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

Головин Р.М.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**Оглавление**

[Постановка задачи 3](#_Toc89867396)

[Методы решения 4](#_Toc89867397)

[Сортировка пузырьком 4](#_Toc89867398)

[Сортировка Шелла 4](#_Toc89867399)

[Сортировка слиянием 5](#_Toc89867400)

[Поразрядная сортировка 6](#_Toc89867401)

[Руководство пользователя 7](#_Toc89867402)

[Описание программной реализации 8](#_Toc89867403)

[Сортировка пузырьком 8](#_Toc89867404)

[Сортировка Шелла 8](#_Toc89867405)

[Сортировка слиянием 9](#_Toc89867406)

[Поразрядная сортировка 9](#_Toc89867407)

[Подтверждение корректности 10](#_Toc89867408)

[Результаты экспериментов 11](#_Toc89867409)

[Сортировка пузырьком 11](#_Toc89867410)

[Сортировка Шелла 12](#_Toc89867411)

[Сортировка слиянием 13](#_Toc89867412)

[Поразрядная сортировка 14](#_Toc89867413)

[Заключение 16](#_Toc89867414)

[Приложения 17](#_Toc89867415)

# Постановка задачи

Целью лабораторной работы являлась реализовать на языке программирования Си сортировку пузырьком, сортировку Шелла , сортировку слиянием и поразрядную сортировки. Сортировки нужно реализовать для данных типа float. 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 пузырьком

Алгоритм является бессмертной классикой жанра, он очень прост в реализации и понимании. Его суть заключается в сравнении двух соседних элементов и их обмене , если они расположены в неправильном порядке. Большие элементы как бы “всплывают” к верху, а маленькие “тонут”. Алгоритму необходимо выполнить n проходов по массиву. В немного улучшенной версии сравнения осуществляются до тех пор ,пока была выполнена хотя бы одна перестановка. Сложность данного алгоритма сортировки равна O(n2).



Рисунок 1 - иллюстрация работы алгоритма

## Сортировка Шелла

Один из самых загадочных алгоритмов. Сортировка Шелла это практически сортировка вставками, но перед сортировкой с помощью вставок, мы проводим сравнение элементов, которые находятся на расстоянии d. После этого сравниваются элементы которые находятся на расстоянии d/2(делим нацело), и так, пока d не станет равным нулю ,после чего получим отсортированный массив. В худшем случае сложность алгоритма составляет O(n2) в худшем случае и O(n\*log2(n)) в лучшем случае.



Рисунок 2 - иллюстрация работы сортировки Шелла

## Сортировка слиянием

Сортировка вставками — это простой алгоритм сортировки. Суть его заключается в том что, мы разбиваем массив на два под массива и сортируем каждый отдельно, в какой то момент получаем массив состоящий из одного элемента , он считается уже отсортированным. После получения массива из одного элемента начинаем их соединять. Сложность алгоритма составляет O(n\*log2(n)) в любом случае, а так же необходимо О(n) памяти.

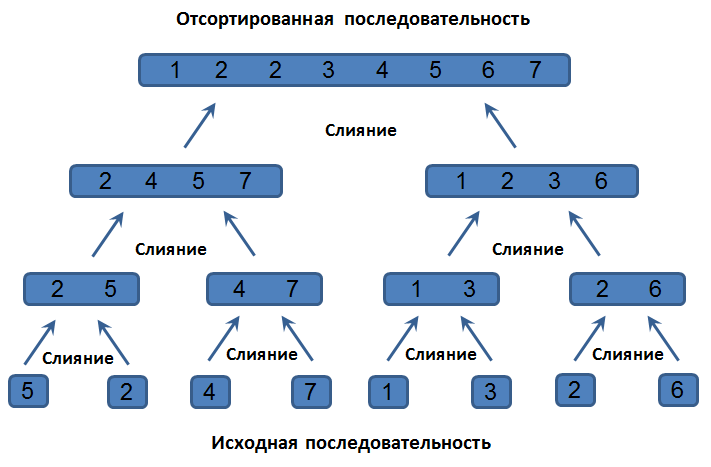


Рисунок 3 - иллюстрация работы сортировки слиянием

## Поразрядная сортировка

Является расширенной версией сортировки подсчетом. Для каждой цифры числа с права на лево поочередно используем поразрядную сортировку. Поскольку сортировка подсчетом является устойчивой ,то цифры располагаются в правильном порядке.

