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 Андрич К.

Группа ИУ7И-56Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение **2**

1 Аналитическая часть **3**
1.1 Сортировка пузырьком . . . 3
1.2 Сортировка вставками . . . 3
1.3 Быстрая сортировка . . . 4
1.4 Вывод . . . 4

2 Конструкторская часть **5**
2.1 Схемы алгоритмов . . . 5
2.2 Модель вычислений . . . 8
2.3 Трудоёмкость алгоритмов . . . 8
2.3.1 Алгоритм сортировки пузырьком . . . 8
2.3.2 Алгоритм сортировки вставками . . . 9
2.4 Вывод . . . 9

3 Технологическая часть **10**
3.1 Выбор ЯП . . . 10
3.2 Требование к ПО . . . 10
3.3 Реализация алгоритмов . . . 10
3.4 Тестовые данные . . . 11
3.5 Вывод . . . 11

4 Исследовательская часть **12**
4.1 Пример работы . . . 12
4.2 Время выполнения алгоритмов . . . 12
4.3 Вывод . . . 13

Заключение **14**

Список литературы **14**

Введение

Сортировка массива — это процесс распределения всех элементов массива в определенном порядке. Очень часто это бывает полезным. Например, в вашем почтовом ящике электронные письма отображаются в зависимости от времени получения; новые письма считаются более релевантными, чем те, которые вы получили полчаса, час, два или день назад; когда вы переходите в свой 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.

Задачи 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 и худшем случае(для двух алгоритмов сделать вывод трудоёмкости).
4. Замерить время работы для лучшего, худшего и произвольного случая.
5. Описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненном как расчётно-пояснительная записка к работе. .

1 | Аналитическая часть

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.

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.

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

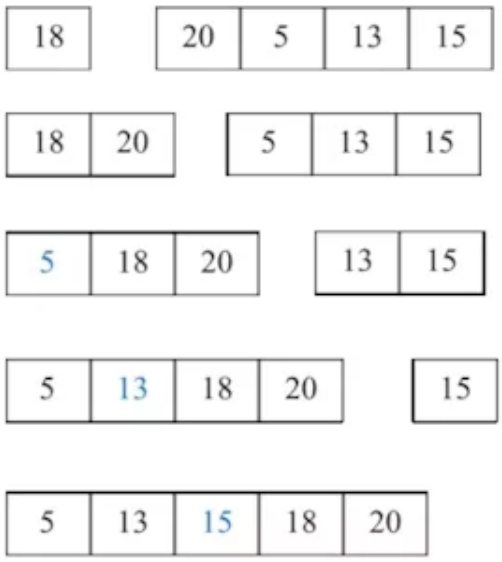

Рис. 1.1: Сортировка вставками.

1.3 Быстрая сортировка

Общая идея алгоритма состоит в следующем:

- выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность
- сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».
- для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

1.4 Вывод

В данном разделе были рассмотрены 3 алгоритмы сортировки: пузырьком, вставками и быстрая сортировка.

