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 по дисциплине "Анализ алгоритмов"

Тема Алгоритмы сортировки

Студент Петрова А. А.

Группа ИУ7-56Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Содержание

Введение

Сортировка массива — это процесс распределения всех элементов массива в определенном порядке. Очень часто это бывает полезным. Например, в почтовом ящике электронные письма отображаются в зависимости от времени получения; новые письма считаются более релевантными, чем те, которые были получены полчаса, час, два или день назад; при переходе в список контактов, имена обычно находятся в алфавитном порядке, потому что так легче что-то найти. Все эти случаи включают в себя сортировку данных перед их фактическим выводом.

Целью данной лабораторной работы является изучение и реализация алгоритмов сортировки, обучение расчету трудоемкости алгоритмов.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- изучить алгоритмы сортировки пузырьком, вставками и выбором;
- реализовать алгоритмы сортировки пузырьком, вставками и выбором;
- дать оценку трудоёмкости в лучшем и худшем случае (для двух алгоритмов сделать вывод трудоёмкости);
- замерить время работы для лучшего, худшего и произвольного случая;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненном как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе будет проанализирована предметная область и рассмотрены 3 алгоритма сортировки: пузырьком, вставками и выбором.

Сортировка массива — одна из самых популярных операций над массивом. Алгоритмы реализуют упорядочивание элементов в списке. В случае, когда элемент списка 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.

1.1 Сортировка пузырьком

Сортировка пузырьком — один из самых известных алгоритмов сортировки. Будем идти по массиву слева направо. Если текущий элемент больше следующего, меняем их местами. Делаем так, пока массив не будет отсортирован. Заметим, что после первой итерации самый большой элемент будет находиться в конце массива, на правильном месте. После двух итераций на правильном месте будут стоять два наибольших элемента, и так далее. Таким образом элементы с большими значениями оказываются в конце списка, а с меньшими остаются в начале [1].

Этот алгоритм считается учебным и почти не применяется на практике из-за низкой эффективности: он медленно работает на тестах, в которых маленькие элементы (их называют «черепахами») стоят в конце массива. Однако на нём основаны многие другие методы, например, шейкерная сортировка и сортировка расчёской.

1.2 Сортировка вставками

Создадим массив, в котором после завершения алгоритма будет лежать ответ. Будем поочередно вставлять элементы из исходного массива так, чтобы элементы в массиве-ответе всегда были отсортированы. Реализовывать алгоритм удобнее по-другому (создавать новый массив и реально что-то вставлять в него относительно сложно): просто сделаем так, чтобы отсортирован был некоторый префикс исходного массива, вместо вставки будем менять текущий элемент с предыдущим, пока они стоят в неправильном порядке [2].

На рисунке 1.1 показаны этапы сортировки вставками массива [18, 20, 5, 13, 15].

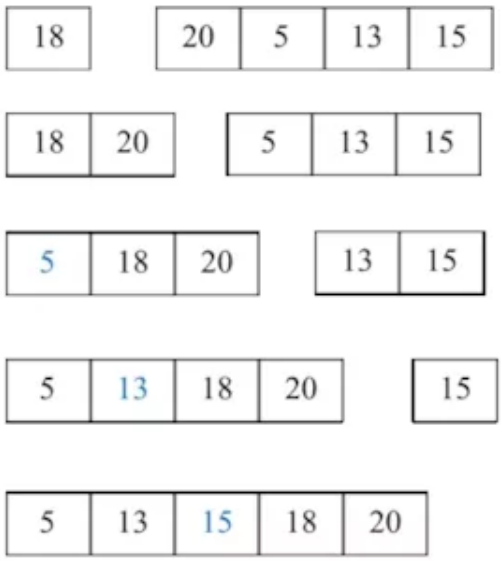

Рис. 1.1: Сортировка вставками.

1.3 Сортировка выбором

Общая идея алгоритма состоит в следующем [3]:

- в неотсортированном подмассиве ищется локальный максимум (минимум);
- найденный максимум (минимум) меняется местами с последним (первым) элементом в подмассиве;
- делать так, пока в массиве остались неотсортированные подмассивы.

