Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

ОТЧЕТ
по лабораторной работе №3

Название: Алгоритмы сортировки

Дисциплина: Анализ алгоритмов

Студент ИУ7-54б _____ Ларин В.Н.
(группа) (подпись, дата) (фамилия, и.о.)

Преподаватель _____ Волкова Л.Л.
(подпись, дата) (фамилия, и.о.)

Москва, 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Алгоритмы сортировки имеют широкое практическое применение. Сортировки используются в большом спектре задач, включая обработку коммерческих, сейсмических, космических данных. Часто сортировка является просто вспомогательной операцией для упорядочивания данных, упрощения последующих алгебраических действий над данными.

Сортировка применяется во многих областях программирования, например, базы данных или математические программы. Упорядоченные объекты содержатся в телефонных книгах, ведомостях налогов, в библиотеках, в оглавлениях, в словарях.

Во многих вычислительных системах на сортировку уходит больше половины машинного времени. Исходя из этого, можно заключить, что либо сортировка имеет много важных применений, либо ею часто пользуются без нужды, либо применяются в основном неэффективные алгоритмы сортировки.

В настоящее время, в связи с экспоненциально возросшими объемами данных, вопрос эффективной сортировки данных снова стал актуальным. В настоящее время в сети Интернет можно найти результаты производительности алгоритмов сортировки для ряда ведущих центров данных.

Целью данной работы является изучение трудоемкости алгоритмов и способов ее минимизации на примере алгоритмов сортировки.

Для достижения данное цели были выделены следующие задачи:

— рассмотреть и изучить сортировки пузырьком, вставками и выбором;

— реализовать каждую сортировку;

— рассчитать их трудоемкость;

— сравнить их временные характеристики экспериментально;

— на основании проделанной работы сделать выводы.

1 Аналитический раздел

1.1 Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и если порядок в паре неверный (возрастание, в случае сортировки по убыванию, и наоборот), выполняется обмен элементов. Проходы по массиву повторяются $N - 1$ раз [1].

Также существует модифицированная версия данного алгоритма: добавляется флаг, которые показывает были ли обмены в данном цикле. Данный флаг показывает отсортирован ли массив.

1.2 Сортировка вставками

Сортировка вставками — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов [1].

Массив условно делится на две части: отсортированную и требующую сортировки. В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма

1.3 Сортировка выбором

Существует семейство сортировок, основанное на идее многократного выбора. Простейшая сортировка, основанная на следующей идее. Массив условно делится на две части: отсортированную и требующую сортировки [1]. Из правой части выбирается наименьший элемент и добавляется в конец отсортированной части с помощью обмена первого элемента не отсортированной части и минимального данной части.

1.4 Требование к ПО

Программа должна отвечать следующем требованиям:

1) На вход подается длинна массива и массив чисел разделенные пробельным символом;

2) ПО должно выводить отсортированный массив.

1.5 Вывод

В данной работе стоит задача реализации 3 алгоритмов сортировки, а именно: пузырьком, вставками и выбором. Необходимо оценить трудоемкость алгоритмов и проверить ее экспериментально.

2 Конструкторский раздел

2.1 Схемы алгоритмов

В данной части будут рассмотрены схемы алгоритмов сортировки пузырьком, вставками и выбором. На рисунках 2.1, 2.2, 2.3 представлены рассматриваемые алгоритмы.

