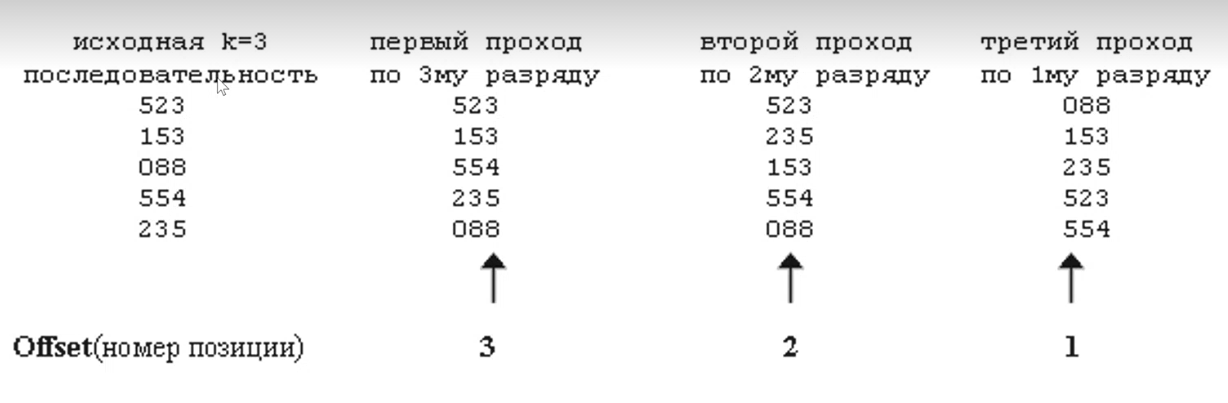


**Поразрядная сортировка**

Поразрядная сортировка относится к линейным сортировкам. В ее основе лежит использование сортировки подсчетом и ее свойство устойчивости.

Предполагается, что каждый ключ сортировки можно рассматривать как k-значное число, каждая цифра которого находится в диапазоне от 0 до m-1. Мы поочередно используем устойчивую сортировку (сортировка подсчетом) для каждой цифры справа налево. Порядок сортировки цифр или символов действительно важен.

Пример на иллюстрации:



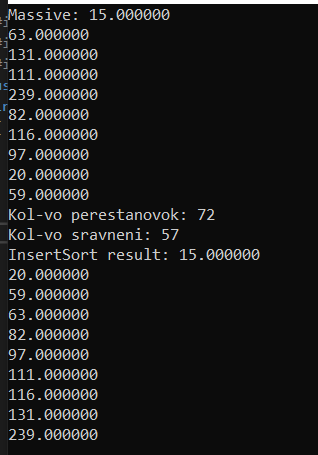
# Руководство пользователя

Руководство для пользователя для использования сортировок массивов очень просто.(Все массивы состоят из элементов типа double).

**1)Сортировка вставками(Insert Sort):** В этой сортировке можно изменить размер вашего массива, вы можете выбрать любой размер (но так как в сортировке вставками используется генератор случайных чисел от 0 до 255, то при больших размерах массива число могут повторяться).



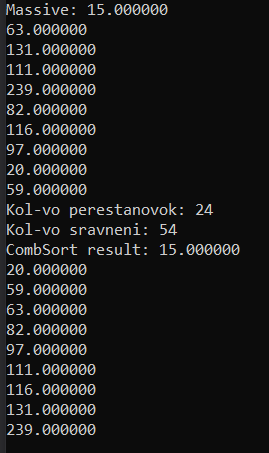
После запуска программы она выведет наш неотсортированный массив, потом кол-во сравнений и перестановок, а также в конце наш упорядоченный по возрастанию массив.



**2)Сортировка расческой(Comb Sort):** Руководство пользователя аналогично с сортировкой вставками. В этой сортировке можно изменить размер вашего массива, вы можете выбрать любой размер (но так как в сортировке вставками используется генератор случайных чисел от 0 до 255, то при больших размерах массива число могут повторяться).



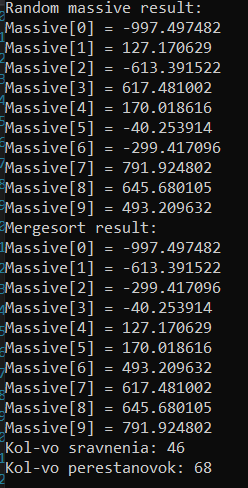
После запуска программы она выведет наш неотсортированный массив, потом кол-во сравнений и перестановок, а также в конце наш упорядоченный по возрастанию массив.



