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 ИУ, Информатика и системы управления

КАФЕДРА ИУ7, Программное обеспечение ЭВМ и информационные технологии

ЛАБОРАТОРНАЯ РАБОТА №2

ПО ДИСЦИПЛИНЕ

“Анализ алгоритмов”

Студент ИУ7-54Б _____ **А.А. Андреев**
(Группа) (Подпись, дата) (И.О.Фамилия)

Преподаватель _____ **Л.Л. Волкова**
(Подпись, дата) (И.О.Фамилия)

2021 г.

Оглавление

Введение.

Данная лабораторная работа посвящена исследованию и сравнению алгоритмов умножения матриц.

Алгоритмы умножения матриц широко применяются в задачах, использующих линейную алгебру, компьютерной графики, физики и экономики.

Цель данной лабораторной работы: исследование и сравнение трех алгоритмов умножения матриц: стандартного алгоритма умножения матриц, алгоритма Винограда и модифицированный алгоритм Винограда.

Задачи данной лабораторной работы:

1. Изучение и реализация трех алгоритмов умножения матриц: стандартного алгоритма умножения матриц, алгоритма Винограда и модифицированный алгоритм Винограда;
2. Проведение сравнительного анализа трудоемкости алгоритмов;
3. Проведение сравнительного анализа алгоритмов на основе экспериментальных данных;

1 Аналитическая часть

В данном разделе будут описаны нерекурсивные алгоритмы сортировки пузырьком, выбором, вставками.

1.1 Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется перестановка элементов. Проходы по массиву повторяются
N - 1 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 Сортировка выбором

Алгоритм состоит из повторяющихся операций сравнения и перемещения элементом из рассматриваемого массива: сначала необходимо найти номер минимального значения в текущем списке, после чего произвести обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции), и только потом отсортировать хвост списка, исключив из рассмотрения уже отсортированные элементы

1.3 Сортировка вставками

Алгоритм состоит из операций последовательного просмотра элементов не рассмотренной и уже отсортированной последовательности: на каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки и вставляем.

Для оптимизации работы алгоритма используют методы разделения последовательности на несколько частей.

1.4 Вывод

В аналитической части были описаны: нерекурсивные алгоритмы сортировки пузырьком, выбором, вставками.

2 Конструкторская часть.

В данном разделе будет приведены блок-схемы алгоритмов, описанных в аналитическом разделе п.1.

2.1 Схемы алгоритмов

Схемы алгоритмов пузырьком, выбором, вставками доступны на рисунках 1-3.

