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 по дисциплине "Анализ алгоритмов"

Тема Алгоритмы сортировки

Студент Жабин Д.В.

Группа ИУ7-54Б

Преподаватель Волкова Л.Л.

Москва, 2021 г.

Оглавление

Введение

Сортировка — это упорядочивание элементов заданной последовательности. Работать с отсортированным списком гораздо удобнее, ведь так в разы быстрее можно найти нужный объект.

Алгоритм сортировки — это алгоритм для упорядочивания элементов в массиве. В случае, когда элемент в массиве имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки. На практике в качестве ключа часто выступает число, а в остальных полях хранятся какие-либо данные, никак не влияющие на работу алгоритма. Почти в каждом программном продукте используется один или даже несколько алгоритмов сортировки.

Любой алгоритм сортировки состоит из сравнений и перестановок, количество которых определяет скорость работы алгоритма. Этот параметр является ключевым при выборе того или иного алгоритма для решения конкретной поставленной задачи.

Целью лабораторной работы является анализ алгоритмов сортировки. Для ее достижения поставлены следующие задачи:

- изучить и реализовать 3 алгоритма сортировки: сортировка выбором, сортировка вставками, сортировка пузырьком;
- провести сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- провести сравнительный анализ алгоритмов на основе экспериментальных данных.

1 Аналитическая часть

Рассмотрим ключевые особенности выбранных для анализа сортировок.

1.1 Сортировка 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 элементом и так далее, пока весь массив не будет отсортирован. Сортировка получила такое название, поскольку 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 (необязательно минимальный) и вставляется на своё место в отсортированной части. В сортировке выбором при этом ищется минимальный элемент в неотсортированной части, который вставляется в конец отсортированной части массива.

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.

Считается, что последовательность из одного элемента — отсортированная. Начиная со второго элемента входных данных, алгоритм помещает его на нужную позицию в уже отсортированную часть. Так до тех пор, пока весь набор входных данных не будет исчерпан [4]. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

1.3 Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов.

Проходы по массиву повторяются $N - 1$ раз. При каждом проходе алгоритма по внутреннему циклу очередной наибольший элемент массива ставится на свое место в конце массива рядом с предыдущим “наибольшим элементом”, а наименьший элемент массива перемещается на одну позицию к началу массива (“всплывает” до нужной позиции, как пузырёк в воде — отсюда и название алгоритма) [4].

1.4 Вывод

В данном разделе были рассмотрены 3 алгоритма сортировки: сортировка выбором, сортировка вставками и сортировка пузырьком.

2 Конструкторская часть

На основе полученных аналитических данных построим схемы алгоритмов сортировок и оценим их трудоемкости.

2.1 Схемы алгоритмов

На рисунках 2.1, 2.2 и 2.3 представлены схемы алгоритмов сортировки выбором, сортировки вставками и сортировки пузырьком соответственно.