**3)Сортировка слиянием(Comb Sort):** В этой сортировке можно изменить размер вашего массива на любую величину (генератор рандомного массива создает элементы от -1000 до 1000)



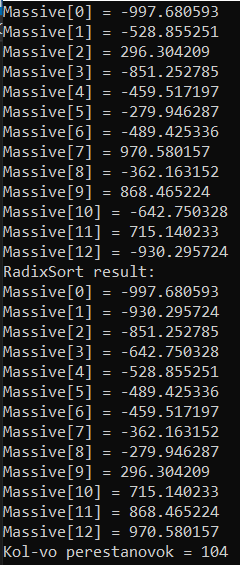
После запуска программы она выведет сначала неотсортированный массив, а следующим она выведет наш сортированный массив, после этого выводится кол-во сравнение и перестановок.



**4)Поразрядная сортировка(Radix Sort):** Руководство пользователя аналогично с сортировкой слиянием. В этой сортировке можно изменить размер вашего массива на любую величину (генератор рандомного массива создает элементы от -1000 до 1000)



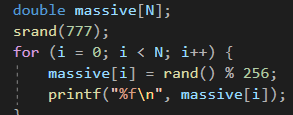
После запуска программы она выведет сначала неотсортированный массив, а следующим она выведет наш сортированный массив, после этого выводится кол-во перестановок.(Так линейная сортировка не используются сравнения).



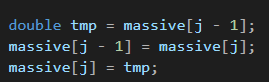
# Описание программной реализации

**1)Сортировка вставками(Insert Sort)**

В самом начале объявляется 3 переменных: кол-во перестановок, кол-во сравнений, размер массива. Главная функция int main() в ней производиться рандомное создания массива с элементами типа double от 0 до 255 и сама сортировка и вывод всех нужных нам значений.



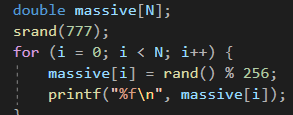
После этого идет сама сортировка. Работа ее такова, второй элемент сравнивается с первым: если первый больше второго, то они меняются местами; на втором шаге третий элемент сравнивается сначала со вторым, потом с первым и т. д. Среди уже отсортированных элементов вставляется новый элемент без нарушения порядка.



После завершения сортировки при помощи функции printf я вывожу сам неотсортированный массив, кол-во сравнений и перестановок, а также сам упорядоченный по возрастанию массив.(Также при помощи функции count я вывожу надписи для удобного нахождения всех нужных значений)

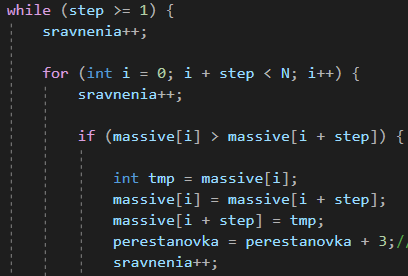
**2)Сортировка расческой(Comb Sort)**

В самом начале объявляется 3 переменных: кол-во перестановок, кол-во сравнений, размер массива. Главная функция int main() в ней производиться рандомное создания массива с элементами типа double от 0 до 255 и сама сортировка и вывод всех нужных нам значений. Дальше идет сама сортировка расческой .



Первоначальный разрыв между сравниваемыми элементами лучше брать с учетом специальной величины, называемой фактором уменьшения, ее значение равно примерно равно 1,247.(Мы вводим эту переменную). Также у нас есть переменная step, которая указывает на наш шаг межу сравниваемыми элементами .Дальше по мере упорядочивания массива сужать это расстояние до минимального. Мы как бы причесываем массив, постепенно разглаживая на все более аккуратные подмножества. Все элементы двигаются сразу на несколько позиций в сторону своего места.

После этого вызывается сортировка пузырьком.



После завершения сортировки при помощи функции printf я вывожу сам неотсортированный массив, кол-во сравнений и перестановок, а также сам упорядоченный по возрастанию массив.(Также при помощи функции count я вывожу надписи для удобного нахождения всех нужных значений)

**3)Сортировка слиянием(Merge Sort)**

В начале вводиться необходимы нам переменные.(размер массива и т.д).

**void MergeSort(double massive[], int l, int r)** — это функция отвечает за разбиение массивов на подмассивы и т.д. Эта функция принимает такие переменные как: наш массив, порядковый номер левой части массива (самый крайний элемент с лево) и правой части массива (самый правый элемент). MergeSort(massive, l, q) - отвечает за сортировку левой части. MergeSort(massive, q + 1, r) - отвечает за сортировку правой части. Merge(massive, l, q, r) - отвечает за слияния частей в 1 результирующий массив.

