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

**Отчёт по лабораторной работе**

**«Сортировки»**

**Выполнил**:

студент группы 3821Б1ПМ2

Борисов С.А.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Методы решения 4](#_Toc26962563)

Сортировка вставками........................................................................................................................4

Сортировка Шелла..............................................................................................................................4

Сортировка слиянием.........................................................................................................................5

Поразрядная сортировка....................................................................................................................6

[Руководство пользователя 8](#_Toc26962564)

[Описание программной реализации 9](#_Toc26962565)

[Подтверждение корректности 1](#_Toc26962566)1

[Результаты экспериментов 1](#_Toc26962567)2

Сортировка вставками......................................................................................................................12

Сортировка Шелла............................................................................................................................15

Сортировка слиянием.......................................................................................................................17

Поразрядная сортировка..................................................................................................................20

[Заключение 2](#_Toc26962568)3

# Постановка задачи

Цель лабораторной работы: реализовать на языке программирования C сортировку вставками, сортировку Шелла, сортировку слиянием и поразрядную сортировку для данных типа float, описать алгоритмы сортировок и их программную реализацию, подтвердить корректность программной реализации, а также провести эксперименты, показывающие теоретическую сложность алгоритмов, описать способ проведения экспериментов и сделать вывод на основе полученных результатов.

# Методы решения

**Сортировка вставками**

Суть данного алгоритма заключается в разделении массива на отсортированную (изначально пустую) и неотсортированную часть и последующем добавлении на каждом шаге элемента в отсортированную часть массива на свою позицию до тех пор, пока неотсортированная часть не станет пустой. Сложность этого алгоритма равна O(n2).



Рисунок 1 - иллюстрация работы сортировки вставками

**Сортировка Шелла**

Сортировка Шелла - это та же сортировка вставками, но с изменением расстояния между сравниваемыми элементами массива. В начале сравниваются элементы, находящиеся на расстоянии step. После этого сравниваются элементы, находящиеся на расстоянии step/2. То есть, каждый раз расстояние делится нацело на 2. И так до тех пор, пока это расстояние не станет равным нулю. Сложность данного алгоритма для худшего случая равна О(n2), а для лучшего О(n\*log2(n))

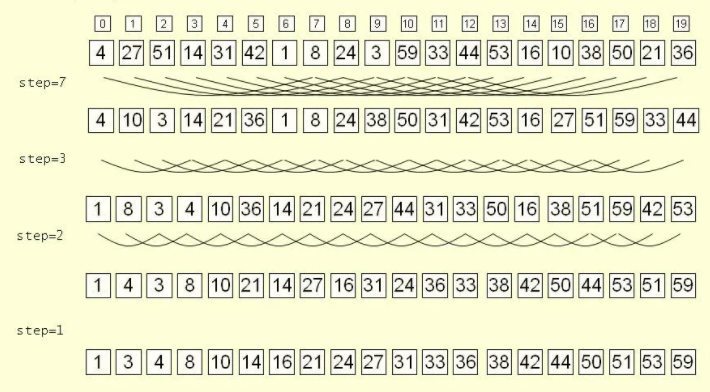


Рисунок 2 - иллюстрация работы сортировки Шелла

**Сортировка слиянием**

Суть данного алгоритма заключается в разделении массива на две части примерно одинакового размера. Обе части также делятся на две одинаковые части половинной длины. И так до тех пор, пока в итоге основной массив не будет разбит на массивы, состоящие из одного элемента. Такие массивы уже отсортированы, и единственное, что остаётся сделать, это соединить эти массивы в результирующий массив в нужном порядке. Сложность данного алгоритма равна O(n\*log(n)). Также данной сортировке необходимо O(n) памяти.