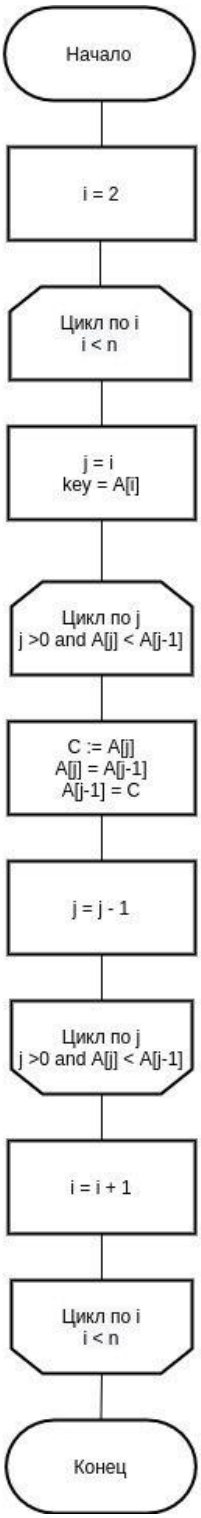

Рисунок 2.1 — Схема алгоритма сортировки пузырьком

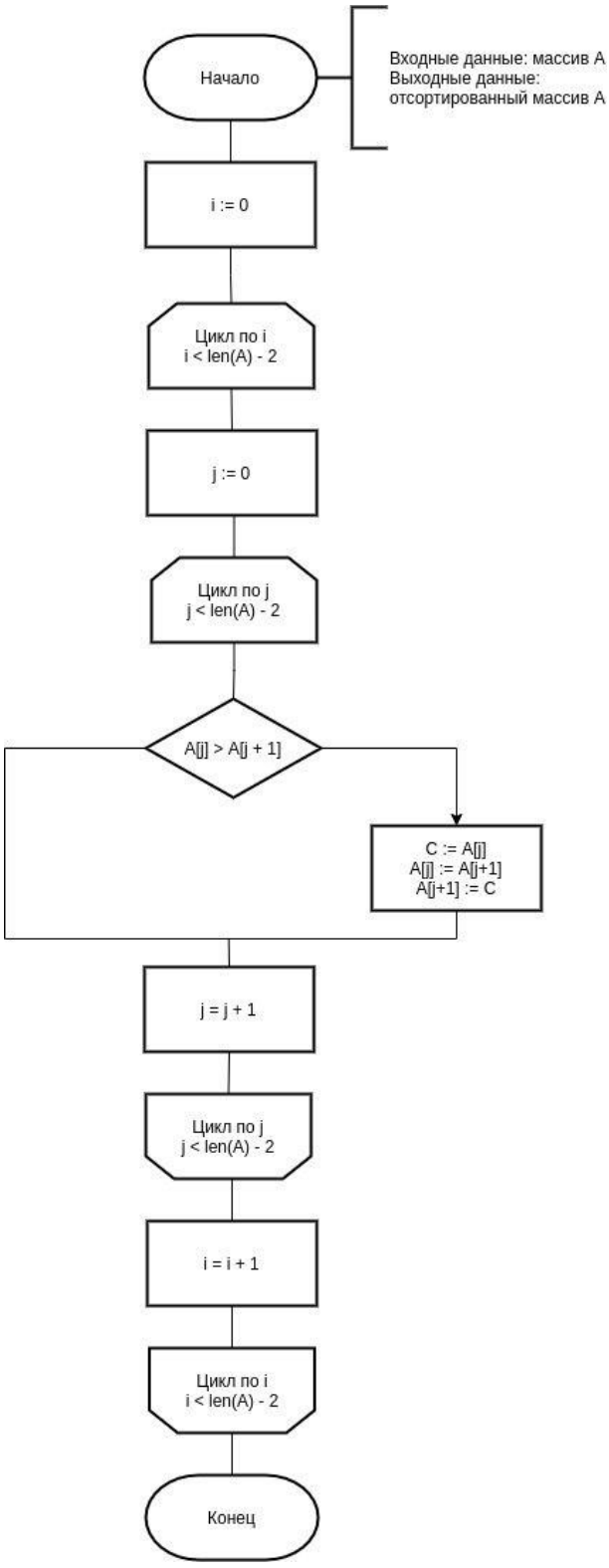

Рисунок 2.2 — Схема алгоритма сортировки вставками

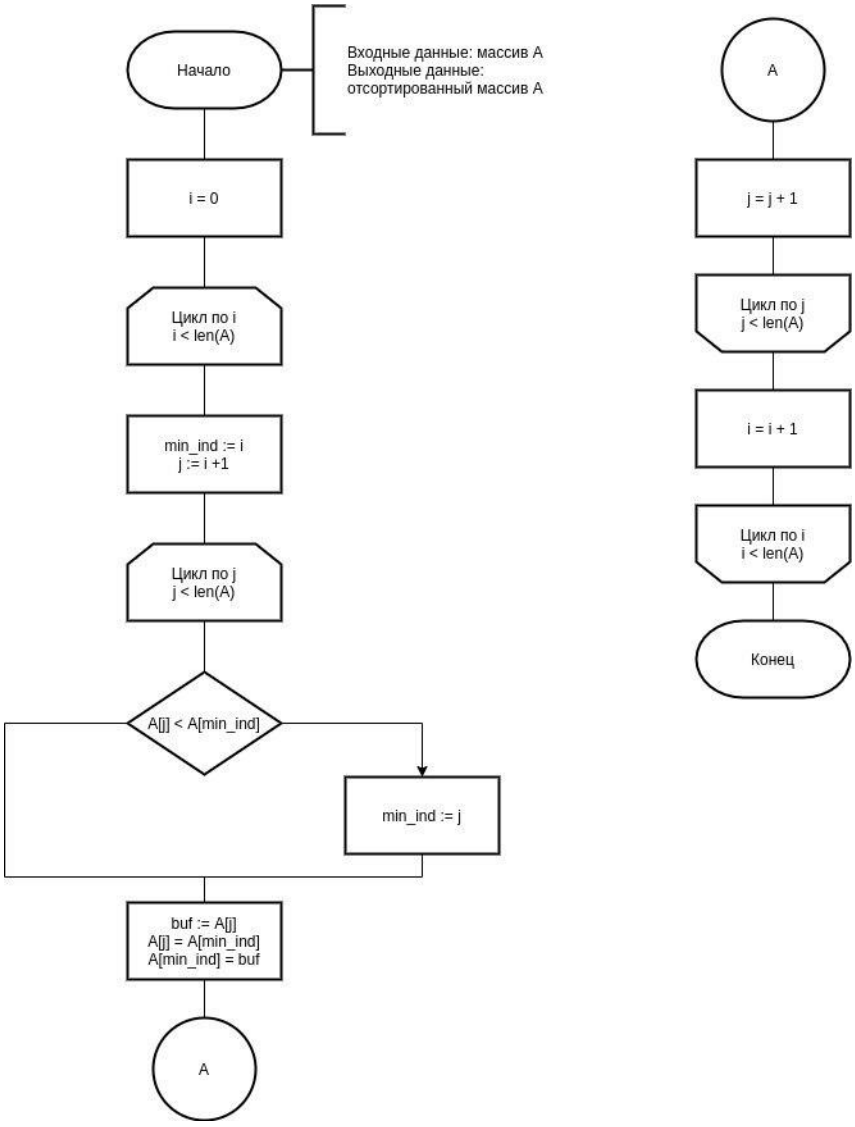

Рисунок 2.3 — Схема алгоритма сортировки выбором

2.2 Трудоёмкость алгоритмов

2.2.1 Модель вычислений

Для последующего вычисления трудоемкости необходимо ввести модель вычислений. Трудоемкость "элементарных"операций оценивается следующим образом:

1) Стоимость операций $+, -, =, <, >, \geqslant, \leqslant, ==, -=, +=, ++, --, [], \&\&, ||, >>, <<$ оценивается в 1 пункт

2) Стоимость операций $*, /, \%$ оценивается в 2 пункта

3) Трудоемкость оператора выбора if условие then A else B рассчитывается, как (2.1).

