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»

Отчёт

по лабораторной работе № 3

Название: Трудоёмкость алгоритмов сортировки

Дисциплина: Анализ алгоритмов

Студент ИУ7-55Б _____ М.А. Козлов
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподователь _____ Л.Л. Волкова
(Подпись, дата) (И.О. Фамилия)

Москва, 2020

Содержание

Введение

Алгоритм сортировки – это алгоритм для упорядочивания элементов в списке. Входом является последовательность из n элементов: $a_1, a_2, \ldots, a_n$ Результатом работы алгоритма сортировки является перестановка исходной последовательности $a'_1, a'_2, \ldots, a'_n$, такая что $a_1 \leqslant a_2 \leqslant \cdots \leqslant a_n$, где $\leqslant$ – отношение порядка на множестве элементов списка. Поля, служащие критерием порядка, называются ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма.

В данной работе рассматриваются три алгоритма:

1) сортировка пузырьком с флагом;
2) сортировка вставками;
3) сортировка выбором.

Целью данной лабораторной работы является реализация алгоритмов сортировки и исследование их трудоемкости.

Задачи данной лабораторной работы:

1) изучить алгоритмы сортировки пузырьком с флагом, вставками, выбором;
2) реализовать алгоритмы сортировки пузырьком с флагом, вставками, выбором;
3) дать оценку трудоёмкости в лучшем, произвольном и худшем случае (для двух алгоритмов сделать вывод трудоёмкости);
4) провести замеры процессорного времени работы для лучшего, худшего и произвольного случая.

1 Аналитический раздел

В данном разделе будут рассмотрены основные теоритические понятия алгоритмов сортировок пузырьком с флагом, вставками, выбором.

1.1 Алгоритмы сортировок

1.1.1 Алгоритм сортировки пузырьком с флагом

Алгоритм сортировки пузырьком или метод простых обменов имеет следующий принцип работы:

1) прохождение по всему массиву;
2) сравнивание между собой пар соседних ячеек;
3) если при сравнении оказывается, что значение ячейки i больше, чем значение ячейки i + 1, то нужно поменять значения этих ячеек местами.

Алгоритм сортировки пузырьком с флагом является модификацией этого алгоритма. Идея состоит в том, что если при выполнении прохода методом пузырька не было ни одного обмена элементов массива, то это означает, что массив уже отсортирован и остальные проходы не требуются.

1.1.2 Алгоритм сортировки вставками

Алгоритм сортировки вставками, просматривает элементы входной последовательности по одному, и для каждого элемента, размещает его в подходящее место среди ранее упорядоченных элементов.

В начальный момент времени отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается в нужную позицию в отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан.

Сортировка методом вставок – простой алгоритм сортировки. Хотя этот метод сортировки намного менее эффективен, чем более сложные алгоритмы (такие как быстрая сортировка), у него есть ряд преимуществ:

1) простота реализации;
2) эффективен на небольших наборах данных;
3) эффективен на частично отсортированных последовательностях;
4) является устойчивым алгоритмом (не меняет порядок элементов, которые уже отсортированы).

Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места вставки текущего элемента в отсортированную часть последовательности.

1.1.3 Алгоритм сортировки выбором

Алгоритм сортировки выбором работает следующим образом: находим наименьший элемент в массиве и обмениваем его с элементом находящимся на первом месте. Повторяем процесс – находим наименьший элемент в последовательности, начиная со второго элемента, и обмениваем со вторым элементном и так далее, пока весь массив не будет отсортирован. Этот метод называется сортировка выбором, поскольку он работает, циклически выбирая наименьший из оставшихся элементов.

Главным отличием сортировки выбором от сортировки вставками является, то что в сортировке вставками извлекается из неотсортированной части массива первый элемент (не обязательно минимальный) и вставляется на своё место в отсортированной части. В отличии от сортировки выбором, где ищется минимальный элемент в неотсортированной части, который вставляется в конец отсортированной части массива.

1.2 Трудоёмкость алгоритма

Трудоёмкость – количество работы, которую алгоритм затрачивает на обработку данных. Является функцией от длины входов алгоритма и позволяет оценить количество работы.

Введём модель вычисления трудоёмкости.

1.2.1 Базовые операции

Ниже представлены базовые операции, стоимость которых единична:

1) $=, +, + =, -, - =, *, * =, /, / =, ++, --, \%$,
2) $<, \leqslant, ==, \neq, \geqslant, >$,
3) [ ].