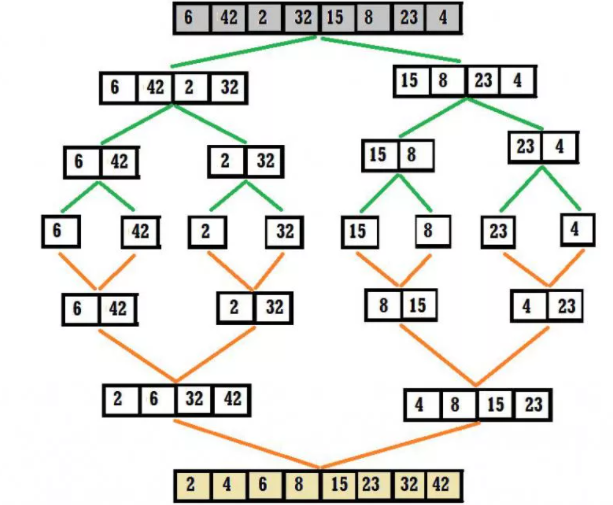


Рисунок 3 - иллюстрация работы сортировки слиянием

**Поразрядная сортировка**

Поразрядная сортировка является расширенной версией сортировки подсчётом. Её суть заключается в сортировке чисел по разрядам. Идя справа налево, то есть, от младшего разряда к старшему, применяется сортировка подсчётом. Так как поразрядная сортировка является устойчивой, то числа с разными младшими разрядами, но одинаковыми старшими не будут перемешаны при сортировке.

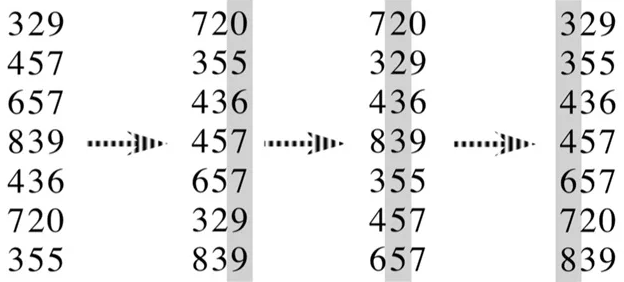


Рисунок 4 - иллюстрация работы поразрядной сортировки

# Руководство пользователя

В начале программа выводит пронумерованный список предлагаемых сортировок (рис. 5)

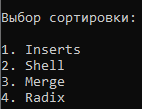


Рисунок 5 - вывод списка сортировок

Также на этом этапе программа выводит фразу «Номер сортировки: », напротив которой пользователю нужно ввести номер сортировки (цифру от 1 до 4) согласно выведенному списку.

Если введённые данные соответствуют одному из четырёх номеров сортировок, программа выведет фразу «Количество сравнений: », напротив которой выведется количество сравнений в ходе сортировки, фразу «Количество присваиваний: », напротив которой выведется количество присваиваний в ходе сортировки, а также фразу «Проверка сортировки: », напротив которой в случае, если выбранный алгоритм правильно отсортировал массив, выведется фраза «Сортировка корректна», а в противном случае выведется фраза «Сортировка НЕкорректна».

Если же введённые пользователем данные не удовлетворяют требованиям, то программа выведет фразу «Error» и сразу завершится.

# Описание программной реализации

void check(float\* array\_check, uint size\_check) - на вход функция принимает изначальную копию сортируемого массива и его размер. Эта функция сортирует копию исходного массива отлаженной сортировкой qsort из стандартной библиотеки языка C, используя ниже описанную функцию.

int compare(const void\* a, const void\* b) - на вход функция принимает два указателя на сравниваемые элементы. Эта функция возвращает отношение этих элементов, то есть, одно из них меньше, больше или равно другому.

**Сортировка вставками** void inserts\_Sort(float\* array\_1, uint size\_inserts) - на вход функция принимает указатель на сортируемый массив и длину этого массива.

**Сортировка Шелла**

void Shell\_Sort(float\* array\_2, uint size\_Shell) - на вход функция принимает указатель на сортируемый массив и длину этого массива.

**Сортировка слиянием**

void merge\_Sort(float\* array\_3, int first, int end, uint size\_merge) - на вход функция принимает указатель на сортируемый массив, крайний левый индекс сортируемой части массива, крайний правый индекс сортируемой части массива и длину сортируемого массива.

