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

Ермолаев Д.А.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

Оглавление

[Постановка задачи 3](#_Toc91294006)

[Методы решения 4](#_Toc91294007)

[Сортировка выбором 4](#_Toc91294008)

[Сортировка Шелла 4](#_Toc91294009)

[Сортировка слиянием 5](#_Toc91294010)

[Поразрядная сортировка 6](#_Toc91294011)

[Руководство пользователя 8](#_Toc91294012)

[Описание программной реализации 9](#_Toc91294013)

[Сортировка выбором 9](#_Toc91294014)

[Сортировка Шелла 9](#_Toc91294015)

[Сортировка слиянием 10](#_Toc91294016)

[Поразрядная сортировка 10](#_Toc91294017)

[Подтверждение корректности 12](#_Toc91294018)

[Результаты экспериментов 13](#_Toc91294019)

[Сортировка выбором 13](#_Toc91294020)

[Сортировка Шелла 15](#_Toc91294021)

[Сортировка слиянием 17](#_Toc91294022)

[Поразрядная сортировка 19](#_Toc91294023)

[Заключение 21](#_Toc91294024)

[Приложение 22](#_Toc91294025)

# Постановка задачи

Целью лабораторной работы являлась реализовать на языке программирования Си сортировку выбором, сортировку Шелла, сортировку слиянием и поразрядную сортировки. Сортировки нужно реализовать для данных типа double. Нужно описать программную реализацию и алгоритмы работы данных сортировок. Необходимо подтвердить корректность реализации данных сортировок. Провести эксперименты для подтверждения сложности, описать способ проведения экспериментов и сделать вывод по полученным результатам.

# Методы решения

## Сортировка выбором

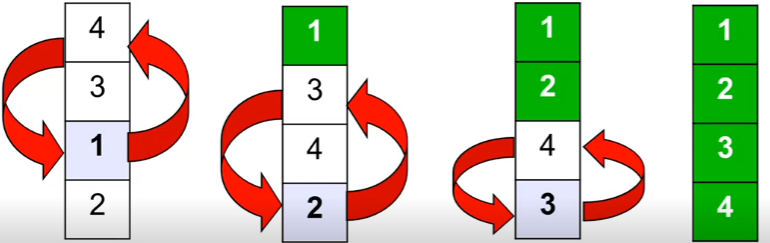
Идея сортировки выбором - проходим по массиву в поисках минимального элемента. Найденный минимум меняем местами с первым элементом. Неотсортированная часть массива уменьшилась на один элемент (не включает последний элемент, куда мы переставили найденный минимум). К этой неотсортированной части применяем те же действия — находим минимум и ставим его на первое место в неотсортированной части массива. И так продолжаем до тех пор, пока неотсортированная часть массива не уменьшится до одного элемента. Сложность данного алгоритма сортировки равна O(n2).

Рисунок 1 - иллюстрация работы алгоритма

## Сортировка Шелла

При сортировке Шелла сначала сравниваются и сортируются между собой значения, отстоящие один от другого на некотором расстоянии d. После этого процедура повторяется для некоторых меньших значений d, а завершается сортировка Шелла упорядочиванием элементов при d=1 (то есть обычной сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше). В худшем случае сложность алгоритма составляет O(n2) в худшем случае и O(n\*log22(n)) в среднем случае.