1.2.2 Условный оператор

```
if (условие) {
// тело A
}
else {
// тело B
}
```

Пусть трудоёмкость тела A равна f_A, а тела B f_B, тогда стоимость условного оператора можно найти по формуле (1.1):

$$f_{if} = f_{\text{условия}} + \begin{cases} min(f_A, f_B) - \text{лучший случай}, \\ max(f_A, f_B) - \text{худший случай} \end{cases} \tag{1.1}$$

1.2.3 Цикл со счётчиком

```
for (int i = 0; i < n; i++) {
// тело цикла
}
```

Начальная инициализация цикла (int i = 0) выполняется один раз. Условие i < n проверяется перед каждой итерацией цикла и при входе в цикл – n + 1 операций. Тело цикла выполняется ровно n раз. Счётчик (i++) выполняется на каждой итерации, перед проверкой условия, т.е. n раз. Тогда, если трудоёмкость тела цикла равна f, трудоёмкость всего цикла определяется формулой (1.2)

$$f_{\text{цикла}} = 2 + n(2 + f) \tag{1.2}$$

2 Конструкторский раздел

В данном разделе будут рассмотрены схемы алгоритмов, требования к функциональности ПО, и опредены способы тестирования.

2.1 Разработка алгоритмов

Ниже будут представлены схемы алгоритмов сортировки:

1) алгоритм сортировки пузырьком с флагом (рисунок 2.1);
2) алгоритм сортировки вставками (рисунок 2.2);
3) алгоритм сортировки выбором (рисунок 2.3).

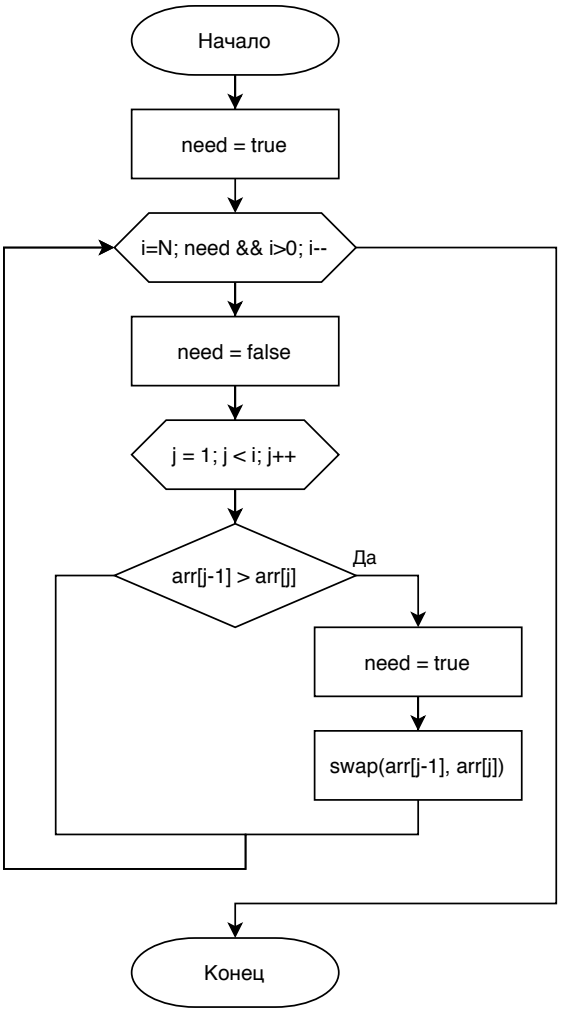

Рисунок 2.1 — Схема алгоритма сортировки пузырьком с флагом

2.2 Требования к функциональности ПО

В данной работе требуется обеспечить следующую минимальную функциональность консольного приложения:

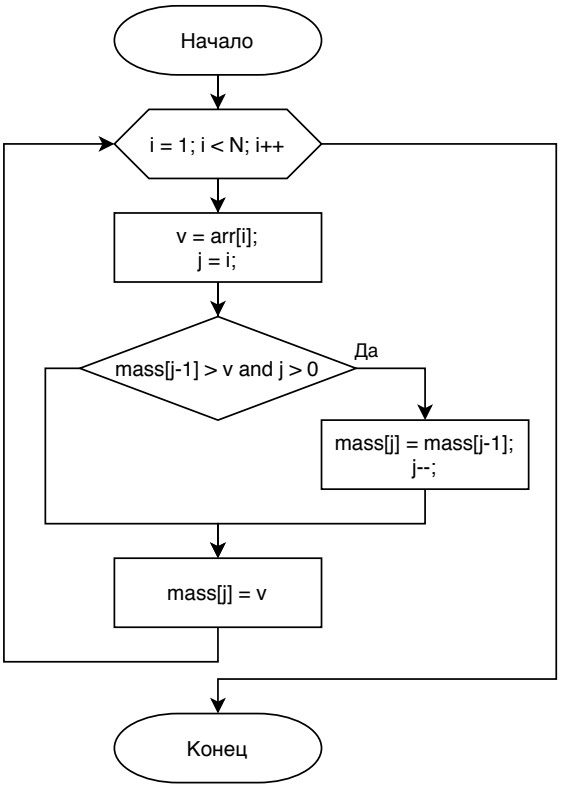

Рисунок 2.2 — Схема алгоритма сортировки вставками

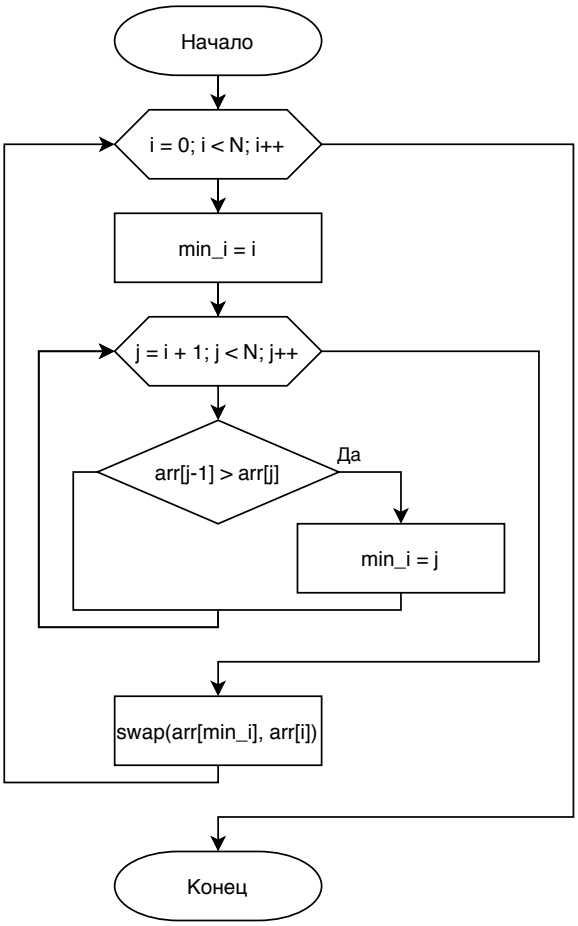

Рисунок 2.3 — Схема алгоритма сортировки выбором

1) предоставить возможность ввода массива, на выходе пользователь должен получить результат сортировки массива, произведенной тремя алгоритмами;

2) обеспечить вывод замеров времени работы каждого из алгоритмов в худшем, лучшем и произвольном случаях.

2.3 Тесты

Тестирование ПО будет проводиться методом чёрного ящика. Необходимо проверить работу системы на тривиальных случаях: список является пустым или содержит один элемент, и несколько нетривальных случаев: список отсортирован по возрастанию / по убыванию, случайный список.

3 Технологический раздел

В данном разделе будут выбраны средства реализации ПО и представлен листинг кода.

3.1 Средства реализации

В данной работе используется язык программирования python, так как он позволяет написать программу в относительно малый срок. В качестве среды разработки использовалась Visual Studio Code. [1][2]

Для замера процессорного времени была использована функция process_time модуля time. [3] Она возвращает значение в долях секунды суммы системного и пользовательского процессорного времени текущего процесса и не включает время, прошедшее во время сна.

3.2 Листинг программы

Ниже представлены листинги кода алгоритмов сортировки:

1) пузырьком с флагом (листинг 3.1);
2) вставками (листинг 3.2);
3) выбором (листинг 3.3).

Листинг 3.1 — Реализация алгоритма сортировки пузырьком с флагом

```
def bubble_sort(mass, cmp = lambda a, b: a - b):
    len_mass= len(mass)
    for i in range(len_mass, 0, -1):
        need = False
        for j in range(1, i):
            if cmp(mass[j-1], mass[j]) > 0:
                mass[j-1], mass[j] = mass[j], mass[j-1]
                need = True
        if not need: break
    return mass
```

Листинг 3.2 — Реализация алгоритма сортировки вставками

```
def insertion_sort(mass, cmp = lambda a, b: a - b):
    len_mass = len(mass)
    for i in range(1, len_mass):
        v = mass[i]
        j = i
        while cmp(mass[j-1], v) > 0 and j > 0:
            mass[j] = mass[j-1]
            j -= 1
        mass[j] = v
    return mass
```

Листинг 3.3 — Реализация алгоритма сортировки выбором

```
def selection_sort(mass, cmp = lambda a, b: a - b):
    len_mass = len(mass)
    for i in range(len_mass):
        min_i = i
        for j in range(i+1, len_mass):
            if cmp(mass[j], mass[min_i]) < 0:
                min_i = j
        mass[min_i], mass[i] = mass[i], mass[min_i]
    return mass
```

3.3 Тестирование

В таблице 3.1 отображён возможный набор тестов для тестирования методом чёрного ящика, результаты которого, представленные на рисунке 3.1, подтверждают прохождение программы перечисленных тестов.

