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/ка группы 3821Б1ПМ2

Соколов И. Д.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

# Постановка задачи

Цель лабораторной работы:

1) Реализовать 4 сортировки для данных типа float на языке СИ

2) Описать программную реализацию

3) Подтвердить корректность данной программы

4) Провести эксперименты подтверждающие теоретическую сложность сортировок

5) Сделать вывод

# Метод решения

**Сортировка пузырьком**

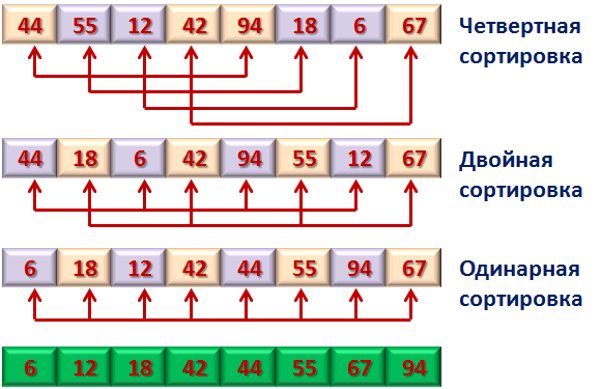
Данную сортировку можно представить в виде столбика, где первый элемент наверху, а последний внизу. Каждый раз проходя через столбик программа сравнивает два соседних элемента, и если нижний меньше верхнего, то он как бы «всплывает», а другой элемент «тонет» как пузырьки, отсюда и название. Обычно данной сортировке нужно пройти массив n раз, но моя улучшенная сортировка будет проходить массив до тех пор пока количество перестановок не будет равно 0. Таким образом теоретическая сложность данной сортировки в лучшем случае O(n), в худшем O(n^2), и в среднем ~O(n^2).



**Сортировка Шелла**

*Иллюстрация сортировки пузырьком*

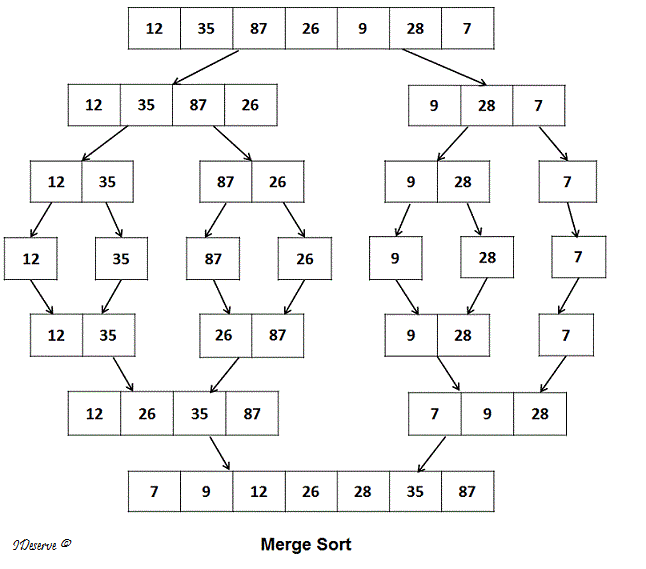
Данная сортировка напоминает сортировку вставками, мы сравниваем элементы на некотором расстоянии ***step*** и меняем местами если правый меньше левого. Каждый раз после прохода по массиву ***step*** делится нацело на 2, до тех пор пока он не будет равен 1. Когда ***step*** становится один программа просто пройдет по всему массиву как пузырьковая сортировка, возможно переставляя последние(неотсортированные) элементы. Моя сортировка использует вместо ***step, step/2, step/4, … , 1*** последовательность Марцина Циура = ***{ 1, 4, 10, 23, 57, 132, 301, 701, 1750 }***, благодаря которой данная сортировка решается за оптимальное время при массивах до *4000* элементов. Таким образом теоретическая сложность данной сортировки в худшем случае O(n^2) и O(n\*log n) в лучшем случае.



