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

Соколов В.Д.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**Оглавление**

[Постановка задачи 3](#_Toc91072720)

[Методы решения 4](#_Toc91072721)

[Сортировка выбором 4](#_Toc91072722)

[Сортировка расчёской 5](#_Toc91072723)

[Сортировка слиянием 6](#_Toc91072724)

[Поразрядная сортировка 7](#_Toc91072725)

[Руководство пользователя 8](#_Toc91072726)

[Описание программной реализации 10](#_Toc91072727)

[Подтверждение корректности 13](#_Toc91072728)

[Результаты экспериментов 14](#_Toc91072729)

[Заключение 16](#_Toc91072730)

# Постановка задачи

Целью лабораторной работы являлась реализовать на языке программирования Си следующие сортировки:

1.Сортировка выбором

2.Сортировка расчёской

3.Сортировка слиянием

4.Поразрядная сортировка

Также необходимо было выполнить **3** **задания:**

1. Реализовать сортировки для типов данных по предложенной таблицы
2. Замерить число перестановок и количество сравнений выполненное при сортировке
3. Написать отчет. Эксперименты поставить так, чтобы показать теоретическую сложность алгоритм

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

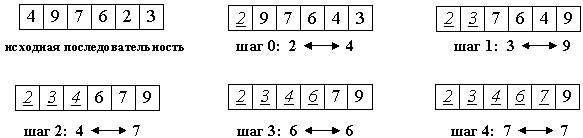
Сортировка массивов методом выбора, пожалуй, самая простая для понимания, хотя и одна из самых медленных.

Суть алгоритма проста, мы ищем наименьший элемент в массиве и перемещаем его на первое место. Затем ищем второй наименьший элемент и перемещаем его уже на второе место после первого наименьшего элемента. Этот процесс продолжается до тех пор, пока в массиве не закончатся неотсортированные элементы. Отсортированные элементы мы больше не трогаем, тем самым уменьшая кол-во последующих сравнений.

min

min

min



min

min

min

Замена 7 на 4

Оставляем на месте

Оставляем на месте

Замена 9 на 3

Замена 4 на 2

Рисунок 1 - иллюстрация работы алгоритма (min)

Также действует и наоборот (ищем наибольший элемент):

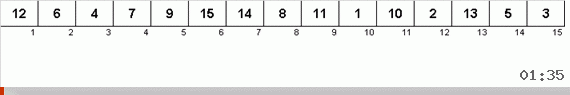


Рисунок 2 - иллюстрация работы алгоритма (max)

На первый взгляд кажется, что это ускоряет алгоритм в 2 раза — после каждого прохода неотсортированная часть массива уменьшается не с одной, а сразу с двух сторон. Но не стоит забывать, что при этом в 2 раза увеличилось количество сравнений, а число замен осталось неизменным. Двойной выбор лишь незначительно увеличивает скорость алгоритма, а на некоторых языках даже почему-то работает медленнее.

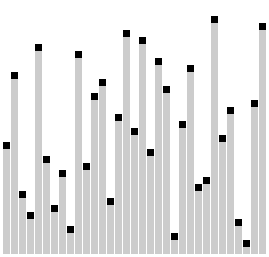
Сложность данного алгоритма сортировки равна **O(n2)**.

## Сортировка расчёской

Сортировка расчёской улучшает сортировку пузырьком, и конкурирует с алгоритмами, подобными быстрой сортировке.

В сортировке пузырьком, когда сравниваются два элемента, промежуток между которыми равен 1. Основная идея же сортировки расчёской состоит в том, что этот промежуток может быть гораздо больше, чем единица.

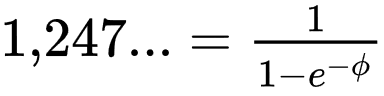
Суть алгоритма сортировки расчёской состоит в том, чтобы первоначально брать достаточно большое расстояние между сравниваемыми элементами и по мере упорядочивания массива сужать это расстояние вплоть до минимального.



*Таким образом, мы как бы причёсываем массив, постепенно разглаживая на всё более аккуратные пряди.*

Рисунок 3 - Визуализация сортировки расчёской

Первоначальный разрыв между сравниваемыми элементами лучше брать с учётом специальной величины, называемой ***фактором уменьшения***, оптимальное значение которой равно примерно ***1,247***. Сначала расстояние между элементами максимально, то есть равно размеру массива минус один. Затем, пройдя массив с этим шагом, необходимо поделить шаг на фактор уменьшения и пройти по списку вновь. Так продолжается до тех пор, пока разность индексов не достигнет единицы. В этом случае сравниваются соседние элементы, как и в сортировке пузырьком, но такая итерация одна.