Таблица 3.1 — Тесты для проверки корректности программы

Массив	Ожидаемый результат
[ ]	[ ]
[1]	[1]
[1, 2, 3]	[1, 2, 3]
[3, 2, 1]	[1, 2, 3]
[1, 212, -5, 1, 64, -75]	[-75, -5, 1, 1, 64, 212]

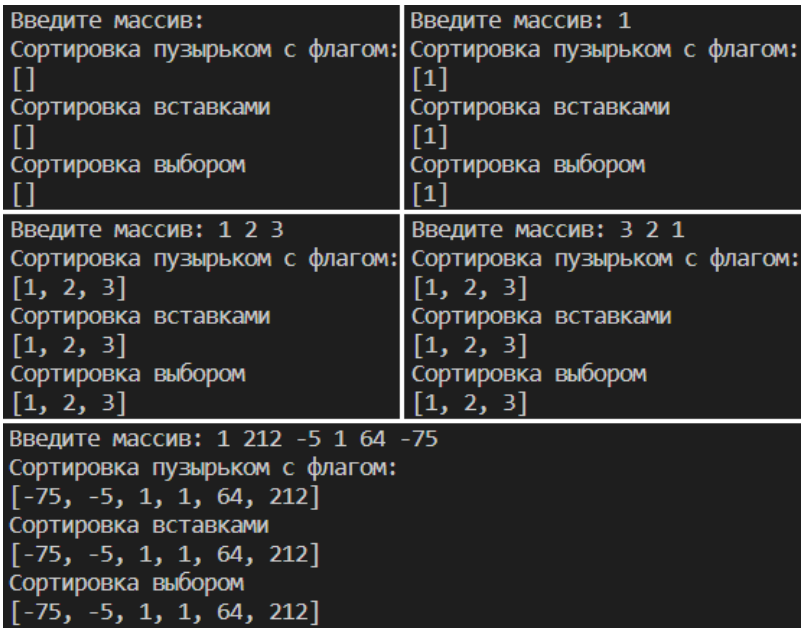

Рисунок 3.1 — Результаты тестирования алгоритмов.

4 Экспериментальный раздел

В данном разделе будут проведены эксперименты для проведения сравнительного анализа трёх алгоритмов по затрачиваемому процессорному времени в зависимости от длины массива и степени его отсортированности.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

В рамках данного проекта были проведёны следующие эксперименты:

1) сравнение времени работы алгоритмов в лучшем случаи (графики 4.1 и 4.2);
2) сравнение времени работы алгоритмов в худшем случаи (график 4.3);
3) сравнение времени работы алгоритмов в произвольном случаи (график 4.4).

Для лучшего случая массивы заполняются в порядке возрастания числами от -1000 до 1000, для худшего случая – в порядке убывания от -1000 до 1000, для произвольного случая – псевдослучайные числами в диапазоне от -1000 до 1000. Для построения графиков указанные массивы генерировались размерами от 1 до 10000.

Тестирование проводилось на ноутбуке с процессором Intel(R) Core(TM) i5-7200U CPU 2.50 GHz [4] под управлением Windows 10 с 8 Гб оперативной памяти.

В ходе экспериментов по замеру времени работы было установлено, что в лучшем случае, когда массив отсортирован, сортировка выбором оказалась самой медленной. Алгоритм сортировки вставками оказался медленнее сортировки вставками. Сортировка пузырьком с флагом в лучшем случае работает быстрее всего.

В худшем случае самой медленной сортировкой является сортировка пузырьком с флагом, а сортировка выбором является самой быстрой (более, чем в 2 раза быстрее сортировки с флагом).

В произвольном случае время работы сортировок вставками и выбором сопоставимо. Самой медленной является сортировка пузырьком с флагом, на длине массива 10000 она работает в ≈ 1.75 раз медленнее, чем другие сортировки.

4.2 Оценка трудоёмкости алгоритмов сортировки

Во всех теоритических оценках трудоёмкости алгоритмов сортировки предполагается, что трудоёмкость сравнения $f_{cmp} = 2$.

4.2.1 Алгоритм сортировки пузырьком с флагом

Найдём трудоёмкость алгоритм сортировки пузырьком с флагом.