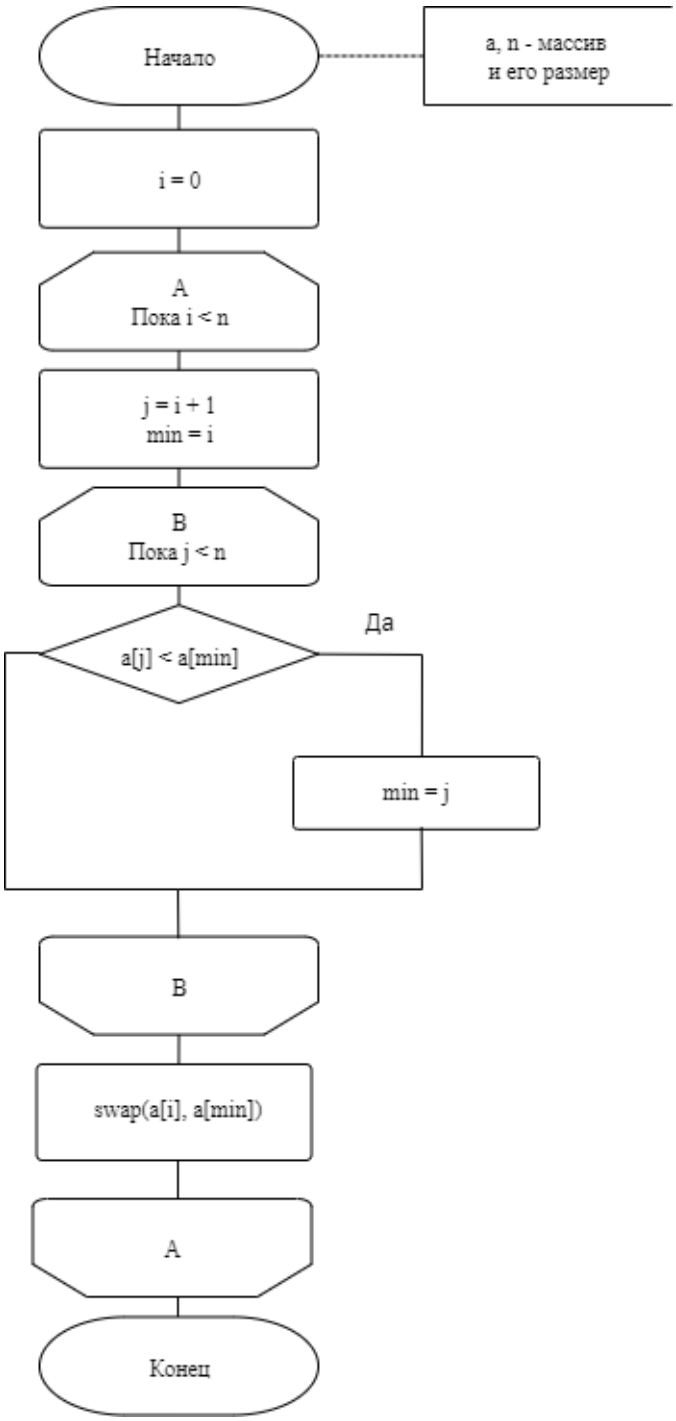

Рисунок 2.1 — Сортировка выбором

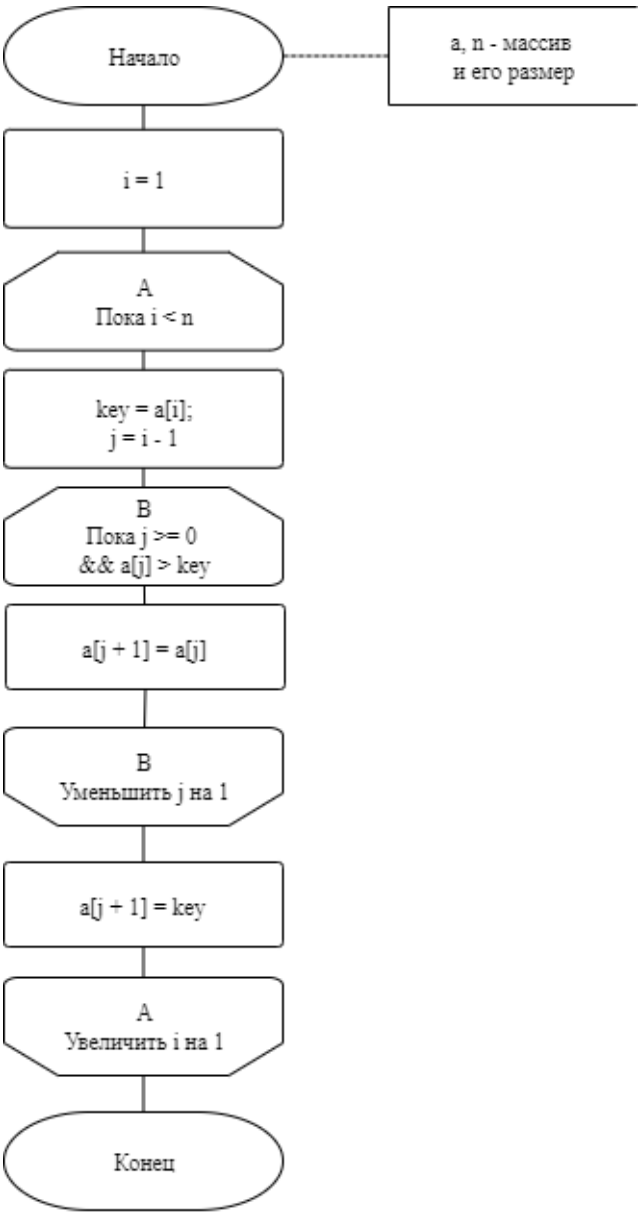

Рисунок 2.2 — Сортировка вставками

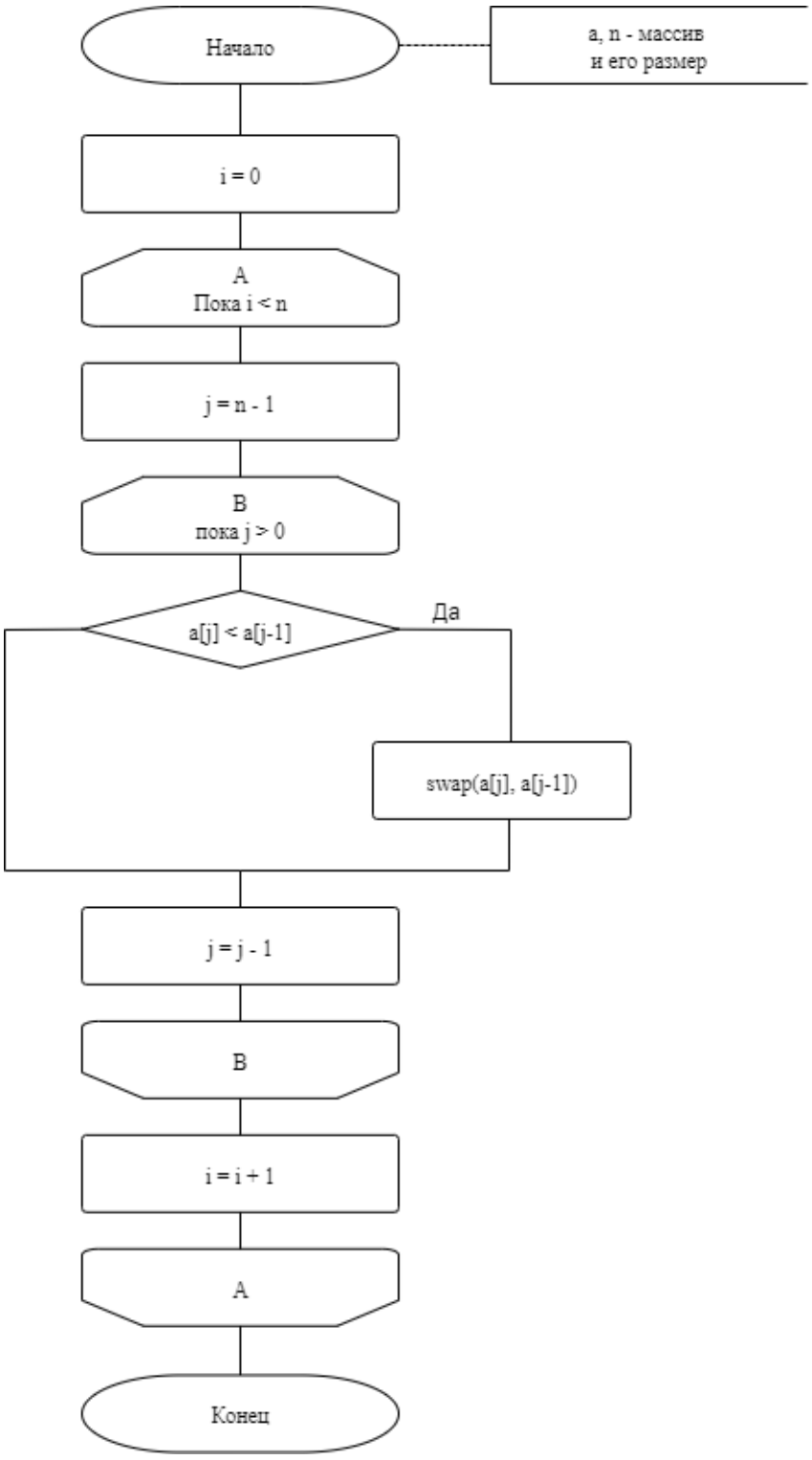

Рисунок 2.3 — Сортировка пузырьком

2.2 Модель вычислений трудоемкости алгоритмов

1. Трудоемкость базовых операций.

   Трудоемкость операций из списка (2.1) равна 2.

$$*, /, \%, * =, / = \tag{2.1}$$

Операции из списка (2.2) имеют трудоемкость 1.

$$=, +, -, + =, - =, ==, ! =, <, >, <=, >=, ||, \&, [] \tag{2.2}$$

2. Трудоемкость условного оператора.

   Трудоемкость условного перехода равна 0, а оператора if условие then A else B рассчитывается, как (2.3) в лучшем случае и как (2.4) в худшем случае.

