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

Рымарь Артем Александрович

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**Содержание**

Постановка задачи……………………………………………….………………………………………………….3

Метод решения………………………………….………………………………….………………………………..4

Insertion sort……………………………………………………………….…………………………………………4

Shell sort………………………………………………………………………………………………………………..4

Merge sort……………………………………………………………………………………………………………..5

Radix sort……………………………………………………………………………………………………………….6

Руководство пользователя......................................................................................... 7

Описание программной реализации ....................................................................... 8

Insertion sort…………………………………………….……………………………………………………..……8

Shell sort……………………………………………………………………………………………………..………..8

Merge sort………………………………………………………………………………………………..…………..8

Radix sort………………………………………………………………………………………………..…………….8

Подтверждение корректности ............................................................................... 10

Результаты экспериментов ..................................................................................... 11

Insertion sort…………………………………………………………………………...…………………..……..11

Shell sort…………………………………………………………….………………………………………….…….12

Merge sort…………………………………………………………………………..……………………………...14

Radix sort………………………………………………………………………………………………………….….15

Заключение…………………………………………………………………………………………………..………17

**Постановка задачи**

Цель лабораторной работы – реализовать “Insertion Sort”, “Shell Sort”, “Merge sort” и “Radix Sort” для типа данных “float” на языке программирования C. После написания кода необходимо написать руководство пользователя, описать программную реализацию сортировк, проверить их корректность, доказать их сложность экспериментальным путем и сделать вывод по полученным результатам.

3

**Метод решения**

**“Insertion sort”**

Суть сортировки вставками заключается в том, что на каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. Стоит отметить, что последовательность из одного элемента является отсортированной.

Иллюстрация работы сортировки вставками для лучшего понимания алгоритма:

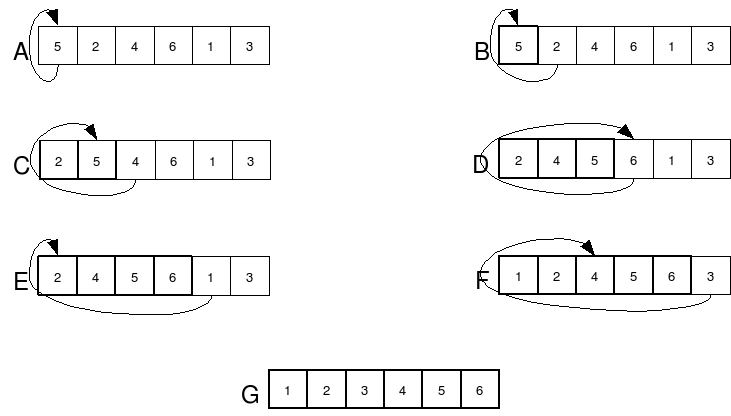
****

Рис. 1

Вычислительная сложность – О(n^2)

**“Shell sort”**

Сортировка Шелла – алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками. При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящиее один от другого на некотором расстоянии d. После этого процедура повторяется для некоторых меньших значений d, а завершается сортировка упорядочиванием элементов при d = 1(обычной сортировкой вставками). Стоит отметить, что среднее время работы алгоритма зависит от длин промежутков - d, на которых будут находиться сортируемые элементы исходного массив. Существует несколько подходов к выбору этих значений, каждый из подходов имеет свои плюсы и минусы.