1.4 Вывод

В данном разделе была проанализирована предметная область и рассмотрены 3 алгоритма сортировки: пузырьком, вставками и выбором.

Входными данными для программы являются:

- длина массива;
- массив действительных чисел.

Выходные данные: отсортированный по возрастанию массив.

Ограничения, в рамках которых будет работать программа:

- элементами массива являются действительные числа;
- массив сортируется по возрастанию;
- корректность данных в пользовательском разделе не проверяется.

Функциональные требования к ПО:

- ПО должно содержать 2 раздела: пользовательский (ручной ввод) и экспериментальный (для замеров времени);
- ПО должно выводить отсортированный массив;
- ПО должно выводить потраченное время.

2 Конструкторская часть

В этом разделе на основе теоретических данных, полученных в аналитическом разделе, будут построены схемы исследуеммых алгоритмов. А также будут описаны: структуры данных, используемые в алгоритмах, способы тестирования и классы эквивалентности, память, используемая алгоритмом, и структура ПО.

2.1 Схемы алгоритмов

На рисунках 2.1 - 2.3 представлены схемы рассматриваемых алгоритмов.

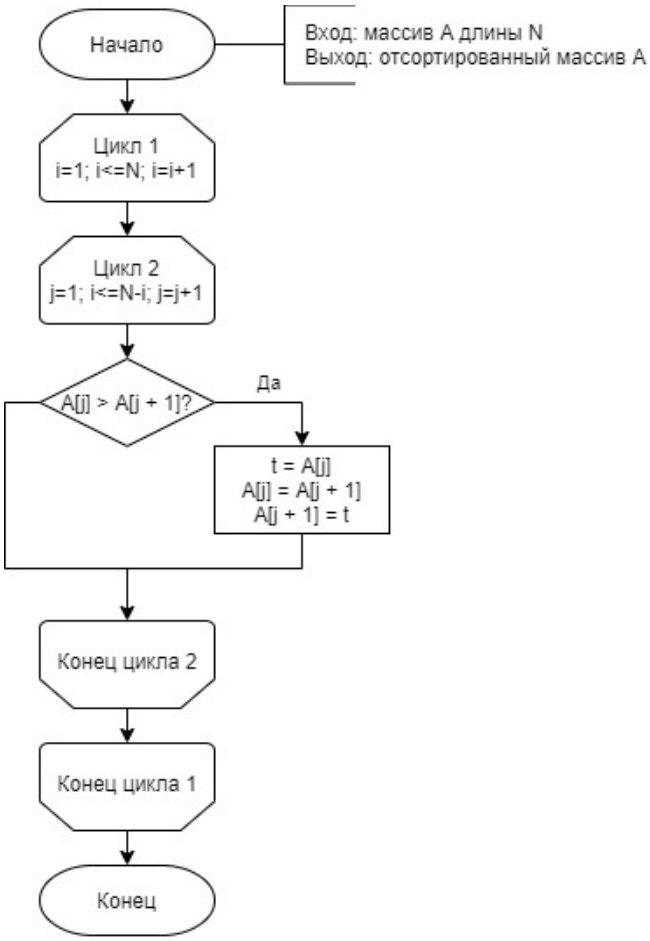

Рис. 2.1: Схема сортировки пузырьком

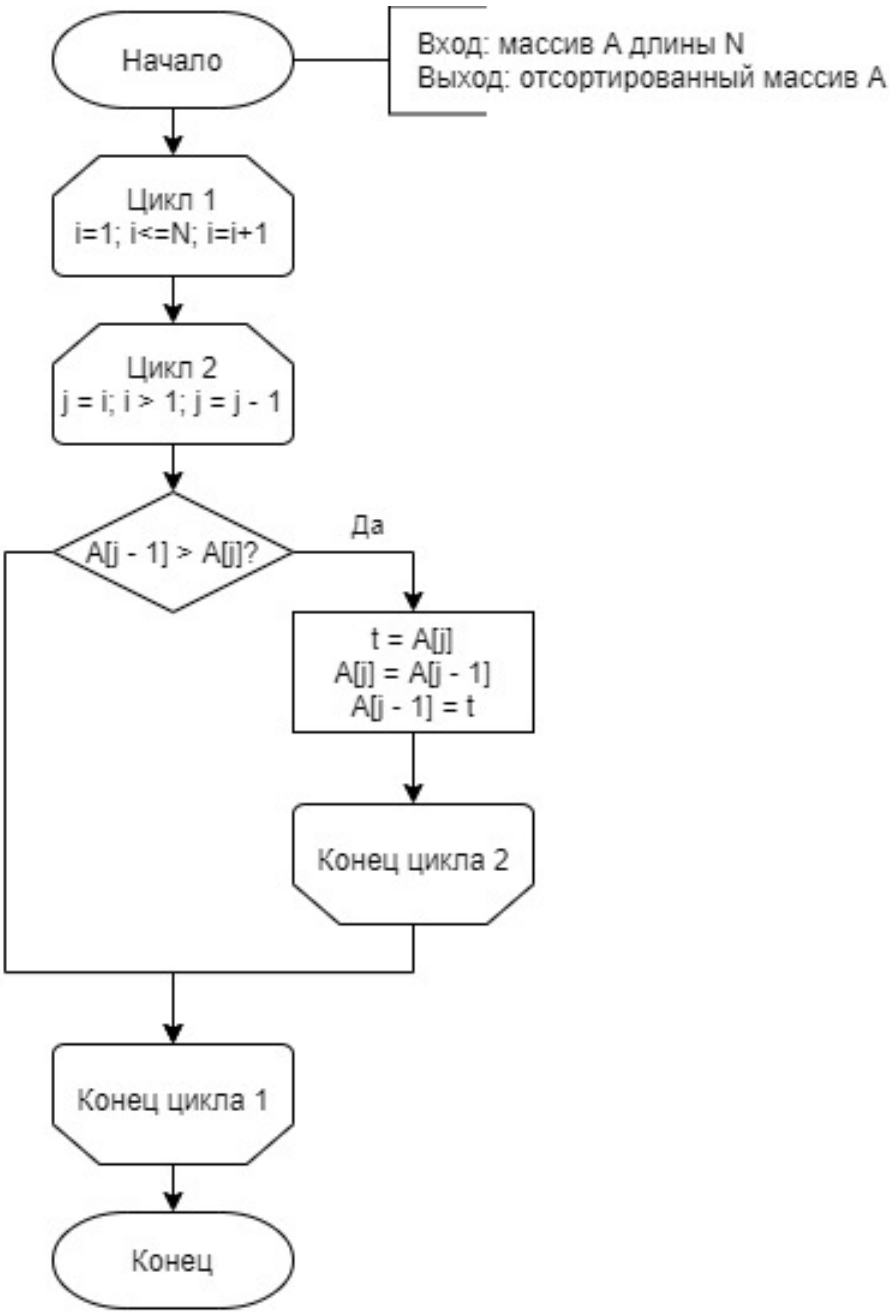

Рис. 2.2: Схема сортировки вставками

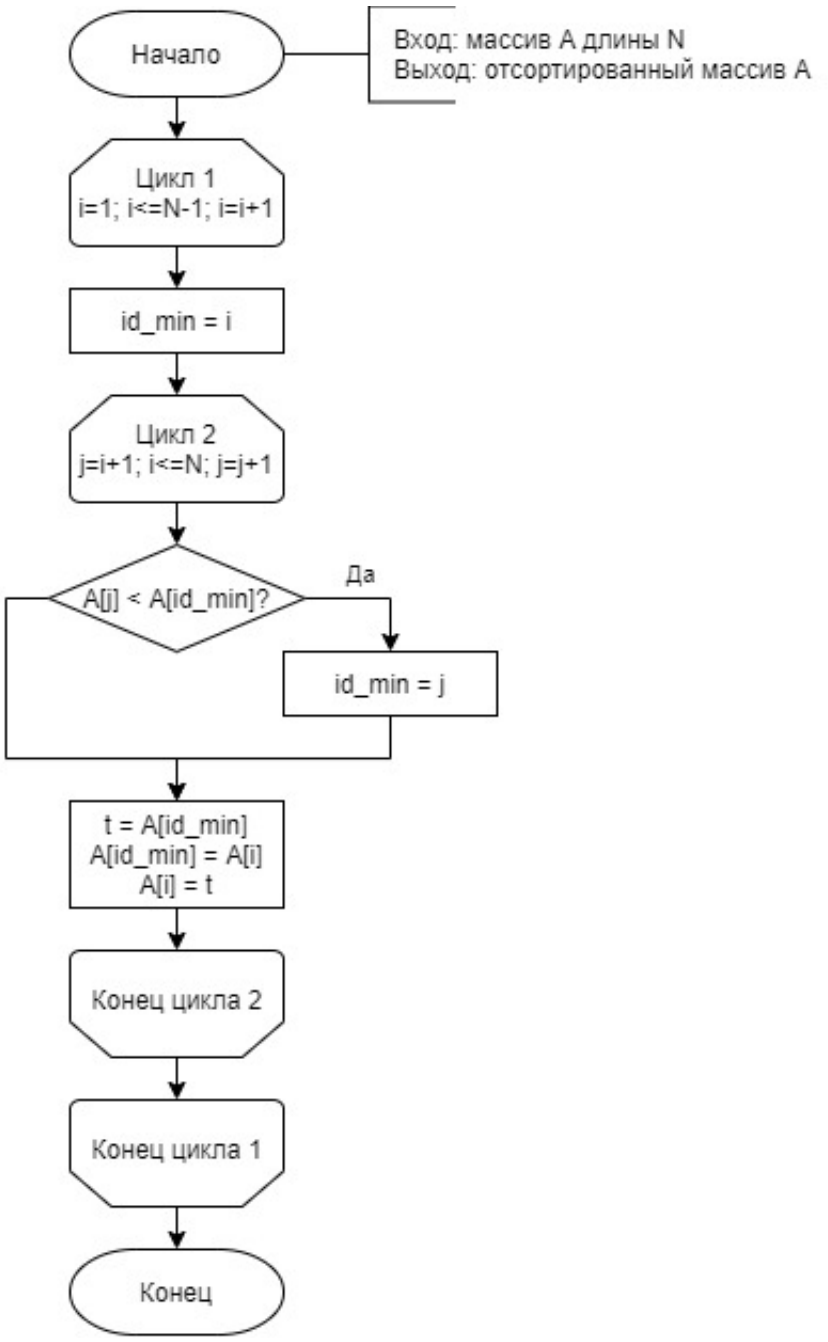

Рис. 2.3: Схема сортировки выбором

2.2 Модель вычислений

Для последующего вычисления трудоемкости введём модель вычислений:

1. Операции из списка (2.1) имеют трудоемкость 1.

$$+, -, /, \%, ==, != , <, >, <=, >=, [], ++, --\tag{2.1}$$

2. Трудоемкость оператора выбора if условие then A else B рассчитывается, как (2.2).

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases}\tag{2.2}$$

3. Трудоемкость цикла рассчитывается, как (2.3).

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}})\tag{2.3}$$

4. Трудоемкость вызова функции равна 0.

2.3 Трудоёмкость алгоритмов

Пусть размер массивов во всех вычислениях обозначается как N.

2.3.1 Алгоритм сортировки пузырьком

Трудоёмкость алгоритма сортировки пузырьком состоит из:

- трудоёмкость сравнения и инкремента внешнего цикла $i \in [1..N)$ (2.4):

$$f_i = 2 + 2(N-1)\tag{2.4}$$

- суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N-1]$ (2.5):

$$f_j = 3(N-1) + \frac{N \cdot (N-1)}{2} \cdot (3 + f_{if})\tag{2.5}$$

- трудоёмкость условия во внутреннем цикле (2.6):

$$f_{if} = 4 + \begin{cases} 0, & \text{в лучшем случае} \\ 9, & \text{в худшем случае} \end{cases}\tag{2.6}$$