$$f_{\text{if}} = f_{\text{условия}} + min(f_{\text{A}}, f_{\text{B}}) \tag{2.3}$$

$$f_{\text{if}} = f_{\text{условия}} + max(f_{\text{A}}, f_{\text{B}}) \tag{2.4}$$

3. Трудоемкость цикла.

   Она рассчитывается, как (2.4).

$$f_{\text{for}} = f_{\text{инициализации}} + f_{\text{условия}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.4}$$

4. Трудоемкость вызова функции равна 0.

2.3 Трудоемкость алгоритмов

Пусть размер массивов во всех вычислениях обозначается как N.

2.3.1 Алгоритм сортировки пузырьком

Трудоёмкость алгоритма сортировки пузырьком:

- трудоёмкость внешнего цикла $i \in [0..N-1)$:

$$f_{\text{i}} = 3 + (N-1) \cdot (4 + f_{\text{j}}) \tag{2.5}$$

- трудоёмкость внутреннего цикла $j \in [0..N-1-i)$:

$$f_{\text{j}} = 4 + (N-1) \cdot (5 + f\text{if}) \tag{2.6}$$

- трудоёмкость условия во внутреннем цикле в лучшем случае (2.7) и в худшем случае (2.8):

$$f_{\text{ifbest}} = 4 + 0 = 4 \tag{2.7}$$

$$f_{\text{ifworst}} = 4 + 9 = 13 \tag{2.8}$$

Трудоёмкость в **лучшем** случае (2.9):

$$f_{\text{best}} = 9N^2 - 10N + 4 \approx 9N^2 = O(N^2) \tag{2.9}$$

Трудоёмкость в **худшем** случае (2.10):

$$f_{\text{worst}} = 18N^2 - 28N + 13 \approx 18N^2 = O(N^2) \tag{2.10}$$

2.3.2 Алгоритм сортировки вставками

Трудоёмкость алгоритма сортировки вставками:

- трудоёмкость внешнего цикла $i \in [1..N)$:

$$f_{\text{i}} = 2 + (N-1) \cdot (5 + f_{\text{j}}) \tag{2.11}$$

- трудоёмкость внутреннего цикла $j >= 0$ в лучшем случае (2.12) и в худшем случае (2.13):

$$f_{\text{jbest}} = 5 + 0 = 5 \tag{2.12}$$

$$f_{\text{jworst}} = 5 + 5 \cdot (N-1) \tag{2.13}$$

Трудоёмкость в **лучшем** случае (2.14):

$$f_{\text{best}} = 10N - 8 \approx 10N = O(N) \tag{2.14}$$

Трудоёмкость в **худшем** случае (2.15):

$$f_{\text{worst}} = 5N^2 - 3 \approx 5N^2 = O(N^2) \tag{2.15}$$

2.3.3 Алгоритм сортировки выбором

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

2.4 Вывод

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

3 Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

3.1 Требование к ПО

К программе предъявляется ряд требований:

- на вход ПО получает массив сравнимых элементов;
- на выходе — этот массив, отсортированный по возрастанию.

3.2 Средства реализации

Для реализации ПО был выбран язык программирования `Python` [1]. Это обусловлено знанием возможностей языка, что обеспечит высокую скорость написания программы без потери ее качества.

В качестве среды разработки выбрана `Visual Studio Code` [3]. Удобства написания кода и его автодополнения стали ключевыми при выборе.

3.3 Реализация алгоритмов

В листингах 3.1 - 3.3 приведена реализация трёх алгоритмов сортировки.

Листинг 3.1 — Сортировка выбором

```
def selection_sort(a):
    for i in range(len(a)):
        min_i = i
        for j in range(i+1, len(a)):
            if a[j] < a[min_i]:
                min_i = j
        a[min_i], a[i] = a[i], a[min_i]
    return a
```

Листинг 3.2 — Сортировка вставками

```
def insertion_sort(a):
    for i in range(1, len(a)):
        key = a[i]
        j = i-1
        while j >= 0 and a[j] > key:
            a[j+1] = a[j]
            j -= 1
        a[j+1] = key
    return a
```

Листинг 3.3 — Сортировка пузырьком

```
def bubble_sort(a):
    for i in range(len(a)):
        for j in range(len(a)-1, 0, -1):
            if a[j-1] > a[j]:
                a[j-1], a[j] = a[j], a[j-1]
    return a
```

3.4 Тестовые данные

В таблице (3.1) приведены тесты для функций, реализующих алгоритмы сортировки. Все тесты пройдены успешно.

Таблица 3.1 — Тестирование функций

Входной массив	Результат	Ожидаемый результат
[37, 44, 45, 52, 88]	[37, 44, 45, 52, 88]	[37, 44, 45, 52, 88]
[88, 52, 45, 44, 37]	[37, 44, 45, 52, 88]	[37, 44, 45, 52, 88]
[−10, −20, −30, −25, −50]	[−50, −30, −25, −20, −10]	[−50, −30, −25, −20, −10]
[40, −10, 20, −30, 75]	[−30, −10, 20, 40, 75]	[−30, −10, 20, 40, 75]
[10]	[10]	[10]
[−10]	[−10]	[−10]
[]	[]	[]

3.5 Вывод

В данном разделе были разработаны исходные коды трёх алгоритмов: сортировки выбором, сортировки вставками и сортировки пузырьком.

4 Исследовательская часть

Исследуем быстродействие сортировок на практических тестах.

4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- Операционная система: Windows 10 64-bit.
- Оперативная память: 16 GB.
- Процессор: Intel(R) Core(TM) i5-4690 @ 3.50GHz.

4.2 Время выполнения алгоритмов

Время выполнения алгоритмов замерялось с помощью специальной функции `process_time()` [2] из модуля `time`, которая возвращает значение в долях секунды процессорного времени текущего процесса. Контрольная точка возвращаемого значения не определена, поэтому допустима только разница между результатами последовательных вызовов.

На рисунках 4.1 - 4.3 и на графиках 4.1 - 4.3 показаны результаты замеров.