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.1}$$

4) Трудоемкость цикла рассчитывается, как (2.2).

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.2}$$

5) Трудоемкость вызова функции равна 0.

Пусть размер массивов во всех вычислениях обозначается как N.

2.2.2 Алгоритм сортировки пузырьком

Трудоёмкость алгоритма сортировки пузырьком состоит из:

— трудоёмкость сравнения и инкремента внешнего цикла $i \in [1..N)$ (2.3):

$$f_i = 2 + 2(N-1) \tag{2.3}$$

— суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N-1]$ (2.4):

$$f_j = 3(N-1) + \frac{N \cdot (N-1)}{2} \cdot (3 + f_{if}) \tag{2.4}$$

— трудоёмкость условия во внутреннем цикле (2.5):

$$f_{if} = 4 + \begin{cases} 0, & \text{в лучшем случае} \\ 9, & \text{в худшем случае} \end{cases} \tag{2.5}$$

Трудоёмкость в **лучшем** случае (2.6):

$$f_{best} = \frac{7}{2}N^2 + \frac{3}{2}N - 3 \approx \frac{7}{2}N^2 = O(N^2) \tag{2.6}$$

Трудоёмкость в **худшем** случае (2.7):

$$f_{worst} = 8N^2 - 8N - 3 \approx 8N^2 = O(N^2) \tag{2.7}$$

2.2.3 Алгоритм сортировки вставками

Трудоёмкость алгоритма сортировки пузырьком состоит из:

— трудоёмкость сравнения и инкремента внешнего цикла $i \in [1..N)$ (2.8):

$$f_i = 2 + 2(N-1) \tag{2.8}$$

— суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N-1]$ (2.9):

$$f_{if} = 4 + \begin{cases} 0, & \text{в лучшем случае} \\ 3(N-1) + \frac{N\cdot(N-1)}{2} \cdot (3 + f_{if}), & \text{в худшем случае} \end{cases} \tag{2.9}$$

— трудоёмкость условия во внутреннем цикле (2.10):

$$f_{if} = 4 + \begin{cases} 0, & \text{в лучшем случае} \\ 9, & \text{в худшем случае} \end{cases} \tag{2.10}$$

Трудоёмкость в **лучшем** случае (2.11):

$$f_{best} = 13N - 10 \approx 13N = O(N) \tag{2.11}$$

Трудоёмкость в **худшем** случае (2.12):

$$f_{worst} = 4.5N^2 + 10N - 13 \approx 4N^2 = O(N^2) \tag{2.12}$$

2.3 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы трёх алгоритмов сортировки. Оценены их трудоёмкости в лучшем и худшем случаях.

3 Технологический раздел

3.1 Средства реализации

Основным средством разработки является язык программирования. Был выбран язык программирования C++. Данный выбор обоснован высокой скоростью работы языка, поддержкой строгой типизации [2]. Для сборки проекта был выбран инструмент CMake [3]. В качестве среды разработки был выбран инструмент JetBrains Clion [4].

3.2 Реализация алгоритмов

В листинге 3.1 представлена реализация сортировки пузырьком. В листинге 3.2 представлена реализация сортировки вставками. В листинге 3.3 представлена реализация сортировки выбором.

Листинг 3.1 — Алгоритм сортировки пузырьком

```
template<class T>
void BubbleSort(std::vector<T> &vector) {
    auto size = vector.size();
    for (size_t i = 0; i < size - 1; ++i) {
        for (size_t j = i + 1; j < size; ++j) {
            if (vector[i] > vector[j]) {
                std::swap(vector[i], vector[j]);
            }
        }
    }
}
```

Листинг 3.2 — Алгоритм сортировки вставками

```
template<class T>
void InsertSort(std::vector<T> &arr) {
    T key;
    int n = arr.size();

    for (int i = 1; i < n; i++) {
        key = arr[i];
        auto j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
```

Листинг 3.3 — Алгоритм сортировки выбором

```
template<class T>
void SelectSort(std::vector<T> &vector) {
    auto size = vector.size();
    for (size_t i = 0; i < size; ++i) {
        size_t minI = i;
        for (auto j = i; j < size; ++j) {
            if (vector[j] < vector[minI]) {
                minI = j;
            }
        }
        std::swap(vector[i], vector[minI]);
    }
}
```

3.3 Тестовые данные

Для генерации тестовых данных были использованы функции представленные на листинге 3.4.

Листинг 3.4 — Генераторы тестовых данных

```
std::vector<int> randomVector(int len) {
    std::vector<int> vector;
    vector.resize(len);

    srand((unsigned) time(NULL) * getpid());
    for (int i = 0; i < len; ++i) {
        vector[i] = rand();
    }

    return vector;
}

std::vector<int> sortedVector(int len) {
    std::vector<int> vector;
    vector.reserve(len);

    for (int i = 0; i < len; ++i) {
        vector.push_back(i);
    }

    return vector;
}

auto reSortedVector = [](int len) {
```

```
25      auto vector = sortedVector(len);
26      std::reverse(vector.begin(), vector.end());
27      return vector;
28  };
```

3.4 Вывод

На основе схем из конструкторского раздела были разработаны реализации требуемых алгоритмов.