*Иллюстрация сортировки Шелла*

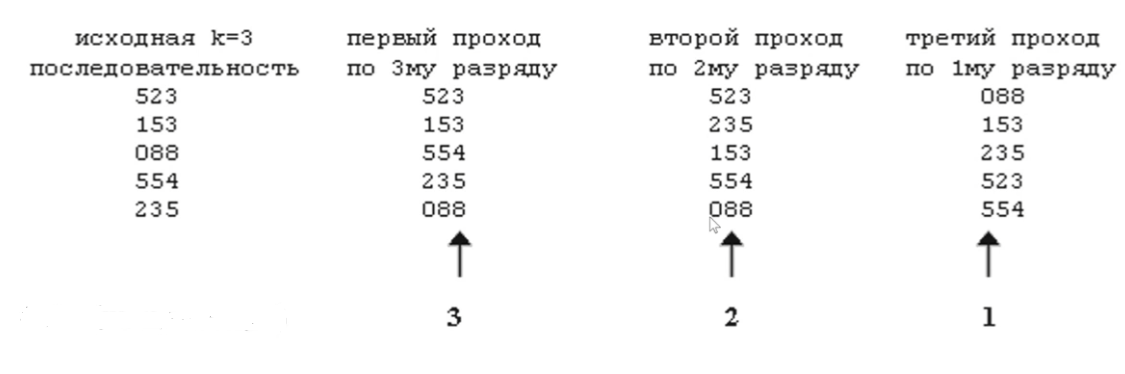
**Сортировка слиянием**

Это сортировка которая продолжает делить массив на под-массивы до тех пор, пока под-массивы не окажутся одним элементом. Потом она объединяет два под-массива в один и одновременно сортируя их образуя один отсортированный под-массив. Таким образом теоретическая сложность в любом случае O(n\*log n).



*Иллюстрация сортировки слиянием*

**Поразрядная сортировка**

Как понятно из названия сортировка сортирует числа по разрядам. Есть два способа считать разряды слева направо и справа налево. Возьмем второй так как он легче. Программа начинает сортировать массив по последнему разряду(последней цифре) числа, потом пред последнему, потом пред-пред последнему и т.д. до первого разряда. Таким образом теоретическая сложность в любом случае O(n\*k), где k – это максимальное количество разрядов.

*Иллюстрация поразрядной сортировки*

# Руководство пользователя

При запуске компилятора в консоли отобразится такая меню:

change the sort method

1) radix

2) merge

3) bubble

4) shell

just input the number of sort method

Программа будет ждать пока вы не введете цифру соответствующую сортировке которую вы хотите проверить, т.е. если вы хотите например проверить поразрядную сортировку введите «1».

После того как вы введете цифру появится следующее:

just input the number of sort method 1

input size of massiv (remember that its scanf (<1364))

Программа будет ждать пока вы не введете число соответствующее длине массива который вы хотите отсортировать, а так же предупреждает что числа придется вводить самому и то что ограничение длинны массива = 1364.

После ввода длины массива программа будет ждать пока вы не напечатаете все элементы массива через Enter.

После ввода длины массива и ввода всех элементов появится следующее:

just input size of massiv (remember that its scanf (<1364)) 5

2

5

3

4

2

2.000000 2.000000 3.000000 4.000000 5.000000

sravnen = 2133 perestanovok = 0

Как видно программа выдала сначала отсортированный массив, а на следующей строке количество сравнений и перестановок.

# Описание программной реализации

**int Main( )**

В начале инициализирую необходимые переменные и создаю 3 массива с максимальной длинной 1364. Потом печатаю первое меню и принимаю на вход переменную для выбора сортировки. Выбрав какую то сортировку печатаю вторую меню и принимаю на вход длину массива. Потом принимаю на вход и записываю элементы массива в пустой массив. Потом вывожу на экран отсортированный массив с количеством сравнений и перестановок.

**Сортировка пузырьком**

void bubsort(float massiv[ ], int n) – на вход функция принимает массив для сортировки и его длинну. Данная функция сортирует массив выталкивая легкие пузырьки на поверхность.

**Сортировка Шелла**

void shellsort(float massiv[ ], int N) – на вход функция принимает массив для сортировки и его длину. Данная функция используя последовательность Марцина Циура отсортировывает массив как улучшенная сортировка вставками.

**Сортировка слиянием**

void mergesort(float array[ ], float second[], int l, int r) – на вход функция принимает массив для сортировки, пустой массив, левый крайний индекс элемента и правый крайний индекс элемента. Данная функция при помощи нижней делит массив на под-массивы а потом собирает отсортированные под-массивы в отсортированный массив.

void merge(float array[ ], float second[], int l,int q, int r) – на вход функция принимает массив для сортировки, пустой массив, левый крайний индекс элемента, индекс элемента находящегося посередине и правый крайний индекс элемента.