2 | Конструкторская часть

2.1 Схемы алгоритмов

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

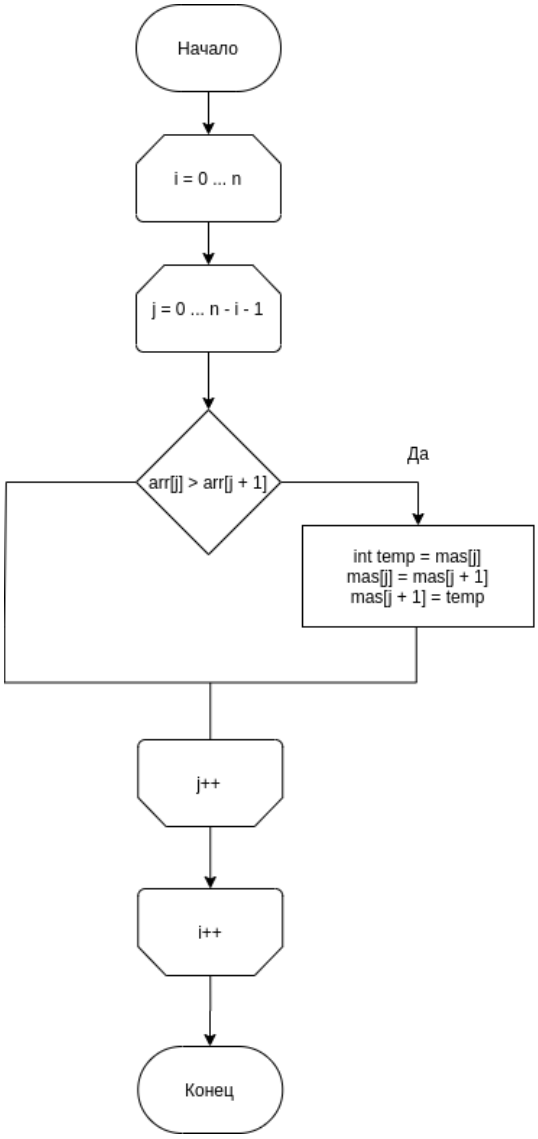

Рис. 2.1: Схема сортировки пузырьком

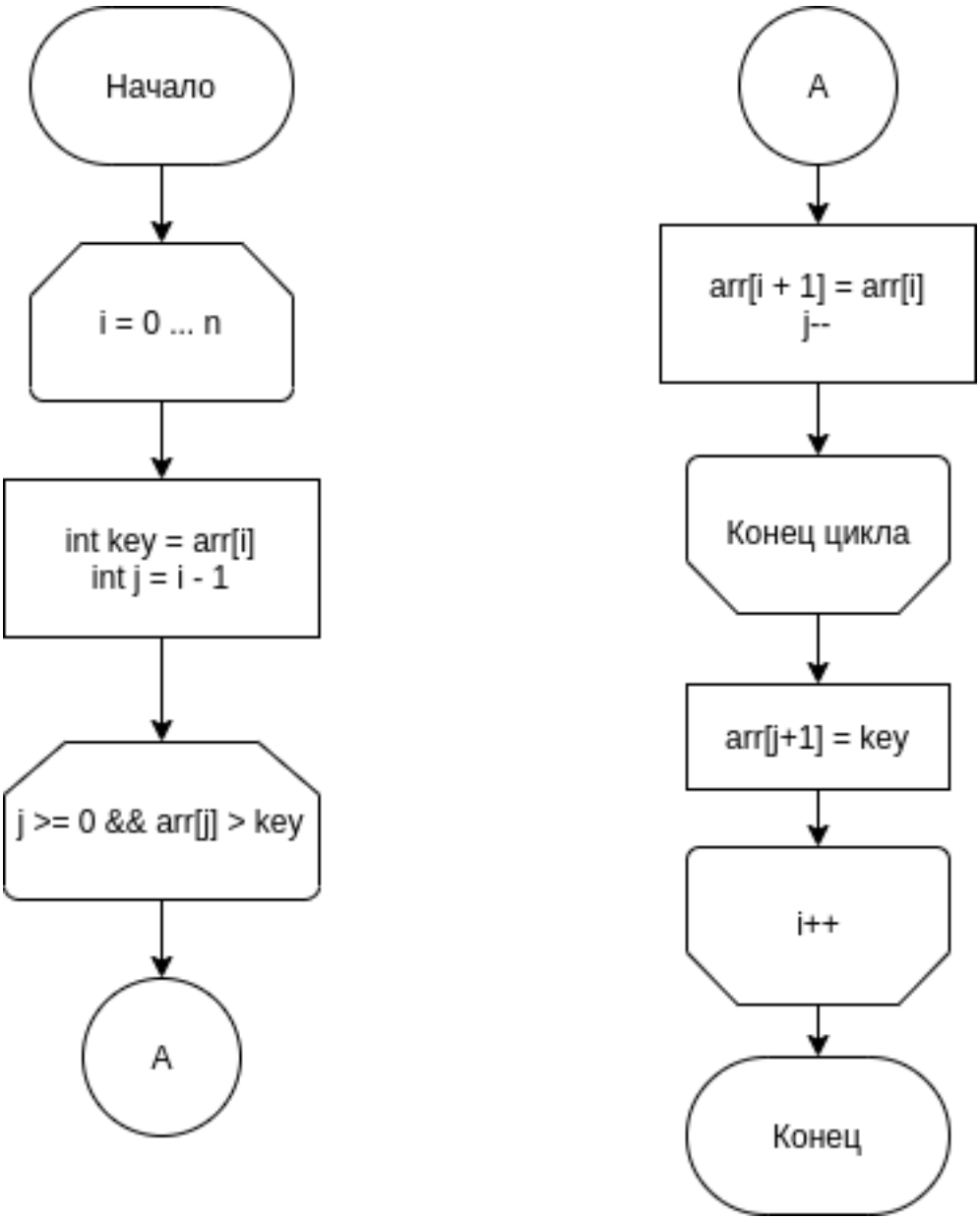

Рис. 2.2: Схема сортировки вставками

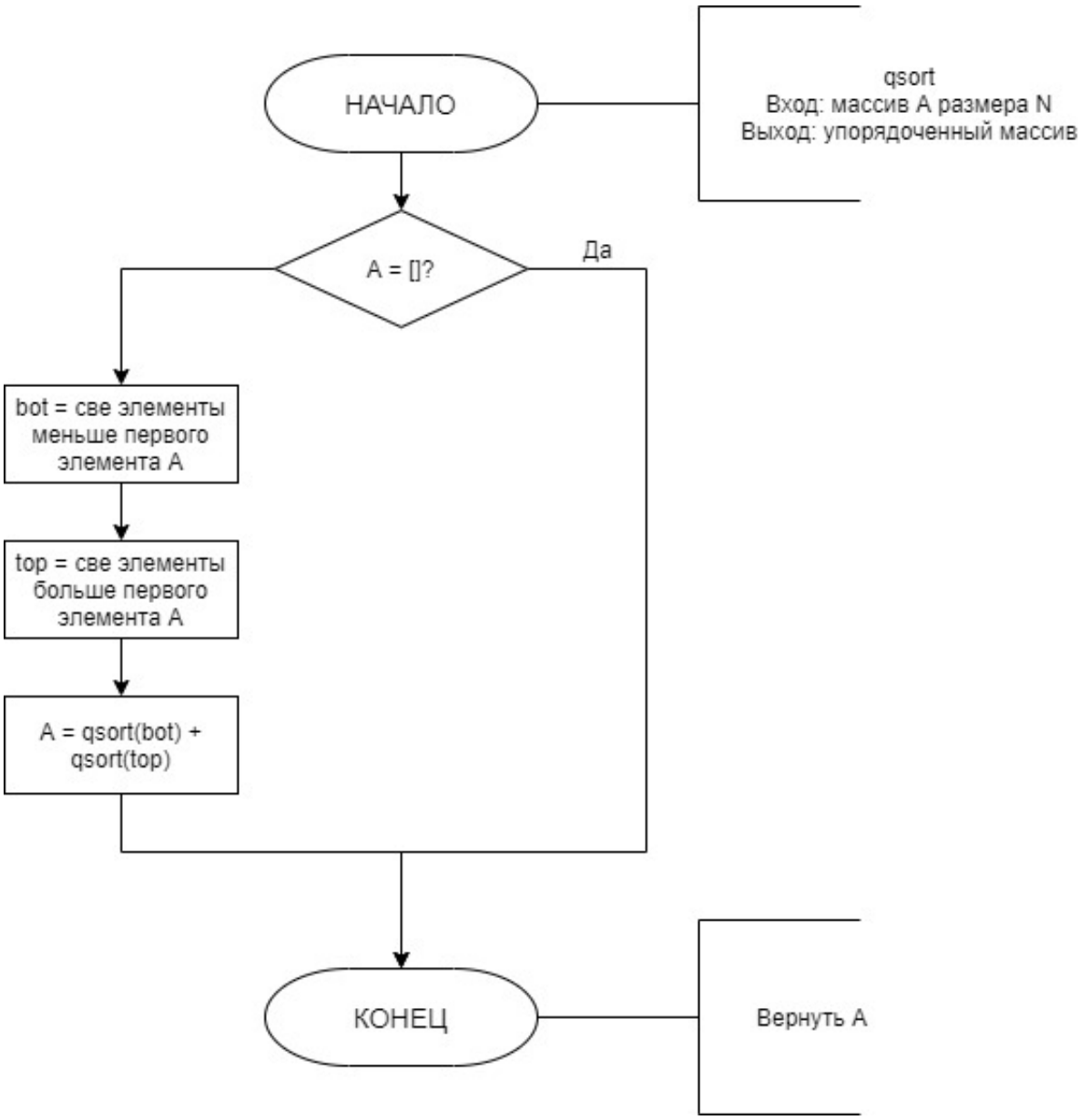

Рис. 2.3: Схема быстрой сортировк

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 Вывод

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

3 | Технологическая часть

3.1 Выбор ЯП

В данной лабораторной работе использовался язык программирования - python. Данный выбор обусловлен тем, что этот язык наиболее удобен для работы со строками, а также тем, что в нём присутсвует функция для измерения процессорного времени. В качестве среды разработки я использовала Visual Studio Code, так как считаю его достаточно удобным.

3.2 Требование к ПО

К программе предъявляется ряд требований:

- на вход ПО получает массив сравнимых элементов;
- на выходе – тот же массив, но отсортированный в заданном порядке.