**Поразрядная сортировка**

void radix\_Sort(float\* array\_4, float\* out\_array\_4, int\* in\_counters, uint size\_radix) - на вход функция принимает указатель на сортируемый массив, указатель на вспомогательный массив размера, равного размеру сортируемого массива, указатель на массив-счётчик размера 256 \* sizeof(float) \* sizeof(int) и размер сортируемого массива. Эта функция сортирует сортирует входной массив, используя ниже описанные вспомогательные функции.

int\* createCounters(float\* data, int\* counters, uint size\_CC) - на вход функция принимает указатель на сортируемый массив, указатель на массив-счётчик длины 256 \* sizeof(float) \* sizeof(int), в который в ходе работы данной функции будет записываться количество каждого байта числа из сортируемого массива, и размер сортируемого массива. Функция возвращает полученный массив-счётчик.

void radixPass(short Offset, float\* sourse, float\* dest, int\* count, uint size\_pass) - на вход функция принимает номер разряда, указатель на сортируемый массив, указатель на вспомогательный массив, указатель на полученный на прошлой итерации подмассив массива-счётчика (созданного функцией createCounters()) с количеством байтов определённого разряда, и размер сортируемого массива. Данная функция сортирует числа по одному разряду.

void signedRadixLastPass(float\* sourse, float\* dest, uint size\_RLP) - на вход функция принимает сортируемый массив, вспомогательный массив и размер сортируемого массива. Данная функция сортирует отрицательные числа.

# Подтверждение корректности

Для подтверждения корректности программной реализации выбранной пользователем сортировки была использована функция check(). На вход подаётся изначальная копия сортируемого массива и его размер. Эта функция при помощи сортировки qsort() из библиотеки stdlib и вспомогательной функции compare(), устанавливающей отношение двух элементов, сортирует данный массив. Так как использующаяся в ходе работы этой функции сортировка входит в стандартную библиотеку языка C, то можно быть уверенным в корректности данной сортировки. В дальнейшем в основной функции main() полученный отсортированный массив поэлементно сравнивается с тем массивом, который был получен в ходе работы выбранной пользователем сортировки. Если найдётся несовпадающая пара чисел, то сравнение завершается и программа сообщит о том, что функция некорректна, выведя следующую строку: «Проверка сортировки: Сортировка НЕкорректна». Если же каждые из двух чисел с одинаковым индексом совпадают, то есть, массивы идентичны, то программа сообщит о том, что выбранная пользователем сортировка, выведя следующую строку: «Проверка сортировки: Сортировка корректна»

**Проверка сортировок для достаточно больших объёмов данных подтвердила корректность их программной реализации.**

# Результаты экспериментов

В ходе эксперимента длина массива изменялась с 10 до 10000 элементов с шагом 10 элементов. На каждом шаге считалось количество сравнений и количество присваиваний внутри сортировки. По полученным данным для каждой сортировки строились графики зависимости от длины массива следующих величин:

1. Количества сравнений
2. Количества присваиваний
3. Количества сравнений, делённого на предполагаемую сложность сортировки
4. Количества присваиваний, делённого на предполагаемую сложность сортировки

Последние два вида графиков строятся для подтверждения сложности сортировок. Для этого значения на графике должны сходиться к какому-то числу или попадать в небольшой диапазон около него.