Оптимальное значение **фактора уменьшения** , где

**e** – основание натурального логарифма, а **ф** - золотое сечение.

Сложность данного алгоритма сортировки равна **O(nlogn)**.

## Сортировка слиянием

Сортировка слиянием — это простой алгоритм сортировки. Суть его заключается в том что, мы разбиваем массив на два массива поменьше 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.

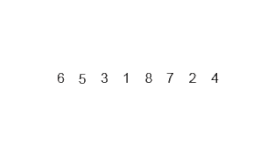
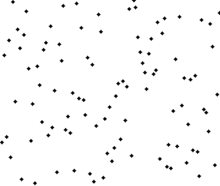


Рисунок 4 - иллюстрация работы сортировки слиянием



Алгоритм состоит из трёх простых шагов:

Шаг 1: Сортируемый массив разбивается на две части примерно одинакового размера;

Шаг 2: Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;

Шаг 3: Два упорядоченных массива половинного размера соединяются в один.

Рисунок 5 -

- Действие алгоритма на примере сортировки случайных точек.

Сложность данного алгоритма сортировки равна **O(nlogn)**.

Необходимо **О(n)** памяти.

## Поразрядная сортировка

Поразрядная сортировка является расширенной версией сортировки подсчетом.

**Суть Алгоритма:** 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

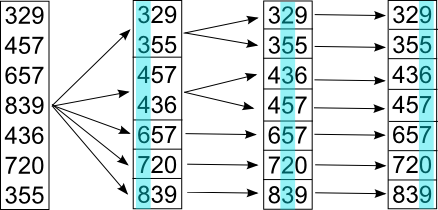


Рисунок 6 - иллюстрация работы поразрядной сортировки (MSD)

Так как выравнивать сравниваемые записи относительно друг друга можно в разную сторону, на практике существуют **два варианта этой сортировки**. Для чисел они называются в терминах значимости разрядов числа, и получается так: можно выровнять записи чисел *в сторону менее значащих цифр* (по правой стороне, в сторону единиц, **LSD**) или *более значащих цифр* (по левой стороне, со стороны более значащих разрядов, **MSD**).

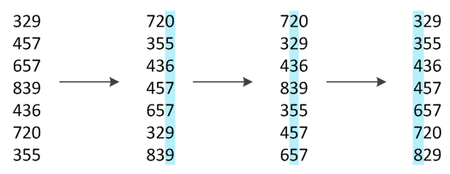


Рисунок 7 - иллюстрация работы поразрядной сортировки (LSD)

**O(w n)**, где w – кол-во бит, требуемых для хранения каждого ключа

Сложность данного алгоритма сортировки равна

Необходимо памяти: **O(w+n)**

# Руководство пользователя

Программы all\_sorts (все сортировки), selection\_sort (сортировка выбором), comb\_sort (сортировка расчёской), mergesort (сортировка слиянием), radixsort (поразрядная сортировка) обладают удобными меню (на английском языке), в которых объясняется, что нужно делать:

Пример all\_sorts:

Вы должны выбрать нужный вам тип сортировки (просто введите число):

Вы выбираете тип сортировки из предложенных ниже и вводите число, обозначенное в скобках левее от выбранного вами типа сортировки.

(1)Selection sort | (1)Сортировка выбором

Если Вы введёте в командной строке единицу (1), то Вас перенесёт в меню сортировки выбором.

(2)Combination sort | (2)Сортировка расчёской

Если Вы введёте в командной строке двойку (2), то Вас перенесёт в меню сортировки расчёской.

(3)Merge sort | (3)Сортировка слиянием

Если Вы введёте в командной строке тройку (3), то Вас перенесёт в меню сортировки слиянием.

(4)Radix sort | (4)Поразрядная сортировка

Если Вы введёте в командной строке четвёрку (4), то Вас перенесёт в меню поразрядной сортировки.