4

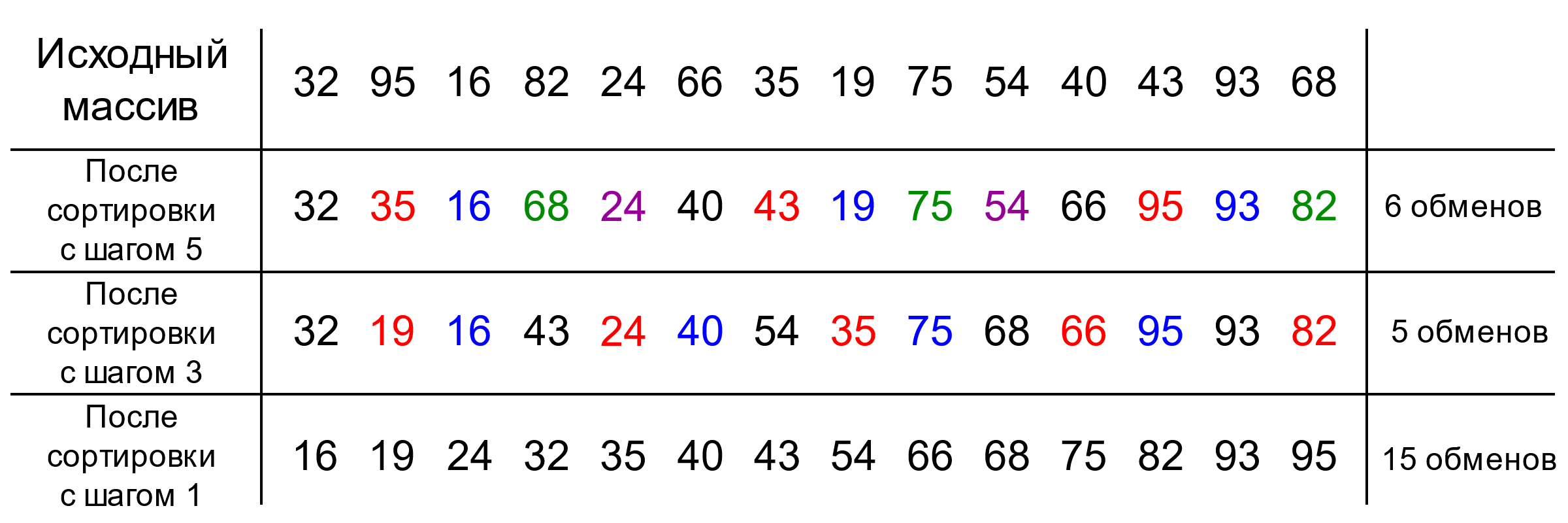
Иллюстрация работы сортировки Шелла для лучшего понимния алгоритма: 

Рис. 2

Вычислительная сложность алгоритма зависит от параметра d и сортируемого массива. В худшем случае, сложность алгоритма составит О(n^2).

