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

Нагорная Е.А.

**Проверил**:

преподаватель каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2021

**Оглавление**

[Постановка задачи 3](#_Toc89867396)

[Методы решения 4](#_Toc89867397)

[Сортировка вставками 4](#_Toc89867398)

[Сортировка расчёской 4](#_Toc89867399)

[Сортировка слиянием 5](#_Toc89867400)

[Поразрядная сортировка 6](#_Toc89867401)

[Руководство пользователя 7](#_Toc89867402)

[Описание программной реализации 8](#_Toc89867403)

[Сортировка вставками 8](#_Toc89867404)

[Сортировка расчёской 8](#_Toc89867405)

[Сортировка слиянием 9](#_Toc89867406)

[Поразрядная сортировка 9](#_Toc89867407)

[Подтверждение корректности 10](#_Toc89867408)

[Результаты экспериментов 11](#_Toc89867409)

[Сортировка вставками 11](#_Toc89867410)

[Сортировка расчёской 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 вставками**

Сортировка вставками (*Insertion Sort*) — это простой алгоритм сортировки. Суть его 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 первого элемента считается отсортированным. Словесное описание алгоритма звучит довольно сложно, но на деле это самая простая в реализации сортировка. Сложность данного алгоритма сортировки в среднем и худшем случаях равна O(n2).

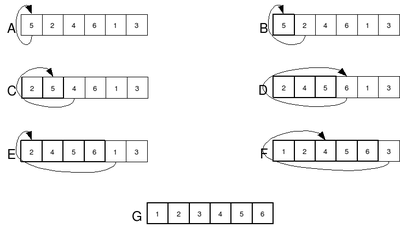


Рисунок 1 – иллюстрация работы алгоритма

**Сортировка расчёской**

Сортировка расческой (*Comb sort*) представляет собой улучшенный вариант сортировки пузырьком. Но в отличие от нее этот алгоритм больше не является устойчивой сортировкой.

Сортировка пузырьком основана на сравнении и перестановке соседних элементов, то есть разница их индексов всегда равна 1. Но если увеличить ее и сравнивать элементы, стоящие на большем расстоянии друг от друга, то минимальные элементы, находящиеся в конце массива, можно передвинуть в его начало за меньшее количество перестановок. Сложность данного алгоритма сортировки в худшем случае равна O(n2).

