МИНИСТЕРСТВО науки и высшего ОБРАЗОВАНИЯ РОссИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ

УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ

(национальный исследовательский университет)»

|  |  |
| --- | --- |
| **Институт №3** | **Кафедра №301** |

Курсовая работа

по дисциплине

«**Программирование и основы алгоритмизации**»

на тему:

«Сортировка обменом и «расчёской»: сравнение методов»

Студента Якимович Марии Михайловны\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Группа \_\_\_М3О-101БВ-24\_\_\_\_\_\_\_\_\_\_\_   
Дата защиты «\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025\_ Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_   
Проверил Кананадзе С.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/

Москва – 2025

Оглавление

[Введение 3](#_Toc198565813)

[1. Теория по алгоритмам сортировки обменом и «расчёской» 5](#_Toc198565814)

[1.1. Алгоритм сортировки обменом (пузырьковая сортировка) 5](#_Toc198565815)

[1.2. Алгоритм сортировки «расчёской» 6](#_Toc198565816)

[2. Реализация алгоритмов сортировки обменом и «расчёской» и их применение в решении практических задач 8](#_Toc198565817)

[2.1. Реализация сортировки обменом 8](#_Toc198565818)

[2.2. Реализация сортировки «расчёской» 9](#_Toc198565819)

[3. Сравнительный анализ алгоритмов 12](#_Toc198565820)

[Заключение 17](#_Toc198565821)

[Список использованных источников 19](#_Toc198565822)

# Введение

Основополагающей темой данной курсовой работы являются алгоритмы сортировки обменом и сортировки «расчёской». Однако прежде чем перейти  
к детальному изучению и сравнению этих методов, необходимо понимать,  
что представляет собой сортировка данных в программировании.

Сортировка – это процесс упорядочивания элементов в определённой последовательности на основе заданных параметров. В программировании сортировка данных – это один из основных и самых распространённых методов. Она необходима для упорядоченного хранения, быстрого поиска и эффективной обработки информации. Упорядоченные (или чаще их называют отсортированные) данные проще анализировать, визуализировать и использовать в других алгоритмах.

Среди множества методов сортировки, стоит упомянуть сортировку обменом и сортировку «расчёской». Первая, известная как пузырьковая сортировка, часто используется в программировании из-за своей простоты. Вторая – сортировка «расчёской» – это улучшенная версия сортировки обменом, которая работает быстрее благодаря уменьшению числа лишних сравнений.

Актуальность выбранной темы заключается в необходимости оценки эффективности различных алгоритмов сортировки для решения задач разной сложности. Несмотря на то, что оба рассматриваемых алгоритма не являются самыми быстрыми на сегодняшний день, они часто используются в задачах небольшого объёма, а также в обучении и отладке.

**Целью исследования** данной курсовой работы является сравнительный анализ алгоритмов сортировки обменом и сортировки «расчёской».

Для достижения поставленной цели можно выделить следующие задачи:

- Рассмотрение принципов работы каждого из алгоритмов.

- Реализация и применение методов сортировки на основе программного кода С++.

- Сравнить алгоритмы по скорости и использованию памяти.

- Проанализировать эффективность алгоритмов при выполнении определенных задач.

**Методы исследования** – сбор, обработка и анализ информации из научных источников, учебной литературы и теоретических пособий при помощи использования Интернет-ресурсов.

**Объект исследования** – алгоритмы сортировки обменом и «расчёской».

**Предмет исследования** – анализ и эффективность алгоритмов сортировки  
обменом и «расчёской».

# 1. Теория по алгоритмам сортировки обменом и «расчёской»

## **1.1. Алгоритм сортировки обменом (пузырьковая сортировка)**

Пузырьковая сортировка (Bubble Sort) – это один из самых простых  
и понятных способов сортировки. Ее название связано с тем, что элементы  
в массиве «всплывают» на свои места, как пузырьки в воде. Идея заключается  
в последовательном сравнении значений соседних элементов в массиве попарно. Если числа в паре не расположены в порядке возрастания меняем их местами. Затем переходим к следующей паре. Сортировка считается законченной,  
если в ходе просмотра не была произведена ни одна перестановка.

