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»

Лабораторная работа № 3

Дисциплина Анализ алгоритмов

Тема Трудоемкость алгоритмов сортировки

Студент Искакова К.М.

Группа ИУ7-52Б

Оценка (баллы) _____

Преподаватель Волкова Л. Л.

Москва — 2021 г.

Оглавление

Введение

Алгоритмы сортировки имеют большое практическое применение. Их можно встретить почти везде, где речь идет об обработке и хранении больших объемов информации. Сортировки используются в самом широком спектре задач, включая обработку коммерческих, сейсмических, космических и прочих данных [1]. Часто сортировка является просто вспомогательной операцией для упорядочивания данных, упрощения последующих алгебраических действий над данными и т. п.

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

В настоящее время, в связи с экспоненциально возросшими объемами данных, вопрос эффективной сортировки данных снова стал актуальным.

Цель работы: оценить трудоемкость алгоритмов сортировки массива и получить практический навык реализации алгоритмов.

Задачи работы:

1) изучить алгоритмы сортировки массива: пузырьком, вставка и выбором;
2) оценить трудоемкость данных алгоритмов сортировки;
3) реализовать данные алгоритмы сортировки массива на одном из языков программирования;
4) сравнить алгоритмы сортировки массива.

1 | Аналитическая часть

Сортировка массива — одна из самых частых 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. Описание алгоритма сортировки «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 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма).

1.2. Описание алгоритма сортировки вставками

Сортировка вставками — алгоритм сортировки, котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов [2].

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.

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

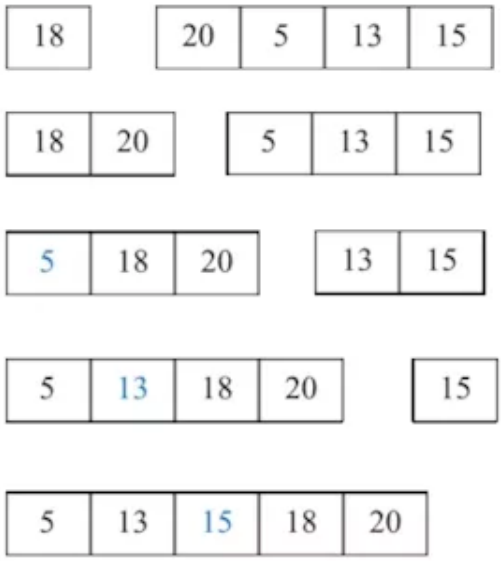

Рис. 1.1. Сортировка вставками

1.3. Описание алгоритма сортировки выбором

Сначала нужно рассмотреть подмножество массива и найти в нём максимум (или минимум). Затем выбранное значение меняют местами со значением первого неотсортированного элемента. Таким образом, после i-ой итерации первые i элементов будут стоять на своих местах. Этот шаг нужно повторять до тех пор, пока в массиве не закончатся неотсортированные подмассивы. Нужно отметить, что эту сортировку можно реализовать двумя способами – сохраняя минимум и его индекс или просто переставляя текущий элемент с рассматриваемым, если они стоят в неправильном порядке.

На рис. 1.2 показаны этапы сортировки выбором массива [4, 9, 7, 6, 2, 3].