**Поразрядная сортировка**

int\* createCounters(float\* data,int N) – на вход принимает указатель на первый элемент массива данных и длину этого массива. Функция подсчитывает сколько раз какое значение каждого байта числа встретилось, записывая все в массив размера 256\*sizeof(float)\*sizeof(int). Функция возвращает указатель на полученный массив.

void radixPass(short Offset, int N, float\* source, float\* dest, int\* count) – 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 позиций сортируемого разряд из массива, созданного функцией createCounters() . Функция выполняет поразрядную сортировку .

void radixSort(float\* in, float\* out, int N) – функция принимает на вход указатель на первый элемент входного массива , указатель на первый элемент выходного массива , их длину. Данная функция при помощи функция указанных выше сортирует массив, но в начале массива ставит положительные по возрастанию, в конце отрицательные по убыванию.

void signedRadixSort(float\* in, float\* out, int N, float\* res) – функция принимает на вход указатель на первый элемент входного массива , указатель на первый элемент выходного массива, необходимого для функции указанной выше, их длину и указатель на массив в который запишем результат. Данная функция с помощью функция radixSort отсортировывает массив как надо.

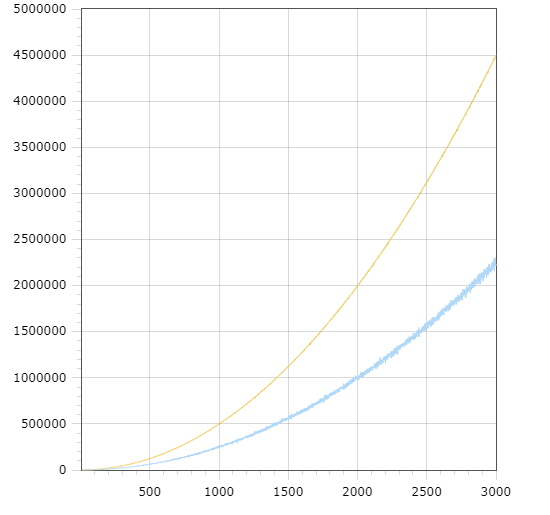
# Подтверждение корректности

Для подтверждения корректности в программе я сравнил 3 сортировки с сортировкой пузырьком потому что сортировка пузырьком достаточно легкая в реализации и в ней я уверен на все 100%.

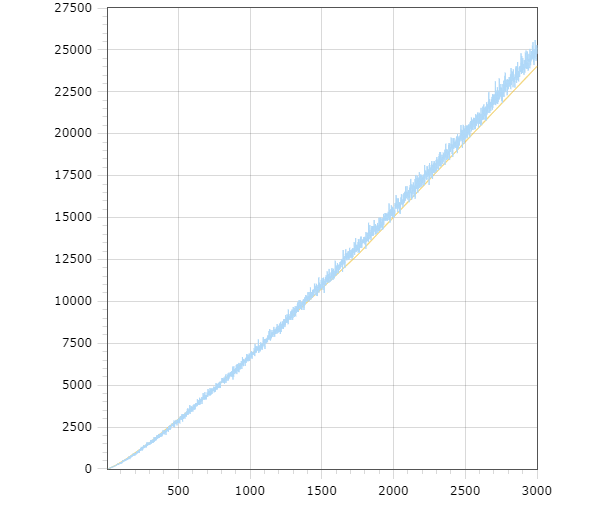
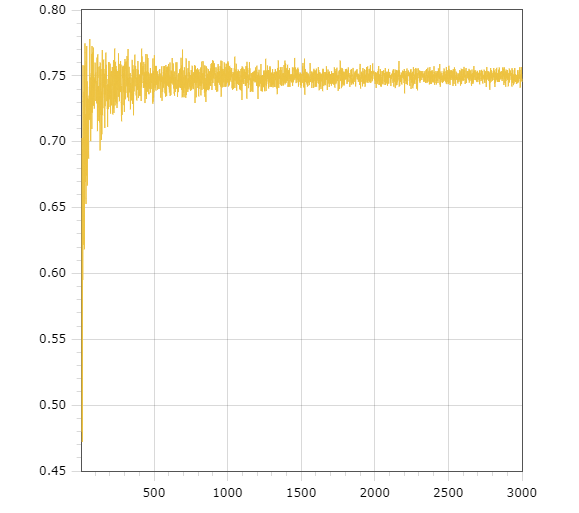
# Результаты экспериментов

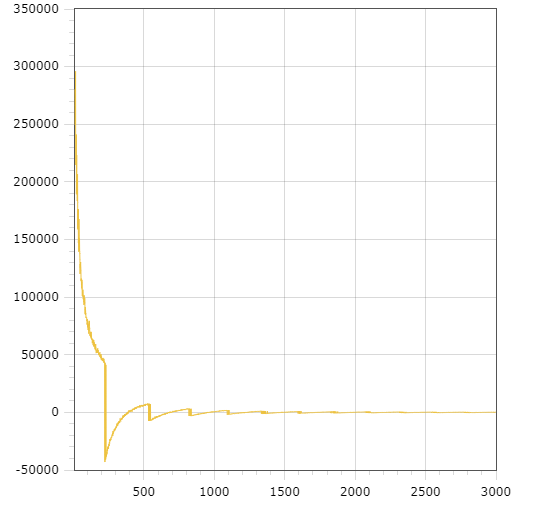
Для проверки сложности я использовал генератор случайного массива. Длинна менялась с 10 элементов до 3000 с увеличением на 1 элемент каждый шаг. После сортировки массива выводил количество перестановок, сравнений и длину массива чтобы построить графики.