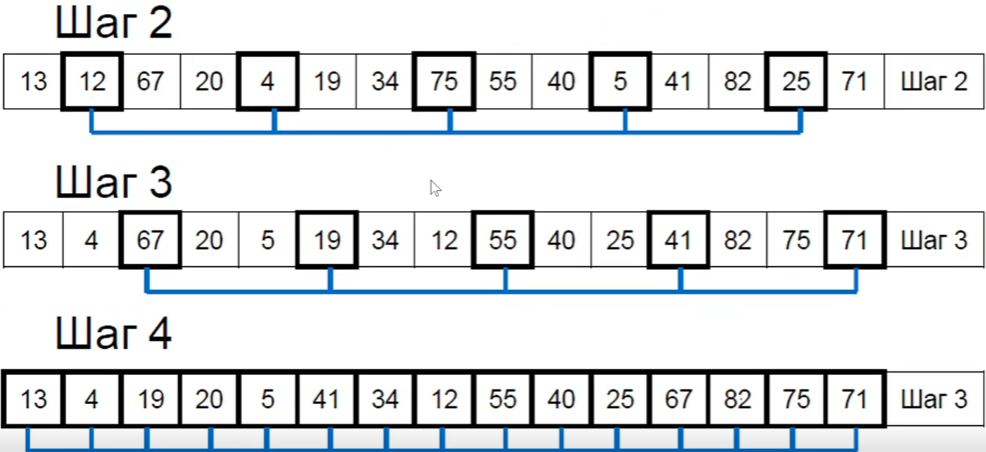
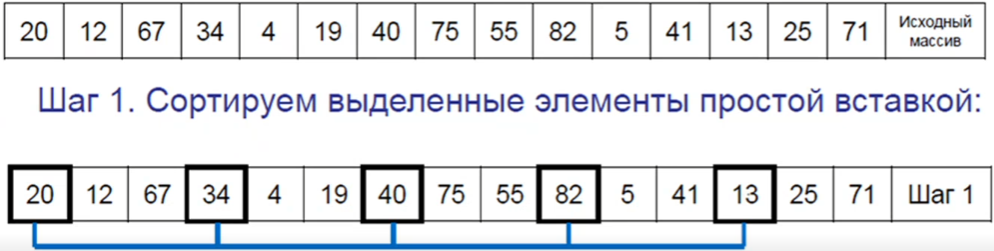


Рисунок 2 - иллюстрация работы сортировки Шелла

Далее полученный массив сортируется простой вставкой

## Сортировка слиянием

Для решения задачи сортировки эти три этапа выглядят так:

1. Сортируемый [массив](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D1%82%D0%B8%D0%BF_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) 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.

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.

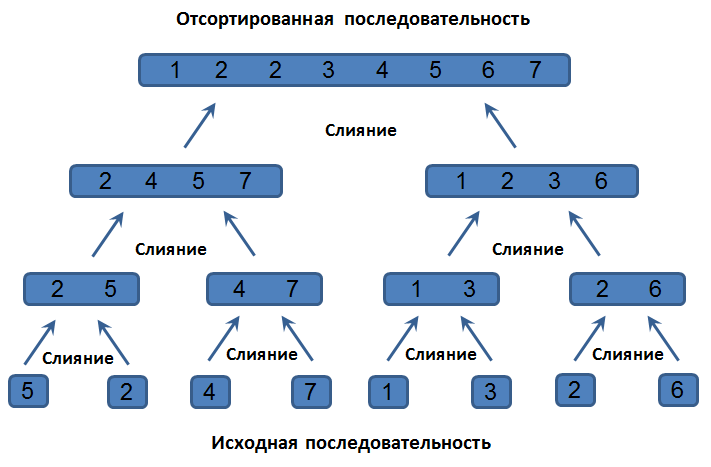
* Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.  
  Сложность алгоритма составляет O(n\*log2(n)) в любом случае, а также необходимо О(n) памяти.