(0)exit program | (0)Сортировка выбором

Если Вы введёте в командной строке ноль (0), то Вы завершите программу. Программа не завершится пока Вы, либо не введёте ноль **в этой меню**, либо не закроете насильно all\_sorts.exe

You must choose which sorting you will use (just put the number):

(1)Selection sort

(2)Combination sort

(3)Merge sort

(4)Radix sort

(0)exit program

Пример selection\_sort: (Вы можете попасть сюда из программы all\_sorts)

Пожалуйста введите числа (размерности double) в массив размером 10

Массив будет отсортирован сортировкой выбором

Пользователь должен ввести десять [10] чисел размерности double (число с плавающей запятой). Число может быть, как положительным, так и отрицательным. После ввода одного числа **нажмите на Enter**, чтобы написать следующее число на другой строке. У Вас должен получиться столбец из десяти введённых Вами чисел.

Нажав очередной раз Enter, пользователь увидит результат сортировки. Результат будет также записан в столбец, но с небольшим отступом.

Пользователь также заметит под отсортированным столбцом два новых объекта.

sort\_times | кол-во перестановок

показывает сколько раз было произведено перестановок.

check\_times | кол-во сравнений

показывает сколько раз было произведено сравнений. (В связи с тем, что массив константного размера – это кол-во всегда одно и то же)

please enter the numbers(double) in 10-size array

It will be sorted by <selection> type

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[пример работы]

3

123

1

4

-0

-4

1.3

1

3

4

-4.000000

-0.000000

1.000000

1.000000

1.300000

3.000000

3.000000

4.000000

4.000000

123.000000

sort\_times = 13 check\_times = 45

# Описание программной реализации

*(Здесь будет описана программа* all\_sorts*, так как в ней присутствуют все функции)*

int main() – это основная функция программы. Программа всегда начинается с main и всегда ею заканчивается (определение main). Заводится бесконечный цикл и в нем сперва на экран выводится меню. Далее программа принимает на вход вводимые пользователем числа. В зависимости от выбранного числа используются определённые функции.

Пользователь ввёл число 1: - программа использует функцию c\_selection\_sort

int c\_selection\_sort() – выводит меню и запрашивает десять значений (чисел double) от пользователя. Затем используется функция selection\_sort.

int selection\_sort(double array[], const size\_t size) – принимает на вход заведённый ранее (int c\_selection\_sort()) массив. Далее мы с первого элемента идём далее по массиву, находим наименьший элемент и производим замену. Также со вторым, третьим и далее идущими элементами.

Делаем так (size – 1) раз, так как последний элемент нам не нужно ни с чем сравнивать. Считается кол-во перестановок и кол-во сравнений. Затем на экран выводится отсортированный массив, кол-во перестановок и кол-во сравнений.

Пользователь ввёл число 2: - программа использует функцию c\_comb\_sort

int c\_comb\_sort() - выводит меню и запрашивает десять значений (чисел double) от пользователя. Затем используется функция comb\_sort.

int comb\_sort(double array[], const size\_t size) - принимает на вход заведённый ранее (int c\_comb\_sort()) массив. В функции используется оптимальный фактор равный 1.2473309 . Заводится цикл while (step >= 1) , а в нём другой цикл, где мы сравниваем элементы находящиеся друг от друга на расстоянии step (шаг) , производим замену, если это требуется, а затем уменьшаем step (step /= factor), делаем всё то же, но уже с новым step и так, пока step не станет строго меньше (<) единицы. Считается кол-во перестановок и кол-во сравнений. Затем на экран выводится отсортированный массив, кол-во перестановок и кол-во сравнений.

Пользователь ввёл число 3: - программа использует функцию c\_merge\_sort

int c\_merge\_sort() - выводит меню и запрашивает десять значений (чисел double) от пользователя. Далее используется функция merge\_sort. После используется функция massive. Затем на экран выводится отсортированный массив, кол-во перестановок и кол-во сравнений.

void mergesort(double array[], double second[], int l, int r) - принимает на вход заведённый ранее (int c\_merge\_sort()) массив, пустой массив (там будет записан отсортированный массив), а также порядки крайних элементов l=0 (первый элемент) r=9 (последний элемент). Затем массив делят на две относительно равные части (5:5) и сортируют их по отдельности функцией mergesort (получается рекурсия). Потом используется функция слияния merge.

void merge(double array[], double second[], int l, int q, int r) – функция слияния. Принимает два массива, сортирует их и объединяет. Здесь также считается кол-во перестановок и кол-во сравнений.

int massive(double array[], const size\_t size) – принимает и выводит массив.