Упорядоченный массив:

Размер	Пузырьком	Вставками	Выбором
100	0.00000	0.00000	0.00156
250	0.00313	0.00000	0.00156
500	0.01562	0.00000	0.00781
1000	0.07812	0.00000	0.02813

Рисунок 4.1 — Зависимость времени выполнения алгоритмов при сортированных массивах (в секундах)

Массив, упорядоченный в обратном порядке:

Размер	Пузырьком	Вставками	Выбором
100	0.00156	0.00000	0.00000
250	0.00781	0.00469	0.00156
500	0.03438	0.01406	0.00781
1000	0.14219	0.06406	0.02969

Рисунок 4.2 — Зависимость времени выполнения алгоритмов при обратно сортированных массивах (в секундах)

Случайный массив:

Размер	Пузырьком	Вставками	Выбором
100	0.00000	0.00156	0.00000
250	0.00469	0.00313	0.00156
500	0.02344	0.00937	0.00625
1000	0.10938	0.03594	0.02813

Рисунок 4.3 — Зависимость времени выполнения алгоритмов при случайных массивах (в секундах)

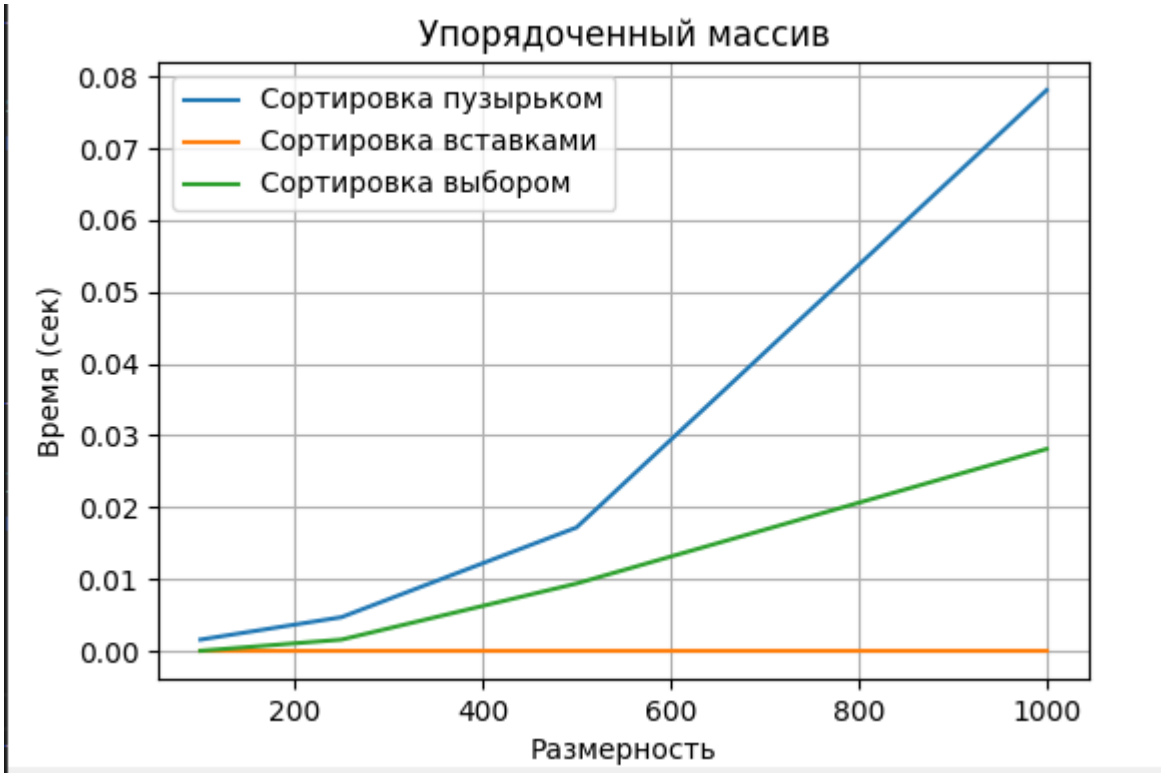

График 4.1 — Зависимость времени выполнения от размерности при сортированных массивах