Далее n - размер (длина) массива

**Сортировка вставками**

Предполагаемая сложность данной сортировки равна О(n2). Это видно по графикам, представленным на рисунках 6 и 7, походящим на параболы. Если разделить каждые полученные значения количества сравнений и количества присваиваний на n2, то получим графики, подтверждающие предполагаемую сложность данной сортировки. На рисунке 8 значения на графике сходятся к единице, а на рисунке 9 значения лежат в небольшом диапазоне около 0,5

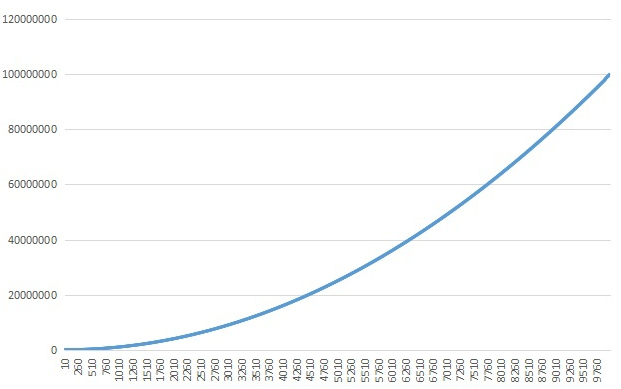


Рисунок 6 - график зависимости количества сравнений от длины массива в сортировке вставками

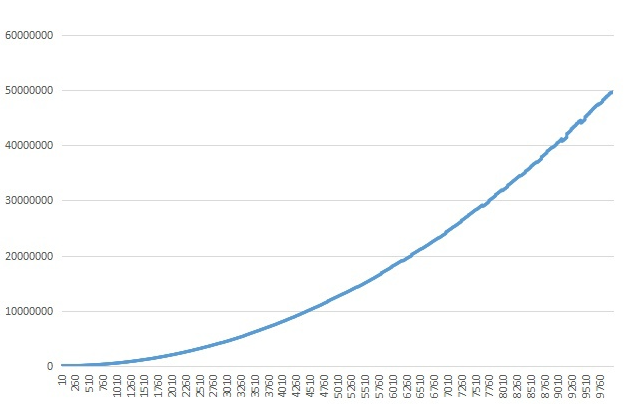


Рисунок 7 - график зависимости количества присваиваний от длины массива в сортировке вставками

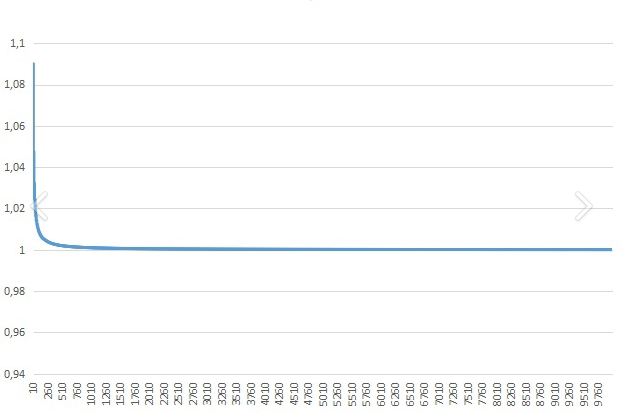


Рисунок 8 - график зависимости количества сравнений, делённого на предполагаемую сложность в сортировке вставками

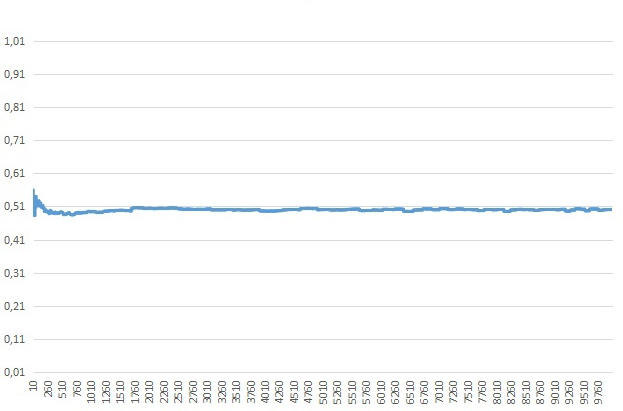


Рисунок 9 - график зависимости количества присваиваний, делённого на предполагаемую сложность в сортировке вставками

**Сортировка Шелла**

Средняя сложность данной сортировки равна O(n\*log2n). Количество сравнений и перестановок в этой сортировке сильно зависит от входных данных и способа уменьшения шага сравнения элементов (рисунки 10 и 11). Если разделить полученные значения на среднюю сложность, то они будут лежать на небольшом промежутке, что подтверждает сложность сортировки (рисунки 12 и 13)

