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: Алгоритмы сортировки

Дисциплина: Анализ алгоритмов

Студент ИУ7-54Б _____ Л.Е.Тартыков

(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ Л.Л. Волкова

(Подпись, дата) (И.О. Фамилия)

Москва, 2021

Содержание

Введение

При работе с большими объемами данных, расположенных в хаотичном порядке относительно друг друга, часто возникает проблема нахождения нужной записи по некоторому ключу. Например, нахождение в телефонной книге некоторого абонента, если при этом записи не находятся в алфавитном порядке по фамилии.

Для решения подобных задач существует такое решение как сортировка данных. Алгоритм сортировки - алгоритм, необходимый для упорядочивания элементов в списке. Сортировка может проводиться по какому-то критерию (ключу). Было разработано множество алгоритмов, которые различаются по трудоемкости и эффективности в связи с затрачиваемыми ими ресурсами ЭВМ. Целью лабораторной работы является изучение и реализация нерекурсивных алгоритмов сортировок. Для её достижения необходимо выполнить следующие задачи:

- изучение алгоритмов нерекурсивной сортировки;
- рассчитать трудоемкость каждого из выбранных алгоритмов;
- разработать данные алгоритмы;
- выполнить тестирование методом черного ящика;
- провести сравнительный анализ этих алгоритмов по затратам памяти и процессорному выполнению времени на основе экспериментальных данных.

1 Аналитический раздел

1.1 Понятие сортировки

Алгоритм сортировки — это алгоритм для упорядочения 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.

1.2 Критерии выбора алгоритма сортировки

К основным параметрам выбора необходимого алгоритма сортировки относят:

- временная сложность - описывает только то, как производительность алгоритма изменяется в зависимости от размера набора данных.;
- память — ряд алгоритмов требует выделения дополнительной памяти под временное хранение данных. При оценке не учитывается место, которое занимает исходный массив и независящие от входной последовательности затраты, например, на хранение кода программы;
- устойчивость - сортировка является устойчивой в том случае, если для любой пары элементов с одинаковым ключами, она не меняет их порядок в отсортированном списке (является важным критерием для баз данных).

1.3 Пузырьковая сортировка

Является одним из самых известных алгоритмов сортировки за счет своей простоты. Идея такой сортировки заключается в том, что все элементы массива сравниваются друг с другом. Они меняются местами в том случае, если предшествующий элемент больше последующего. Этот процесс повторяется до тех пор, пока перестановок в массиве не будет.

Данный алгоритм почти не применяется на практике ввиду своей низкой временной эффективности: он работает медленнее на больших данных. Исключение составляет сортировка малого числа элементов (примерно, до 10 штук), когда такой подход выигрывает по скорости выполнения алгоритмам с меньшей временной сложностью на больших данных.

1.4 Сортировка простыми вставками

Основная идея алгоритма сортировки вставками заключается в том, что исходный массив делится на две части: отсортированную и неотсортированную. В качестве отсортированной части считается первый элемент массива. Затем берется элемент из неотсортированной и добавляется в отсортированную так, чтобы упорядоченность в первой части не нарушилась. Такие дейсвтия продолжаются до тех пор, пока все элементы не окажутся на своих местах в отсортированной части массива.

1.5 Сортировка методом Шелла

Данный алгоритм является модификацией сортировки простыми вставками. Отличие такого метода заключается в том, что сначала выполняется сравнение элементов, находящихся друг от друга на некотором расстоянии. Изначально оно задается как d или N/2, где N - общее число элементов последовательности. На первом шаге каждая группа включает в себя два элемента, расположенных друг от друга на расстоянии N/2. На последующих шагах также происходит проверка и обмен, но

расстояния d при этом сокращается на d/2. Постепенно расстояни между элементами уменьшается, и на d=1 проход по массиву происходи в последний раз.

1.6 Вывод

В данном разделе были рассмотрены основные теоретические сведения о трех алгоритмах сортировки.

2 Конструкторский раздел

2.1 Модель оценки трудоемкости алгоритмов

Введем модель оценки трудоемкости.

1. Трудоемкость базовых операций. Пусть трудоемкость следующих операций равна 2.

$$*, /, //, \%, * =, / =$$

   Примем трудоемкость следующих операций равной 1.

$$=, +, -, + =, - =, ==, ! =, <, >, <=, >=, |, \&\&, ||, []$$

2. Трудоемкость цикла. Пусть трудоемкость цикла определяется по формуле (2.1).

$$f = f_{init} + f_{comp} + N_{iter} * (f_{in} + f_{inc} + f_{comp}) \tag{2.1}$$

   где:

   - f_{init} - трудоемкость инициализации переменной-счетчика;
   - f_{comp} - трудоемкость сравнения;
   - N_{iter} - номер выполняемой итерации;
   - f_{in} - трудоемкость команд из тела цикла;
   - f_{inc} -трудоемкость инкремента;
   - f_{comp} - трудоемкость сравнения.