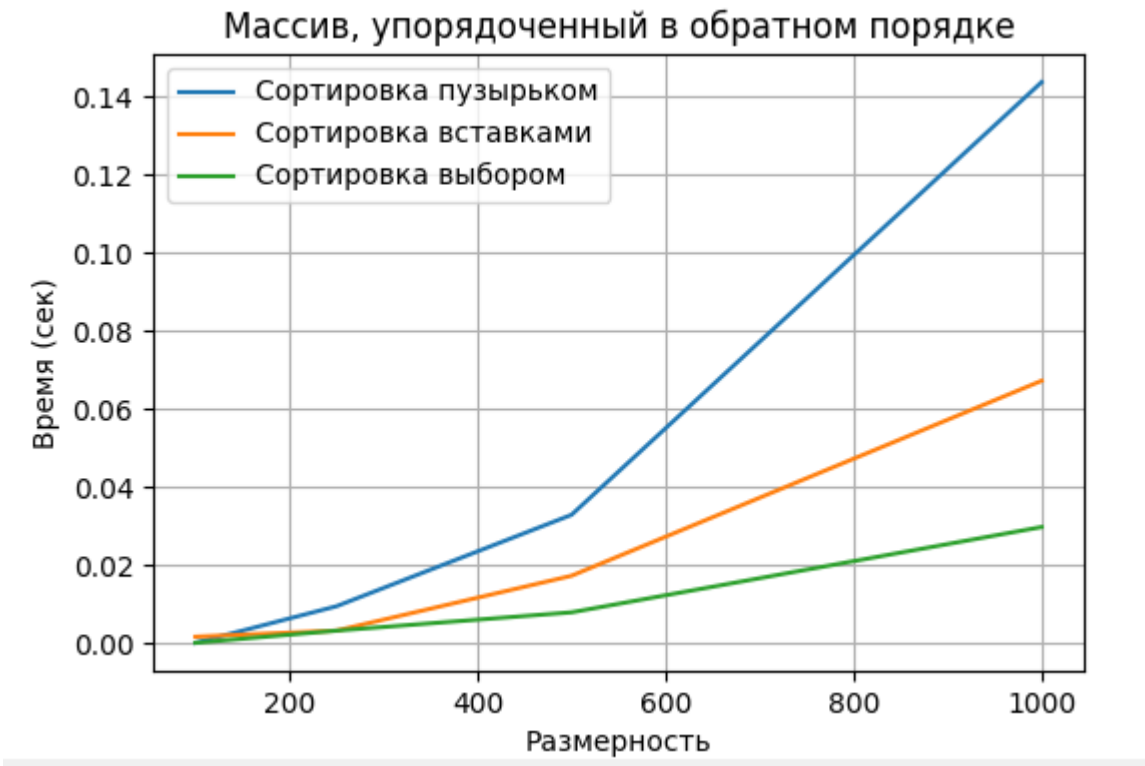

График 4.2 — Зависимость времени выполнения от размерности при обратно сортированных массивах