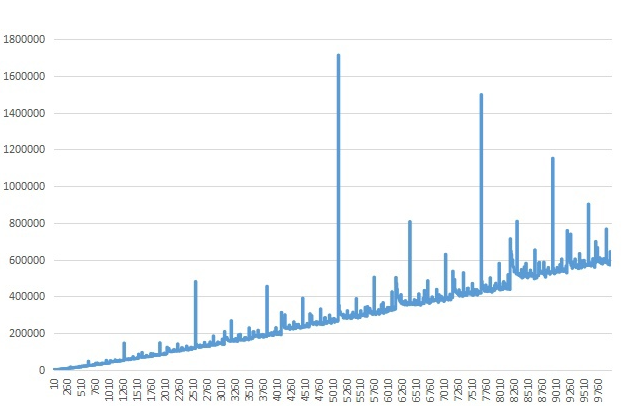


Рисунок 10 - график зависимости количества сравнений от длины массива в сортировке Шелла

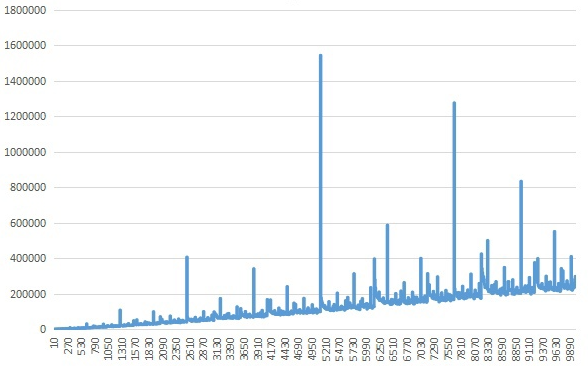


Рисунок 11 - график зависимости количества присваиваний от длины массива в сортировке Шелла

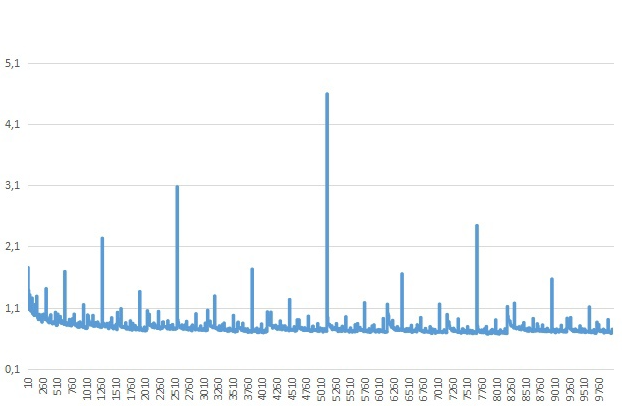


Рисунок 12 - график зависимости количества сравнений, делённого на предполагаемую сложность от длины массива в сортировке Шелла

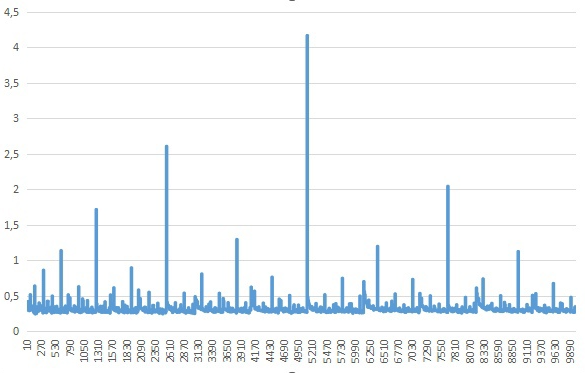


Рисунок 13 - график зависимости количества присваиваний, делённого на предполагаемую сложность от длины массива в сортировке Шелла

**Сортировка слиянием**

Сложность данной сортировки равна O(n\*log(n)). Графики зависимости количества сравнений и присваиваний от длины массива представлены на рисунках 14 и 15. Если разделить полученные значения на предполагаемую сложность O(n\*log(n)),

то по графикам, представленным на рисунках 16 и 17 видно, что полученные значения сходятся к какому-то числу, что подтверждает сложность данной сортировки.