3.3 Реализация алгоритмов

В листингах 3.1 - 3.3 приведена реализация трёх алгоритмов сортировки.

Листинг 3.1: Функция сортировки массива пузырьком

```
def bubble(arr):
  for i in range(len(arr)):
    for j in range(len(arr)-i-1):
      if arr[j] > arr[j+1]:
        arr[j], arr[j+1] = arr[j+1], arr[j]
  return arr
```

Листинг 3.2: Функция сортировки массива вставками

```
def insertion(arr):
  for i in range(len(arr)):
    current = arr[i]
    j = i
    while (arr[j-1] > current) and (j > 0):
      arr[j] = arr[j-1]
      j = j - 1
    arr[j] = current
  return arr
```

Листинг 3.3: Функция быстрой сортировки

```
def quick(arr):
  if len(arr) <= 1:
    return arr
  else:
    q = arr[0]
    s_nums = []
    m_nums = []
    e_nums = []
    for n in arr:
      if n < q:
        s_nums.append(n)
      elif n > q:
        m_nums.append(n)
      else:
        e_nums.append(n)
    return quick(s_nums) + e_nums + quick(m_nums)
```

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
$[8, 12, 17, 25, 34]$	$[8, 12, 17, 25, 34]$	$[8, 12, 17, 25, 34]$
$[41, 35, 21, 17, 5]$	$[5, 17, 21, 35, 41]$	$[5, 17, 21, 35, 41]$
$[-7, -12, -32, -78, -92]$	$[-92, -78, -32, -12, -7]$	$[-92, -78, -32, -12, -7]$
$[35, -7, 12, -51, 101]$	$[-51, -7, 12, 35, 51]$	$[-51, -7, 12, 35, 51]$
$[10]$	$[10]$	$[10]$
$[-3]$	$[-3]$	$[-3]$
Пустой массив	Пустой массив	Пустой массив

3.5 Вывод

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

4 | Исследовательская часть

4.1 Пример работы

Демонстрация работы программы приведена на рисунке 4.1.

```
Menu:
1 - sort list
2 - measure time
0 - exit

Choice: 1

Input array:
-10 5 1 54 -2 -78 100
Bubble sort:  [-78, -10, -2, 1, 5, 54, 100]

Insertion sort:  [-78, -10, -2, 1, 5, 54, 100]

Quick sort:  [-78, -10, -2, 1, 5, 54, 100]
```

Рис. 4.1: Работа алгоритмов сортировки

4.2 Время выполнения алгоритмов

Время выполнения алгоритмов замерялось с помощью функции process_time модуля time в Python.