3. Трудоемкость условного оператора.
   Пусть трудоемкость самого условного перехода равна 0, но она определяется по формуле (2.2).

$$f_{if} = f_{comp\_if} + \begin{cases} f_a \\ f_b \end{cases} \tag{2.2}$$

2.2 Вычисление трудоемкости алгоритмов

Пусть размер массивов во всех дальнейших вычислениях обозначается как N.

2.2.1 Трудоемкость алгоритма сортировки пузырьком

Трудоемкость этого алгоритма сортировки состоит из:

- трудоемкость внешнего цикла вычисляется по формуле (2.3);

$$f_i = \underset{=}{1} + \underset{<}{1} + N = 2 + N \tag{2.3}$$

- трудоемкость внутреннего цикла вычисляется по формуле (2.4).

$$f_j = \underset{i++}{2} + \underset{=}{1} + \underset{<}{1} + \underset{-}{1} + \frac{N-1}{2} * (\underset{j++}{2} + f_{if}) = 6 + \frac{N-1}{2} * (2 + f_{if}) \tag{2.4}$$

где вычисление худшего/лучшего случая определяется по формуле (2.5):

$$f_{if} = \underset{>}{1} + \underset{[\,]}{2} + \underset{+}{1} + \begin{cases} 0 \\ \underset{=}{3} + \underset{[\,]}{4} + \underset{+}{2} \end{cases} = 4 + \begin{cases} 0, \text{ лучший случай} \\ 9, \text{ худший случай} \end{cases} \tag{2.5}$$

Трудоемкость пузырька для лучшего случая (2.6):

$$f_{sum} = 3N^2 + 3N + 2 \approx O(N^2) \tag{2.6}$$

Трудоемкость пузырька для худшего случая (2.7):

$$f_{sum} = 7.5N^2 - 1.5N + 2 \approx O(N^2) \tag{2.7}$$

2.2.2 Трудоемкость алгоритма сортировки простыми вставками

Трудоемкость этого алгоритма сортировки состоит из:

- трудоемкость внешнего цикла вычисляется по формуле (2.8);

$$f_i = \underset{=}{1} + \underset{<}{1} + N = 2 + N \tag{2.8}$$

- трудоемкость тела внутреннего цикла определяется по формуле (2.9).

$$f_{body} = \underset{i++}{2} + \underset{=}{2} + \underset{[\,]}{1} + \underset{-}{1} + f_{if} + \underset{[\,]}{1} + \underset{+}{1} + \underset{=}{1} \tag{2.9}$$

  где f_{if} вычисляется по формуле (2.10):

$$f_{if} = \underset{>=}{1} + \underset{\&\&}{1} + \underset{<}{1} + \underset{[\,]}{1} + \begin{cases} 0 \\ \underset{=}{3} + \underset{[\,]}{4} + \underset{+}{2} \end{cases} = 4 + \begin{cases} 0, \text{ лучший случай} \\ \frac{N-1}{4} * (\underset{[\,]}{2} + \underset{+}{1} + \underset{-=}{1} + \underset{=}{1}), \\ \text{худший случай} \end{cases} \tag{2.10}$$

  Трудоемкость алгоритма сортировки простыми вставками для лучшего случая (2.11):

$$f_{sum} = 13N + 2 \approx O(N) \tag{2.11}$$

  Трудоемкость алгоритма сортировки простыми вставками для худшего случая (2.12):

$$f_{sum} = \frac{5N^2}{4} - \frac{47N}{4} + 2 \approx O(N^2) \tag{2.12}$$

2.2.3 Трудоемкость алгоритма сортировки методом Шелла

Пусть шаг в методе Шелла будет равен N/2, где N - размер массива. Трудоемкость этого алгоритма сортировки состоит из:

- трудоемкость внешнего цикла вычисляется по формуле (2.13);

$$f_i = \underset{//=}{2} + \underset{=}{1} + \underset{<}{1} + \underset{i++}{2} + \underset{=}{1} + \frac{N}{2} * (\underset{i++}{2} + \underset{=}{1} + \underset{>=}{1} + \underset{[\,]}{2} + \underset{>}{1} + \underset{+}{1} + \underset{\&\&}{1} + \frac{N}{2}(f_{if})) \tag{2.13}$$

- трудоемкость f_{if} вычисляется по формуле (2.14).

$$f_{if} = \underset{=}{3} + \underset{[\,]}{4} + \underset{+}{2} + \underset{-=}{1} \tag{2.14}$$

Трудоемкость алгоритма сортировки методом Шелла для лучшего случая (2.15):

$$f_{sum} = \frac{9N}{2} + 7 \approx O(N) \tag{2.15}$$

Трудоемкость алгоритма сортировки методом Шелла для худшего случая (2.16):

$$f_{sum} = \frac{10N^2}{4} + \frac{9N}{2} + 7 \approx O(N^2) \tag{2.16}$$

2.3 Схемы алгоритмов сортировок

Ниже представлены следующие схемы алгоритмов:

- рис. 2.1 - схема алгоритма сортировки пузырьком;

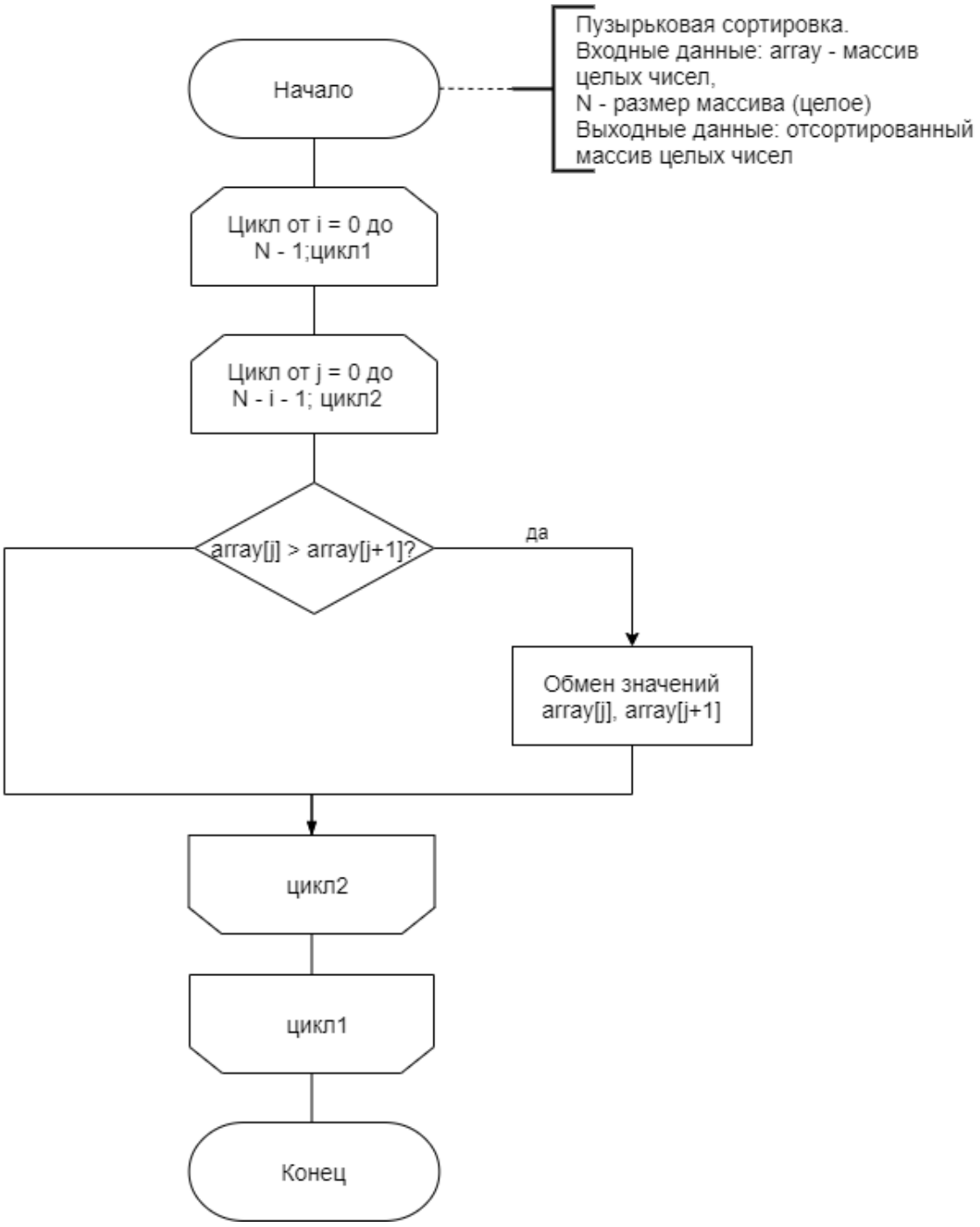

Рисунок 2.1 – Схема алгоритма сортировки пузырьком.