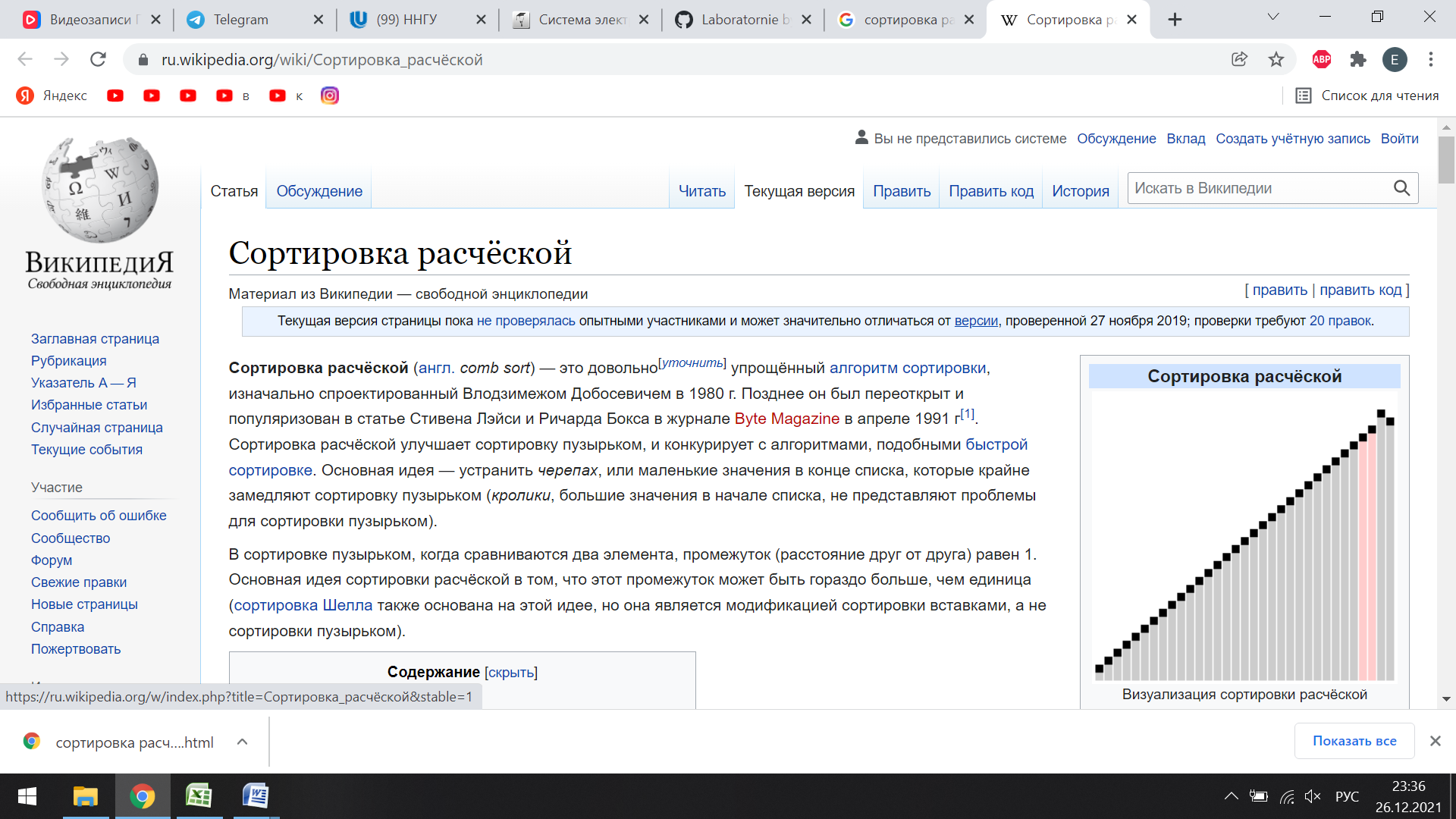
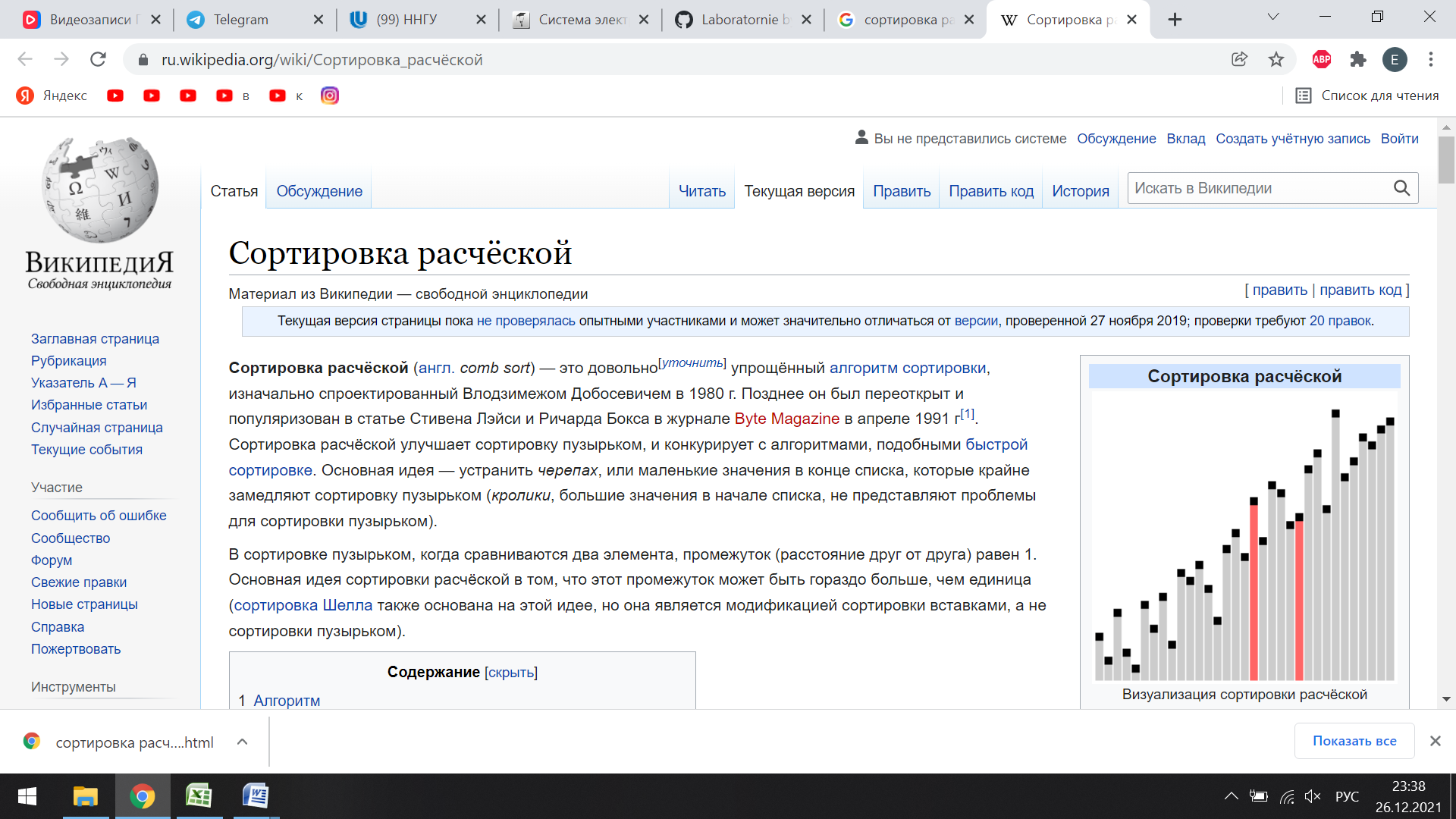
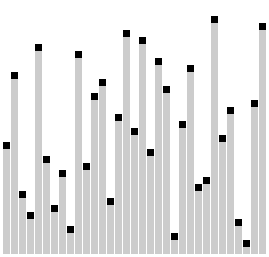
****