Таблица 4.1: Таблица времени выполнения сортировок на случайных данных

Размер	bsort	isort	qsort
1000	0.453125	0.281250	0.015625
2000	1.796875	1.125000	0.046875
3000	4.046875	2.421875	0.093750
5000	11.296875	7.015625	0.140625

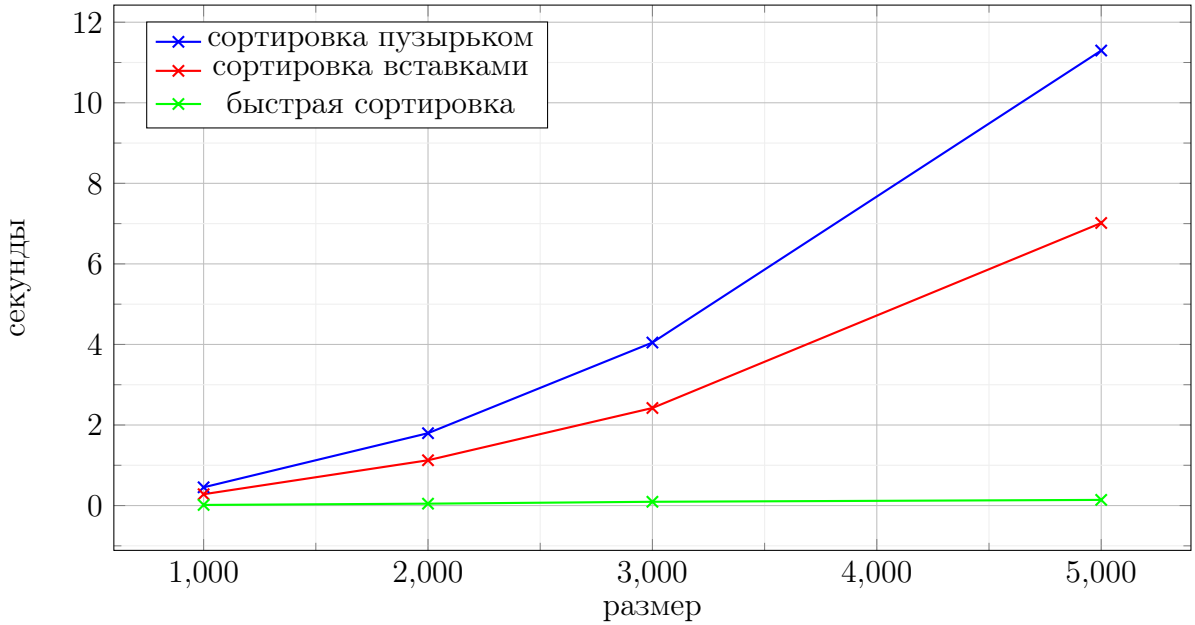

Рис. 4.2: Сравнение времени выполнения сортировок на случайных данных

4.3 Вывод

При прямом порядке элементов в массиве, сортировка пузьком работает медленее чем сортировка вставками, и ещё медленее чем быстрая.

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 обранто 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 быстрая и еще медленее чем сортировка вставкам. Так же был доказана выведенная трудоемкость алгоритма сортировки вставками – при уже отсортированном массиве сортировка работает очень быстро. На случайных данных быстрее всех работает алгоритм быстрой сортировки.

Литература

[1] Основные виды сортировок и примеры их реализации. [Электронный ресурс]. Режим доступа:

https://academy.yandex.ru/posts/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii Дата обращения: 17.10.2021

[2] Описание алгоритмов сортировки и сравнение их производительности. [Электронный ресурс.] Режим доступа:

https://habr.com/ru/post/335920/ Дата обращения: 17.10.2021

[3] Знай сложности алгоримтмов. [Электронный ресурс.] Режим доступа:

https://habr.com/ru/post/188010/ Дата обращения: 17.10.2021

[4] Python documentation. [Электронный ресурс.] Режим доступа:

https://www.python.org/doc/ Дата обращения: 17.10.2021