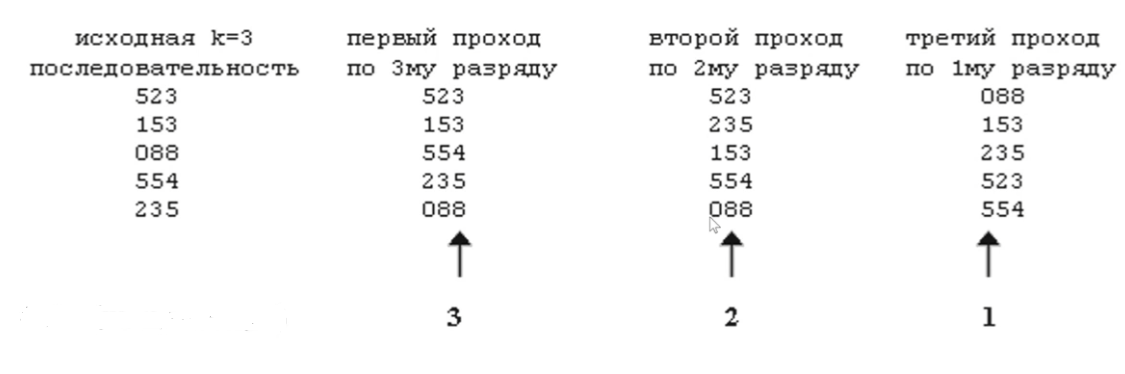


Рисунок 4 - иллюстрация работы сортировки

# Руководство пользователя

Все сортировки выводят в консоль длину сортируемого массива, сумму количества перестановок и сравнений, а так же сумму количества перестановок и сравнений деленную на теоретическую сложность алгоритма. Вывод осуществляется в следующем формате “размер массива; сумма перестановок и сравнений; сумма перестановок и сравнений / теоретическая сложность”. Для каждого массива вывод начинается с новой строки.

# Описание программной реализации

(Эти функции содержаться во всех сортировках поэтому я их вынес сюда)

float\* createMas(int length) – функция принимает на вход длину создаваемого массива, возвращает указатель на массив указанной длинны ,содержащий числа в диапазоне от -1000.999 до 1000.999.

int compareFunction(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.

int isSortCorrect(float\* orig\_mas, float\* sort\_mas, int length) – на вход передаются указатель на исходный массив ,копия исходного массива отсортированная некоторой сортировкой и их длинна. Функция сортирует исходный массив при помощи проверенной сортировки из стандартной библиотеки и поэлементно сравнивает полученный массив с отсортированным другой сортировкой. Если массивы не совпадают ,возвращается 0 ,иначе 1.

## Сортировка пузырьком

void bubleSort(float\* data,int length,int\* compare,int\* swap) – на вход функция принимает указатель на сортируемый массив, длинна сортируемого массива, указатель на переменную в которую будет записано количество сравнений , указатель на переменную в которую будет записано количество перестановок.

## Сортировка Шелла

void shellSort(float\* data,int length,int\* swap,int\*compare) - на вход функция принимает указатель на сортируемый массив, длинна сортируемого массива, указатель на переменную в которую будет записано количество сравнений , указатель на переменную в которую будет записано количество перестановок.

## Сортировка слиянием

float\* mergeSort(float \*sortable, int left, int right) – на вход функция принимает указатель на сортируемый массив, левый индекс сортируемой части, правый индекс сортируемой части. Возвращает указатель на отсортированную часть массива.

float \*merge(float \*mas1,int l\_mas1,float \*mas2,int l\_mas2) - на вход функция принимает указатель на первый отсортированный массив, длину первого массива, указатель на второй отсортированный массив, длину второго массива.

Возвращает указатель на отсортированный массив, состоящий из элементов входных массивов. Возвращаемый массив имеет длину l\_mas1+l\_mas\_2.

## Поразрядная сортировка

void signedRadixSort(float\* in, float\* out, int N) – функция принимает на вход указатель на первый элемент входного массива , указатель на первый элемент выходного массива , их длину. Функция при помощи нижеописанных функций сортирует входной массив.

void radixPass(short Offset,int N, uint\* source, uint\* 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 signedRadixLastPas(short Offset, int N, uint\* source, uint\* 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(). Данная функция сортирует отрицательные числа.

int\* createCounters(uint\* 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(uint)\*sizeof(int). Функция возвращает указатель на полученный массив.

# Подтверждение корректности

Для проверки корректности реализации всех сортировок была использована функция isSortCorrect().

На вход подаются указатели на первый элемент оригинального и отсортированного массив, а так же их длинна. В начале происходит сортировка оригинального массива при помощи стандартной сортировки qsort() из библиотеки stdlib. Поскольку данная сортировка входит в стандартную библиотеку языка Си, то можно быть уверенным в корректности ее работы. После сортировки начинается поэлементное сравнения оригинального и отсортированного массива. Если элементы массивов не совпадают, сравнение заканчивается и возвращается значение 0.Если массивы идентичны, то будет возвращено значение 1.

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

# Результаты экспериментов

n обозначает количество элементов в массиве.

Сложность вычислялась по сложности указанной для каждой сортировки. 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.

Перестановка элементов в сортировках осуществлялась в три присвоения.

## Сортировка пузырьком

Сложностью данной сортировки является O(n2).

На рисунке 5 график изменения отношения количества перестановок к сложности при росте количества элементов в массиве. Видно ,что значения сходятся к ~0.25. Поскольку диапазон значений с ростом количества элементов уменьшается и значения становится все ближе к некоторому числу, то это является экспериментальным подтверждением указанной сложности по сравнениям 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 2. Поскольку график сходится к некоторому значению, то это является экспериментальным подтверждением указанной сложности по сравнениям для данной сортировки.

Рисунок 5 – перестановки/сложность

Рисунок 6 - сравнения/сложность

## Сортировка Шелла

Сложностью данной сортировки является O(n\*log2(n)).

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 перестановкам 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 - перестановки/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 значению приблизительно равному 2,9. Поскольку график сходится к некоторому значению, то это является экспериментальным подтверждением указанной сложности по сравнениям 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 значению приблизительно равному 5,5. 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(4\* (n+256)+n).

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 4,2. Поскольку график сходится к некоторому значению, то это является экспериментальным подтверждением указанной сложности по сравнениям 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 2,8. 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 пузырьком, 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.