Лучший случай – массив отсортирован; не произошло ни одного обмена за 1 проход, следовательно выходим из цикла, тогда трудоёмкость определяется по следующей формуле:

$$f = 1 + 2 + (2 + 1 + 2 + (N - 1)(2 + 3) + 2) = 5N + 8 = O(N) \tag{4.1}$$

Худший случай – массив отсортирован в обратном порядке; На каждой итерации происходил обмен, тогда трудоёмкость определяется по следующей формуле:

$$f = 1 + 2 + \sum_{i=1}^{N}(2 + 1 + 2 + (N - i)(2 + 4) + 1) = 3N^2 + 3N + 3 = O(N^2) \tag{4.2}$$

4.2.2 Алгоритм сортировки вставками

Найдём трудоёмкость алгоритма сортировки вставками.

Лучший случай – массив отсортирован. При этом все внутренние циклы состоят всего из одной итерации, тогда трудоёмкость определяется по следующей формуле:

$$f = 1 + 2 + (N - 1)(2 + 3 + 2 + 2 + 1) = 10N - 7 = O(N) \tag{4.3}$$

Худший случай – массив отсортирован в обратном порядке. Каждый новый элемент сравнивается со всеми в отсортированной последовательности. Все внутренние циклы будут состоять из j итераций, тогда трудоёмкость определяется по следующей формуле:

$$f = 1 + 2 + (N - 1)(2 + 3 + \sum_{i=1}^{N-1}(2 + 2 + 1 + 4 + 1) + 1) = 10N^2 - 14N + 7 = O(N^2) \tag{4.4}$$

4.2.3 Алгоритм сортировки выбором

Трудоёмкость сортировки выбором в худшем и лучшем случаях совпадает и оценивается как $O(N^2)$.

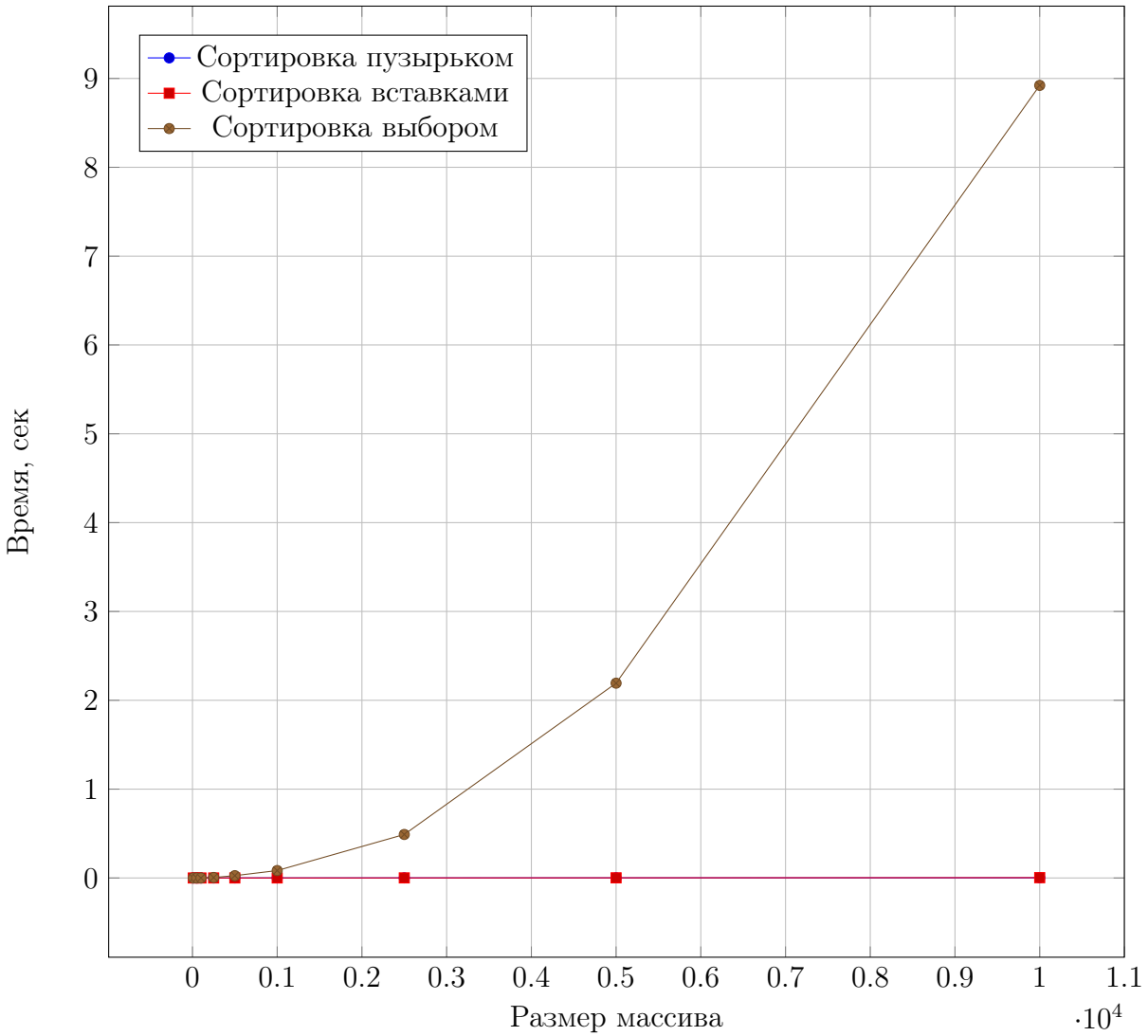

Рисунок 4.1 — График зависимости времени работы реализации алгоритмов сортировки в лучшем случае

