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

Голубев А.М.

**Проверил**:

преподаватель каф. МОСТ,

ВолокитинВ.Д.

Нижний Новгород

2021

**Содержание**

[Постановка задачи 3](#_Toc91532633)

[Метод решения 4](#_Toc91532634)

[Руководство пользователя 6](#_Toc91532635)

[Описание программной реализации 7](#_Toc91532636)

[Подтверждение корректности 8](#_Toc91532637)

[Результаты экспериментов 9](#_Toc91532638)

[Заключение 12](#_Toc91532639)

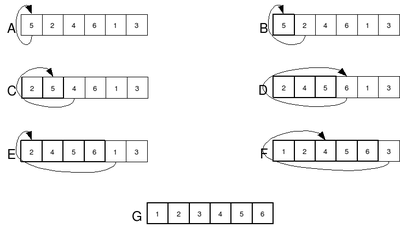
# Постановка задачи

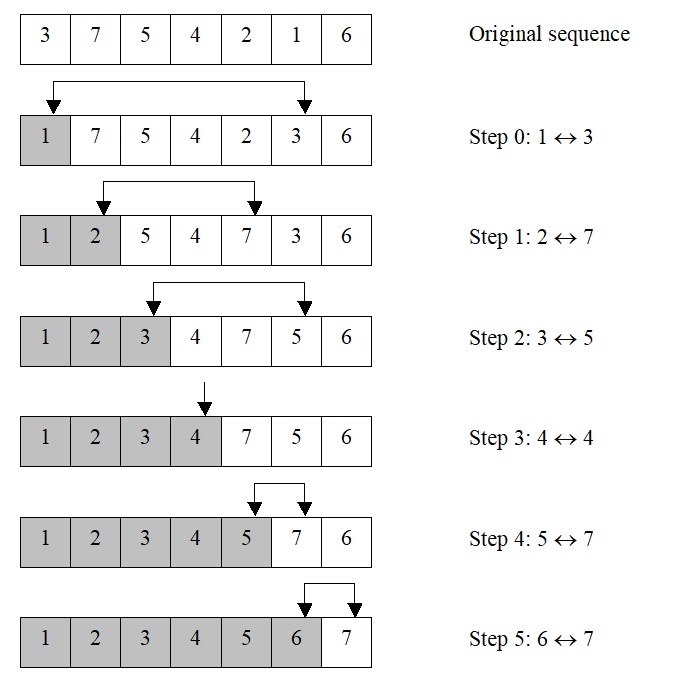
Задачей лабораторной работы являлось реализовать на языке программирования Си сортировку вставками, сортировка расчёской, сортировку слиянием и поразрядную сортировку для данных типов double. Нужно описать программную реализацию и алгоритмы работы данных сортировок. Необходимо подтвердить корректность реализации данных сортировок . Провести эксперименты для подтверждения сложности, описать способ проведения экспериментов и сделать вывод по полученным результатам.

# Метод решения

**Сортировка вставками**

Простой сортировка. Суть которого заключается в том что, на каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем. Стоит отметить что массив из 1-го элемента считается отсортированным. Вычислительная сложность алгоритма **O(n2)**.