**“Merge sort”**

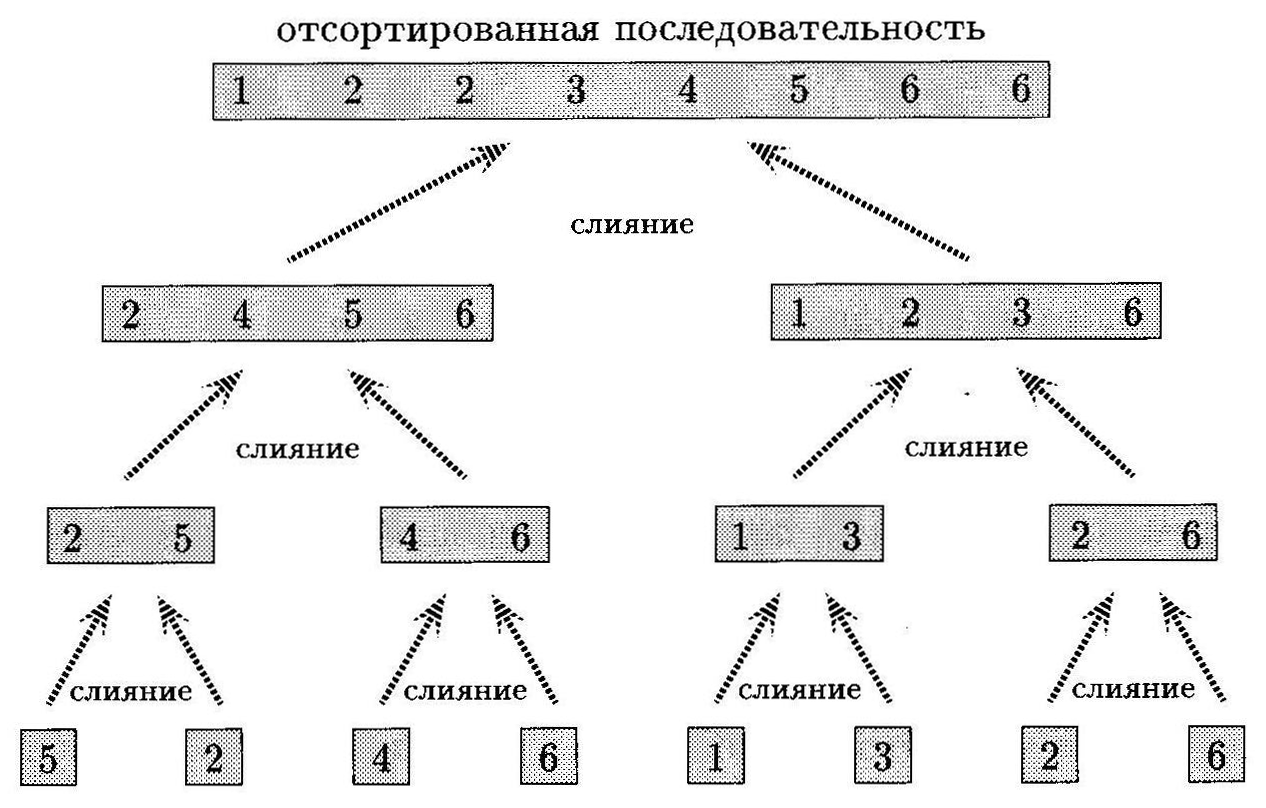
Сортировка слиянием – хороший пример использования принципа “разделяй и властвуй”. Алгоритм сортировки можно описать следующим образом: Сортируемый массив рекурсивно разбивается на две части примерно одинакового размера. Рекурсивное разбиение происходит до тех пор, пока размер массива не достигнет единицы, т. к массив длины 1 можно считать упорядоченным. Далее мы соединяем два отсортированных массива в результатирующий массив(когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив). Делаем это со всеми подмассивами до тех пор, пока не получим отсортированный результатирующий массив с длиной исходного массива.

Иллюстрация работы сортировки слиянием для лучшего понимания алгоритма:

Вычислительная сложность составляет O(n\*log2(n)

Рис. 3

5

**“Radix sort”**

Поразрядная сортировка – линейная сортировка в которой поочередно используется сортировка подсчетом и ее свойство устойчивости для каждой цифры справа налево. Порядок цифр или символов действительно важен.

Иллюстрация работы поразрядной сортировки для лучшего понимния алгоритма:

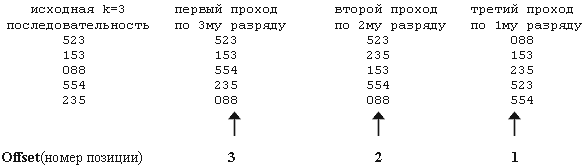


Рис. 4

Худшая сложность О(w \* n), где w – количество бит, требуемых для хранения каждого ключа.

6

**Руководство пользователя**

Во всех сортировках после директивов “include” объявлен “define”, который задает размер сортируемого массива. Каждая сортировка выводит в консоль исходный массив, отсортированный массив, маркер корректности массива в виде “Sort is correct”, количество элементов, количество сравнений, деленное на сложность, количество присваиваний, деленных на сложность.

7

**Описание программной реализации**

Функции, используемые во всех сортировках:

void cout(const float\* arr, int i) – функция вывода массива. На вход функции подается указатель массив и его длина.

void rands(float\* arr, int l) – функция заполнения массива рандомными отрицательными и положительными значениями типа float. На вход функции подается указатель и длина массива заполняемого массива.

int comp(const void\* i, const void\* j) – вспомогательная функция для встроенной в стандартную библиотеку сортировку qsort(). На вход функции подается два указателя на сравниваемые числа и возвращает их отношение (больше, или равно, или меньше).