Подробный принцип работы алгоритма по действиям:

1. Проходим по массиву от начала до конца.

2. Сравниваем каждую пару соседних элементов.

3. Если текущий элемент больше следующего, меняем их местами.

4. После каждого полного прохода элемент с наибольшим значением оказывается в конце массива.

5. Процесс повторяем до тех пор, пока массив не будет полностью отсортирован.

На рисунке 1 представлен пример процесса полной работы алгоритма пузырьковой сортировки. Массив из 5-ти элементов был отсортирован  
за 9 действий.

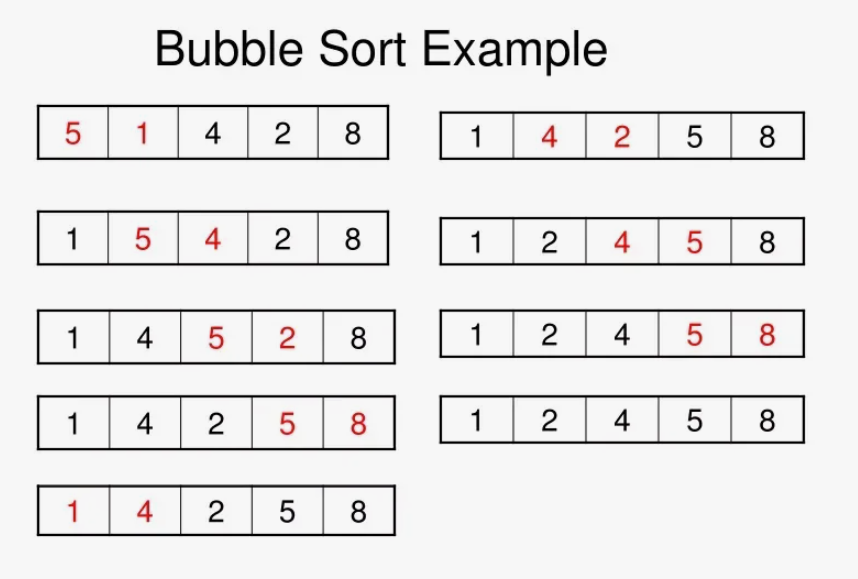


Рисунок 1

## **1.2. Алгоритм сортировки «расчёской»**

Сортировка «расчёской» (Comb Sort) была предложена в 1991 году  
как усовершенствование пузырьковой сортировки. Она улучшает сортировку обменом за счет устранения маленьких значений в конце списка. Достигается  
это тем, что вместо сравнения соседних элементов, сравниваются элементы  
на достаточно большом расстоянии друг от друга, постепенно уменьшая  
это расстояние. Таким образом, мы как бы причёсываем массив, постепенно разглаживая на всё более аккуратные пряди. Сначала разрыв между элементами берётся максимальный, т.е. на единицу меньше, чем размер массива. 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 сравниваемых элементов не достигнет единицы. В этом случае сравниваются уже соседние элементы, как и в сортировке пузырьком, но эта итерация будет последней.

Оптимальное значение фактора уменьшения – 1,247 (допускается также  
1,25 или 1,3). Почему мы используем именно коэффициент, а не уменьшаем расстояние на единицу? Это связано с тем, что если уменьшать расстояние  
на единицу, то это свело бы алгоритм к стандартной пузырьковой сортировке  
уже после нескольких шагов, поэтому таким способом мы не добьемся оптимизации процесса.

Подробный принцип работы алгоритма по действиям:

1. Устанавливаем начальное расстояние между сравниваемыми элементами   
(gap — шаг или интервал) равным длине массива минус один.

2. На каждом проходе по массиву gap уменьшается на фактор уменьшения  
по формуле.

3. Сравниваем элементы, которые расположены на расстоянии gap друг  
от друга, и при необходимости меняем их местами.