2.1.1. Сортировка пузырьком

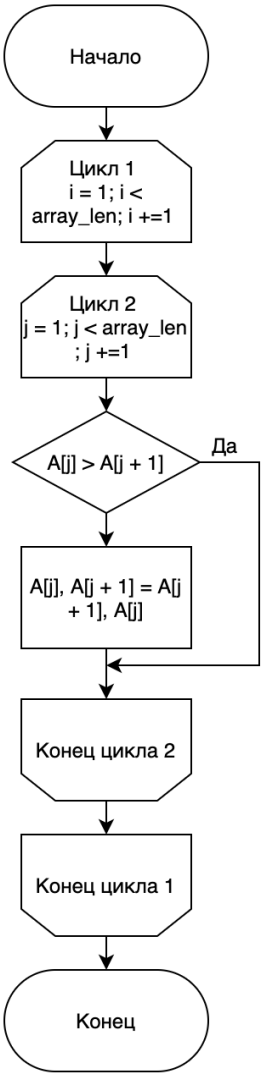

Рисунок 1: Схема итеративного алгоритма сортировки пузырьком

2.1.2. Сортировка выбором

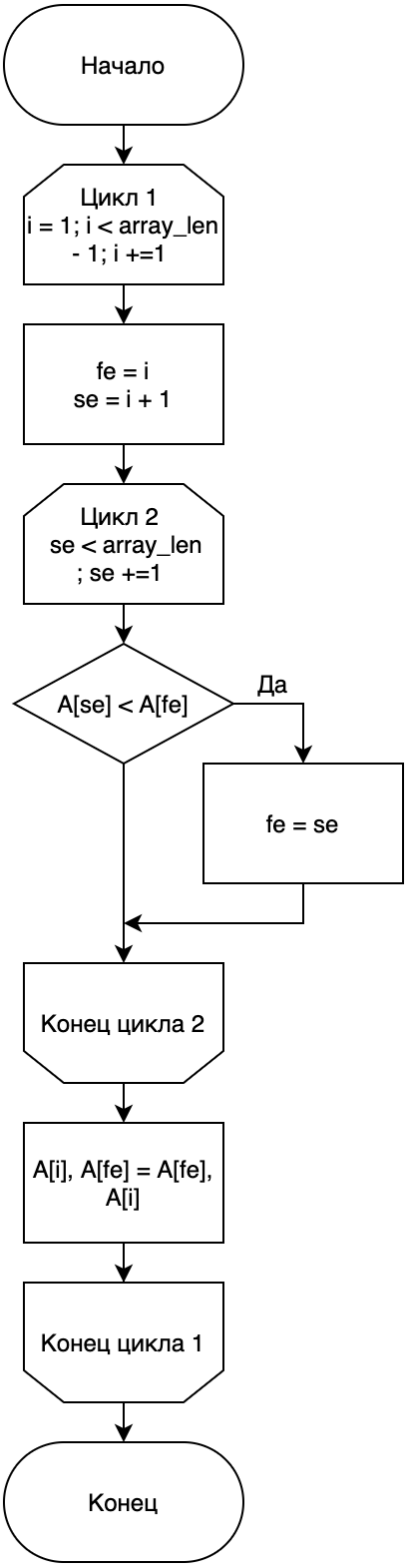

Рисунок 2: Схема итеративного алгоритма сортировки выбором

2.1.3. Сортировка вставками

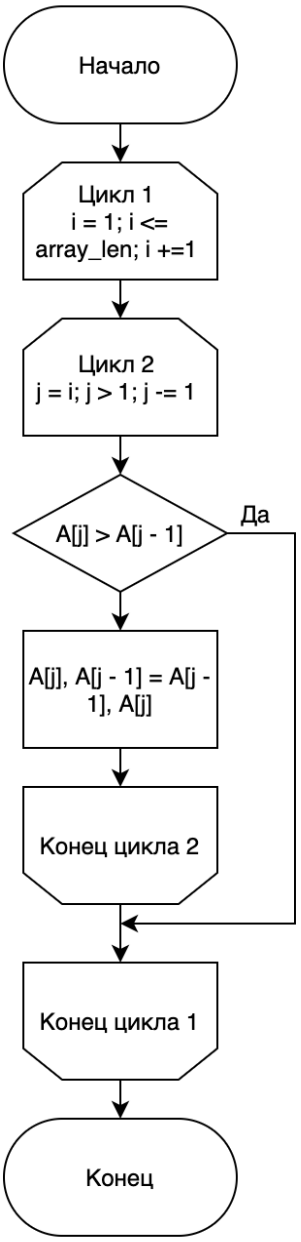

Рисунок 3: Схема итеративного алгоритма сортировки вставками

2.2 Модель вычислений

Введенная модель вычислений трудоемкости доступна в Таблице 1.

Таблица 1: Введенная модель вычислений

№	Операции	Трудоемкость
1	$+, -, /, \%, ==, != , <, >, <=, >=, [], ++, --$	1
2	Условие	$f_{if} = f_{\text{услов.}} + f_{\text{выполняемая операция}}$
3	Цикла	$f_{for} = f_{\text{иниц.}} + f_{\text{сравн.}} + N(f_{\text{тела}} + f_{\text{инк.}}$
4	Вызов функции	0

2.3 Трудоемкость алгоритмов

Обозначим размер массива N, а сам массив A.

2.3.1 Сортировка пузырьком

На основе вычислений трудоемкости составляющих алгоритма сортировки пузырьком в Таблице 2 сделано суммирование общей трудоемкости в Таблице 3.