4 Экспериментальный раздел

4.1 Технические характеристики

Тестирование выполнялось на устройстве со следующими техническими характеристиками:

— Операционная система Ubuntu 21.04;

— Память 16 GiB (4,5 GiB выделено для нужд графического ядра)

— Процессор AMD® Ryzen 5 5500u with radeon graphics × 12

4.2 Время выполнения алгоритмов

Для замеров времени использовалась стандартная функция языка clock [5]. Данная функция возвращает суммарное процессорное время, использованное программой. В случае ошибки, функция возвращает значение -1. На листинге 4.1 показан способ применения данной функции при замерах.

Листинг 4.1 — Замер времени функции

```
1 auto start = clock();
2 for (int j = 0; j < counts; ++j) {
3     rec_l(left, right);
4 }
5 res.rl = double(clock() - start) / counts;
```

Результаты тестирования приведены в таблице 4.1.

Таблица 4.1 — Таблица зависимости времени работы реализаций алгоритмов от длины входных данных

Длина массива	Пузырьком	Вставками	Выбором
1	0.17	0.42	0.39
2	0.16	0.14	0.17
3	0.16	0.16	0.19
4	0.19	0.17	0.2
5	0.22	0.18	0.24
6	0.27	0.22	0.26
7	0.28	0.22	0.3
8	0.34	0.26	0.34
9	0.43	0.31	0.38
10	0.53	0.35	0.43
20	1.56	0.88	1.2
30	3.67	1.91	3.98
50	10.33	4.47	6.6
100	38.3	16.85	24.64
200	158.61	55.62	90.96
500	974.95	294.31	524.33
1000	4715.04	1558.64	2031.2

Представлены зависимости времени работы для случайного набора данных представлены на рисунках 4.1 и 4.2, для отсортированного набора данных на рисунках 4.3 и 4.4, для обратно отсортированного набора данных на рисунках 4.5 и 4.6 .

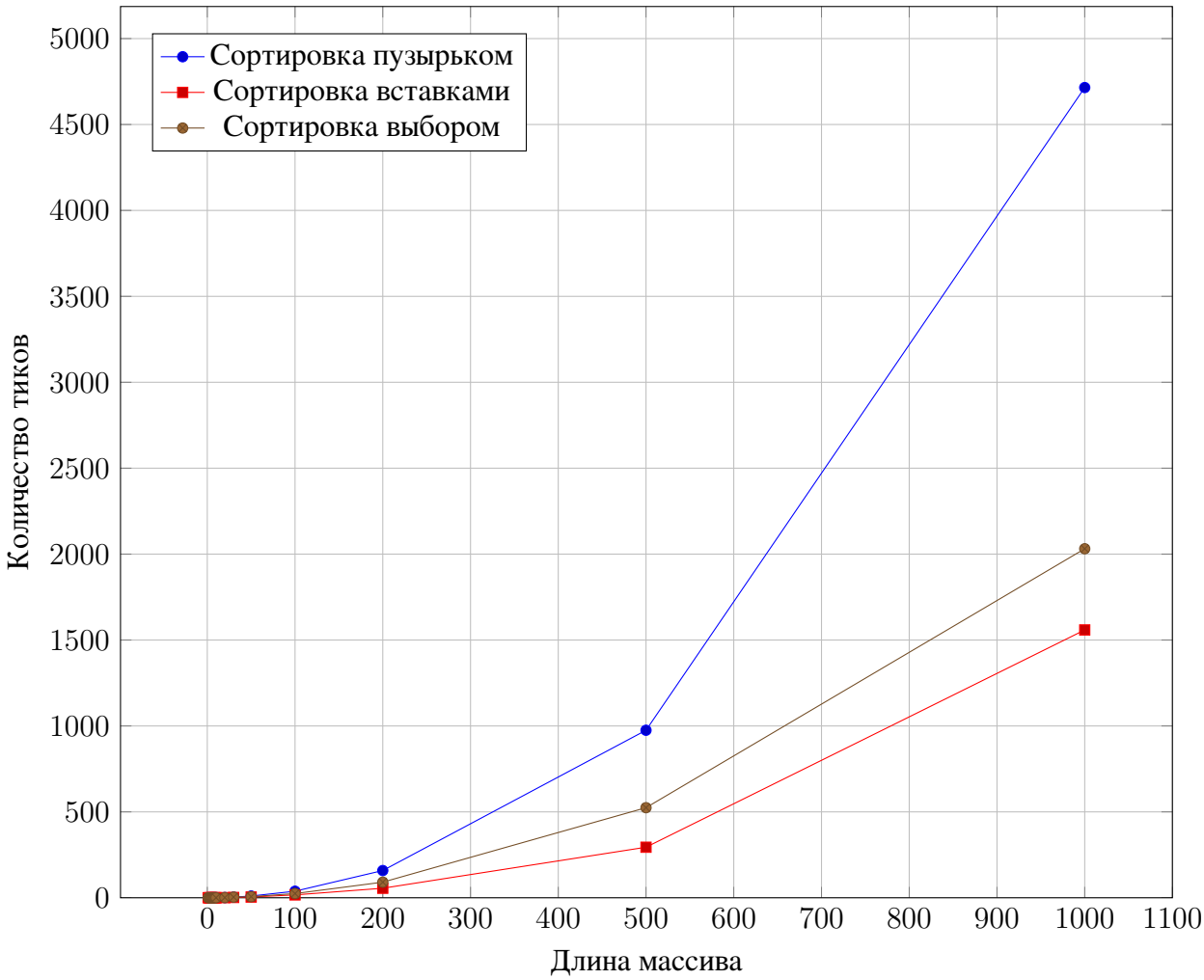

Рисунок 4.1 — Зависимость времени работы реализаций алгоритмов сортировки от длины массивов, содержащих случайный набор данных

Следует рассмотреть участок длинны массива от 1 до 10 более детально. Данная зависимость представлена на рисунке 4.2. На малых размерах длинны массива 1-2 результаты не имеет смысла, так как задачи такого рода не решаются с помощью сортировки, а с помощью простого обмена чисел.