**Сортировка пузырьком**



Первый график показывает кол-во сравнений и перестановок в зависимости от длины массива, а второй кол-во сравнений и перестановок деленное на теоретическую сложность n^2. По данным второго графика видно что теоретическая сложность верна с точностью до константы.

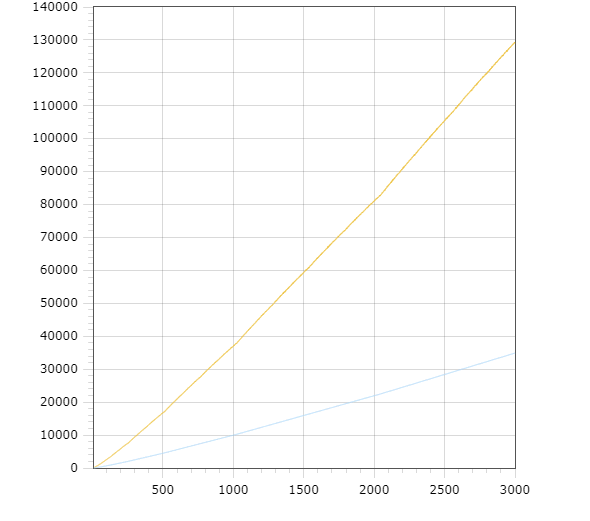
**Сортировка Шелла**

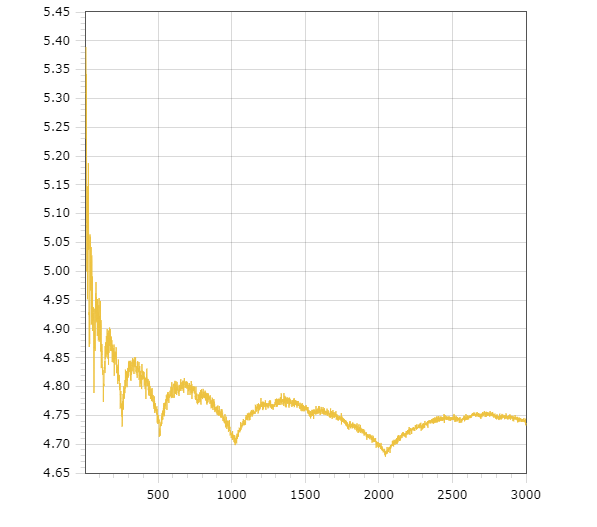


Первый график показывает кол-во сравнений и перестановок в зависимости от длины массива, а второй кол-во сравнений и перестановок деленное на теоретическую сложность n^2 как худший случай. По данным второго графика видно что с некоторого момента теоретическая сложность верна с точностью до константы.

**Сортировка слиянием**

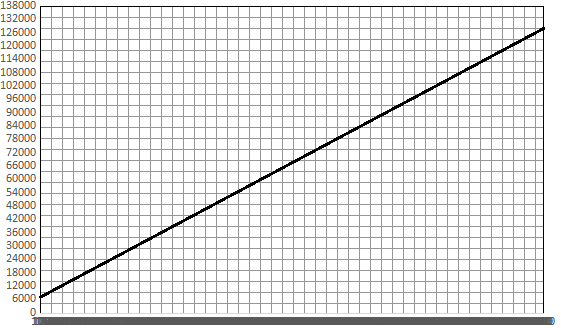
Первый график показывает кол-во сравнений и перестановок в зависимости от длины массива, а второй кол-во сравнений и перестановок деленное на теоретическую сложность n\*log n. По данным второго графика видно что с некоторого момента теоретическая сложность верна с точностью до константы.





**Поразрядная сортировка**

Первый график показывает кол-во сравнений и перестановок в зависимости от длины массива, а второй кол-во сравнений и перестановок деленное на теоретическую сложность k\*(n+m)+n , где k - длина 1 сортируемого числа в байтах, m – возможные значения этого числа. По данным второго графика видно что с некоторого момента теоретическая сложность верна с точностью до константы.



# Заключение

В результате проведенной лабораторной работы были реализованы 4 сортировки на языке СИ для данных типа float. Была описана программная реализация. Была подтверждена корректность сортировок. И были проведены эксперименты подтверждающие теоретическую сложность сортировок.