4. В конце делаем стандартный проход при gap=1, как при пузырьковой сортировке. После этого массив будет полностью отсортирован.

На рисунке 2 представлен пример процесса полной работы алгоритма сортировки «расчёской». Массив из 7-ми элементов был отсортирован  
за 5 действий.

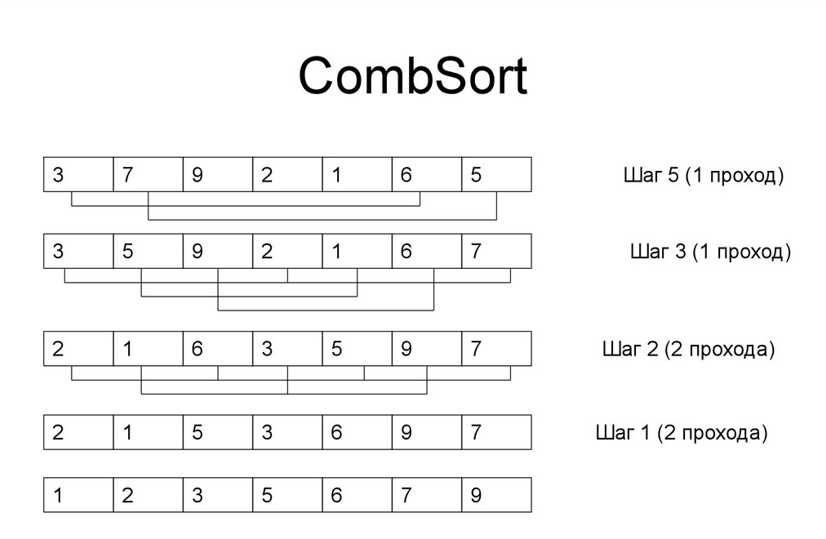


Рисунок 2

# 2. Реализация алгоритмов сортировки обменом и «расчёской» и их применение в решении практических задач

## **2.1. Реализация сортировки обменом**

Рассмотрим реализацию алгоритма пузырьковой сортировки на языке программирования С++. Поставим следующую задачу: отсортировать массив  
{1, 98, 11, -7, 41, 56, 22} по возрастанию при помощи алгоритма сортировки обменом (программа 1).

Программа 1

#include <iostream>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

//создаем неотсортированный массив

const int n = 7; //кол-во элементов массива

int arr[n] = { 1, 98, 11, -7, 41, 56, 22 };

//внешний цикл - кол-во проходов по массиву (i - номер прохода)

for (int i = 0; i < n - 1; i++) { //n-1 достаточное кол-во проходов для отсортировки всего массива

//внутренний цикл (j - элемент массива)

for (int j = 0; j < n - 1; j++) { //n-1 необходим, чтобы мы не вышли за границы массива

if (arr[j] > arr[j + 1]) { //проверка сравнения значений рядом стоящих парных элементов

int temp = arr[j]; //временно сохраняем значение текущего элемента

arr[j] = arr[j + 1]; //в текущую ячейку записываем значение из следующей

arr[j + 1] = temp; //возвращаем старое значение из temp

//произошла перестановка двух элементов

}

}

}

cout << "Результат: ";

for (int i = 0; i < n; i++) //проход по отсортированному массиву

cout << arr[i] << " "; //выводим все элементы через пробел

return 0;

}

Результат работы программы 1 представлен на рисунке 3.