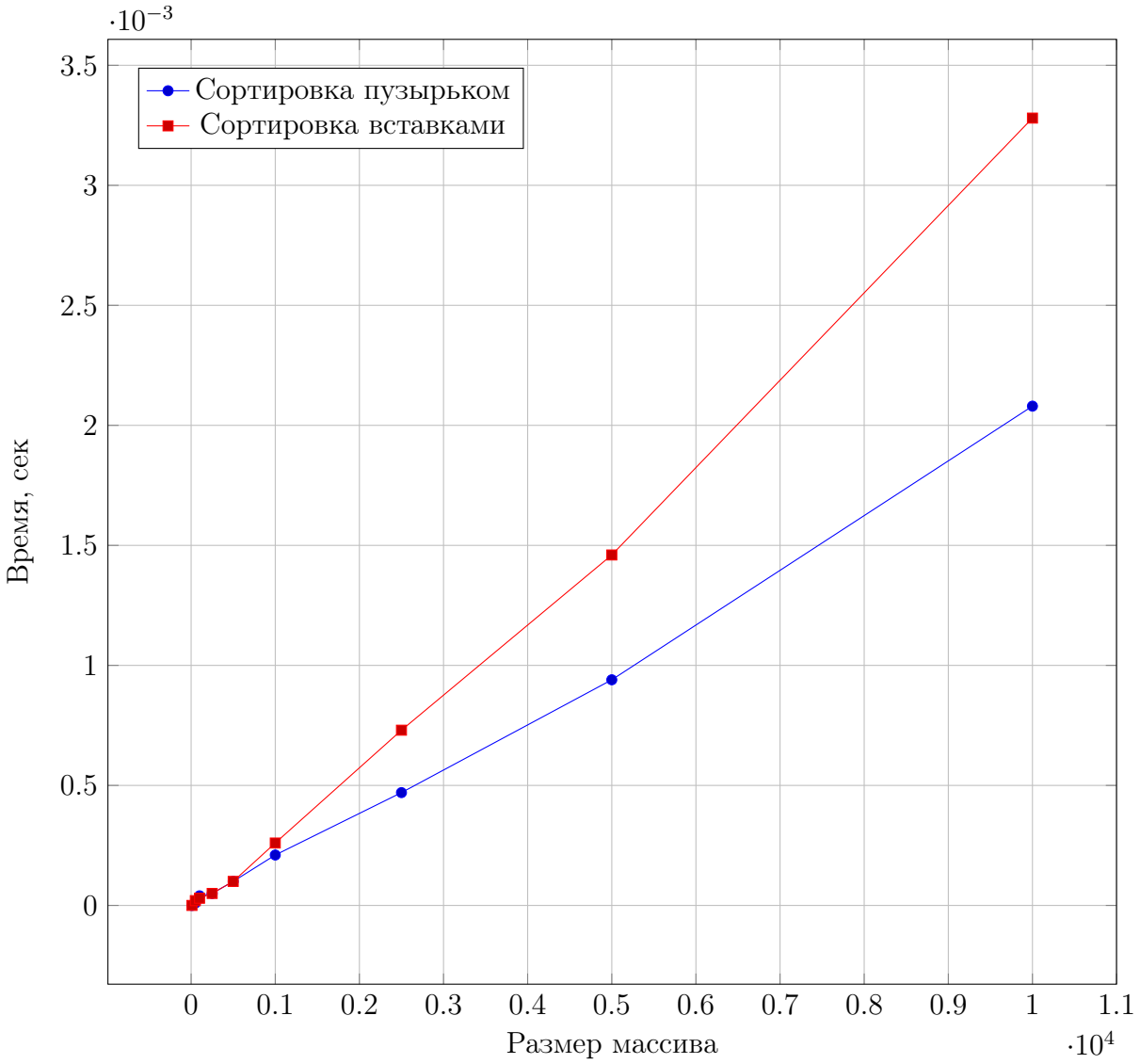

Рисунок 4.2 — График зависимости времени работы реализации алгоритмов сортировки пузырьком и вставками в лучшем случае