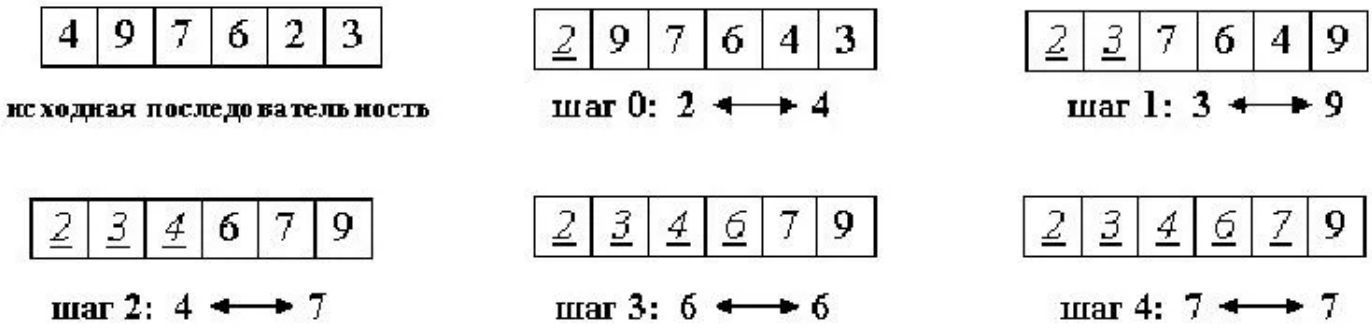

Рис. 1.2. Сортировка выбором

2 | Конструкторская часть

В этом разделе содержатся схемы алгоритмов сортировки массива. На вход алгоритмы принимают массив и размер массива. На выходе выдают отсортированный массив.

2.1. Схемы алгоритмов

На рис. 2.1-2.3 приведены схемы следующих алгоритмов:

1) алгоритм сортировки массива «пузырьком»;
2) алгоритм сортировки массива вставками;
3) алгоритм сортировки массива выбором.