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

$$f_{best} = \frac{7}{2}N^2 + \frac{3}{2}N - 3 \approx \frac{7}{2}N^2 = O(N^2)\tag{2.7}$$

Трудоёмкость в **худшем** случае (2.8):

$$f_{worst} = 8N^2 - 8N - 3 \approx 8N^2 = O(N^2)\tag{2.8}$$

2.3.2 Алгоритм сортировки вставками

Трудоёмкость алгоритма сортировки пузырьком состоит из:

- трудоёмкость сравнения и инкремента внешнего цикла $i \in [1..N)$ (2.9):

$$f_i = 2 + 2(N-1) \tag{2.9}$$

- суммарная трудоёмкость внутренних циклов, количество итераций которых меняется в промежутке $[1..N-1]$ (2.10):

$$f_{if} = 4 + \begin{cases} 0, & \text{в лучшем случае} \\ 3(N-1) + \frac{N\cdot(N-1)}{2} \cdot (3 + f_{if}), & \text{в худшем случае} \end{cases} \tag{2.10}$$

- трудоёмкость условия во внутреннем цикле (2.11):

$$f_{if} = 4 + \begin{cases} 0, & \text{в лучшем случае} \\ 9, & \text{в худшем случае} \end{cases} \tag{2.11}$$

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

$$f_{best} = 13N - 10 \approx 13N = O(N) \tag{2.12}$$

Трудоёмкость в **худшем** случае (2.13):

$$f_{worst} = 4.5N^2 + 10N - 13 \approx 4N^2 = O(N^2) \tag{2.13}$$

2.4 Структуры данных

В данной работе из структур данных используются только одномерные массивы.

2.5 Тестирование и классы эквивалентности

Для проверки работоспособности ПО будет применяться функциональное тестирование. Классы эквивалентности:

- массив уже отсортирован по возрастанию;
- массив отсортирован в обратном порядке;
- массив никак не отсортирован;
- массив из одного элемента;
- пустой массив.

2.6 Используемая память

Все используемые в данной работе массивы будут динамическими. В связи с этим, нужно будет контролировать проблемы с памятью, в частности при выделении памяти под массивы и её освобождении из-под них.

Из всего вышесказанного следует, что количество памяти, необходимой алгоритмам, пропорционально длинам массивов.

2.7 Структура ПО

ПО будет состоять из набора функций:

- основная функция, работающая с меню;
- ввод исходных данных с клавиатуры;
- сортировка пузырьком;
- сортировка вставками;
- сортировка выбором;
- замеры времени.

2.8 Вывод

На основе теоретических данных, полученные в аналитическом разделе были построены схемы иследуеммых алгоритмов и оценены их трудоёмкости в лучшем и худшем случае. Также были описаны: структуры данных, используемые в алгоритмах, способы тестирования и классы эквивалентности, память, используемая алгоритмом, и структура ПО.

3 Технологическая часть

В этом разделе будут разработаны исходные коды трёх алгоритмов сортировки: пузырьком, вставками и выбором.

3.1 Средства реализации

Для реализации программы сортировки массивов был выбран язык программирования Python. Данный выбор обусловлен тем, что в этом языке присутсвует функция для измерения процессорного времени.

3.2 Реализация алгоритмов

В листингах 3.1 - 3.3 приведена реализация трёх алгоритмов сортировки.

Листинг 3.1: Функция сортировки пузырьком

```
1     def bubble_sort(arr, n):
2       res = [arr[x] for x in range(n)]
3       for i in range(n):
4         for j in range(n - i - 1):
5           if res[j] > res[j + 1]:
6             res[j], res[j + 1] = res[j + 1], res[j]
7       return res
```

Листинг 3.2: Функция сортировки вставками

```
1     def insert_sort(arr, n):
2       res = [arr[x] for x in range(n)]
3       for i in range(n):
4         current = res[i]
5         j = i
6         while (res[j - 1] > current) and (j > 0):
7           res[j] = res[j - 1]
8           j -= 1
9         res[j] = current
10      return res
```