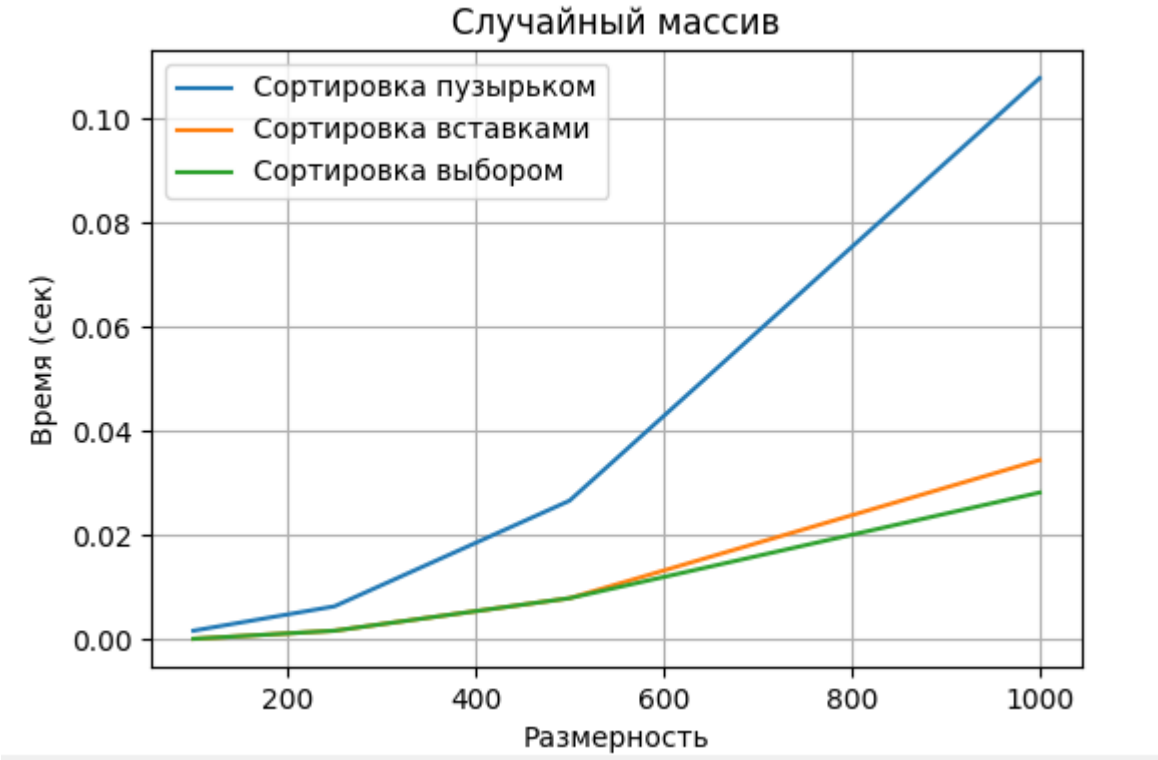

График 4.3 — Зависимость времени выполнения от размерности при случайных массивах

4.3 Вывод

По результатам замеров сортировка выбором работает быстрее остальных рассмотренных сортировок, но на отсортированных массивах выигрывает сортировка вставками. Сортировка пузырьком работает значительно медленнее на любых массивах.

Заключение

В ходе проделанной работы была достигнута поставленная цель и решены следующие задачи:

- были изучены и реализованы 3 алгоритма сортировки: сортировка выбором, сортировка вставками, сортировка пузырьком;
- был проведён сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- был проведён сравнительный анализ алгоритмов на основе экспериментальных данных.

На основании анализа трудоемкости алгоритмов в выбранной модели вычислений, было показано, что алгоритм сортировки вставками имеет трудоемкость меньше, чем другие сортировки в лучшем случае. В худшем случае все три сортировки имееют квадратическую трудоемкость. На основании замеров времени исполнения алгоритмов, был сделан вывод, что сортировка выбором имеет наименьшее время работы на случайных и обратно сортированных массивах. Также была доказана выведенная трудоемкость алгоритма сортировки пузырьком — на любых данных имеет сложность $O(N^2)$.

Литература

[1] Python [Электронный ресурс]. Режим доступа: https://python.org. Дата обращения: 17.10.2021.

[2] Модуль time [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html. Дата обращения: 17.10.2021.

[3] Visual Studio Code - Code Editing [Электронный ресурс]. Режим доступа: https://code.visualstudio.com. Дата обращения: 17.10.2021.

[4] Сортировки [Электронный ресурс]. Режим доступа: https://function-x.ru/cpp-algoritmy-sortirovki.html. Дата обращения: 17.10.2021.