Рисунок 2 – иллюстрация работы алгоритма

**Сортировка слиянием**

Сортировка слиянием (*Merge Sort*) — это алгоритм сортировки, который упорядочивает исходный массив в определённом порядке.  
Слияние означает объединение двух (или более) последовательностей в одну упорядоченную последовательность при помощи циклического выбора элементов, доступных в данный момент.  
Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Затем их решения комбинируются, и получается решение исходной задачи. Сложность данного алгоритма сортировки в худшем случае равна O(n\*log2 n).

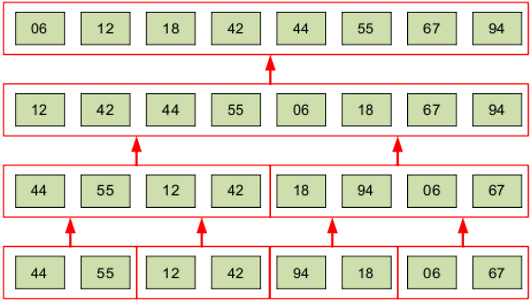


Рисунок 3 – иллюстрация работы алгоритма

**Поразрядная сортировка**

 Поразрядная сортировка (*Radix Sort*) — алгоритм сортировки, который выполняется за линейное время. 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.

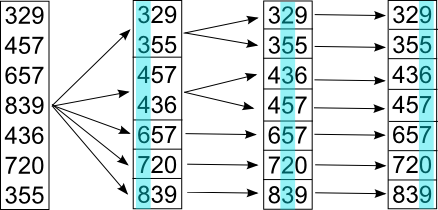


Рисунок 4 - иллюстрация работы алгоритма

# Руководство пользователя

Все сортировки выводят в консоль длину сортируемого массива, количество перестановок и сравнений. Вывод осуществляется в следующем формате:

«Исходный массив

Отсортированный массив

Массив, отсортированный другим алгоритмом

Подтверждение корректности

Размер массива; сумма перестановок; сумма сравнений»

# Описание программной реализации

Рассмотрим алгоритмы, применяемые во всех сортировках.

Ввод массива A[n]:

for (i = 0; i < n; i++) {A[i] = -100+ rand()%1000;

printf("%.0f ", A[i]);

}

Сортировка массива методом пузырька для проверки:

for (i = 0; i < n-1; i++)

for (j=n-2; j>(i-1);j--)

if (B[j] > B[j + 1]) {tmp = B[j];

B[j] = B[j + 1];

B[j + 1] = tmp;

}

Проверка корректности сортировки:

for (i = 0; i < n; i++) {if (A[i] == B[i]) { k = k + 1; };

printf("%.0f ", B[i]);

}

if (k == n) { printf("Correct");}

Вывод необходимых для анализа сложности данных:

printf("%i %i %i", n, sravn, perest);

**Сортировка вставками**

Цикл для сортировки, который ищет место для вставки выбранного элемента и переставляет его:

for (i = 1; i < n; i++)

{for (j = i; (j > 0) && (A[j - 1] > A[j]); j--)

{tmp = A[j - 1];

A[j - 1] = A[j];

A[j] = tmp;

}

}

**Сортировка расчёской**

Изначальная величина шага (n-1), далее она уменьшается. В это время происходит сравнение элементов, стоящих на позициях разностью в этот шаг. Они сравниваются между собой и при необходимости меняются местами.