Таблица 2: Составляющие трудоемкости алгоритма сортировки пузырьком

№	Операции	Трудоемкость
1	Сравнение, увеличение внешнего цикла *от 1 до N*	$2 + 2 \cdot (N - 1)$
2	Итерации внутренних циклов *от 1 до N - 1*	$3 \cdot (N - 1) + 0.5 \cdot N \cdot (N - 1) \cdot (3 + f_{y}$
3	Внутреннее условие	В лучшем: $4 + 0$ В худшем: $4 + 9$

Таблица 3: Трудоемкость алгоритма сортировки пузырьком

Случай	Трудоемкость
Лучший	$7/2 \cdot N^2 + 3/2 \cdot N - 3 \approx 7/2 \cdot N^2 = O(N^2)$
Худший	$8 \cdot N^2 - 8 \cdot N - 3 \approx 8 \cdot N^2 = O(N^2)$

2.3.2 Сортировка выбором

На основе вычислений трудоемкости составляющих алгоритма сортировки выбором в Таблице 4 сделано суммирование общей трудоемкости в Таблице 5.

Таблица 4: Составляющие трудоемкости алгоритма сортировки выбором

№	Операции	Трудоемкость
1	Сравнение, увеличение внешнего цикла *от 1 до N*	$2 + 2 \cdot (N - 1)$
2	Итерации внутренних циклов *от 1 до N - 1*	$3 \cdot (N - 1) + 0.5 \cdot N \cdot (N - 1) \cdot (3 + f$ у
3	Внутреннее условие	В лучшем: $4 + 0$ В худшем: $4 + 3 \cdot (N - 1) + N \cdot (N - 1)/2 \cdot (3 + f$ усл

Таблица 5: Трудоемкость алгоритма сортировки выбором

Случай	Трудоемкость
Лучший	$13 \cdot N + 10 \cdot N \approx 13 \cdot N = O(N)$
Худший	$4.5 \cdot N^2 + 10 \cdot N - 13 \approx 4 \cdot N^2 = O(N^2)$

2.3.3 Сортировка вставками

На основе вычислений трудоемкости составляющих алгоритма сортировки вставками в Таблице 6 сделано суммирование общей трудоемкости в Таблице 7.

Таблица 6: Составляющие трудоемкости алгоритма сортировки вставками

№	Операции	Трудоемкость
1	Сравнение, увеличение внешнего цикла *от 1 до N*	$2 + 2 \cdot (N - 1)$
2	Итерации внутренних циклов *от 1 до N - 1*	$3 \cdot (N - 1) + 0.5 \cdot N \cdot (N - 1) \cdot (3 + f$ у
3	Внутреннее условие	В лучшем: $4 + 0$ В худшем: $4 + 3 \cdot (N - 1) + N \cdot (N - 1)/2 \cdot (3 + f$ усл

Таблица 7: Трудоемкость алгоритма сортировки вставками

Случай	Трудоемкость
Лучший	$13 \cdot N + 10 \cdot N \approx 13 \cdot N = O(N)$

Худший	$4.5 \cdot N^2 + 10 \cdot N - 13 \approx 4 \cdot N^2 = O(N^2)$

2.4 Вывод

Блок-схемы в данном разделе позволяют перейти к технологической части - непосредственно к программной реализации решения.

3 Технологическая часть.

В данном разделе будут рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач.

3.1 Требования к программному обеспечению

Программное обеспечение должно реализовывать поставленную на лабораторную работу задачу. Интерфейс для взаимодействия с программой - командная строка. Программа должна сравнение работы трех нерекурсивных алгоритмов сортировки пузырьком, выбором, вставками, показывать потраченное на это время, строить графики.

3.2 Выбор и обоснование языка и среды программирования.

Для разработки данной программы применён язык Python 3 с библиотекой time.clock() [4] для вычисления времени работы процессора, чтобы расширить знания в области данного языка программирования.

3.3 Реализация алгоритмов

В листингах 1-4 приведена реализация алгоритмов нахождения расстояния Левенштейна и Дамерау-Левенштейна.