**void Merge(double massive[], int left, int sredni, int right)** — это функция отвечает за упорядочивание 2-х частей нашего массива. Является важной функцией в сортировке слиянием. Также это функция принимает такие переменные как: наш массив, левую часть массива, правая часть массива и средний элемент.

**int main()** — это функция отвечает за генерацию нашего рандомного массива, а также вывод нашего отсортированного массива, кол-во перестановок и кол-во сравнений.

**4)Поразрядная сортировка(Radix Sort)**

В начале программы вводим переменные , размер массива и кол-во перестановок.

**void RadixSortVizov(double\* in, long int N)** - функция является основной, при помощи нее мы вызываем нашу сортировку, а также выделяет память на работу сортировку с нашими массивами. Это функция принимает такие перменные как размер нашего массива и сам указатель на него.

**void RadixSort(double\* in, double\* out, long int\* counters, long int N)** - это функция обозначает о перемещении от младшего байта к самому старшему. При этом выполняется конечная часть сортировки. Принимаемые переменные: указатель на вспомогательный массив, длина массива, указатель на наш массив.

**void RadixSortsign(short int offset, long int N, double\* source, double\* dest, long int\* count)** - это функция подсчитывает кол-во отрицательных чисел. Для положительных чисел создаем вспомогательный массив .Следующие числа отрицательные, аналогично делаем это и с ними. Дальше записываем элементы массива на их позиции. Принимаемые переменные: размер массива, номер нашей позиции, массив dest, массив для счетчиков.

**void RadixPass(short int offset, long int N, double\* source, double\* dest, long int\* count)** - это функция помогает найти вспомогательный массив(в этом массиве будет показываться кол-во цифр, т.е. с какой позиции будут начинаться числа с нашей цифрой).Дальше все сортируется при этом кладя числа в нужные нам позиции. Принимаемые переменные: размер массива, сам наш массив, offset- номер позиции, dest- массив чисел, которые отсортированы по байту offset, массив из счетчиков(соответствует текущему шагу).

**void Counterscreate(double\* data, long int\* counters, long int N)** - это функция создает наш вспомогательный массив, подсчитывающий одинаковые значения в нашем массиве, по индексу значения +1.Принмаемые переменные: Размер массива, указатель на наш исходный массив, указатель на наш вспомогательный массив.

# Подтверждение корректности

Для подтверждения корректности моих сортировок я использовал сортировку вставками и сравнивал с другими моими сортировками. Сортировка вставками является простой сортировкой и ее правильность работы очевидна .Так как у поразрядной сортировки и сортировки слиянием разная генерации массива, то эту генерацию массива я поставил на сортировку вставками(Все сортировки работают корректно и без ошибок с отрицательными числами, также я поставил такую же генерацию массива на сортировку расческой для работы с отрицательными числами).

# Результаты экспериментов

**1)Сортировка вставками(Insert Sort)**

Сортировка вставками - простая сортировка. Теоретическая сложность сортировки ставками составляет О(n^2).

Лучший случай сортировки - массив уже отсортирован по возрастанию

Худший случай сортировки - вводимый массив в самом начале отсортирован по убыванию.

Сортировка вставками – квадратичный алгоритм, время его работы пропорционально квадрату от размера массива. Так как в худшем случае массив является упорядочен по убыванию то по алгоритму кол-во перестановок равно (n-1)\*n\*(½)

O(n2) сравнений и столько же перестановок

График сложности:

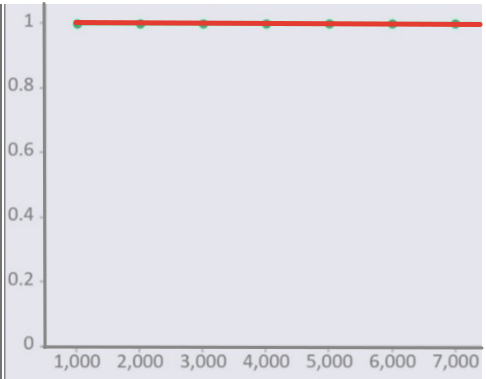
Столбец обозначает значение после выполнения сортировки

Строка обозначает длину массива



Если мы делим наш столбец на длину массива в квадрате(наша строка)

После этого можно сказать, что после этого мы получили константу для всех наших n (размеров массивов).Округлив значение получим примерно 1.