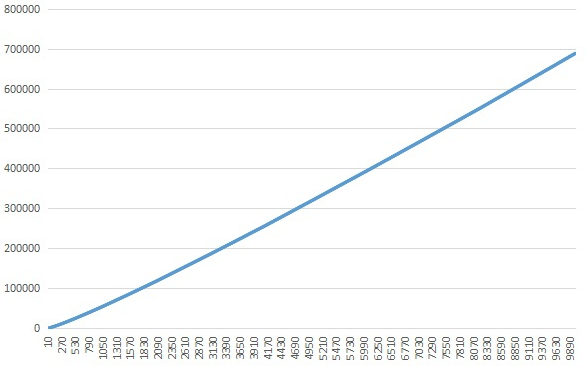


Рисунок 14 - график зависимости количества сравнений от длины массива в сортировке слиянием

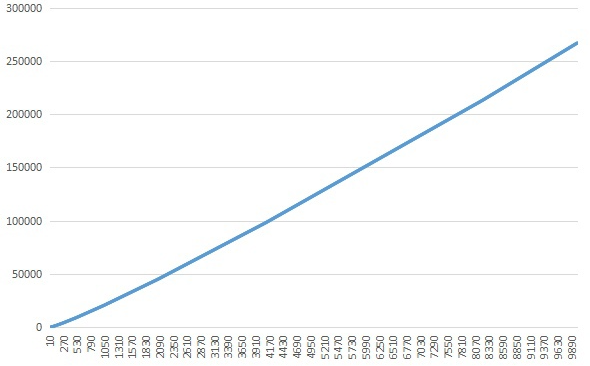


Рисунок 15 - график зависимости количества присваиваний от длины массива

в сортировке слиянием

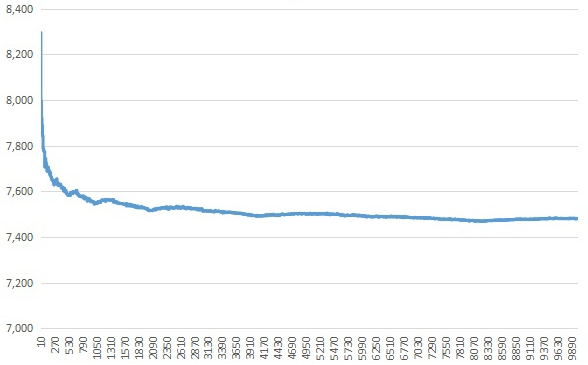


Рисунок 16 - график зависимости количества сравнений, делённого на предполагаемую сложность от длины массива в сортировке слиянием

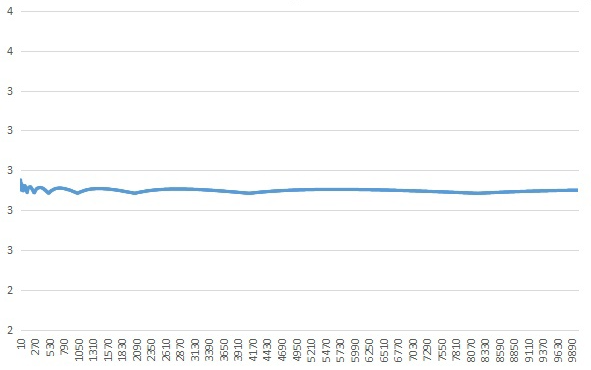


Рисунок 17 - график зависимости количества присваиваний, делённого на предполагаемую сложность от длины массива в сортировке слиянием