Программа была реализована в парадигме ООП [2], где в базовый класс был вынесен объект Sort (Листинг 1-3), внутри него с доступом protected, используемая алгоритмами итеративная функция проверки последовательности на отсортированность *is_already_sorted()*.

Наследуемые объекты итеративной сортировки пузырьком (Листинг 4-5), выбором (Листинг 6-7), вставками (Листинг 8-9) имеют публичную функцию получения времени работы *get_time()* для измерения времени работы алгоритмов;

Листинг 1: Базовый класс Sort, Часть 1

```
1. # Базовый класс сортировок
2. class Sort:
3.     # Затраченное время процессора на сортировку
4.     _time = None
5.
6.     # Статус ошибки сортировки
7.     _status = None
8.
9.     # Защищенные поля с наследуемым уровнем доступа
10.       _number_of_elements = None
11.       _elements = None
12.
13.       # Инициализация базового класса сортировки
14.       def __init__(self, number_of_elements, elements):
```

Листинг 2: Базовый класс Sort, Часть 2

```
15.             #  Назначение защищенных полей
16.             self._number_of_elements = number_of_elements
17.             self._elements = elements
18.             self._time = config.START_TIME_ZERO_VALUE
19.
20.             # Назначение статуса сортировки
21.             self._status = self._is_input_data_correct()
22.
23.         # Общая функция сортировки
24.         def sort(self):
25.
26.             # Базовый класс не имеет возможности сортировки
27.             self._status = config.ERROR_STATUS
28.
29.         # Верификация вводимых на сортивроку данных
30.         def _is_input_data_correct(self):
31.             status = config.SUCCESS_STATUS
32.
33.             # Проверка заявленных размеров массива
34.             if len(self._elements) != self._number_of_elements:
35.                 status = config.ERROR_STATUS
36.
37.             return status
38.
39.         # Проверка на существующую отсортированность
40.         def is_already_sorted(self):
41.             is_already_sorted_status = config.ERROR_STATUS
42.
43.             if self._status == config.SUCCESS_STATUS:
44.                 if self._number_of_elements ==
    config.ONE_ELEMENT:
45.                     is_already_sorted_status =
    config.SUCCESS_STATUS
46.
47.                 for number_of_checked_symbols in
    range(self._number_of_elements - 1):
48.                     if self._elements[number_of_checked_symbols +
    1] < self._elements[number_of_checked_symbols]:
49.                         return is_already_sorted_status
50.
51.                 is_already_sorted_status = config.SUCCESS_STATUS
52.
53.             return is_already_sorted_status
54.
55.         # Получение статуса сортировки
56.         def get_status(self):
57.             return self._status
58.
59.         # Получение количества элементов
60.         def get_number_of_elements(self):
61.             if self._status == config.SUCCESS_STATUS:
62.                 return self._number_of_elements
63.             return config.ZERO_ELEMENTS
```

Листинг 3: Базовый класс Sort, Часть 3

```
64.             # Получение массива элементов
65.         def get_elements(self):
66.             if self._status == config.SUCCESS_STATUS:
67.                 return self._elements
68.             return config.ZERO_ELEMENTS
69.
70.         # Получение времени выполнения сортировки
71.         def get_time(self):
72.             return self._time
```

Листинг 4: Наследуемый объект сортировки пузырьком, Часть 1

```
1. # Наследуемый объект сортировки Пузырьком
2. class BubbleSort(Sort):
3.     _temp_number_of_elements = None
4.     _temp_elements = None
5.
6.     def sort(self):
7.         if self._status == config.SUCCESS_STATUS:
8.             self._sort_proc()
9.
10.        def _swap(self, first_element_position,
   second_element_position):
11.
12.            # Меняемое значение подвешено
13.            temp_element =
   self._temp_elements[first_element_position]
14.
15.            # Непосредственная смена позиций
16.            self._temp_elements[first_element_position] =
   self._temp_elements[second_element_position]
17.            self._temp_elements[second_element_position] =
   temp_element
18.
19.        def _sort_proc(self):
20.            # Установка условно виртуальных переменных
21.            self._temp_number_of_elements =
   self._number_of_elements
22.            self._temp_elements = self._elements
23.
24.            while self.is_already_sorted() !=
   config.SUCCESS_STATUS:
25.                for number_of_checked_elements in
   range(self._temp_number_of_elements - 1):
26.                    if
   self._temp_elements[number_of_checked_elements] >
   self._temp_elements[
27.                            number_of_checked_elements + 1]:
28.                        self._swap(number_of_checked_elements,
   number_of_checked_elements + 1)
```