- рис. 2.2 - схема алгоритма сортировки простыми вставками;

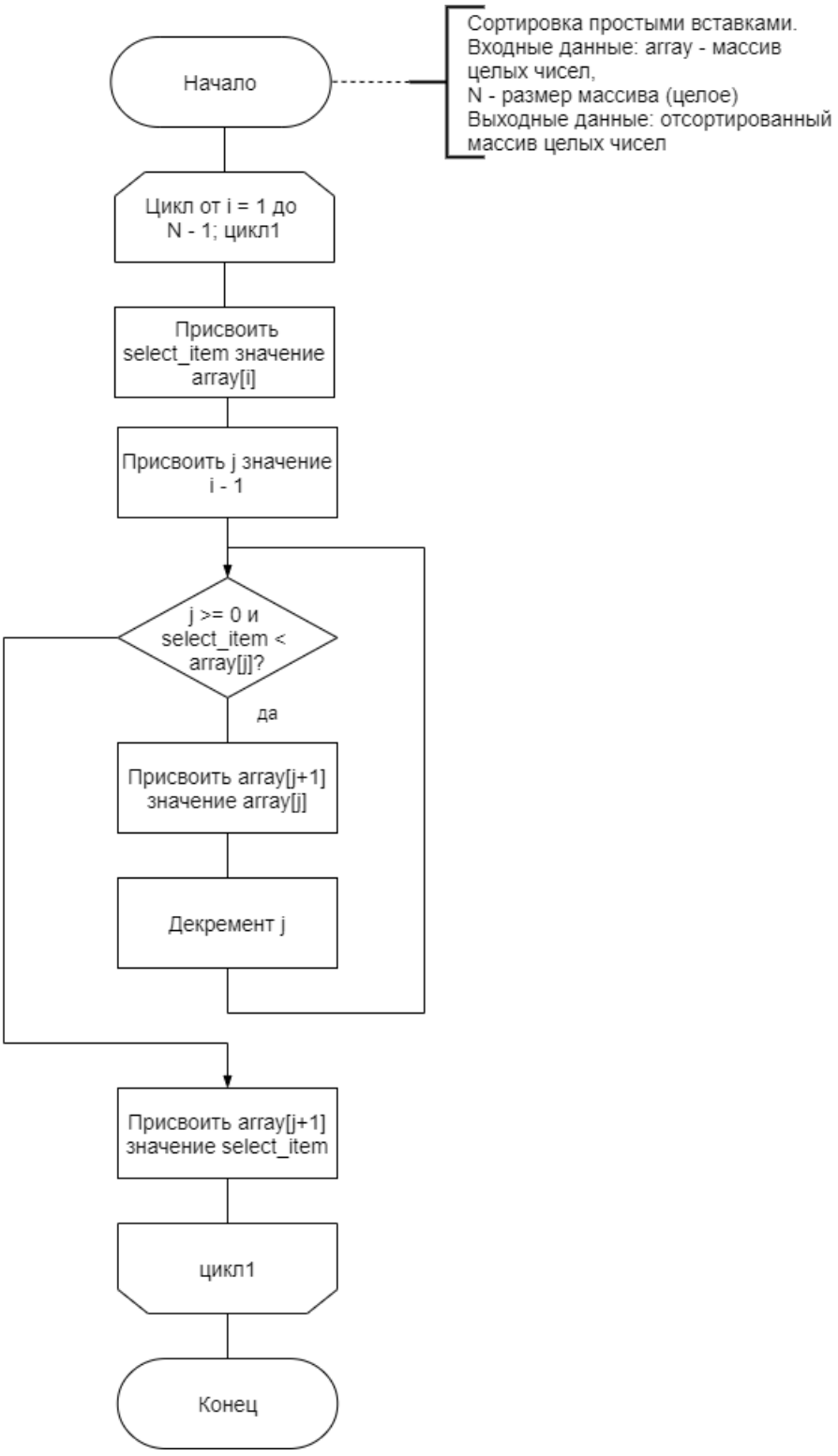

Рисунок 2.2 – Схема алгоритма сортировки простыми вставками.

- рис. 2.3 - схема алгоритма сортировки методом Шелла.