Сортировка завершается при шаге, меньшем 1.

while (step >= 1)

{for (i = 0; i + step < n; i++)

{if (A[i] > A[i + step])

{tmp = A[i];

A[i] = A[i + step];

A[i + step] = tmp;

}

} step /= factor;

}

**Сортировка слиянием**

merge(float A[], int p, int q, int r) - процедура слияния, которая сливает две части массива.

mergesort(float A[], int l, int r) - функция, которая сортирует подотрезок массива A[n] с индексами в полуинтервале.

void main()

{ . . .

mergesort(A, 0, n - 1);

. . .

} - основная часть кода, запускающая сортировку.

**Поразрядная сортировка**

velich\_razr(int chislo, int razr) – распределяет числа в массивы по разрядам.

sort\_razr(float dop\_mas[(n < 10) ? 10 : n][(n < 10) ? 10 : n], float mas[n], int razr) – сортирует полученные по разрядам массивы.

int main()

{ . . .

for (razr = 1; razr < 4; razr++)

sort\_razr(dop\_mas, A, razr);

. . .

} - основная часть кода, объединяющая отсортированные массивы в искомый отсортированный массив.

# Подтверждение корректности

Для проверки корректности всех сортировок использовался следующий алгоритм:

for (i = 0; i < n; i++) {if (A[i] == B[i]) { k = k + 1; };

printf("%.0f ", B[i]);

}

if (k == n) { printf("Correct");}

Изначально, при вводе исходного массива, создавался идентичный ему массив.

Первый сортировался заданной сортировкой, а второй – методом пузырька.

Далее при помощи цикла подсчитывается число одинаковых элементов на одинаковых позициях обоих массивов.

Если это количество совпадает с длиной массива, то программа работает корректно и пользователь видит в консоли слово «Correct», иначе – «No correct».

# Результаты экспериментов

Длина массива изменялась с 5 до 195 элементов на 5 элементов каждый шаг. На каждом шаге после сортировки массива осуществлялась проверка на корректность. В случае корректного выполнения сортировки выводилась длинна массива, количество сравнений и присвоений.

По полученным данным заполнялась таблица, по которой строился график .

Поскольку нотация O не учитывает константу, то 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.

* perest – количество перестановок;
* sravn – количество сравнений;
* n – количество элементов в массива.
* Sum – сумма сравнений и перестановок

**Сортировка вставками**

Сложность данной сортировки равна O(n2),что видно на рисунке 5. График явно является параболой. Если разделить каждое полученное значение на n2 ,то получим график константы, равной ~0,2 ,что подтверждает сложность алгоритма с точностью до константы.

Рисунок 5 – сортировка вставками

**Сортировка расчёской**

Сложность данной сортировки равна O(n2),что видно на рисунке 6. График явно является параболой. Если разделить каждое полученное значение на n2 ,то получим график константы, равной ~0,1 ,что подтверждает сложность алгоритма с точностью до константы.

Рисунок 6 – сортировка расчёской

**Сортировка слиянием**

Средняя ложность данной сортировки равна O(n\*log2n).Время работы сортировки явно зависит от расположения элементов во входном массиве, рисунок 7. Если разделить каждое полученное значение на n\*log2n ,то получим график, значения которого ~7,5. Это позволяет сделать вывод, что сложностью данной сортировки с точность до константы является n\*log2n.

Рисунок 7 – сортировка слиянием

**Поразрядная сортировка**

Сортировка имеет линейную сложность O(k\*(n+m)+n), где k-длинна 1 сортируемого числа в байтах,m – возможные значения этого числа. По результатам замеров сложности был получен график, изображенный на рисунке 8. Если разделить каждое полученное значение на теоретическую сложность, то получим график константы ~0,3.

Рисунок 8 – поразрядная сортировка

# Заключение

В ходе лабораторной работы на языке программирования Си были реализованы сортировки вставками, расчёской, слиянием и поразрядная сортировка для массива, элементы которого имеют тип данных float.

Были описаны алгоритмы работы данных сортировок и их программная реализация, а также проведены эксперименты для замера и подтверждения их теоретической сложности и проверки корректности на большом объёме данных.