Листинг 5: Наследуемый объект сортировки пузырьком, Часть 2

```
29.
30.             # Обратный переход от виртуальных адресов к реальным
31.             self._elements = self._temp_elements
32.
33.         def get_time(self):
34.             t_0 = clock()
35.             self.sort()
36.             t_1 = clock()
37.
38.             self._time = t_1 - t_0
39.
40.             return self._time
```

Листинг 6: Наследуемый объект сортировки выбором, Часть 1

```
1. # Наследуемый объект Сортировки выбором
2. class SelectionSort(Sort):
3.     _temp_number_of_elements = None
4.     _temp_elements = None
5.
6.     def sort(self):
7.         self._temp_number_of_elements =
   self._number_of_elements
8.         self._temp_elements = self._elements
9.
10.            if self._status == config.SUCCESS_STATUS:
11.                self._sort_proc(self._temp_elements)
12.
13.        def _sort_proc(self, _temp_elements):
14.
15.            if self.is_already_sorted != config.SUCCESS_STATUS:
16.                number_of_checked_elements = 0
17.                while number_of_checked_elements <
   self._temp_number_of_elements - 1:
18.                    first_element = number_of_checked_elements
19.                    second_element = first_element + 1
20.                    while second_element <
   self._number_of_elements:
21.                        if self._temp_elements[second_element] <
   self._temp_elements[first_element]:
22.                            first_element = second_element
23.                        second_element += 1
24.
25.
   self._temp_elements[number_of_checked_elements],
   self._temp_elements[first_element] = self._temp_elements[
26.
   first_element], \
27.
   self._temp_elements[
28.
   number_of_checked_elements]
```

Листинг 7: Наследуемый объект сортировки выбором, Часть 2

```
29.
30.                     number_of_checked_elements += 1
31.
32.             self._elements = self._temp_elements
33.
34.         def _swap(self, first_element_position,
    second_element_position):
35.
36.             # Меняемое значение подвешено
37.             temp_element =
    self._temp_elements[first_element_position]
38.
39.             # Непосредственная смена позиций
40.             self._temp_elements[first_element_position] =
    self._temp_elements[second_element_position]
41.             self._temp_elements[second_element_position] =
    temp_element
42.
43.         def get_time(self):
44.             t_0 = clock()
45.             self.sort()
46.             t_1 = clock()
47.
48.             self._time = t_1 - t_0
49.
50.             return self._time
```

Листинг 8: Наследуемый объект сортировки вставками, Часть 1

```
1. # Наследуемый объект сортировки вставками
2. class InsertionSort(Sort):
3.     _temp_number_of_elements = None
4.     _temp_elements = None
5.
6.     def sort(self):
7.         if self._status == config.SUCCESS_STATUS:
8.             self._sort_proc()
9.
10.         def _swap(self, first_element_position,
    second_element_position):
11.
12.             # Меняемое значение подвешено
13.             temp_element =
    self._temp_elements[first_element_position]
14.
15.             # Непосредственная смена позиций
16.             self._temp_elements[first_element_position] =
    self._temp_elements[second_element_position]
17.             self._temp_elements[second_element_position] =
    temp_element
```