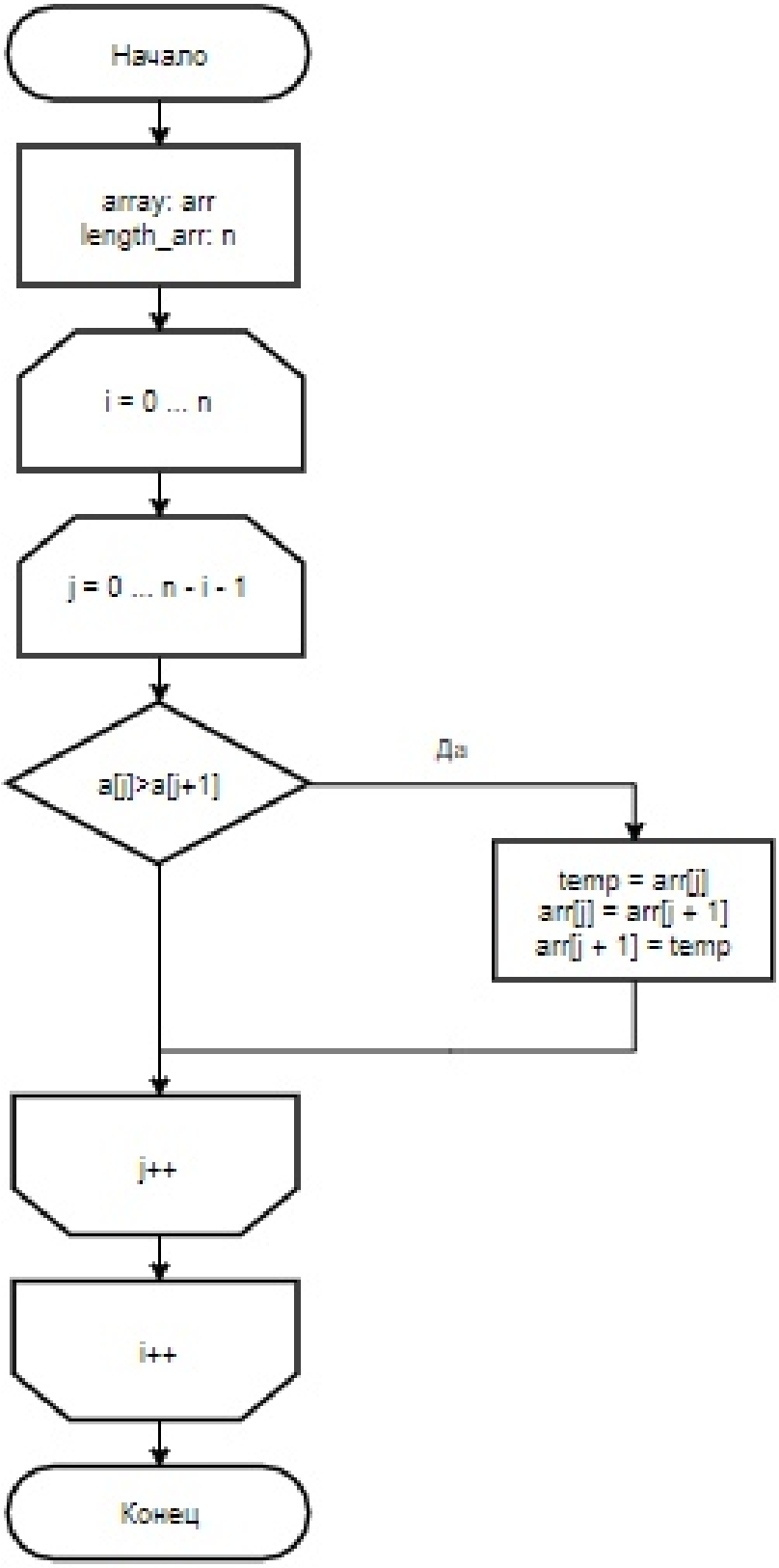

Рис. 2.1. Алгоритм сортировки массива «пузырьком»