**Insertion Sort и Shell sort**

Сама сортировка написана в main(). Объявляются два размера одной длины(1 – для написанной сортировки, 2 - для сортировки qsort) и перемненные. Далее 1 массив заполняется рандомными элементами функцией rands() и все элементы первого массива копируются во 2 массив циклом for. Далее идет код сортировки с подсчетом количества сравнений и присваиваний, сортировка 2 массива встроенной сортировкой и сравнение 1 и 2 массивов

**Merge sort**

void merge(float\* a, int l, int mid, int r) – функция слияния двух подмассивов принимает указатель на сортируемый массив, первый элемент первого подмассива, размер первого подмассива и в то же время первый элемент второго подмассива, размер второго подмассива. Результат слияния двух подмассивов записывается в вспомогательный массив, созданный “mallocом”, после чего записывается в сортируемый массив, тем самым я получал отсортированный отрезок (от l до r) в сортируемом массиве.

void mergesort(float\* array, int l, int r) – на вход функция получает указатель на сортируемый массив, левый и правй индекс сортируемый части.

**Radix sort**

void createCounters(float\* data, long\* counters, long N) – функция на вход принимает указатель на 1 элемент исходного массива, указатель на 1 элемент массива counterов размером sizeof(float)\*256, размер исходного массива. Функция подсчитывает какое количество раз значение каждого байта числа встретилось и записывает в массив counterов.

8

void radixPass(short Offset, long N, float\* source, float\* dest, long\* count) – на вход функции подается номер сортируемого разряда, размер исходного массива, указатель на 1 элемент исходного массива, указатель на результатирующий массив, указатель на подмассив массива counterов, зависящий от offset Функция сортирует массив в рамках одного разряда.

void radixSort(float\* in, float\* out, long\* counters, long N) – на вход функции подается указатель на исходный массив, указатель на результатирующий массив, указатель на массив counterов, длину исходного и результатирующего массива. Функция, используя вышеописанные функции сортирует исходный массив. После сортировки с помощью 2 циклов отрицательные чила в отсортированном массиве становятся в правильную позицию.

9

**Подтверждение корректности**

Встроенная сортировка qsort(b, n, sizeof(float), comp) и цикл for для проверки корректности написанной сортировки. На фход функции подается указатель на копию исходного массива, размер массива, размер типа данных и вышеописанная функция comp. Функция сортирует копию исходного массива. Далее циклом for происходит поэлементное сравнение массива, отсортированного проверенной сортировки, и массива, отсортированного написанной сортировкой. Если все элементы двух отсортированных массивов совпадают, то в консоль выводится “Sort is correct”.

10

**Результаты экспериментов**

Во всех сортировках использовалось рандомное задание сортируемого массива, включая отрицательные числа. Длина массива менялась от 0 до 10000 с шагом 1000. На каждом шаге выводился маркер корректности сортировки(Sort is correct), количество сравнений, деленное на сложность и количество присваиваний, деленное на сложность. По полученным данным строился график. Для подтверждения сложности сортировки необходимо по результатам экспериментов получить график константы или значения, попадающего в небольшой диапазон.

**Insertion sort**

У сортировки вставками сложность равна О(n^2). Если сортировка написана верно и сохранила свою сложность, то графики отношения количества присваиваний и количества сравнений к сложности для разных длин массива должны образовывать константу или значения, попадающие в небольшой диапазон.

На рисунке 5 изображен график отношения количества сравнений к сложности сортировки(О(n^2)). По результатам эксперимента был получен график константы, примерно равной 0.5, из чего можно сделать вывод, что у написанной сортировки правильная сложность.