Листинг 3.3: Функция сортировки выбором

```
    def select_sort(arr, n):
      res = [arr[x] for x in range(n)]
      for i in range(n - 1):
        id_min = i
        for j in range(i + 1, n):
          if res[j] < res[id_min]:
            id_min = j
        res[i], res[id_min] = res[id_min], res[i]
      return res
```

3.3 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
$[1, 3, 5, 7, 9]$	$[1, 3, 5, 7, 9]$	$[1, 3, 5, 7, 9]$
$[9, 7, 5.34, 3, 1]$	$[1, 3, 5.34, 7, 9]$	$[1, 3, 5.34, 7, 9]$
$[-7, 5, 9, -3, -1]$	$[-7, -3, -1, 5, 9]$	$[-7, -3, -1, 5, 9]$
$[10.2]$	$[10.2]$	$[10.2]$
Пустой массив	Пустой массив	Пустой массив

3.4 Вывод

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

4 Исследовательская часть

В этом разделе будет проведён эксперимент для определения эффективности по времени работы каждого из разработанных алгоритмов.

4.1 Пример работы

Демонстрация работы программы приведена на рисунке 4.1.

```
Меню:
1. Ручной ввод
2. Замеры процессорного времени выполнения алгоритмов
0. Выход

Выберите пункт меню: 1

Длина массива: 5
Массив чисел:
-7
5
9
-3
-1

Сортировка пузырьком:  [-7.0, -3.0, -1.0, 5.0, 9.0]
Сортировка вставками:  [-7.0, -3.0, -1.0, 5.0, 9.0]
Сортировка выбором:  [-7.0, -3.0, -1.0, 5.0, 9.0]

Меню:
1. Ручной ввод
2. Замеры процессорного времени выполнения алгоритмов
0. Выход

Выберите пункт меню: 0
```

Рис. 4.1: Работа алгоритмов сортировки

4.2 Технические характеристики

Ниже приведеные технические характеристики устройства, на котором было проведенно тестирование ПО:

- Операционная система: Windows 10 64-bit Home [4].
- Оперативная память: 8 GB.
- Процессор: 11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz [5].

4.3 Время выполнения алгоритмов

Время выполнения алгоритмов замерялось с помощью функции process_time модуля time в Python [6]. Данная функция возвращает значение в долях секунды суммы системного и пользовательского процессорного времени текущего процесса.

В таблицах 4.1 - 4.3 представлены замеры времени работы для каждого из алгоритмов.

Таблица 4.1: Время выполнения отсортированных по возрастанию данных (лучший случай)

Размер	Пузырёк	Вставки	Выбор
100	0.000313	0	0.000313
300	0.002188	0	0.001719
500	0.006094	0	0.004844
700	0.012031	0	0.009844
900	0.020625	0	0.016250
1100	0.030937	0.000156	0.024375
1300	0.044531	0.000156	0.034063
1500	0.059062	0.000156	0.045937
1700	0.076406	0.000156	0.057656
1900	0.095781	0.000156	0.072500
2100	0.117969	0.000313	0.087656

Таблица 4.2: Время выполнения отсортированных по убыванию данных (худший случай)

Размер	Пузырёк	Вставки	Выбор
100	0.000625	0.000469	0.000313
300	0.005469	0.004219	0.001875
500	0.015625	0.012188	0.005313
700	0.031563	0.024688	0.010156
900	0.052969	0.041875	0.016875
1100	0.080469	0.063594	0.025469
1300	0.112500	0.089375	0.035469
1500	0.153125	0.120781	0.047969
1700	0.201719	0.156406	0.060937
1900	0.253125	0.197500	0.076563
2100	0.303594	0.240156	0.093125

Таблица 4.3: Время выполнения алгоритмов на случайных данных

Размер	Пузырёк	Вставки	Выбор
100	0.000469	0.000313	0.000313
300	0.004219	0.002344	0.002031
500	0.012344	0.006719	0.005625
700	0.023906	0.013594	0.010937
900	0.040469	0.022656	0.017969
1100	0.060937	0.033906	0.026719
1300	0.085000	0.047500	0.036719
1500	0.113906	0.063750	0.049063
1700	0.146875	0.081250	0.063125
1900	0.184375	0.102031	0.078437
2100	0.225469	0.124531	0.095469

4.4 Вывод

В лучшем случае (на отсортированном по возрастанию массиве) быстрее всего работает сортировка вставками, а медленнее всего - пузырьком (сортировка выбором эффективнее пузырька в среднем примерно на 16,3%).