Пользователь ввёл число 4: - программа использует функцию c\_radix\_sort

void c\_radix\_sort() - выводит меню и запрашивает десять значений (чисел double) от пользователя. Записывает вводимые значения в два массива (массивы идентичны). Далее используется функция signedRadixSort . Затем выводится отсортированный массив, а также выводится кол-во сравнений.

void signedRadixSort(double\* in, double\* out, int N, double\* res) – принимает на вход два, заведённых пользователем, массива. Далее используется функция radixSort . Затем получившийся массив разделяется на массив с отрицательными значениями и на массив с положительными значениями. Эти два массива сортируются и объединяются.

void radixSort(double\* in, double\* out, int N) – принимает на вход два массива. Используется функция createCounters . Затем массив сортируется и используется функция radixPass.

long\* createCounters(double\* data, int N) – функция выводит число. Функция принимает на вход массив и его размер. Создаётся counters (кол-во подсчётов).

void radixPass(short Offset, int N, double\* sourse, double\* dest, long\* count) – принимает на вход три числа (номер элемента, размер массива, счёт) и два массива. Производится поразрядная сортировка.

Пользователь ввёл число 0: - это просто выход из цикла break

# Подтверждение корректности

Для проверки корректности реализации всех сортировок был использован сложный массив.

[-999999999;999999999;-0;0;1;1.0000001;1.00000004;8;3;-5]

It will be sorted by <selection> type

-999999999

999999999

-0

0

1

1.0000001

1.00000004

8

3

-5

-999999999.000000

-5.000000

-0.000000

0.000000

1.000000

1.000000

1.000000

3.000000

8.000000

999999999.000000

sort\_times = 4 check\_times = 45

В результате получилось:

Можно заметить, что в результате у нас исчезли значения шестого и далее элементов после запятой, но посмотрев на кол-во перестановок можно установить, что эти числа всё-таки отсортировались.

Далее приведён список результатов других видов сортировок:

It will be sorted by <Radix> type

-999999999

999999999

-0

0

1

1.0000001

1.00000004

8

3

-5

-999999999.000000

-5.000000

0.000000

1.000000

1.000000

1.000000

3.000000

8.000000

999999999.000000

-0.000000

sort\_times =8

It will be sorted by <combination(rascheska)> type

-999999

999999999

-0

0

1

1.0000001

1.00000004

8

3

-5

-999999.000000

-5.000000

-0.000000

0.000000

1.000000

1.000000

1.000000

3.000000

8.000000

999999999.000000

sort\_times = 11 check\_times = 39

It will be sorted by <merge> type

-999999999

999999999

-0

0

1

1.0000001

1.00000004

8

3

-5

-999999999.000000

-5.000000

0.000000

-0.000000

1.000000

1.000000

1.000000

3.000000

8.000000

999999999.000000

# Результаты экспериментов

В связи с тем, что мой массив был размером 10, разница в скорости программ вообще не ощущалась.

По данным эксперимента выяснилось следующее:

1)Скорость программ (что быстрее?):

#1 Radix sort (поразрядная сортировка)

#2 Merge sort (сортировка слиянием)

#3 Combination sort (сортировка расчёской)

#4 Selection sort (сортировка выбором)

2)Сложность программ:

Поразрядная сортировка:

Сложность алгоритма: O(w n)

Необходимо памяти: O(w+n)

#1

Сортировка слиянием:

Сложность алгоритма: O(n logn)

Необходимо памяти: O(n)

#2

Сортировка расчёской:

Сложность алгоритма: O(n logn)

#3

#4

Сортировка выбором:

Сложность алгоритма: O(n2)

3)Графики показывающие сложность относительно кол-ва элементов:

Сложность: O(n2)

Сортировка выбором

Сложность: O(n logn)

Сортировка расчёской

Сложность: O(n logn)

Необходимо памяти: O(n)

Сортировка слиянием

Сложность: O(w n)

Необходимо памяти: O(w+n)

Поразрядная сортировка

# Заключение

В ходе лабораторной работы на языке программирования Си были реализованы сортировка выбором, 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 связи с тем, что мой компьютер не способен обрабатывать большой объём данных и сильно тупит даже в очень простых программах (all\_sorts запускал 5 минут) результаты экспериментов были взяты с интернет ресурсов.