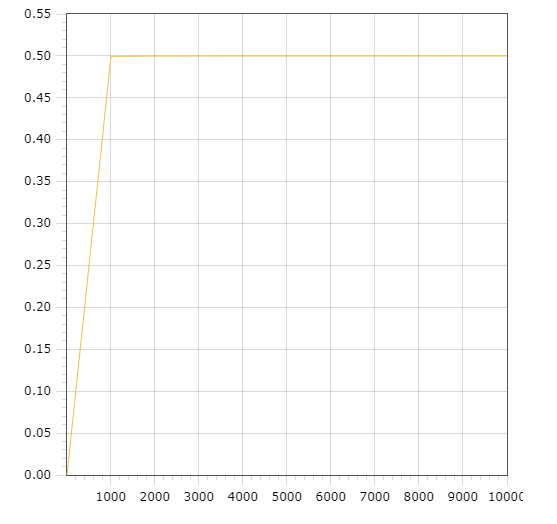


Рис. 5

11

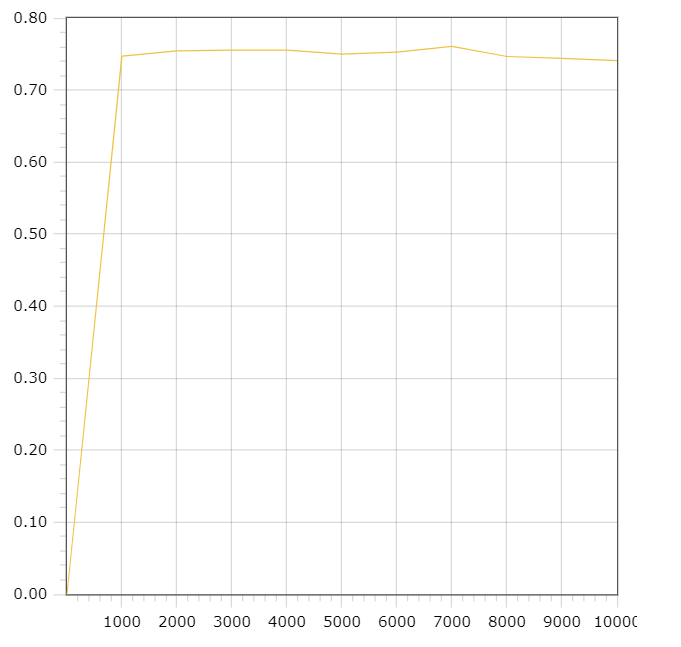
На рисунке 6 изображен график отношения количества присваиваний к сложности сортировки(О(n^2)). По результатам эксперимента был получен график значения, попадающего в диапазон (0.7 – 0.8), из чего можно сделать вывод, что у написанной сортировки правильная сложность.

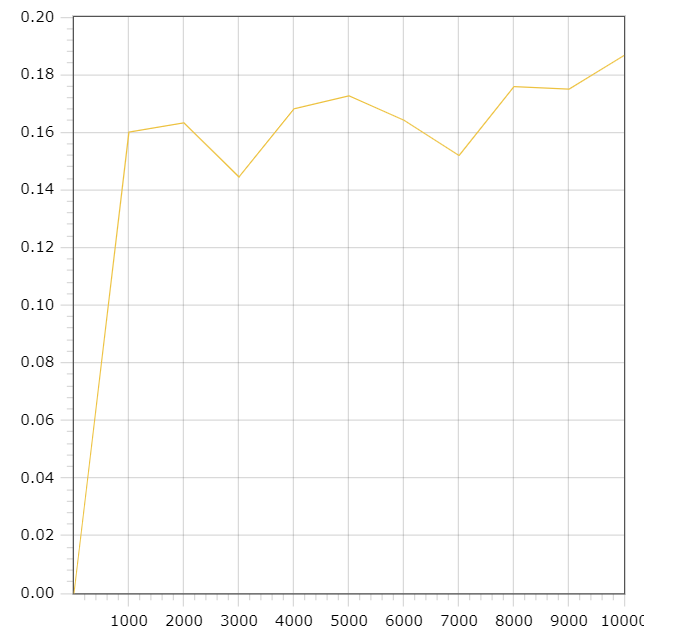
Рис. 6

**Shell sort**

У сортировки Шелла сложность равна O(n \* log^2 (n)). Если сортировка написана верно и сохранила свою сложность, то графики отношения количества присваиваний и количества сравнений к сложности для разных длин массива должны образовывать константу или значения, попадающие в небольшой диапазон.