На случайных данных быстрее всего работает сортировка выбором. Она эффективнее сортировки вставками в среднем примерно на 14% и эффективнее сортировки пузырьком в среднем примерно на 50%.

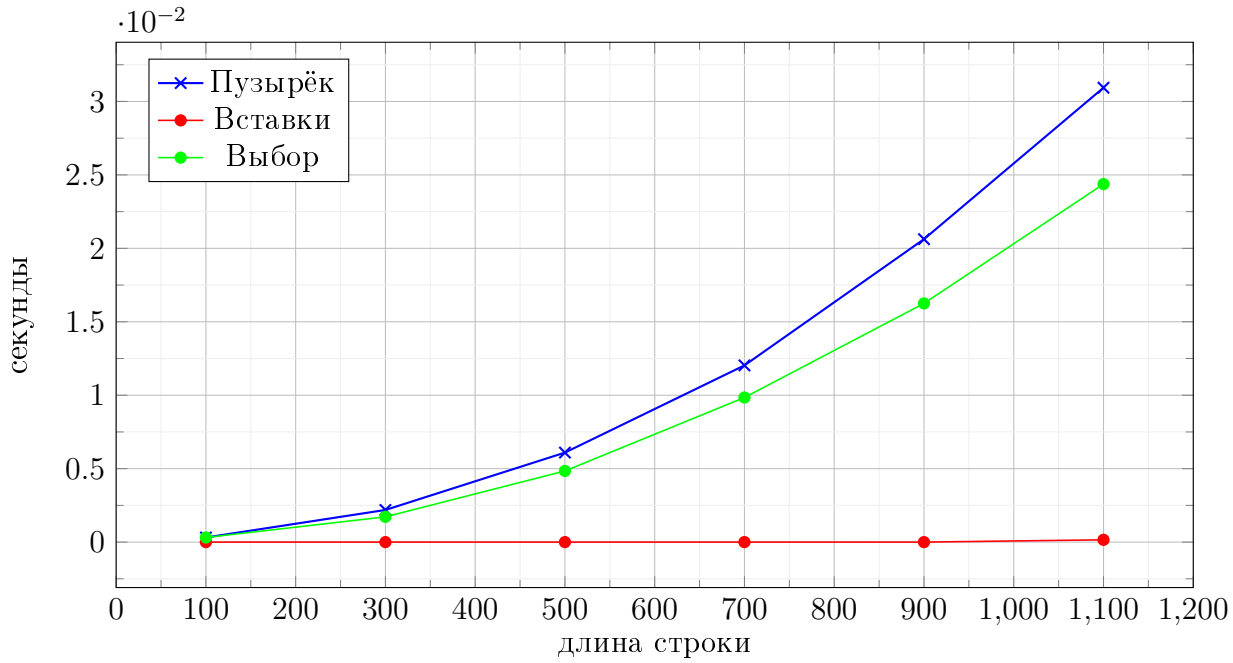

Рис. 4.2: Сравнение алгоритмов сортировки пузырьком, вставками и выбором на лучшем случае