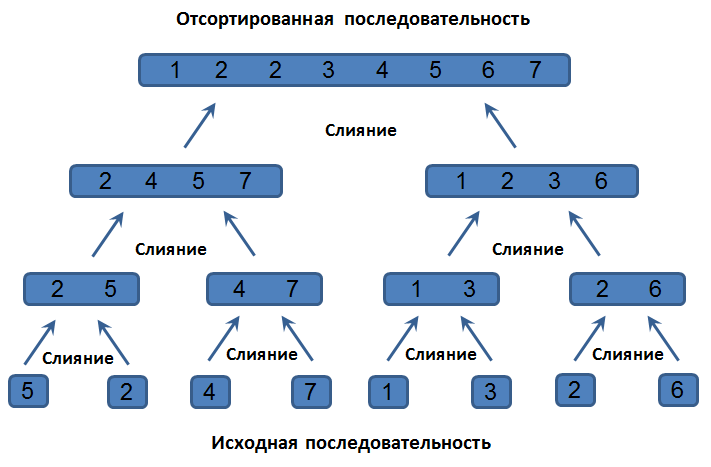


**Сортировка расчёской**

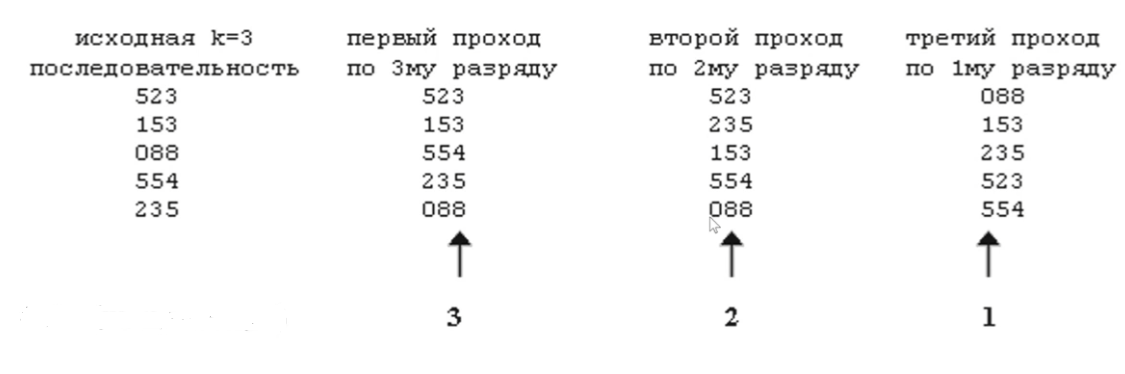
Быстрая сортировка. Сортировка расчёской улучшает сортировку пузырьком. Основная идея — устранить черепах, или маленькие значения в конце списка, которые крайне замедляют сортировку пузырьком. В сортировке пузырьком, когда сравниваются два элемента, промежуток равен 1, а в сортировке расчёской уже этот промежуток может быть гораздо больше, чем единица. Вычислительная сложность алгоритма **O(n log(n))**.

**Сортировка слиянием**

Быстрая сортировка. Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи сортируются. Наконец, их решения комбинируются, и получается решение исходной задачи. Вычислительная Сложность алгоритма **O(n log2(n))** в любом случае, а так же необходимо **О(n)** памяти.



**Поразрядная сортировка**

 Линейная сортировка. Является расширенной версией сортировки подсчетом. Для каждой цифры числа справа налево поочередно используем поразрядную сортировку. Поскольку сортировка подсчетом является устойчивой, то цифры располагаются в правильном порядке. Вычислительная сложность алгоритма **O(n)**.

# Руководство пользователя

При запуске программы выводится окно, где пользователь должен выбрать сортировку которую он хочет использовать, после ввести размер массива и потом уже сам массив. Тогда выведется отсортированный массив, число сравнений и перестановок.

# Описание программной реализации

void main – меню для программы, где задаётся массив и выбирается нужная сортировка.

**Сортировка вставками**

void insertion\_sort(double mas[], int n) – на вход функция принимает заданный массив и его размер. Возвращает отсортированный массив.

**Сортировка расчёской**

void comb\_sort(double array[], const size\_t size) - на вход функция принимает заданный массив и его размер. Возвращает отсортированный массив.

**Сортировка слиянием**

void mergesort(double array[], double second[], int l, int r) - на вход принимает массив и его копия, левый индекс массива и правый. От туда он идёт в функцию merge. После чего выводит отсортированный массив

void merge(double array[], double second[], int l, int q, int r) – на вход функция принимает указатель на первый отсортированный массив, указатель на второй отсортированный массив, левый индекс массива, середина и правый. Оттуда он отдаёт новые данные в функцию mergesort.

**Поразрядная сортировка**

void radixSort(double\* data, double\* sorted\_data, int N) – функция принимает на вход указатель на данный массив ,указатель на копию массива и их длину. Функция при помощи нижеописанных функций сортирует входной массив.

void obmen(double\* out, int N, double\* mas) - на вход функция принимает отсортированный массив, размер массива и копию отсортированного массива. Тут идёт перенос отсортированной отрицательной части массива из конца в начало.