На рисунке 7 изображен график отношения количества сравнений к сложности сортировки (О(n \* log^2 (n)). По результатам эксперимента был получен график значения, попадающего в диапазон (0.14 – 0.2), из чего можно сделать вывод, что у написанной сортировки правильная сложность.

12

**** Рис. 7

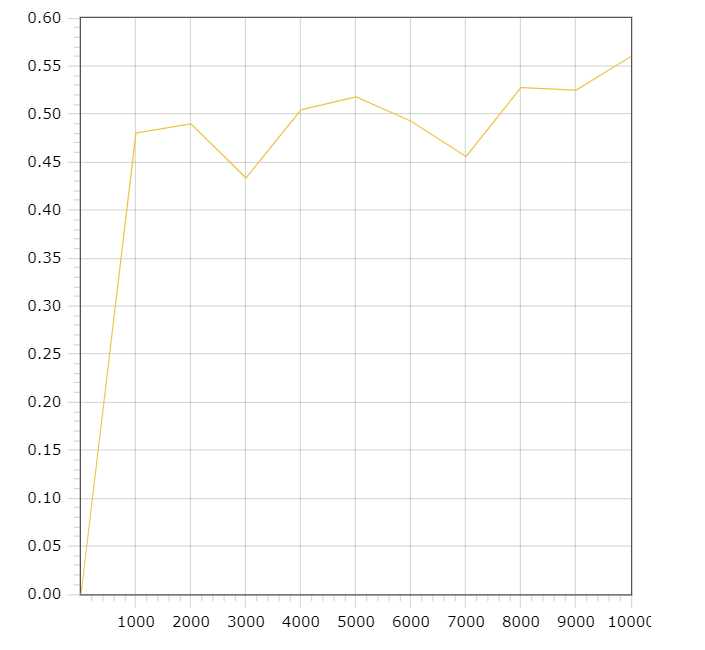
На рисунке 8 изображен график отношения количества присваиваний к сложности сортировки (О(n \* log^2 (n)).По результатам эксперимента был получен график значения, попадающего в диапазон (0.42 – 0.58), из чего можно сделать вывод, что у написанной сортировки правильная сложность.

Рис. 8

13

**Merge sort**

Сложность данной сортировки составляет O(n \* log(n)). Если сортировка написана верно и сохранила свою сложность, то графики отношения количества присваиваний и количества сравнений к сложности для разных длин массива должны образовывать константу или значения, попадающие в небольшой диапазон.

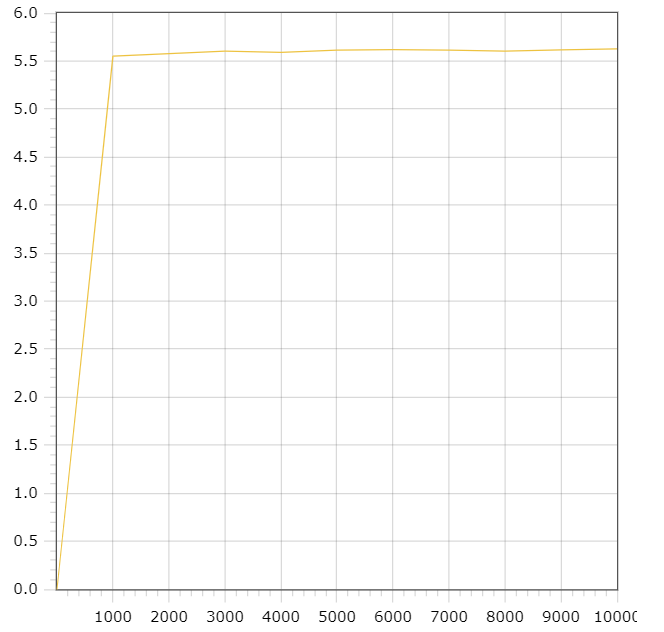
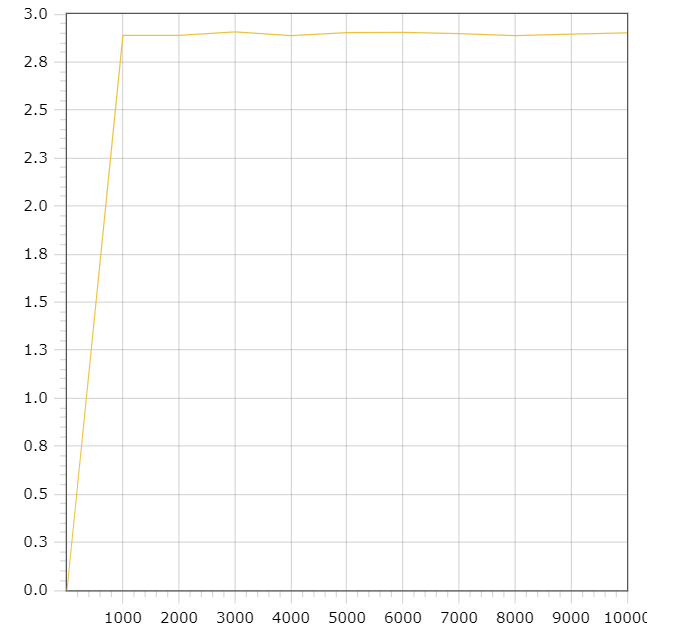
На рисунке 9 изображен график отношения количества сравнений к сложности сортировки (О(n \* log(n)). По результатам эксперимента был получен график значения, попадающего в диапазон (5.5 – 5.63), из чего можно сделать вывод, что у написанной сортировки правильная сложность.