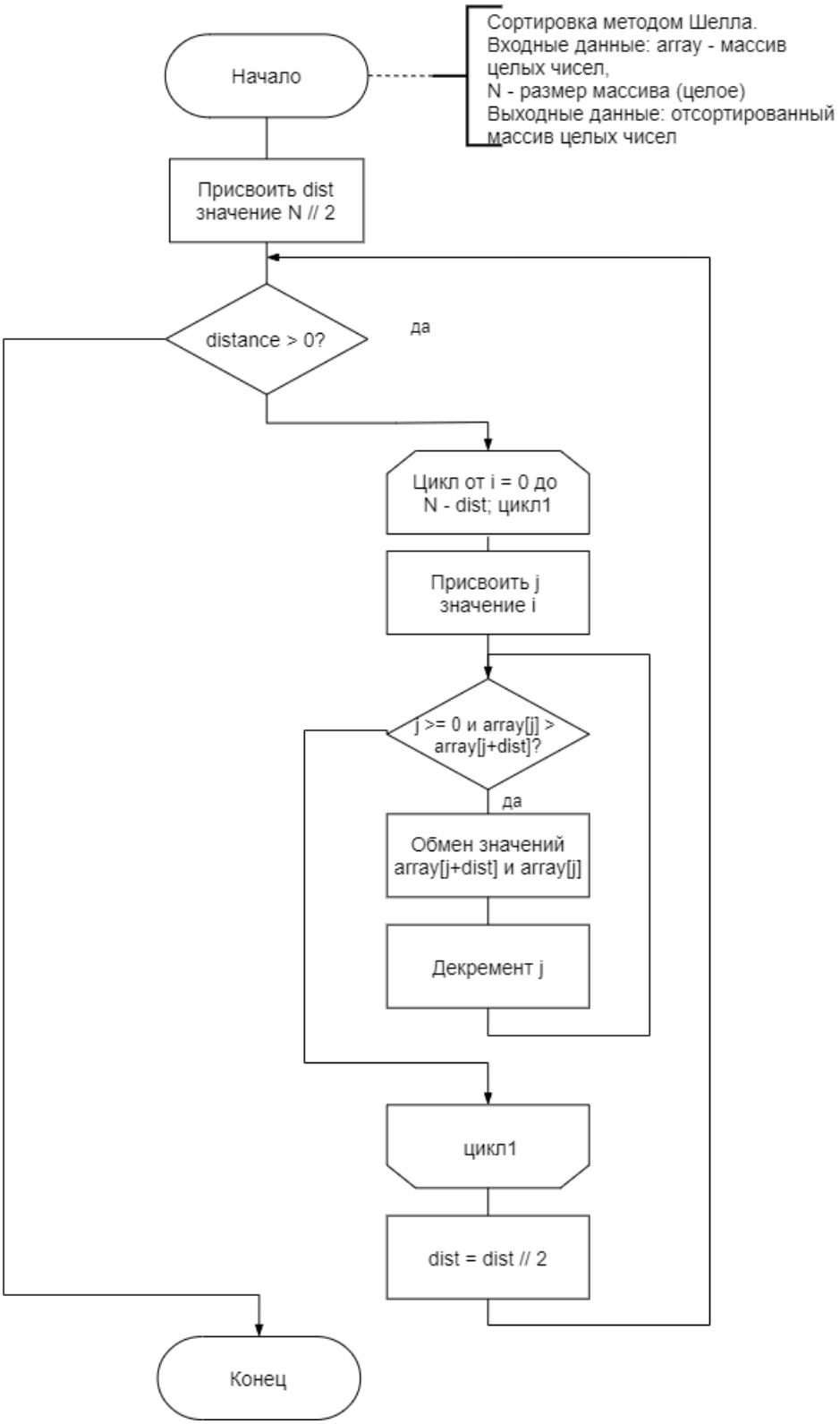

Рисунок 2.3 – Схема алгоритма сортировки методом Шелла.

2.4 Вывод

На основе теоретических данных, полученных в аналатическом разделе, были построены схемы нужных алгоритмов. Были получена трудоемкость каждого из трех алгоритмов для анализа худшего и лучшего случаев асимптотической сложности. Полученная трудоемкость приведена в таблице 2.1:

Таблица 2.1 – Сложность алгоритмов.

Алгоритм	Сложность	
	Лучший случай	Худший случай
Пузырек	$O(N^2)$	$O(N^2)$
Простые вставки	$O(N)$	$O(N^2)$
Шелл	$O(N)$	$O(N^2)$

3 Технологический раздел

3.1 Требования к программному обеспечению

Программа должна отвечать следующим требованиям:

- на вход программе подается массив целых чисел и размерность этого массива;
- осуществляется выбор алгоритма сортировки из меню;
- на выход программа выдает отсортированный массив целых чисел;
- программа должна визуализировать графики времени выполнения алгоритмов при выборе соответствующего пункта меню.

3.2 Выбор средств реализации

Для реализации алгоритмов в данной лабораторной работе был выбран язык программирования Python 3.9.7[1]. Он является кроссплатформенным. Имеется опыт разработки на этом языке. В качестве среды разработки был использован Visual Studio Code[2], так как в нем можно работать как на операционной системе Windows, так и на дистрибутивах Linux. При замере процессорного времени был использован модуль time[3].

3.3 Листинги программ

Ниже представлены листинги разработанных алгоритмов сортировки.