**2)Сортировка расческой(Comb Sort)**

Сортировка вставками - быстрая сортировка. Теоретическая сложность сортировки ставками составляет O(n^2).

Лучший случай сортировки - массив уже отсортирован по возрастанию(Сложность О(n\*logn)

Худший случай сортировки - вводимый массив в самом начале отсортирован по убыванию.(Сложность O(n^2)

График сравнения от длины нашего массива:

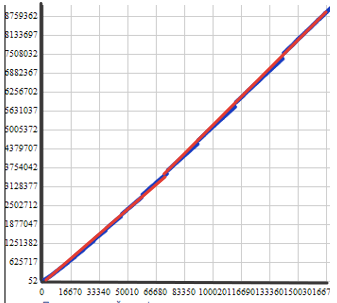


График сравнений от длины массива(если мы делим сравнения на n\*log(n)):

График стремиться к константе примерно равной 3.2.

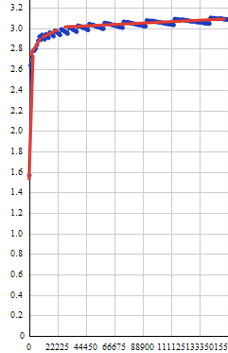


График присвоений(перестановки) от длины нашего массива:

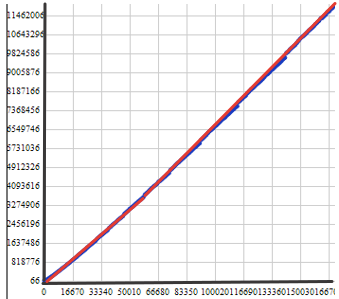
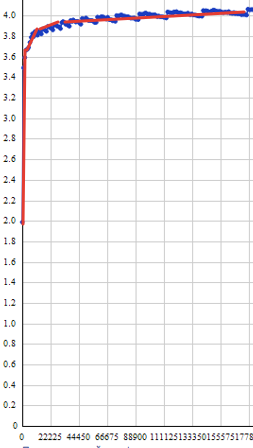


График присвоений(перестановки) от длины массива(если мы делим присовения на n\*log(n)):

График стремиться к константе примерно равной 4.05.



**3)Сортировка слиянием(Merge Sort)**

Сортировка слиянием - быстрая сортировка. Теоретическая сложность алгоритма О(n\*log(n)).

График сравнения от длины нашего массива:

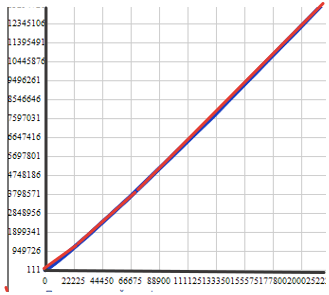


График сравнений от длины массива(если мы делим сравнения на n\*log(n)):

График стремиться к константе примерно равной 3.5

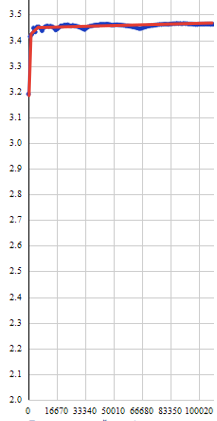


График присвоений(перестановки) от длины нашего массива:

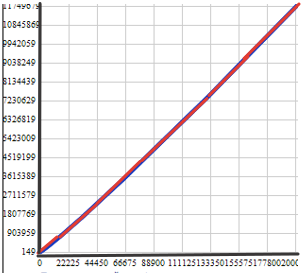
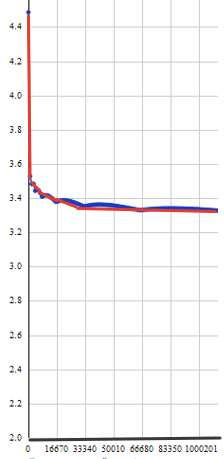


График присвоений(перестановки) от длины массива(если мы делим присовения на n\*log(n)):

График стремиться к константе примерно равной 3.35.