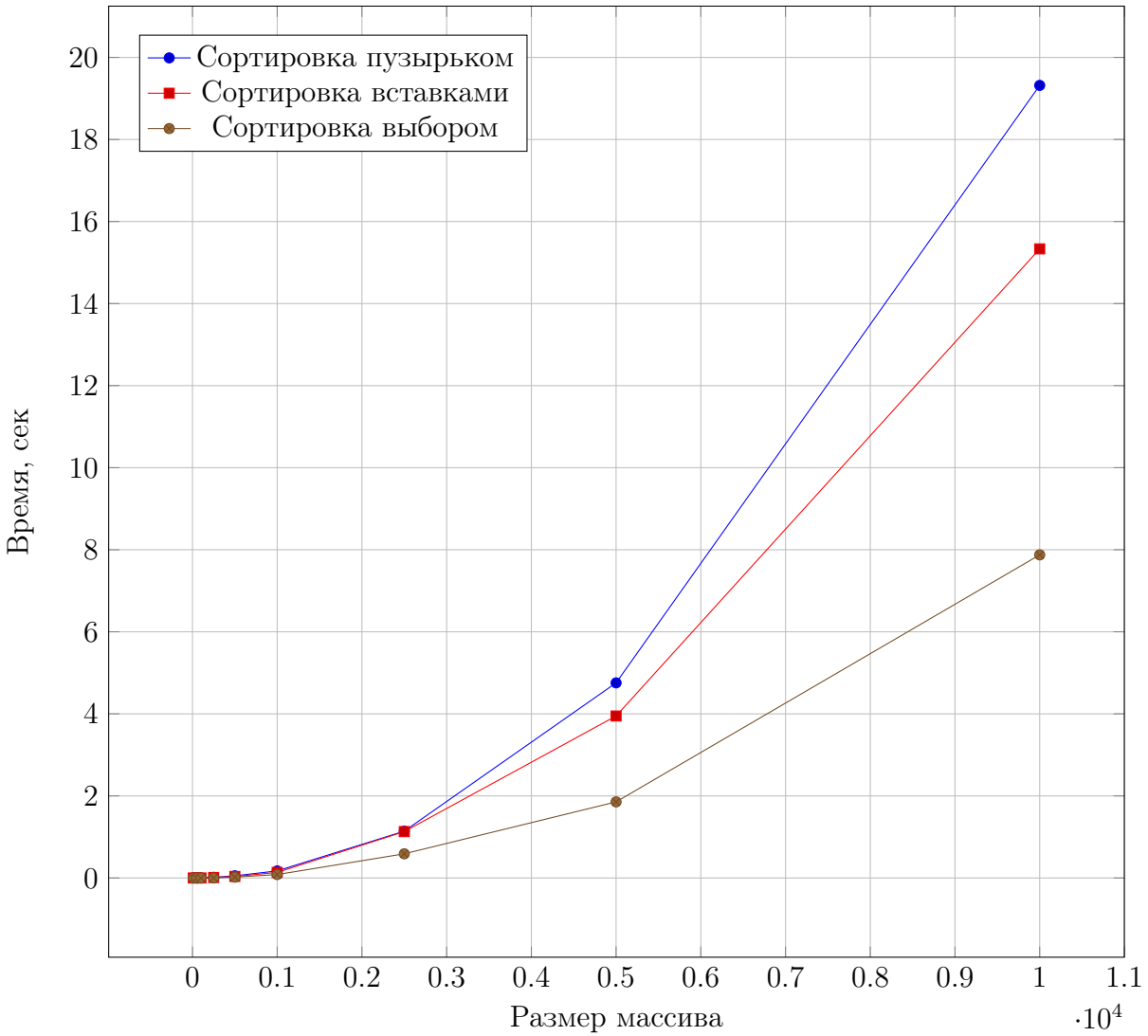

Рисунок 4.3 — График зависимости времени работы реализации алгоритмов сортировки в худшем случае