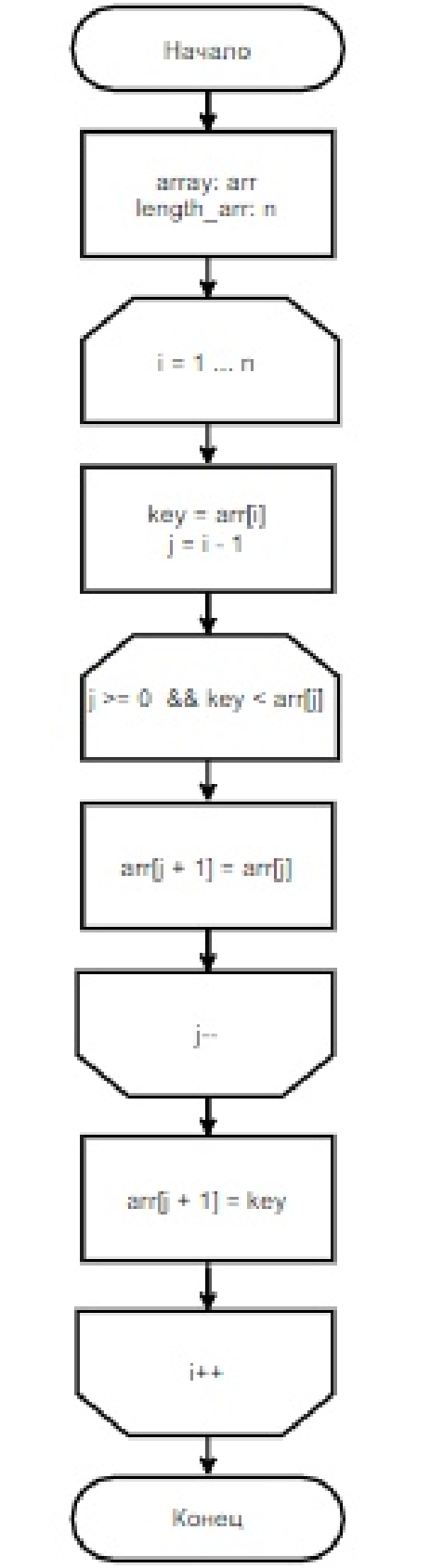

Рис. 2.2. Алгоритм сортировки массива вставками

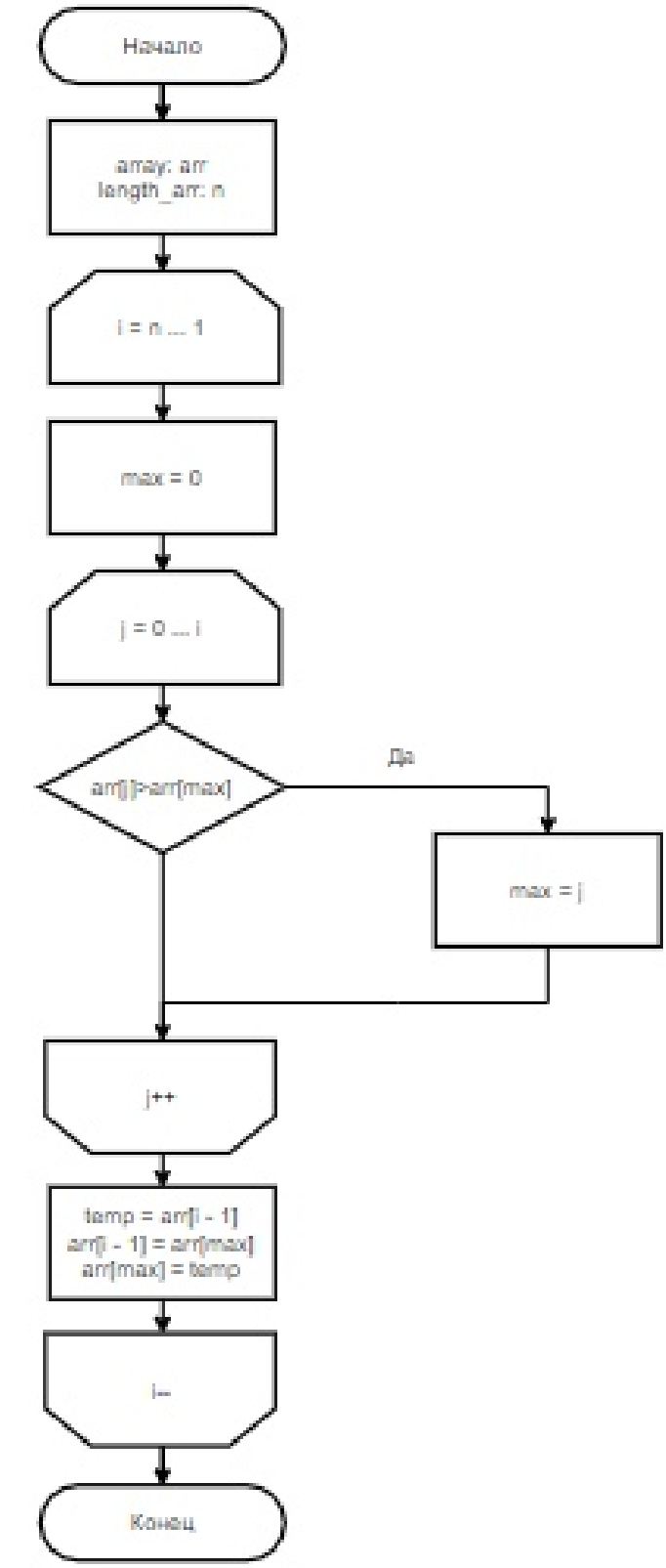

Рис. 2.3. Алгоритм сортировки массива выбором

3 | Технологическая часть

В данном разделе будут рассмотрены требования к программному обеспечению, средства реализации, представлен листинг кода, трудоемкость алгоритмов и описание тестирования.

3.1. Требования к программному обеспечению

Требования к вводу: на вход подаются массив и размер массива.

Требования к программе:

1) на выходе необходимо получить отсортированный массив;
2) требуется замерить время работы каждого из алгоритмов.

3.2. Средства реализации

В качестве языка программирования был выбран Python т.к. я знаком с данным языком, он простой и лаконичный, имеющий немногословный и понятный синтаксис, похожий на псевдокод, обладающий сильной динамической типизацией, которая способствует быстрому написанию кода.

Среда разработки — PyCharm, которая предоставляет умную проверку кода, быстрое выявление ошибок и оперативное исправление, вкупе с автоматическим рефакторингом кода, и богатыми возможностями в навигации.

Время работы алгоритмов было замерено с помощью функции process_time() из библиотеки time [3].

3.3. Листинг кода

В листингах 3.1-3.3 представлена реализация алгоритмов сортировки массива.