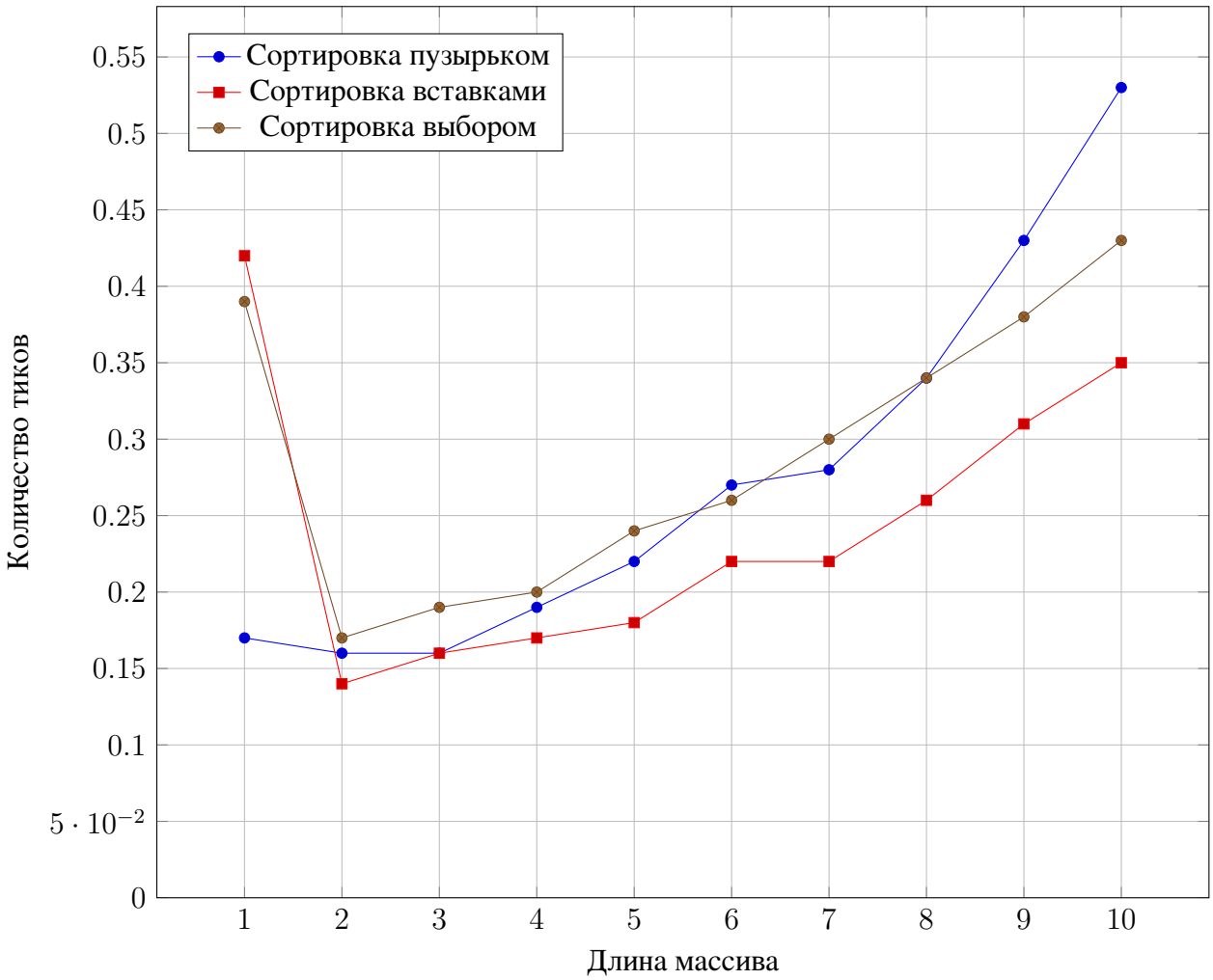

Рисунок 4.2 — Зависимость времени работы реализаций алгоритмов сортировки от длины массивов, содержащих случайный набор данных (крупно)

Представлены зависимости времени работы для отсортированного набора данных представлены на рисунках 4.3 и 4.4.