Рисунок 3 - иллюстрация работы сортировки слиянием

## Поразрядная сортировка

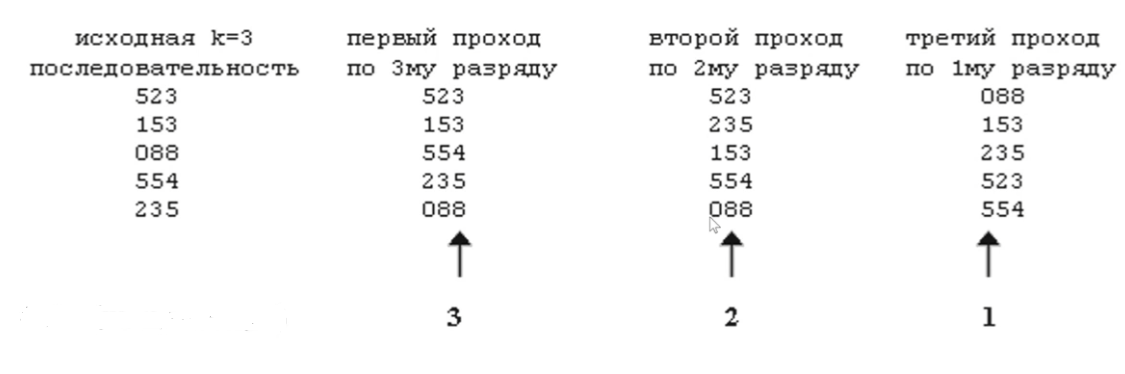
Сравнение производится поразрядно: сначала сравниваются значения одного крайнего разряда, и элементы группируются по результатам этого сравнения, затем сравниваются значения следующего разряда, соседнего, и элементы либо упорядочиваются по результатам сравнения значений этого разряда внутри образованных на предыдущем проходе групп, либо переупорядочиваются в целом, но сохраняя относительный порядок, достигнутый при предыдущей сортировке. Затем аналогично делается для следующего разряда, и так до конца.

Рисунок 4 - иллюстрация работы сортировки

# Руководство пользователя

Все сортировки выводят в консоль неотсортированный массив, отсортированный массив, количество присвоений и количество сравнений. Вывод осуществляется в следующем формате:

* + - 1. Выводится неотсортированный массив
      2. Выводится отсортированный массив
      3. Выводятся количество присвоений и количество сравнений

# Описание программной реализации

Функции, содержащиеся во всех сортировках:

double DRand(double fMin, double fMax) – функция принимает на вход минимальный и максимальный элементы. Возвращает псевдослучайное число типа double в указанном диапазоне.

void randarr(double arr[], int n) – функция принимает на вход указатель на массив и длину данного массива. Заполняет массив указанной длины, числами типа double в диапазоне от -1000 до 1000.

void printarr (double arr[], int n) – функция принимает на вход указатель на массив и длину данного массива. Выводит массив, соответствующий длины, в консоль.

int cmp(const void\* a, const void\* b) – функция принимает на вход два указателя на сравниваемые числа любого типа данных и возвращает их взаимное отношение, больше/равно/меньше. Эта функция является вспомогательной для qsort() из стандартной библиотеки Си.

## Сортировка выбором

void selection\_sort(double arr[], int n, int\* assignments, int\* comparisons) – на вход функция принимает указатель на сортируемый массив, длину сортируемого массива, указатель на переменную, в которую будет записано количество присвоений, указатель на переменную, в которую будет записано количество сравнений.

## Сортировка Шелла

void shell\_sort(double arr[], int n, int\* assignments, int\* comparisons) – на вход функция принимает указатель на сортируемый массив, длину сортируемого массива, указатель на переменную, в которую будет записано количество присвоений, указатель на переменную, в которую будет записано количество сравнений.

## Сортировка слиянием

void merge\_sort(double arr[], int l, int r, int\* assignments, int\* comparisons) – на вход функция принимает указатель на сортируемый массив, левый индекс сортируемой части, правый индекс сортируемой части, указатель на переменную, в которую будет записано количество присвоений, указатель на переменную, в которую будет записано количество сравнений.

void merge(double arr[], int l, int r, int\* assignments, int\* comparisons) - на вход функция принимает указатель на первый отсортированный массив, левый индекс, правый индекс, указатель на переменную, в которую будет записано количество присвоений, указатель на переменную, в которую будет записано количество сравнений.

## Поразрядная сортировка

void createCounters(double data[], long\* counters, long n, int\* assignments) – на вход принимает указатель на первый элемент массива данных, указатель на первый элемент массива для счётчиков, длину сортируемого массива, указатель на переменную, в которую будет записано количество сравнений. Функция подсчитывает сколько раз какое значение каждого байта числа встретилось, записывая все в массив размера 256\*sizeof(double)\*sizeof(long). Функция возвращает указатель на полученный массив.