Листинг 3.1. Сортировка массива «пузырьком»

```
def BubbleSort(a, n):
  i = 0
  while i < n:
    j = 0
    while j < n - i - 1:
      if a[j] > a[j + 1]:
        temp = a[j]
        a[j] = a[j + 1]
        a[j + 1] = temp
      j += 1
    i += 1
```

Листинг 3.2. Сортировка массива вставками

```
def InsertSort(a, n):
  i = 1
  while i < n:
    key = a[i]
    j = i - 1
    while j >= 0 and key < a[j]:
      a[j + 1] = a[j]
      j -= 1
    a[j + 1] = key
    i += 1
```

Листинг 3.3. Сортировка массива выбором

```
def SelectionSort(a, n):
  i = n
  while i > 1:
    maximum = 0
    j = 0
    while j < i:
      if a[j] > a[maximum]:
        maximum = j
      j += 1
    temp = a[i - 1]
    a[i - 1] = a[maximum]
    a[maximum] = temp
    i -= 1
```

3.4. Трудоемкость алгоритмов

Введем модель вычисления трудоемкости для оценки алгоритмов:

- базовые операции стоимостью 1 — +, -, *, /, =, ==, <=, >=, !=, +=, [], получение полей класса
- оценка трудоемкости цикла: Fц = init + N*(a + Fтела + post) + a, где a - условие цикла, init - предусловие цикла, post - постусловие цикла
- стоимость условного перехода применим за 0, стоимость вычисления условия остаётся

Оценим трудоемкость алгоритмов сортировки массива по коду программы.

3.4.1. Трудоемкость алгоритма сортировки «пузырьком»

Рассмотрим трудоемкость алгоритма сортировки «пузырьком».

Наилучший случай: 2 + (N - 1) * (3 + 1 + 2 + (N / 2) * 8) = 4 * N * N + 2 * N - 6
Наихудший случай: 2 + (N - 1) * (3 + 1 + 2 + (N / 2) * (8 + 9)) = 8,5 * N * N - 2,5 * N - 4

3.4.2. Трудоемкость алгоритма сортировки вставками

Рассмотрим трудоемкость алгоритма сортировки вставками.

Наилучший случай: 2 + N * (2 + 2 + 3 + 6) = 13 * N + 2

Наихудший случай: 2 + N * (2 + 2 + 3 + 6 + (N - 1) / 4 * (5 + 4)) = 4.5 * N * N + 8.5 * * N + 2

3.4.3. Трудоемкость алгоритма сортировки выбором

Рассмотрим трудоемкость алгоритма сортировки выбором.

Наилучший случай: 2 + (N - 1) * (3 + 11 + 2 + (N / 2) * (2 + 3)) = 2,5 * N * N + 12,5 * *N - 13
Наихудший случай: 2 + (N - 1) * (3 + 11 + 2 + (N / 2) * (2 + 3 + 1)) = 3 * N * N + 12 * *N - 13

3.5. Тестирование

Реализовано функциональное тестирование отдельным файлом test.py. Полученные результаты функций сравниваются с контрольными значениями.

В таблице 3.1 представлены функциональные тесты для проверки работы программы.

Таблица 3.1. Функциональные тесты

Входные данные	Ожидаемый результат
0 0 0 0 0	0 0 0 0 0
1 2 3 4 5	1 2 3 4 5
5 4 3 2 1	1 2 3 4 5
4 2 3 5 1	1 2 3 4 5
3 2 3 2 3 2	2 2 2 3 3 3

Программа успешно прошла все тестовые случаи.

4 | Экспериментальная часть

В данном разделе приведены примеры работы программы и сравнительный анализ алгоритмов на основе экспериментальных данных.

4.1. Примеры работы

На рис. 4.1 представлено пример работы программы.

```
Введите размер массива n: 10
[20, 100, 59, -90, -75, 87, -17, -15, -33, 100]

Результат сортировки пузырьком с флагом:
[-90, -75, -33, -17, -15, 20, 59, 87, 100, 100]

Результат сортировки вставками:
[-90, -75, -33, -17, -15, 20, 59, 87, 100, 100]

Результат сортировки выбором:
[-90, -75, -33, -17, -15, 20, 59, 87, 100, 100]
```

Рис. 4.1. Результат сортировки массива длиной 10

4.2. Постановка эксперимента по замеру времени

Для произведения замеров времени выполнения реализации алгоритмов будет использована формула:

$$t = \frac{T}{N} \tag{4.1}$$

где t — среднее время выполнения алгоритма, N — количество замеров, T — время выполнения N замеров. Неоднократное измерение времени необходимо для получения более точного результа.