Рис. 9

На рисунке 10 изображен график отношения количества присваиваний к сложности сортировки (О(n \* log(n)). По результатам эксперимента был получен график значения, попадающего в диапазон (2.8 – 2.95), из чего можно сделать вывод, что у написанной сортировки правильная сложность.

14

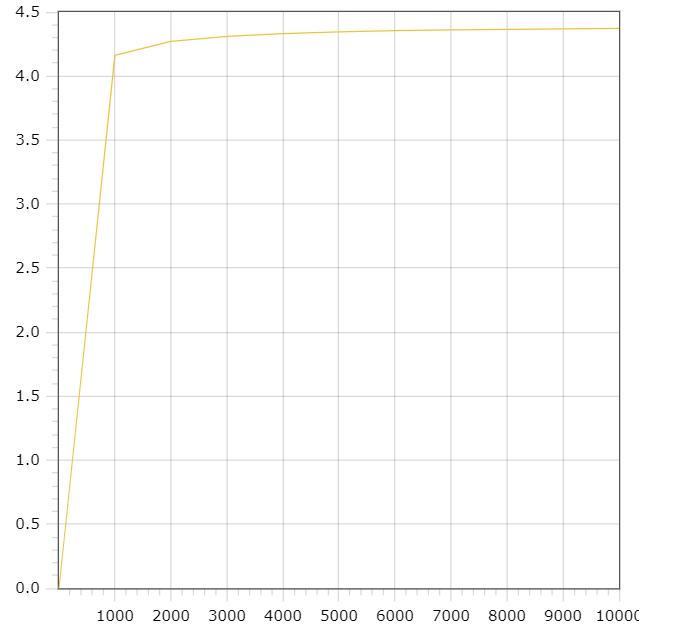
 Рис. 10

**Radix sort**

Сложность данной сортировки составляет O(4 \* (n+256) + n). Если сортировка написана верно и сохранила свою сложность, то графики отношения количества присваиваний и количества сравнений к сложности для разных длин массива должны образовывать константу или значения, попадающие в небольшой диапазон.

На рисунке 9 изображен график отношения количества присваиваний к сложности сортировки O(4 \* (n+256) + n). По результатам эксперимента был получен график значения, попадающего в диапазон (4 – 4.5), из чего можно сделать вывод, что у написанной сортировки правильная сложность

15

 Рис. 11

16

**Заключение**

В ходе лабораторной работы были реализованы сортировка вставками, сортировка Шелла, сортировка слиянием и поразрядная сортировка для типа данных float на языке программирования C. После реализации были проведены проверка корректности сортировок и подтверждение сложности написанных сортировок экспериментальным путем с приложением графиков.

17