void radixPass(short Offset, long n, double\* source, double\* dest, long\* count, int\* assignments) – функция принимает позицию, начиная с которой нужно вставлять число в выходной массив, длину сортируемого массива, указатель на первый элемент исходного массива, указатель на первый элемент выходного массива, указатель на начало позиции сортируемого разряда из массива, созданного функцией createCounters(), и указатель на переменную, в которую будет записано количество присвоений. Функция выполняет последовательную сортировку массива по одному разряду.

void signedRadixLastPass(short Offset, long n, double\* source, double\* dest, long\* count, int\* assignments) - функция принимает позицию, начиная с которой нужно вставлять число в выходной массив, длину сортируемого массива, указатель на первый элемент исходного массива, указатель на первый элемент выходного массива, указатель на начало позиции сортируемого разряда из массива, созданного функцией createCounters(), и указатель на переменную, в которую будет записано количество присвоений. Данная функция переставляет отрицательные числа в правильном порядке.

void signedRadixSort(double\* in, double\* out, long\* counters, long n, int\* assignments) – функция принимает на вход указатель на первый элемент входного массива, указатель на первый элемент выходного массива, указатель на первый элемент массива для счётчиков, длину сортируемого массива и указатель на переменную, в которую будет записано количество присвоений. Функция, при помощи вышеописанных функций, сортирует входной массив.

# Подтверждение корректности

Для проверки корректности реализации всех сортировок была использована функция qsort() из библиотеки stdlib.

В начале происходит сортировка оригинального массива при помощи стандартной сортировки любой из вышеописанных сортировок. Далее тот же массив сортируется функцией qsort(). Поскольку данная сортировка входит в стандартную библиотеку языка Си, то можно быть уверенным в корректности ее работы. После выполнения qsort() и вывода двух отсортированных массивов в консоль начинаем поэлементное сравнение массивов и определяем правдивость нашей сортировки.

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

# Результаты экспериментов

Во всех экспериментах была использована генерации случайных чисел. Длина массива изменялась с 500 элементов до 65000 с увеличением на 500 элементов каждый шаг для сортировки выбором, с 500 элементов до 500000 с увеличением на 500 элементов каждый шаг для сортировки Шелла и сортировки слиянием, с 500 элементов до 1000000 с увеличением на 500 элементов каждый шаг для поразрядной сортировки. Для каждого шага длины массива выводились количество присвоений и количество сравнений. По полученным данным строился график.

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.

## Сортировка выбором

Сложностью данной сортировки является O(n2).

На рисунке 5 график изменения отношения количества присвоений к сложности при росте количества элементов в массиве. Видно, что значения сходятся к ~0,0001. С ростом количества элементов значения становится все ближе к некоторому числу, это является экспериментальным подтверждением указанной сложности по присвоениям для данной сортировки.

На рисунке 6 изображен график изменения отношения количества сравнений к сложности при росте количества элементов в массиве. Как можно заметить, график сходится к значению приблизительно равному 0,5. Поскольку график сходится к некоторому значению, то это является экспериментальным подтверждением указанной сложности по сравнениям для данной сортировки.