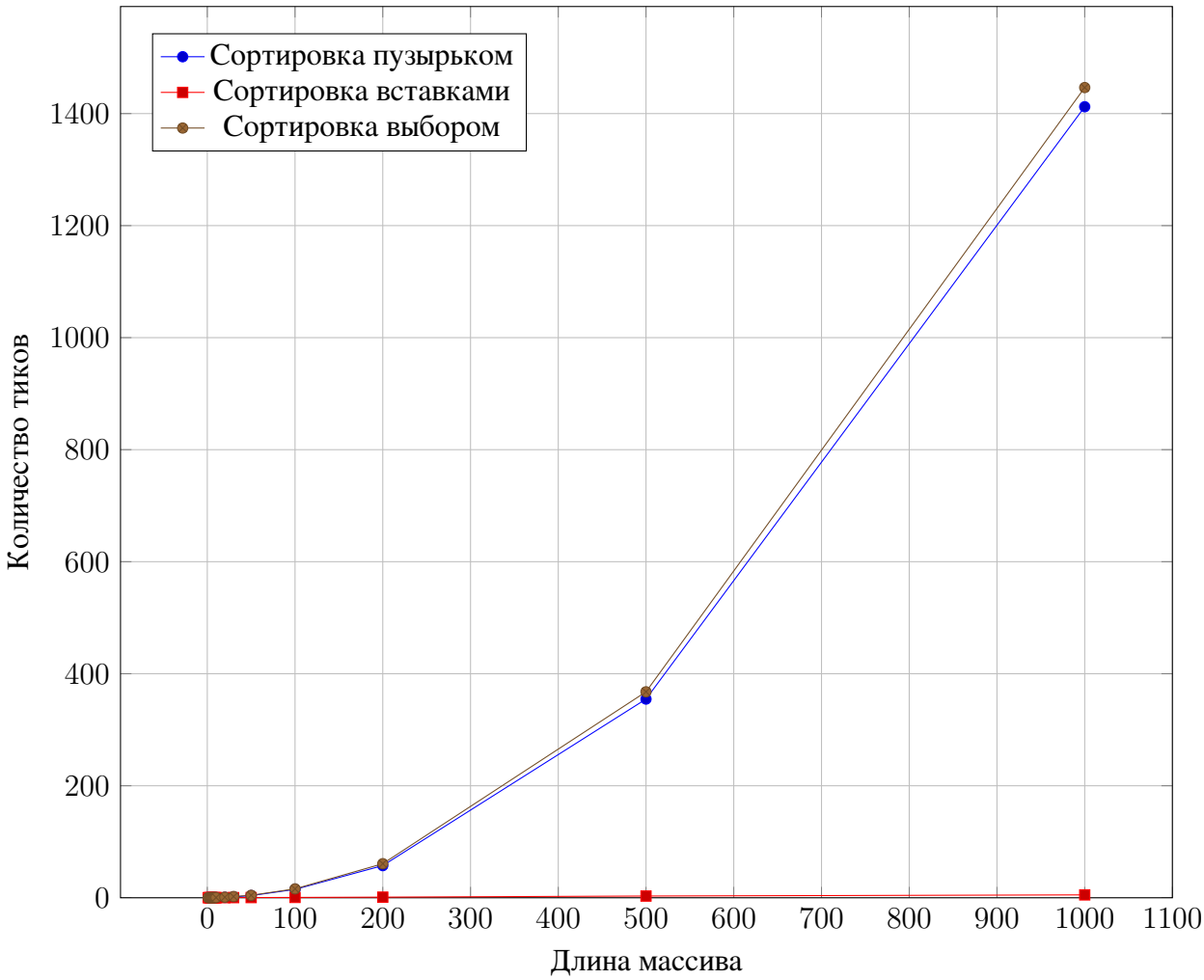

Рисунок 4.3 — Зависимость времени работы реализаций алгоритмов сортировки от длины массивов, содержащих отсортированный набор данных