Количество замеров взято равным 100. Эксперимент проводится на квадратных матрицах, заполненных произвольными значениями.

Был проведен замер времени работы каждого из алгоритмов, результат представлен в Таблицах 4.1 - 4.2.

В таблице 4.1 представлены замеры времени работы алгоритмов в произвольном случае.

Таблица 4.1. Время работы алгоритмов в произвольном случае

длина	пузырьком, сек	вставками, сек	выбором, сек
100	0.00100114	0.00034327	0.00048422
250	0.00595716	0.00197251	0.00277946
500	0.03068482	0.00909589	0.01221364
750	0.08308200	0.02045400	0.02661800
1000	0.12647001	0.03678904	0.04858748

В таблице 4.2 представлены замеры времени работы алгоритмов в наилучшем и нахудшим случаях.

Таблица 4.2. Время работы алгоритмов в наилучшем и нахудшим случаях

сортировка	размер массива	время, сек
пузырьком (наилучший случай)	100	0.00072565
пузырьком (наихудший случай)	100	0.00131949
выбором (наилучший случай)	100	0.00053439
выбором (наихудший случай)	100	0.00065156
вставками (наилучший случай)	100	0.00002002
вставками (наихудший случай)	100	0.00064407
пузырьком (наилучший случай)	250	0.00401841
пузырьком (наихудший случай)	250	0.00943829
выбором (наилучший случай)	250	0.00286945
выбором (наихудший случай)	250	0.00333708
вставками (наилучший случай)	250	0.00004235
вставками (наихудший случай)	250	0.00453917
пузырьком (наилучший случай)	500	0.02004994
пузырьком (наихудший случай)	500	0.03530062
выбором (наилучший случай)	500	0.01109605
выбором (наихудший случай)	500	0.01114292
вставками (наилучший случай)	500	0.00009701
вставками (наихудший случай)	500	0.01583085
пузырьком (наилучший случай)	750	0.05675093
пузырьком (наихудший случай)	750	0.08493499
выбором (наилучший случай)	750	0.02583242
выбором (наихудший случай)	750	0.03231260
вставками (наилучший случай)	750	0.00018080
вставками (наихудший случай)	750	0.03757022
пузырьком (наилучший случай)	1000	0.08380121
пузырьком (наихудший случай)	1000	0.15293604
выбором (наилучший случай)	1000	0.04562732
выбором (наихудший случай)	1000	0.04601358
вставками (наилучший случай)	1000	0.00019316
вставками (наихудший случай)	1000	0.06719751

Вывод: При работе с упорядоченными или практически упорядоченными следует использовать сортировку вставкой (при выборе из трех предложенных алгоритмов). Сортировка выбором показывает стабильный результат, практически независимый от входных данных, в отличии от сортировки вставкой, время которой сильно разбросано. Время сортировки пузырьком имеет меньший разброс, однако из-за того, что верхний предел совпадает с пределом сортировки вставкой, мы можем сказать, что сортировка вставкой в среднем будет лучше сортировки пузырьком. С учетом посчитанных трудоемкостей алгоритмов сортировки, для массива больших размеров следует использовать алгоритм сортировки вставками.

Заключение

В ходе работы были изучены алгоритмы сортировки массива. Реализованы 3 алгоритма: пузырьком, вставка, выбор. Приведен программный код реализации этих алгоритмов.

Была подсчитана трудоемкость каждого из алгоритмов. А также было проведено сравнение их по времени и трудоемкости.

Цель работы достигнута. Получены практические навыки реализации алгоритмов сортировки, а также проведена исследовательская работа по вычислению трудоемкости алгоритмов.

Список литературы

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: 16.10.2020)

2. Описание алгоритмов сортировки и сравнение их производительности. [электронный ресурс] Режим доступа: https://habr.com/ru/post/335920/ (Дата обращения: 16.10.2020)

3. Официальный сайт Python, документация [электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html, свободный (Дата обращения: 16.09.20)