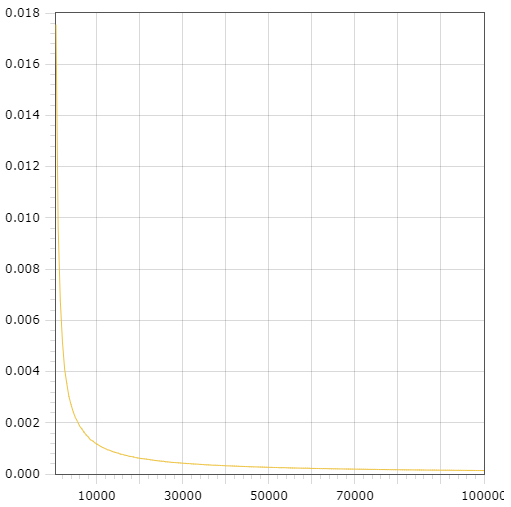


Рисунок 5 – присвоения / сложность

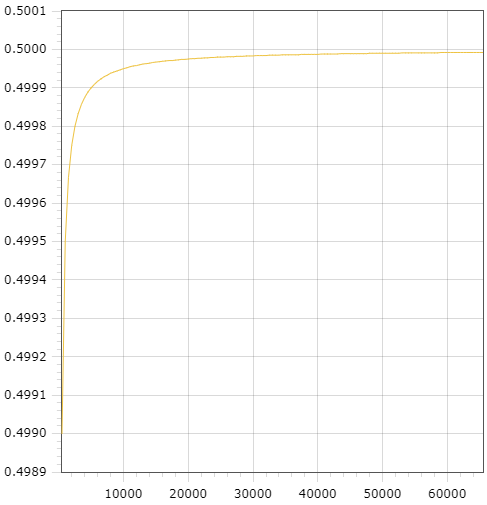


Рисунок 6 – сравнения / сложность

## Сортировка Шелла

Средняя ложность данной сортировки равна O(n\*log2n).

На рисунке 7 изображен график изменения отношения количества присвоений к сложности при росте количества элементов в массиве. Сортировка является чувствительной к данным из-за чего графики очень неровные. Как можно заметить, по большей части график сходится к некоторому значению. Поскольку диапазон значений с ростом количества элементов уменьшается для большинства точек, то это является экспериментальным подтверждением указанной сложности по присвоениям для данной сортировки.

На рисунке 8 график изменения отношения количества сравнений к сложности при росте количества элементов в массиве. Видно, что по большей части значения сходятся. Поскольку диапазон значений с ростом количества элементов уменьшается для большинства точек, то это является экспериментальным подтверждением указанной сложности по сравнениям для данной сортировки.

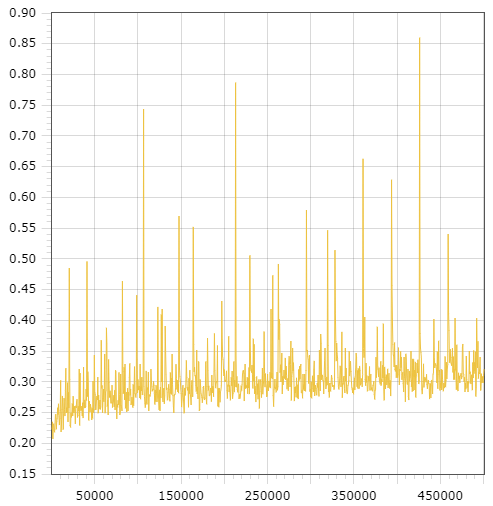


Рисунок 7 – присвоения / сложность

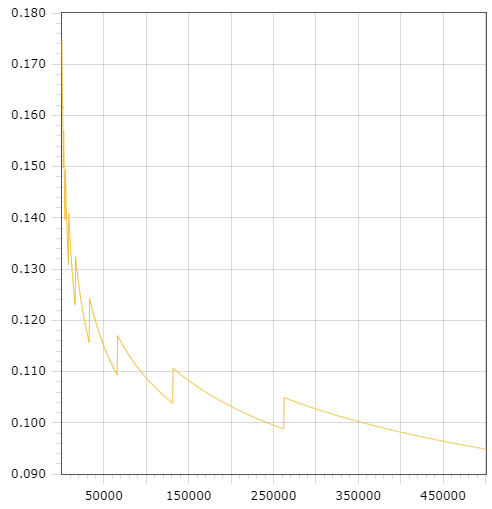


Рисунок 8 – сравнения / сложность

## Сортировка слиянием

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

На рисунке 9 изображен график изменения отношения количества присвоений к сложности при росте количества элементов в массиве. Как можно заметить, график сходится к некоторой константе. Поскольку график сходится к некоторому значению, то это является экспериментальным подтверждением указанной сложности по присвоениям для данной сортировки.

На рисунке 10 изображен график изменения отношения количества сравнений к сложности при росте количества элементов в массиве. Как можно заметить, график сходится к некоторой константе. Поскольку график сходится к некоторому значению, то это является экспериментальным подтверждением указанной сложности по сравнениям для данной сортировки.