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

void radixPass(short Offset, int N, ulonglong\* sourse, ulonglong\* 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.

# Подтверждение корректности

Проверили корректность программы с помощью функции qsort().

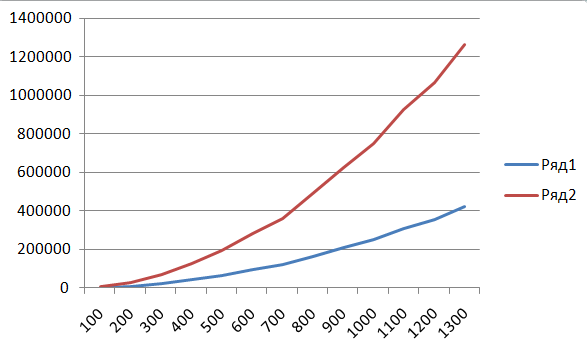
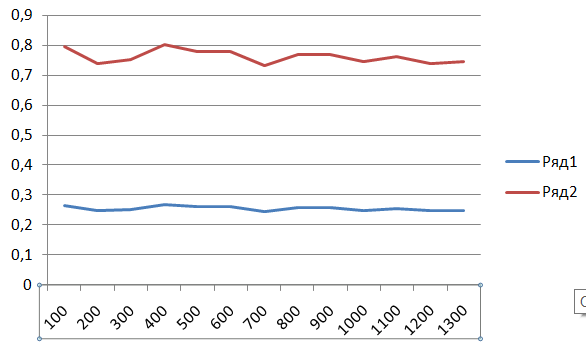
На вход подаются указатели на первый элемент оригинального и отсортированного массив, а так же их длинна. В начале происходит сортировка оригинального массива при помощи стандартной сортировки qsort() из библиотеки stdlib. Поскольку данная сортировка входит в стандартную библиотеку языка Си, то можно 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.

# Результаты экспериментов

Для подтверждения сложности необходимо разделить результаты замеров перестановок и сравнения на предполагаемую сложность. В результате данной операции должна получиться константа или значения попадающие в небольшой диапазон.

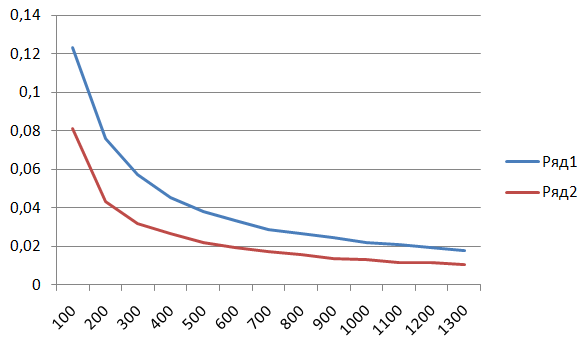
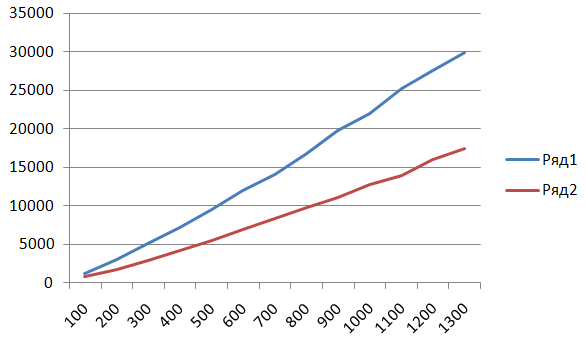
* z – количество перестановок;
* sr – количество сравнений;
* n – количество элементов в массива.

**Сортировка вставками**

Сложность сортировки **O(n2)**. По графику видно что количество слияний и перестановок параболически растёт. Если разделить каждое полученное значение на её сложность, то получим график константы, равной ~0,75 для слияния и ~0,25 ,что подтверждает сложность алгоритма с точностью до константы.