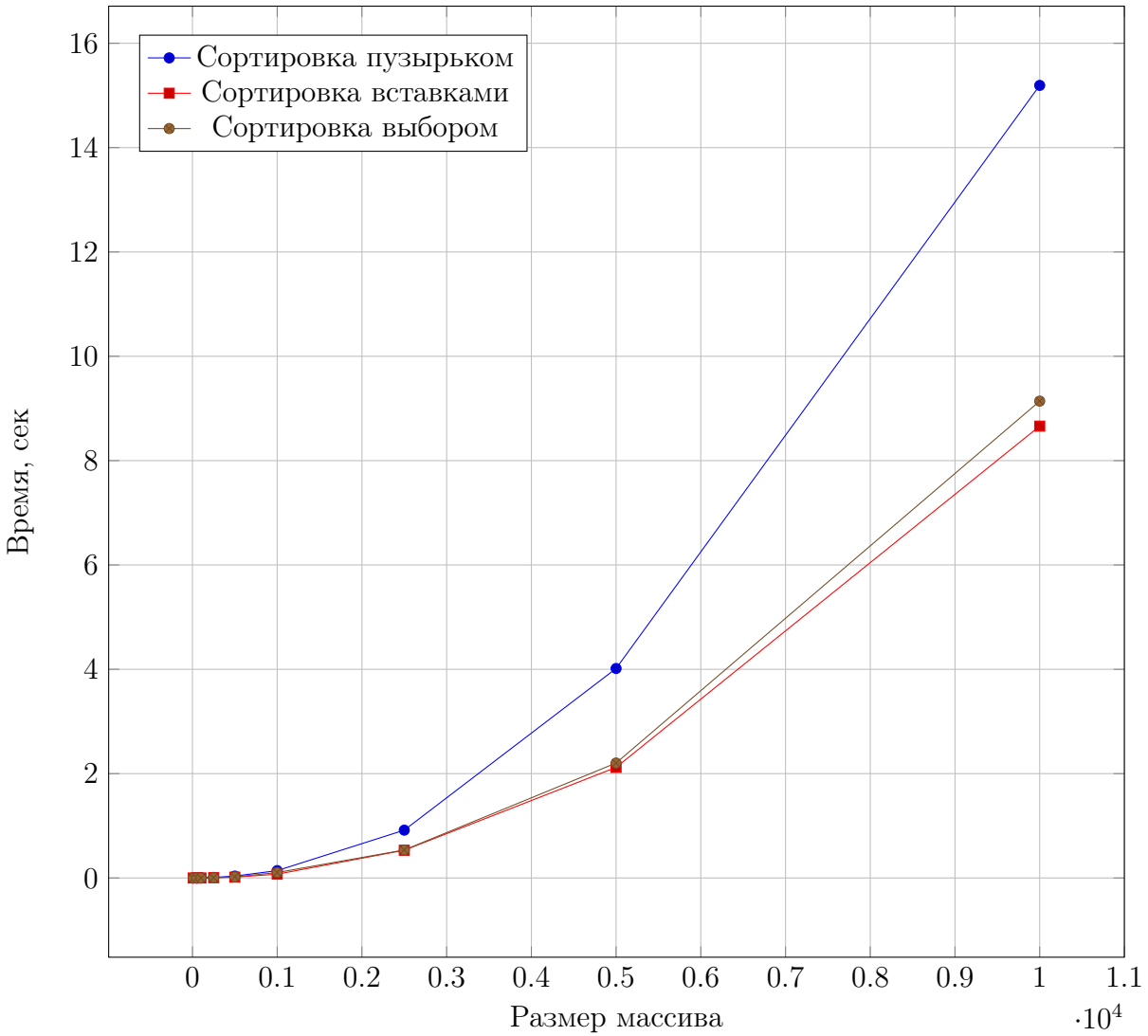

Рисунок 4.4 — График зависимости времени работы реализации алгоритмов сортировки в произвольном случае

Заключение

В ходе работы были изучены и реализованы алгоритмы сортировки (пузырьком с флагом, вставки и выбором).

В ходе экспериментов по замеру времени работы было установлено, что в лучшем случае, когда массив отсортирован, сортировка выбором оказалась самой медленной. Алгоритм сортировки вставками оказался медленнее сортировки вставками. Сортировка пузырьком с флагом в лучшем случае работает быстрее всего.

В худшем случае самой медленной сортировкой является сортировка пузырьком с флагом, а сортировка выбором является самой быстрой (более, чем в 2 раза быстрее сортировки с флагом).

В произвольном случае время работы сортировок вставками и выбором сопоставимо. Самой медленной является сортировка пузырьком с флагом, на длине массива 10000 она работает в ≈ 1.75 раз медленнее, чем другие сортировки.

Список использованных источников

1. Python. // [Электронный ресурс]. Режим доступа: https://www.python.org/, (дата обращения: 01.10.2020).

2. Visual Studio Code - Code Editing. // [Электронный ресурс]. Режим доступа: https://code.visualstudio.com, (дата обращения: 01.10.2020).

3. Process time. // [Электронный ресурс]. Режим доступа: https://docs-python.ru/standart-library/modul-time-python/funktsija-process-time-modulja-time, (дата обращения: 01.10.2020).

4. Intel® Core™ i5-7200U Processor. // [Электронный ресурс]. Режим доступа: https://www.intel.com/content/www/us/en/products/processors/core/i5-processors/i5-7200u.html, (дата обращения: 26.09.2020).