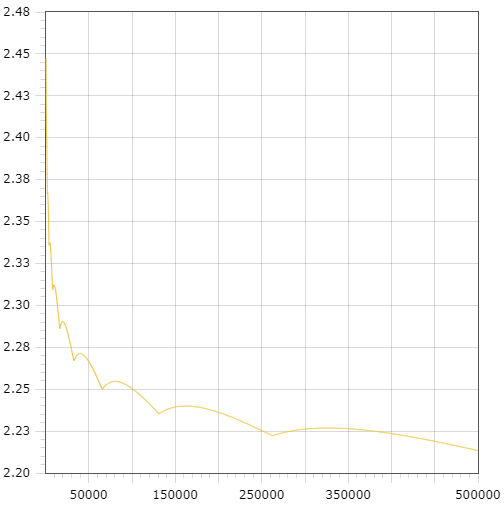


Рисунок 9 – присвоения / сложность

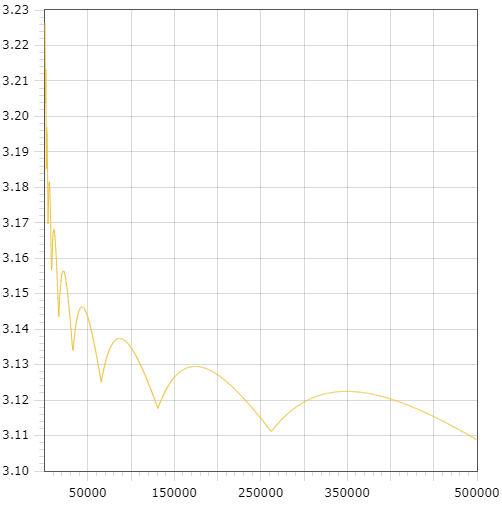


Рисунок 10 – сравнения / сложность

## Поразрядная сортировка

Сложностью данной сортировки является O(8\*(n+256)) для типа double.

На рисунке 11 изображен график изменения отношения количества присвоений к сложности при росте количества элементов в массиве. Как можно заметить, график сходится к значению приблизительно равному 4,8. Поскольку график сходится к некоторому значению, то это является экспериментальным подтверждением указанной сложности по присвоениям для данной сортировки.

На рисунке 12 изображен график изменения отношения количества сравнений к сложности при росте количества элементов в массиве. Как можно заметить, график сходится к значению приблизительно равному 2,5. Поскольку график сходится к некоторому значению, то это является экспериментальным подтверждением указанной сложности по сравнениям для данной сортировки.