Листинг 3.1 – Программный код сортировки пузырьком.

```
def bubble_sort(array: list[int], count: int) -> list[int]:
    for i in range(count):
        for j in range(count - i - 1):
            if array[j] > array[j + 1]:
                array[j], array[j + 1] = array[j + 1], array[j]
```

Листинг 3.2 – Программный код сортировки вставками.

```
def insert_sort(array: list[int], count: int) -> list[int]:
    for i in range(1, count):
        select_item = array[i]
        j = i - 1
        while j >= 0 and select_item < array[j]:
            array[j+1] = array[j]
            j -= 1
        array[j + 1] = select_item
```

Листинг 3.3 – Программный код сортировки методом Шелла.

```
def shell_sort(array: list[int], count:int) -> list[int]:
    distance = count // 2
    while distance > 0:
        for i in range(count - distance):
        j = i
        while j >= 0 and array[j] > array[j + distance]:
            array[j + distance], array[j] = array[j], array[j + distance
                ]
            j -= 1
        distance //= 2
```

3.4 Вспомогательные функции

На листингах представлены программные модули, которые используются в данных функциях:

Листинг 3.4 – Программный код формирования упорядоченного массива целых чисел.

```
def form_order_array(count: int) -> list[int]:
    array = list()
    for i in range(count):
        array.append(i)

    return array
```

Листинг 3.5 – Программный код формирования массива случайных целых чисел.

```
def form_random_array(count: int) -> list[int]:
    array = list()
    for i in range(count):
        array.append(randint(MIN_NUMBER, MAX_NUMBER))

    return array
```

Листинг 3.6 – Программный код формирования массива целых чисел упорядоченных в обратном порядке.

```
    def form_random_array(count: int) -> list[int]:
    array = list()
    for i in range(count):
    array.append(randint(MIN_NUMBER, MAX_NUMBER))

    return array
```

3.5 Тестирование

Для тестирования используется метод черного ящика. В данном разделе приведена таблица 3.1, в которой указаны классы эквивалентностей тестов:

Таблица 3.1 – Тесты

№	Описание теста	Вход	Результат		
			Пузырек	Вставки	Шелл
1	Один элемент	1	1	1	1
2	Отсортированный массив	1,2,3,4,5	1,2,3,4,5	1,2,3,4,5	1,2,3,4,5
2	Отсортированный в обратном порядке	5,4,3,2,1	1,2,3,4,5	1,2,3,4,5	1,2,3,4,5
2	Случайные числа	-8,3,1,6,-2	-8,-2,1,3,6	-8,-2,1,3,6	-8,-2,1,3,6

3.6 Вывод

В данном разделе был выбран язык программирования, среда разработки. Реализованы функции, описанные в аналитическом разделе, и проведено их тестирование методом черного ящика по таблице 2.1.

4 Исследовательский раздел

4.1 Технические характеристики

Технические характеристии устройства, на котором выполнялось тестирование:

- операционная система: Windows 10 Pro;
- память: 8 GiB;
- процессор: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz.

Тестирование проводилось на ноутбуке, который был подключен к сети питания. Во время проведения тестирования ноутбук был нагружен только встроенными приложениями окружения, самим окружением и системой тестирования.

4.2 Временные харастеристики выполнения

Ниже был проведен анализ времени работы алгоритмов. Исходными данными является массив. Единичные замеры выдадут крайне маленький результат, поэтому проведем работу каждого алгоритма n = 1000 раз и поделим на число n. Получим среднее значение работы каждого из алгоритмов.

Выполним анализ для случая, когда массив целых чисел упорядочен по возрастанию. Результат приведен на рис 4.1.