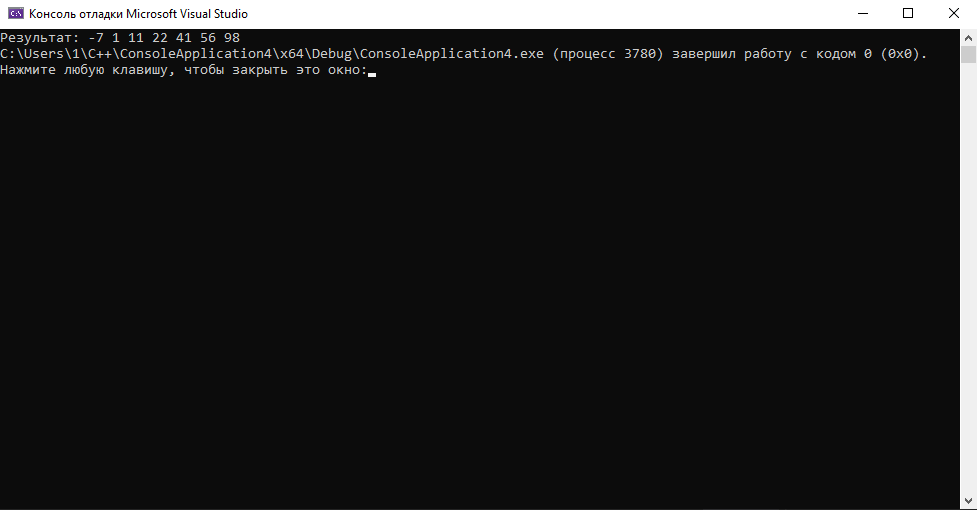


Рисунок 3

Преимуществом данной реализации является её простота. Однако  
при больших объёмах, данных пузырьковая сортировка работает медленно  
из-за высокой временной сложности – O(n²) в среднем и в худшем случае.

## **2.2. Реализация сортировки «расчёской»**

Сортировка «расчёской» реализуется также на языке C++. Используем  
тот же массив, как и в предыдущей задаче для точного сравнения (программа 2).

Программа 2

#include <iostream>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

//создаем неотсортированный массив

const int n = 7; //кол-во элементов массива

int arr[n] = { 1, 98, 11, -7, 41, 56, 22 };

int gap = n - 1; //начальный шаг - длина массива минус 1

float shrink = 1.3; //коэффициент уменьшения

bool sorted = false; //флаг для проверки сортировки

while (sorted == false) {

gap = int(gap / shrink); //уменьшаем шаг

if (gap <= 1) {

gap = 1; //последний проход

sorted = true; //предполагаем, что массив отсортирован

}

for (int i = 0; i + gap < n; i++) { //идем циклом с шагом gap

if (arr[i] > arr[i + gap]) { //проверка сравнения значений элементов с шагом gap

int temp = arr[i]; //временно сохраняем значение текущего элемента

arr[i] = arr[i + gap]; //в текущую ячейку записываем значение новой

arr[i + gap] = temp; //возвращаем старое значение из temp

//произошла перестановка двух элементов

sorted = false; //продолжаем сортировку повторно

}

}

}

cout << "Результат: ";

for (int i = 0; i < n; i++) //проход по отсортированному массиву

cout << arr[i] << " "; //выводим все элементы через пробел

return 0;

}

Результат работы программы 2 представлен на рисунке 4.