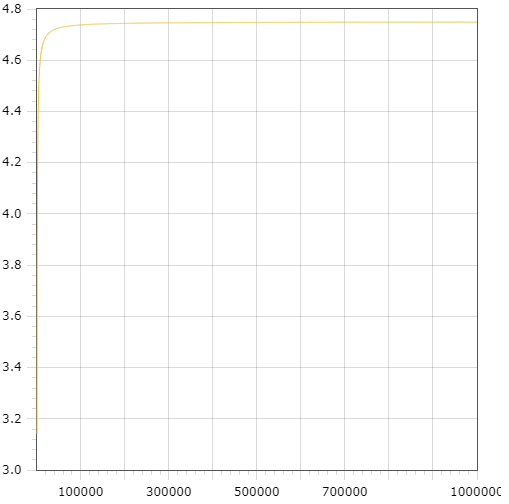


Рисунок 11 – присвоения / сложность

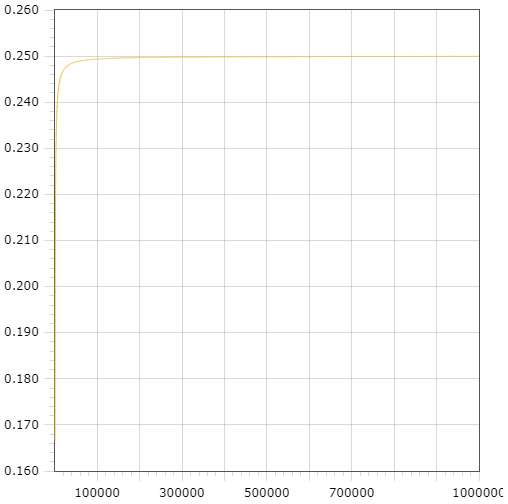


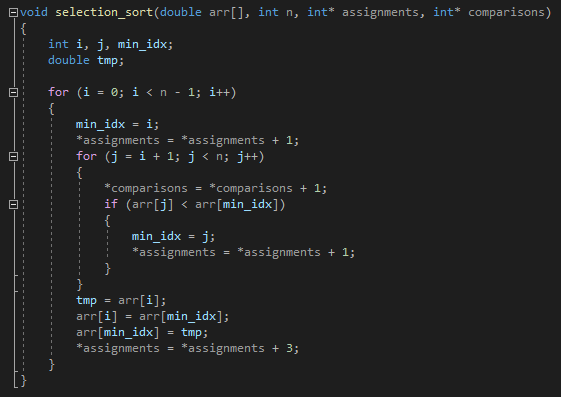
Рисунок 12 – сравнения / сложность

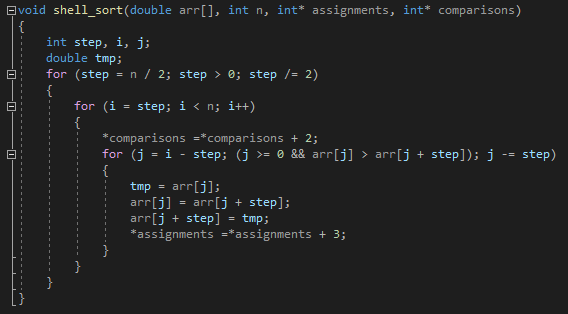
# Заключение

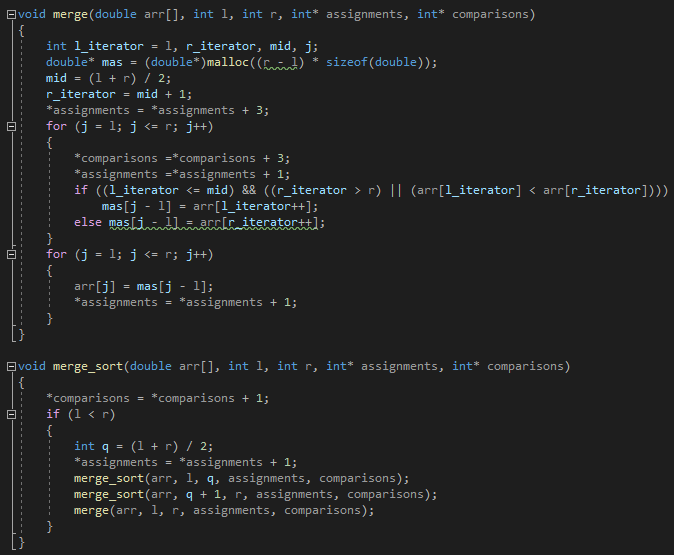
В ходе лабораторной работы на языке программирования Си были реализованы сортировка выбором, сортировка Шелла, сортировка слиянием и поразрядная сортировка. Были описаны алгоритмы работы данных сортировок, их программная реализация и проведенные эксперименты для замера и подтверждения их теоретический сложности. В ходе проведения экспериментов была проведена проверка корректности сортировок на большом объеме данных и подтверждена их теоретическая сложность.

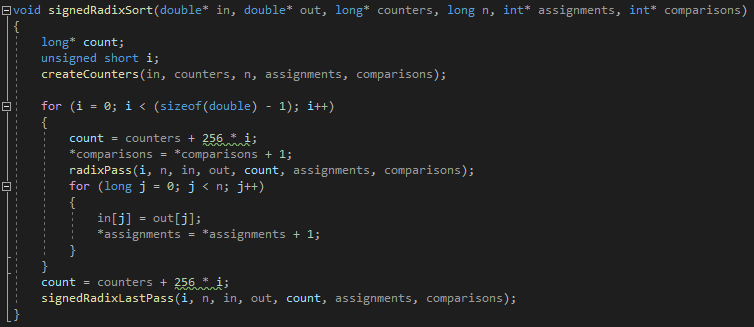
# Приложение

**Сортировка выбором**



**Сортировка Шелла**

**Сортировка слиянием**

**Поразрядная сортировка**