**Поразрядная сортировка**

Сложность данной сортировка равна O(k\*(n+m)+n), где k - длина числа из сортируемого массива в байтах, а m - количество возможных значений одного байта. Графики зависимости количества сравнений (в данном случае сравнений внутри циклов) и присваиваний от длины массива представлены на рисунках 18 и 19. Если разделить полученные значения на предполагаемую сложность, то полученные в результате деления значения будут сходиться к какому-то числу, что подтверждает сложность сортировки (рисунки 20 и 21)

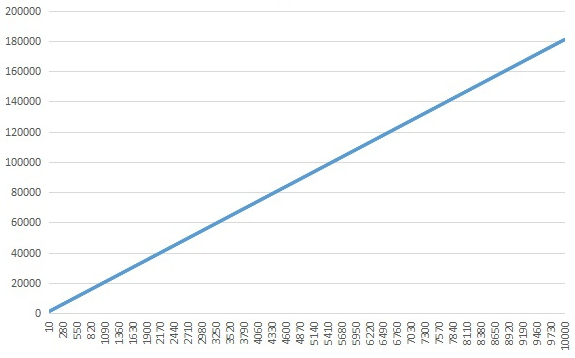


Рисунок 18 - график зависимости количества сравнений от длины массива в поразрядной сортировке

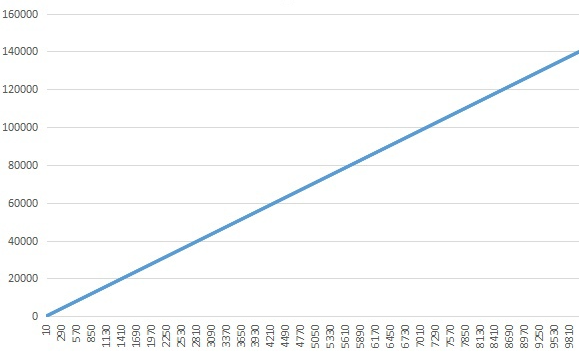


Рисунок 19 - график зависимости количества присваиваний от длины массива в поразрядной сортировке

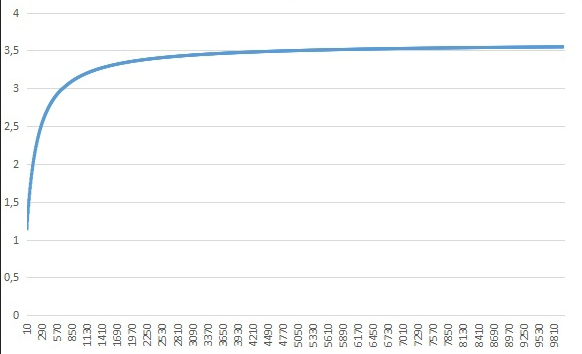


Рисунок 20 - график зависимости количества сравнений, делённого на предполагаемую сложность от длины массива в поразрядной сортировке

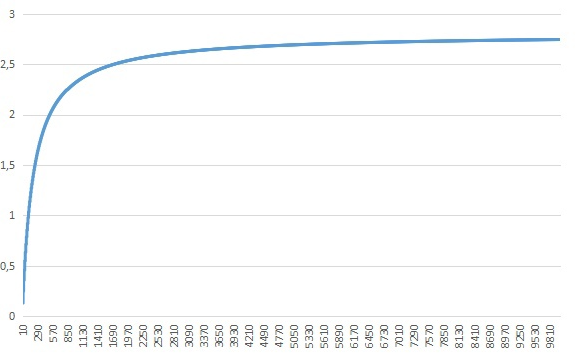


Рисунок 21 - график зависимости количества присваиваний, делённого на предполагаемую сложность от длины массива в поразрядной сортировке

# Заключение

В ходе лабораторной работы были реализованы на языке программирования C сортировка вставками, сортировка Шелла, сортировка слиянием и поразрядная сортировка для типа данных float. Также были описаны алгоритмы данных сортировок и их программная реализация, была подтверждена корректность программной реализации, были проведены эксперименты, показывающие теоретическую сложность алгоритмов, был описан способ проведения экспериментов и сделан вывод на основе полученных результатов.