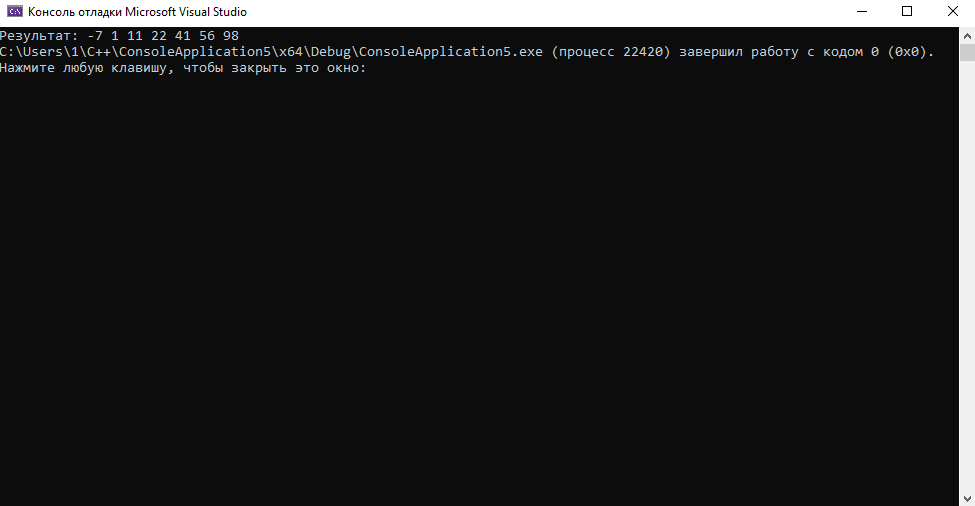


Рисунок 4

Сортировка «расчёской» работает эффективнее, чем пузырьковая, за счёт сокращения числа ненужных сравнений на ранних этапах. Несмотря на худшую временную сложность по теории (O(n²)), на практике она показывает гораздо лучшие результаты.

В обеих программах ответ на поставленную задачу оказался верным. Отличие было лишь в скорости работы программы.

# 3. Сравнительный анализ алгоритмов

Для сравнения двух методов сортировок проведем анализ на основе таких параметров как: временная сложность, потребление памяти, устойчивость, простота реализации и практическое время выполнения на больших массивах данных.

Временная сложность:

|  |  |  |
| --- | --- | --- |
| Время | Сортировка обменом (Пузырьковая) | Сортировка «расчёской» |
| Худшее время | O(n²) | O(n²) |
| Лучшее время | O(n) | O(n log n) |
| Среднее время | O(n²) | O(n² / 2p) |

Анализ сортировки обменом (пузырьковой):

– в лучшем случае, когда массив уже отсортирован, пузырьковая сортировка завершает работу за время O(n);

– в среднем и худшем случаях, когда массив неупорядоченный  
или отсортирован в обратном порядке, то алгоритм требует O(n²) сравнений  
и обменов.

Анализ сортировки «расчёской»:

– алгоритм эффективнее работает на начальных этапах за счёт больших расстояний между сравниваемыми элементами, что позволяет быстрее перенести крупные значения массива в конец;

– лучшее время – O(n log n), среднее время – O(n² / 2p), где p – это количество проходов с уменьшением gap, но на деле оно приближается к O(n²), поскольку  
при увеличении массива оно сводится к худшему времени, особенно на последних этапах сортировки, когда gap становится малым. В реальных условиях алгоритм работает быстрее пузырьковой сортировки за счёт уменьшения количества сравнений.

Потребление памяти:

Обе сортировки являются **внутренними** алгоритмами, то есть они сортируют массив **без использования дополнительной памяти**, кроме временных переменных для обмена. Затраты памяти для обоих алгоритмов минимальны и не зависят  
от размера массива.

|  |  |  |
| --- | --- | --- |
|  | Сортировка обменом (Пузырьковая) | Сортировка «расчёской» |
| Потребление памяти | O(1) | O(1) |

Устойчивость:

Устойчивость – это способность алгоритма сохранять относительный порядок элементов с одинаковыми ключами. Ключ – поле элемента, по которому происходит сортировка.

|  |  |  |
| --- | --- | --- |
|  | Сортировка обменом (Пузырьковая) | Сортировка «расчёской» |
| Устойчивость | Да | Нет |

Пузырьковая сортировка является устойчивой: если два элемента равны,  
их порядок останется прежним после сортировки.

Сортировка «расчёской» не гарантирует сохранение порядка одинаковых элементов, так как на начальных этапах происходит сравнение элементов, находящихся на большом расстоянии друг от друга, что может привести к изменению их взаимного расположения.

Простота реализации:

|  |  |  |
| --- | --- | --- |
|  | Сортировка обменом (Пузырьковая) | Сортировка «расчёской» |
| Простота реализации | Очень простая | Простая |

Практическое время выполнения на больших массивах данных:

Видоизменим Программу 1 и Программу 2. Создадим генератор случайных чисел и заполним ими массив. Для сравнения возьмем 100, 1000, 10000, 100000 элементов для двух алгоритмов сортировок. Измерим время каждого алгоритма  
и построим график зависимости времени выполнения программы от количества элементов в массиве.