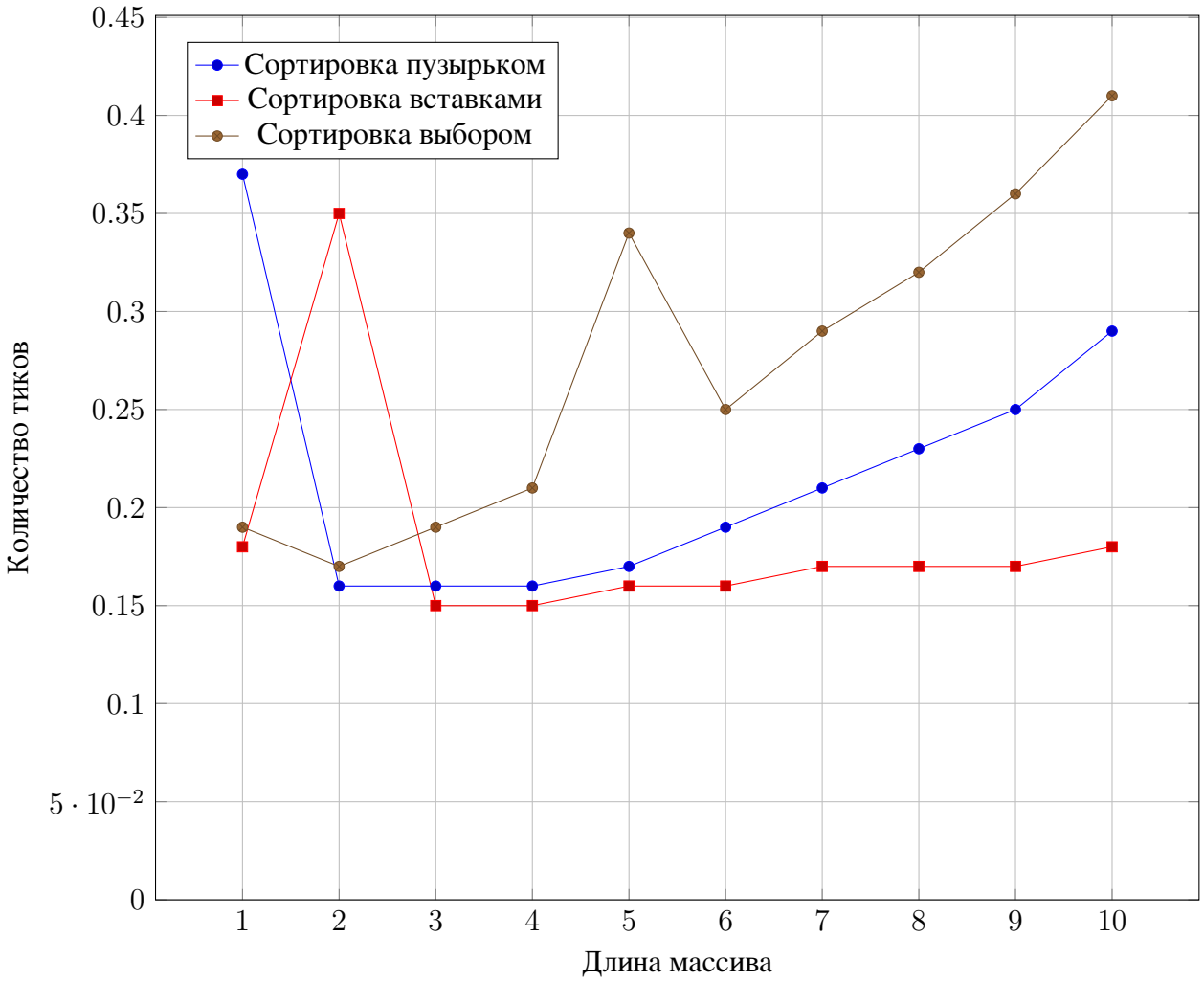

Рисунок 4.4 — Зависимость времени работы реализаций алгоритмов сортировки от длины массивов, содержащих отсортированный набор данных (крупно)

Представлены зависимости времени работы для обратно отсортированного набора данных представлены на рисунках 4.5 и 4.6.

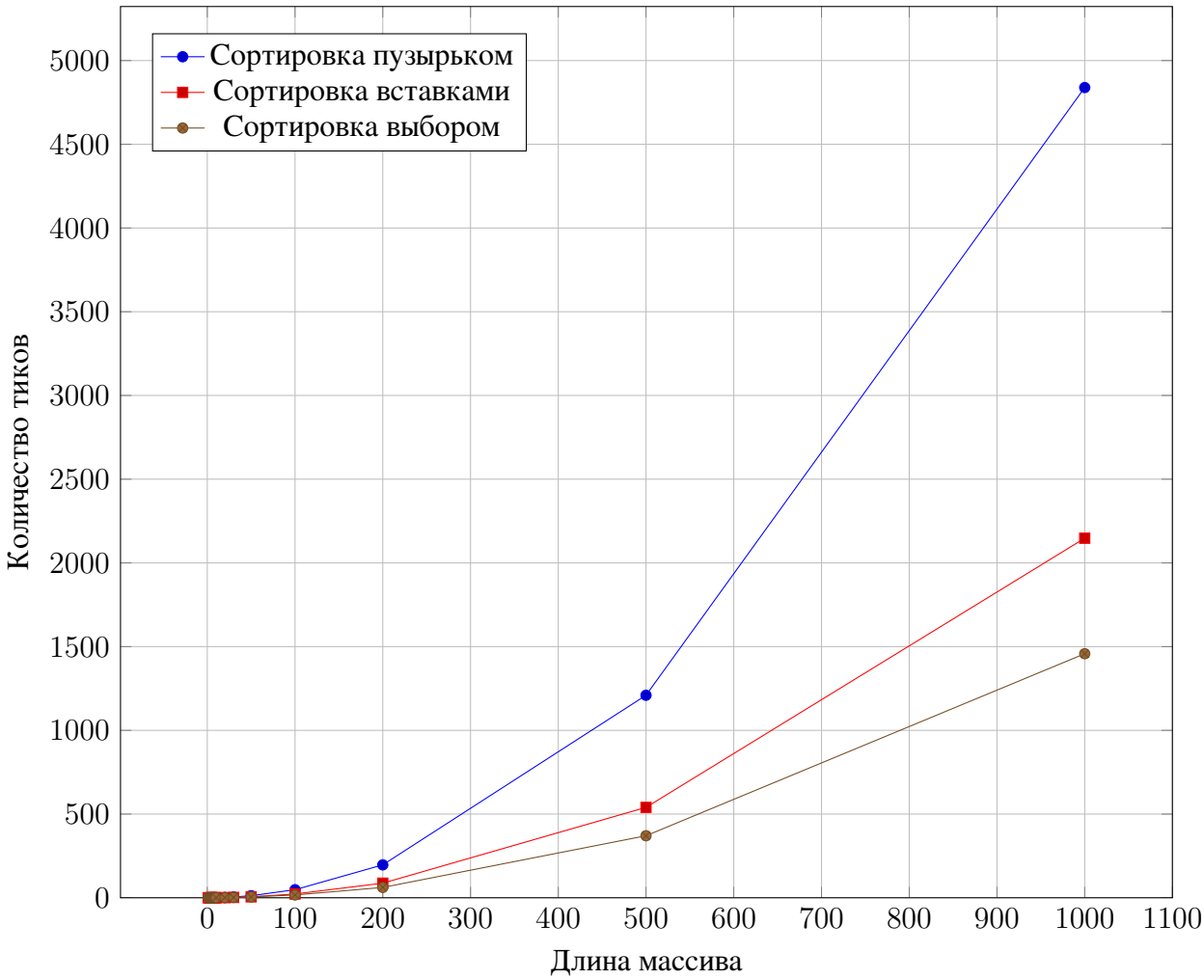

Рисунок 4.5 — Зависимость времени работы реализаций алгоритмов сортировки от длины массивов, содержащих обратно отсортированный набор данных