**4)Поразрядная сортировка(Radix Sort)**

Поразрядная сортировка - линейная сортировка. Сложность составляет О(k\*(n+m)+n).m- кол-во цифр в разряде. k-кол-во цифр в числе(или число разрядов)

График присвоений(перестановки) от длины нашего массива:

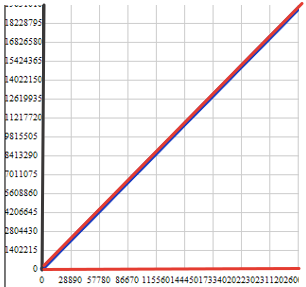


График присвоений(перестановки) от длины нашего массива(Если мы делим на сложность О(k\*(n+m)+n)):

График стремиться к константе примерно равной 8.25

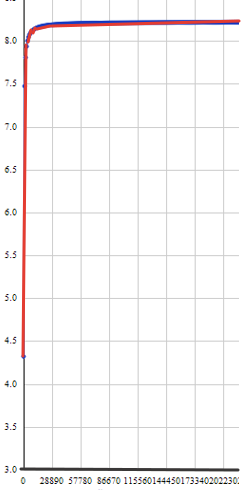


График сравнения от длины нашего массива:

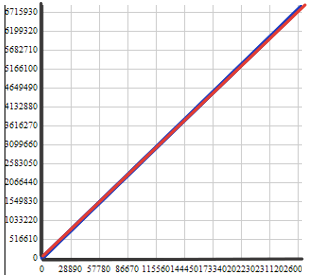
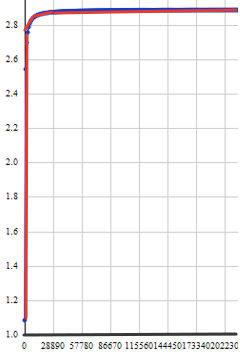


График сравнений от длины массива(если мы делим сравнения на О(k\*(n+m)+n)):

График стремиться к константе примерно равной 2.9

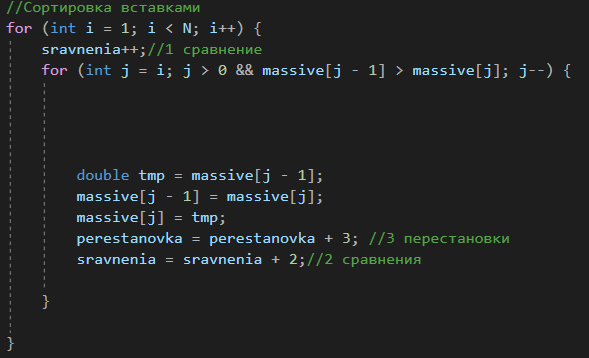


# Заключение

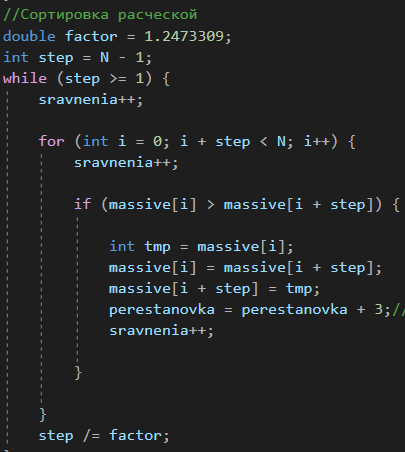
Я выполнил данную лабораторную работу. Реализовал все 4 сортировки : сортировка вставками, сортировка расческой, сортировка слиянием и поразрядная сортировка, для элементов типа double. Также я показал метод решения и описал программную реализацию каждой сортировки. Я подтвердил корректность каждой сортировки и провел ряд экспериментов, показывающих теоретическую сложность каждой моей сортировки.

# Приложение

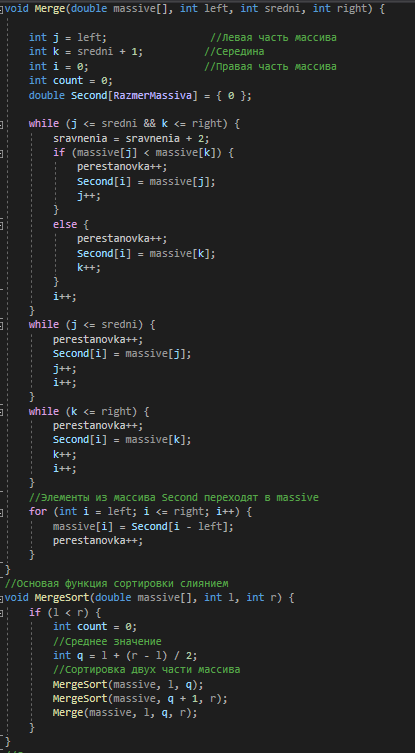
**1)Сортировка вставками(Insert Sort)**



**2)Сортировка расческой(Comb Sort)**



**3)Сортировка слиянием(Merge Sort)**



**4)Поразрядная сортировка(Radix Sort)**