Программа 3

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

//создаем неотсортированный массив

const int n = 100; //кол-во элементов массива(100, 1000, 10000, 100000)

int arr[n];

srand(time(0)); // инициализация генератора случайных чисел

// Генерация случайного массива

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

arr[i] = rand() % 20001 - 10000; // от -10000 до 10000

}

clock\_t start = clock();

//внешний цикл - кол-во проходов по массиву (i - номер прохода)

for (int i = 0; i < n - 1; i++) { //n-1 достаточное кол-во проходов для отсортировки всего массива

//внутренний цикл (j - элемент массива)

for (int j = 0; j < n - 1; j++) { //n-1 необходим, чтобы мы не вышли за границы массива

if (arr[j] > arr[j + 1]) { //проверка сравнения значений рядом стоящих парных элементов

int temp = arr[j]; //временно сохраняем значение текущего элемента

arr[j] = arr[j + 1]; //в текущую ячейку записываем значение из следующей

arr[j + 1] = temp; //возвращаем старое значение из temp

//произошла перестановка двух элементов

}

}

}

clock\_t end = clock();

double duration\_ms = double(end - start) \* 1000 / CLOCKS\_PER\_SEC;

cout << "Результат: ";

for (int i = 0; i < n; i++) //проход по отсортированному массиву

cout << arr[i] << " "; //выводим все элементы через пробел

cout << "\nВремя сортировки: " << duration\_ms << " мс.\n";

return 0;

}

Программа 4

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

//создаем неотсортированный массив

const int n = 100; //кол-во элементов массива(100, 1000, 10000, 100000)

int arr[n];

srand(time(0)); // инициализация генератора случайных чисел

// Генерация случайного массива

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

arr[i] = rand() % 20001 - 10000; // от -10000 до 10000

}

clock\_t start = clock();

int gap = n - 1; //начальный шаг - длина массива минус 1

float shrink = 1.3; //коэффициент уменьшения

bool sorted = false; //флаг для проверки сортировки

while (sorted == false) {

gap = int(gap / shrink); //уменьшаем шаг

if (gap <= 1) {

gap = 1; //последний проход

sorted = true; //предполагаем, что массив отсортирован

}

for (int i = 0; i + gap < n; i++) { //идем циклом с шагом gap

if (arr[i] > arr[i + gap]) { //проверка сравнения значений элементов с шагом gap

int temp = arr[i]; //временно сохраняем значение текущего элемента

arr[i] = arr[i + gap]; //в текущую ячейку записываем значение новой

arr[i + gap] = temp; //возвращаем старое значение из temp

//произошла перестановка двух элементов

sorted = false; //продолжаем сортировку повторно

}

}

}

clock\_t end = clock();

double duration\_ms = double(end - start) \* 1000 / CLOCKS\_PER\_SEC;

cout << "Результат: ";

for (int i = 0; i < n; i++) //проход по отсортированному массиву

cout << arr[i] << " "; //выводим все элементы через пробел

cout << "\nВремя сортировки: " << duration\_ms << " мс.\n";

return 0;

}

График

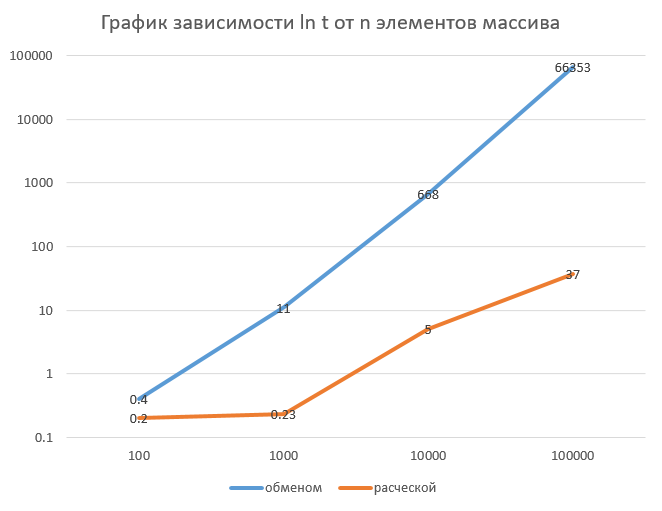


Рисунок 5

Проанализировав график зависимости (рисунок 5), можем сделать вывод,  
что сортировка «расческой» работает намного быстрее пузырьковой, причем сильный отрыв начинается уже после 100 элементов массива, чем выше – тем больше разрыв во времени работы программы.