Листинг 8: Наследуемый объект сортировки вставками, Часть 2

```
18.
19.         def _sort_proc(self):
20.             # Установка условно виртуальных переменных
21.             self._temp_number_of_elements =
    self._number_of_elements
22.             self._temp_elements = self._elements
23.
24.             # Проверка на наличие отсортированности массива
25.             if self.is_already_sorted() != config.SUCCESS_STATUS:
26.
27.                 # Рзабиваем исходный массив на отсортированный
    подмассив и неотсортированный
28.                 for number_of_checked_elements in range(1,
    self._temp_number_of_elements):
29.                     key =
    self._temp_elements[number_of_checked_elements]
30.                     previous_to_check_element =
    number_of_checked_elements - 1
31.
32.                     # Выполняем вставку путем перемещения
    элемента внутри отсортированного подмассива
33.                     while previous_to_check_element >= 0 and key
    < self._temp_number_of_elements[previous_to_check_element]:
34.
    self._temp_number_of_elements[previous_to_check_element + 1] =
    self._temp_number_of_elements[
35.                                 previous_to_check_element]
36.                         previous_to_check_element -= 1
37.
38.             # Обратный переход от виртуальных адресов к реальным
39.             self._elements = self._temp_elements
40.
41.         def get_time(self):
42.             t_0 = clock()
43.             self.sort()
44.             t_1 = clock()
45.
46.             self._time = t_1 - t_0
47.
48.             return self._time
```

3.4 Тестовые данные

Тестовые данные, на которых было протестировано разработанное программное обеспечение, представлено в Таблице 1.

Таблица 8: Тестовые данные

№	Входной массив	Результат	Ожидаемый результат
1	1, 2, 3, 4, 5, 6, 7, 8	1, 2, 3, 4, 5, 6, 7, 8	1, 2, 3, 4, 5, 6, 7, 8
2	8, 7, 6, 5, 4, 3, 2, 1	8, 7, 6, 5, 4, 3, 2, 1	8, 7, 6, 5, 4, 3, 2, 1
3	0, 0, 0, 0, 0	0, 0, 0, 0, 0	0, 0, 0, 0, 0
4	Пустой массив	Пустой массив	Пустой массив
5	1	1	1
6	1, 990, 12, 345, -95	-95, 1, 12, 345, 990	-95, 1, 12, 345, 990

3.5 Вывод

В данном разделе были рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, приведены результаты работы программы на тестовых данных.

4 Исследовательская часть.

4.1. Демонстрация работы программы

Пример работы программы представлен на рисунке 6.