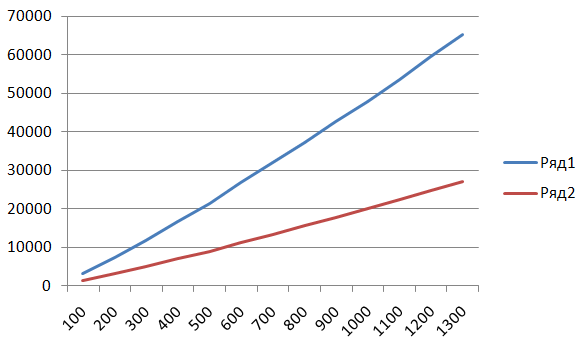
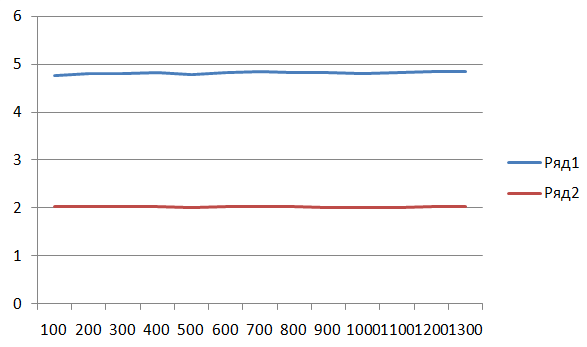
Перестановка Перестановка/сложность  
Сравнение Сравнение/сложность

**Сортировка расчёской**

Сложность сортировки **O(n log(n))**. Если разделить каждое полученное значение на n \* log(n), то получится график сходящийся к константе. Для сравнений эта константа равна ~0,018. Для присвоений же эта константа равна ~0,01. Это доказывает теоретическую сложность сортировки.

Перестановка Перестановка/сложность  
Сравнение Сравнение/сложность

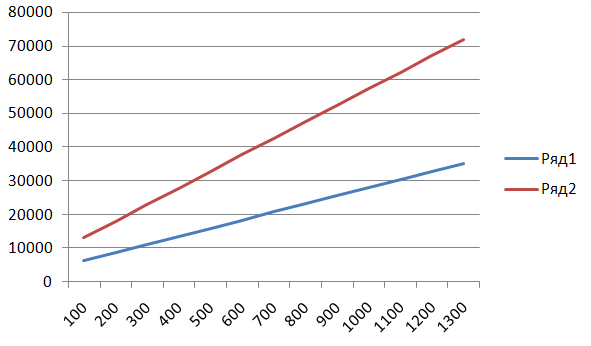
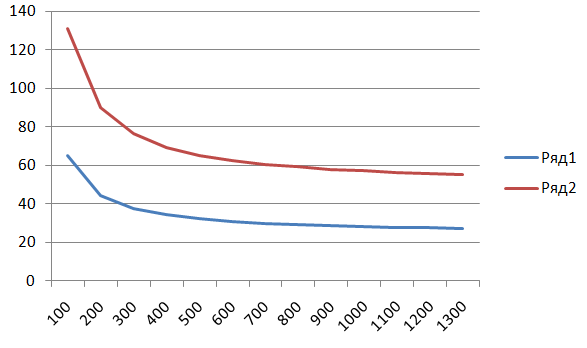
**Сортировка слиянием**

Сложность сортировки **O(n log2(n))**. По результатам замеров получен график роста сложности в зависимости от числа элементов. Если разделить каждое полученное значение на n\*log(n) ,то получим константу для перестановки ~4,8 и для сравнения ~2. Поскольку при делении получена константа, то можно заключить, что сложностью данного алгоритма действительно является O(n\*log(n)).

Перестановка Перестановка/сложность  
Сравнение Сравнение/сложность

**Поразрядная сортировка**

Сложность сортировки **O(n)**.При делении полученных значений на предполагаемую сложность, то получатся графики, сходящиеся к константе. Для графика, на котором изображено отношение перестановок деленных на сложность, эта константа равна ~57. Для графика, на котором изображено отношение сравнений деленных на сложность, это константа равна ~25. В совокупность это доказывает предполагаемую сложность сортировки.



Перестановка Перестановка/сложность  
Сравнение Сравнение/сложность

# Заключение

В ходе лабораторной работы на языке программирования Си были реализованы сортировка вставками, сортировка расчёской, 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.