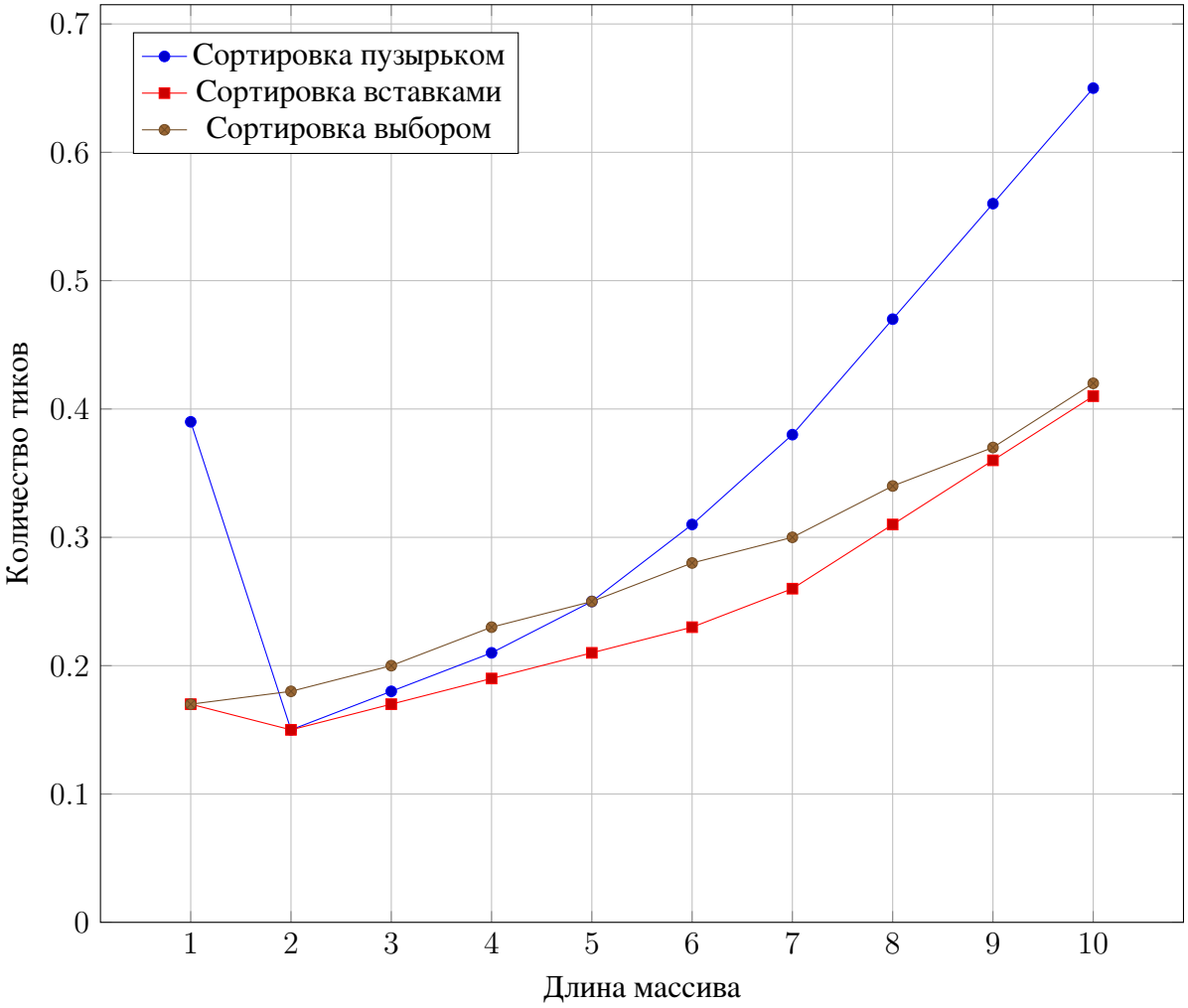

Рисунок 4.6 — Зависимость времени работы реализаций алгоритмов сортировки от длины массивов, содержащих обратно отсортированный набор данных (крупно)

4.3 Вывод

В результате исследования было установлено, что сортировка вставками работает быстрее сортировки пузырьком в 3 раза и быстрее сортировки выбором в 1,3 раза на всех наборах данных.

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе сортировки были проанализированы по их временным характеристикам, а также выделены их преимущества и недостатки. Сортировка пузырьком оказалась самой медленной из всех анализируемых. Сортировка вставками выигрывает по времени в 1,3 раза по сравнению с сортировкой выбором и в 3 раза по сравнению с сортировкой пузырьком на отсортированном, обратно отсортированном и случайном наборе данных.

СПИСОК ЛИТЕРАТУРЫ

1. Кнут Д. Э. Искусство программирования. Сортировка и поиск. — Издательский дом Вильямс, 2000. — Т. 3.

2. Страуструп Б. Программирование. Принципы и практика использования C++. — ООО ИД Вильямс, 2011. — ISBN: 9785457707351. — Режим доступа: https://books.google.ru/books?id=kttwBgAAQBAJ.

3. CMake [Электронный ресурс]. — Режим доступа: https://cmake.org/ (дата обращения: 10.10.2021).

4. CLion: A Cross-Platform IDE for C and C++ by JetBrains [Электронный ресурс]. — Режим доступа: https://www.jetbrains.com/clion/ (дата обращения: 10.10.2021).

5. Clock: Интерактивная система просмотра системных руководств [Электронный ресурс]. — Режим доступа: https://www.opennet.ru/man.shtml?topic=clock&category=3&russian=0 (дата обращения: 10.10.2021).