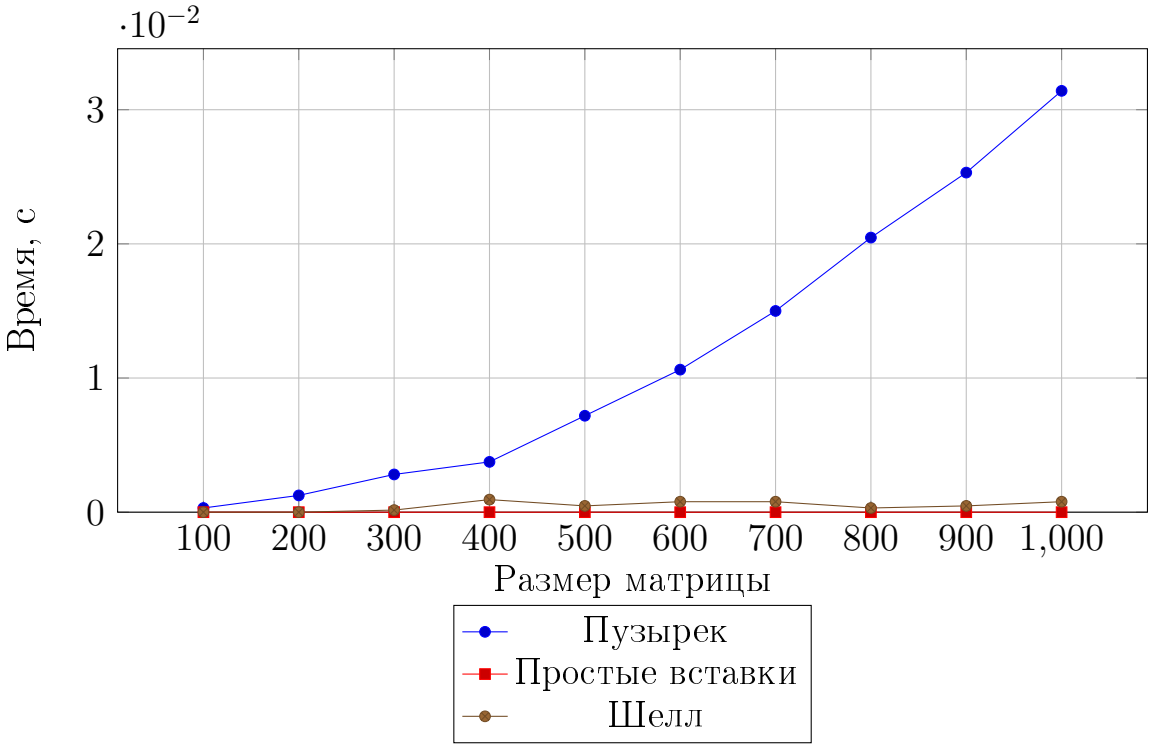

Рисунок 4.1 – График зависимости времени работы алгоритмов сортировки для лучшего случая.

Как видно из результатов, алгоритм сортировки пузырьком работает медленне, чем сортировка вставкой или методом Шелла. Две последние сортировки работают примерно с одинаковой скоростью выполнения.

Выполним анализ для случая, когда массив целых чисел имеет случайные значения. Результат приведен на рис. 4.2.

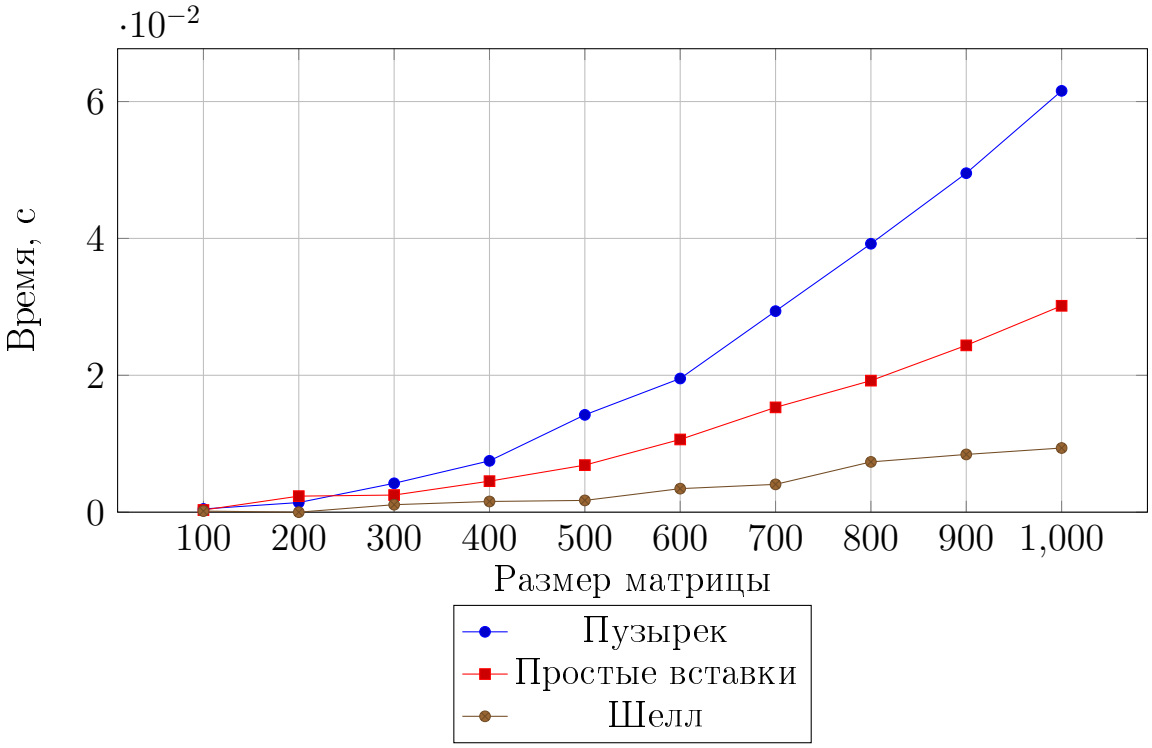

Рисунок 4.2 – График зависимости времени работы алгоритмов сортировки для обычного случая.

Как видно из результатов алгоритм пузырька вновь работает медленнее двух остальных алгоритмов. Метод простых вставок уступает по скорости выполнения сортировки методу Шелла.

Выполним анализ для случая, когда массив целых чисел отсортирован в обратном порядке. Результат приведен на рис. 4.3.