# Заключение

В ходе выполнения курсовой работы были изучены, реализованы  
и проанализированы два алгоритма сортировки: пузырьковая сортировка (обменом) и сортировка «расчёской». Несмотря на схожесть, методы значительно различаются по эффективности работы. Пузырьковая сортировка, хотя и остаётся интуитивно понятной и широко используемой в учебных целях, на практике оказывается слишком медленной при работе с большими массивами из-за высокой временной сложности.

Пузырьковая сортировка отличается простотой реализации и наглядностью, что делает её удобной для изучения. Она широко применяется при первом знакомстве с алгоритмами сортировки благодаря своей логической прозрачности, минимальному количеству используемых конструкций и возможности легко отследить каждый шаг выполнения. Однако её высокая временная сложность (O(n²) в среднем и худшем случае) делает этот алгоритм малопригодным для сортировки больших массивов данных. При увеличении размера массива количество необходимых операций быстро возрастает, что приводит к значительным задержкам в работе программы.

Сортировка «расчёской» является усовершенствованной версией сортировки обменом. Благодаря использованию переменного шага при сравнении элементов она позволяет быстрее устранять крупные нарушения порядка в массиве и снижает количество сравнений, особенно на начальных этапах. Это достигается за счёт того, что элементы, находящиеся далеко друг от друга, могут быть сравнимы  
и перемещены ближе к нужным позициям, не дожидаясь множества проходов,  
как в пузырьковой сортировке. Такое поведение делает алгоритм более гибким  
и эффективным.

Это делает её значительно эффективнее в реальных задачах, несмотря  
на теоретически схожую сложность с пузырьковой сортировкой. На практике сортировка «расчёской» демонстрирует гораздо лучшие результаты по времени выполнения, что делает её предпочтительным выбором в ситуациях, когда важна производительность, но нет необходимости использовать более сложные алгоритмы.

Практические замеры на массивах разного размера подтвердили,  
что сортировка «расчёской» в десятки раз быстрее пузырьковой при росте объёма данных. В ходе эксперимента производилось сравнение времени выполнения программ при сортировке массива из 10 000 случайных чисел, и результаты показали явное превосходство алгоритма «расчёской» по скорости.  
Это подтверждает, что даже при использовании простых алгоритмов можно достичь высокой эффективности за счёт правильного выбора подхода.

Таким образом, при необходимости сортировки небольших массивов можно использовать оба метода, однако для более серьёзных задач предпочтительнее применять сортировку «расчёской». Кроме того, реализация обоих алгоритмов  
на языке C++ продемонстрировала их наглядность и применимость в учебных целях, а также возможность провести количественное сравнение за счёт измерения времени выполнения.

Цель работы – провести сравнительный анализ двух алгоритмов –достигнута. В процессе реализации были не только исследованы теоретические аспекты сортировок, но и произведены практические замеры, а также выведены критерии, по которым можно оценивать эффективность алгоритмов. Исследование подтвердило значительное преимущество сортировки «расчёской» по времени выполнения при сохранении простоты реализации.

# Список использованных источников

1. https://kpfu.ru/staff\_files/F1066117885/Metody\_sortirovki.pdf
2. https://bimlibik.github.io/posts/sorting-algorithm/
3. https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0\_%D0%BF%D1%83%D0%B7%D1%8B%D1%80%D1%8C%D0%BA%D0%BE%D0%BC
4. https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0\_%D1%80%D0%B0%D1%81%D1%87%D1%91%D1%81%D0%BA%D0%BE%D0%B9
5. https://en.cppreference.com/w/