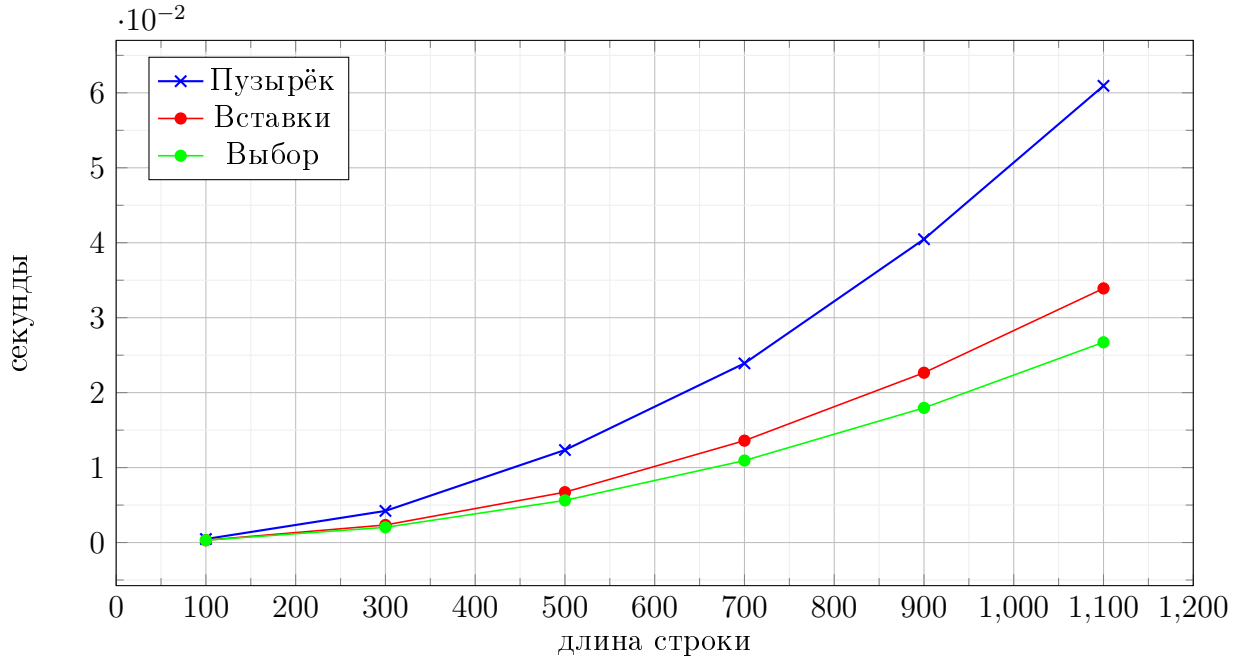

Рис. 4.3: Сравнение алгоритмов сортировки пузырьком, вставками и выбором на случайных данных

Заключение

На основании анализа трудоемкости алгоритмов в выбранной модели вычислений, было показано, что алгоритм сортировки вставками имеет наименьшую сложность (линейную) в уже отсортированном массиве. В случае обратно отсортированного массива, сортировка вставками и пузырьком имееют квадратическую сложность. На основании замеров времени исполнения алгоритмов, был сделан вывод, что при прямом порядке элементов в массиве, сортировка пузьком работает медленее, чем выбором (примерно на 16,3%). Так же был доказана выведенная трудоемкость алгоритма сортировки вставками - при уже отсортированном массиве сортировка работает очень быстро, она является в таком случае наиболее эффективной. На случайных же данных быстрее всех работает алгоритм сортировки выбором.

Литература

[1] Левитин А. В. Глава 3. Метод грубой силы: Пузырьковая сортировка // Алгоритмы. Введение в разработку и анализ. 2006. с. 144-146.

[2] Кнут Д. Э. 5.2.1 Сортировка путём вставок // Искусство программирования. Том 3. Сортировка и поиск. 2007. с. 832.

[3] Левитин А. В. Глава 3. Метод грубой силы: Сортировка выбором // Алгоритмы. Введение в разработку и анализ. 2006. с. 143-144.

[4] Windows 10 Pro и Windows 10 Домашняя. https://www.microsoft.com/ru-ru/windows/compare-windows-10-home-vs-pro. Дата обращения: 18.10.2021.

[5] Процессор Intel® Core™ i3-1115G4 (6 МБ кэш-памяти, до 4,10 ГГц). Режим доступа: https://www.intel.ru/content/www/ru/ru/products/sku/208652/intel-core-i31115g4-processor-6m-cache-up-to-4-10-ghz/specifications.html. Дата обращения: 14.10.2021.

[6] Функция process_time() модуля time в Python. https://docs-python.ru/standart-library/modul-time-python/funktsija-process-time-modulja-time/. Дата обращения: 5.10.2021.