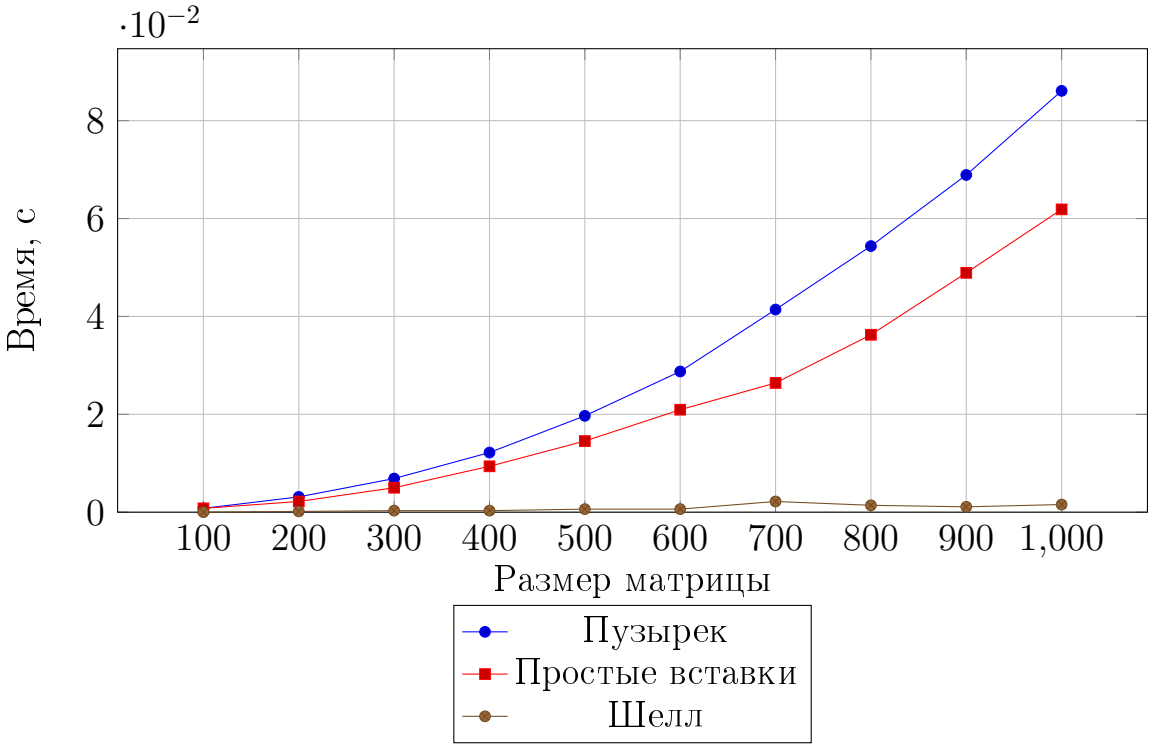

Рисунок 4.3 – График зависимости времени работы алгоритмов сортировки для худшего случая.

Как видно из результатов в данном случае алгоритм простых вставок работает медленее по сравнению с массивом, который имеет случайные целые числа. Метод Шелла уменьшил скорость выполнения по сравнению с предыдущим анализом времени выполнения.

4.3 Вывод

Алгоритм сортировки методом пузырька работает медленнее двух других рассмотренных алгоритмов, причем на больших размерах массива скорость выполнения этого алгоритма увеличивается значительно. Это проявляется также и при любом виде хранения целых чисел в массиве (упорядоченный, случайные числа, упорядоченный в обратном порядке).

Алгоритм сортировки простыми вставками работает также быстро, как и метод Шелла при упорядоченных данных в массиве. Метод Шелла работает значительно быстрее всех других приведенных алгоритмов, даже когда массив отсортирован в обратном порядке.

Заключение

В ходе выполнения лабораторной работы были рассмотрены алгоритмы сортировки. Были выполнено описание каждого из этих алгоритмов, приведены соотвествующие математические расчёты. Была рассчитана трудоемкость каждого из этого алгоритмов. При исследовании трудоемкости и времени выполнения алгоритмов сортировок можно сделать следующие выводы: временная сложность алгоритма не эквивалентна времени выполнения. Это можно хорошо увидеть на исследовании скорости выполнения алгоритма пузырька и метода Шелла. В худшем случае, когда на вход алгоритму подается массив целых чисел, отсортированный в обратном порядке, метод пузырька и метод Шелла имеет одинаковую вычислительную сложности, но второй работает значительно быстрее по сравнению с первым алгоритмом. Самым медленным из приведенных алгоритмов является алгоритм сортировки пузырьком, самым быстрым - сортировка простыми вставками.

Список литературы

[1] Python. [Электронный ресурс]. Режим доступа: https://docs.python.org/3/ (дата обращения: 27.09.2021).

[2] Documentation for Visual Studio Code. [Электронный ресурс]. Режим доступа: https://code.visualstudio.com/docs (дата обращения: 27.09.2021).

[3] time — Time access and conversions — Python 3.9.7 documentation. [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html (дата обращения: 27.09.2021).