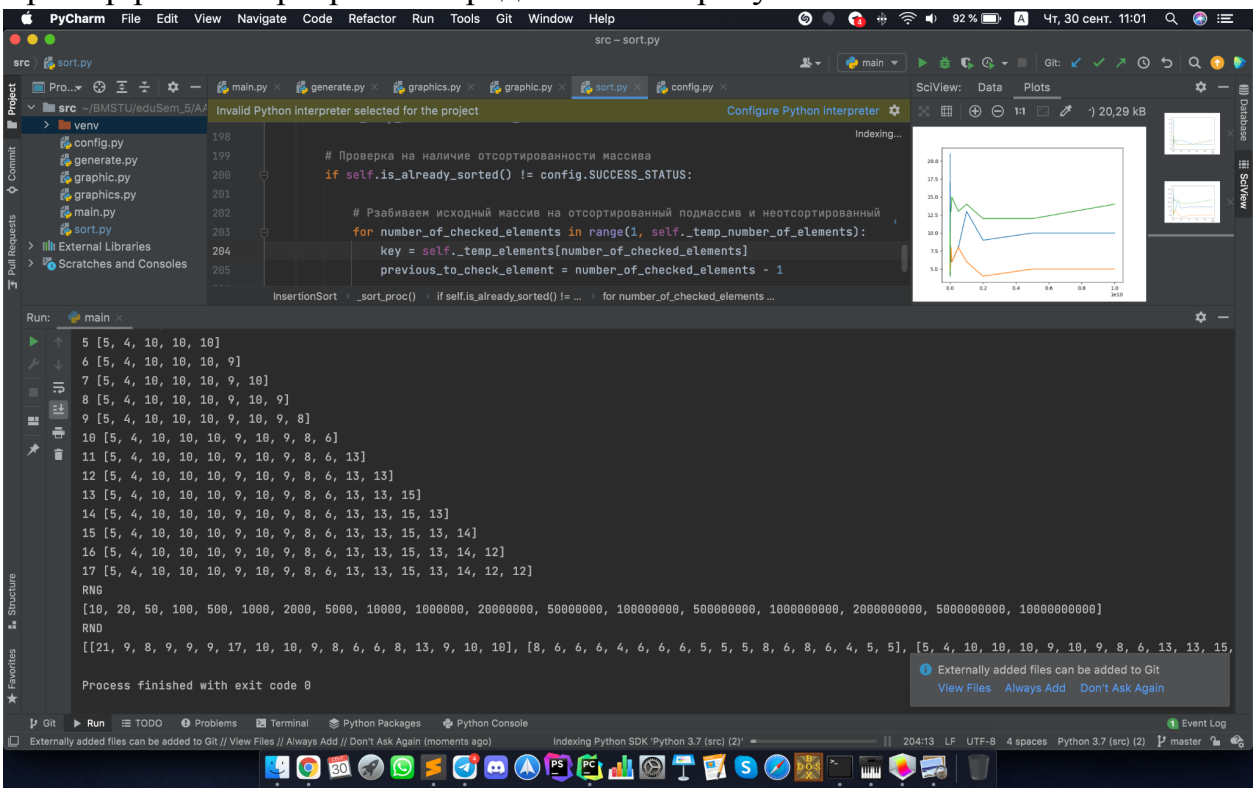

Рисунок 4: Демонстрация работы программы

4.2. Технические характеристики

В Таблице 3. приведены технические характеристики ЭВМ, на котором проводилось тестирование разрабатываемого программного обеспечения.

Таблица 9: Технические характеристики ЭВМ, на котором проводилось тестирование разрабатываемого программного обеспечения

ОС	Mac OS Mojave 64-bit
ОЗУ	8 Gb 2133 MHz LPDDR3
Процессор	2,3 GHz Intel Core i5

4.3. Время выполнения алгоритмов

В Таблице 10. приведена информация о времени выполнения алгоритмов на отсортированных данных в микросекундах, в Таблице 11 на отсортированных данных в обратном порядке, в Таблице 12 на случайных числах.

Таблица 10: Таблица времени выполнения алгоритмов на отсортированных данных (в микросекундах)

№	Длина строк	Время		
		Пузырек	Выбором	Вставками
1	10	223	21	11
2	20	605	22	12
3	1000	19 300	30	23
4	1600	45 400	20	14
5	2000	65 910	13	91

Таблица 11: Таблица времени выполнения алгоритмов на отсортированных данных (в микросекундах)

№	Длина строк	Время		
		Пузырек	Выбором	Вставками
1	10	238	210	199
2	20	829	801	793
3	1000	24 642	23 998	23 638
4	1600	77 411	75 630	69 779
5	2000	89 193	87 904	83 786

Таблица 12: Таблица времени выполнения алгоритмов на отсортированных данных (в микросекундах)

№	Длина строк	Время		
		Пузырек	Выбором	Вставками
1	10	193	125	112
2	20	675	239	217
3	1000	19 389	13 001	12 390
4	1600	47 611	18 992	18 429
5	2000	67 560	22 305	21 129

4.5. Вывод

Лучше всего себя показывает сортировка вставками, она делает это стабильно на всех трех видах последовательности, затрачивая примерно одинаковое время на одну длину последовательности. Когда последовательность упорядочена сортировка вставками работает в 190 раз быстрее сортировки пузырьком, сортировка выбором и вставками выдает примерно равные показатели.

Заключение.

В рамках данной лабораторной работы были изучены и реализованы трех нерекурсивные алгоритмы сортировки: пузырек, выбором, вставками, проведен сравнительный анализ трудоемкости алгоритмов, сделан сравнительный анализ алгоритмов на основе экспериментальных данных.

Экспериментальным путем было выявлено, что лучше всего себя показывает сортировка вставками, она делает это стабильно на всех трех видах последовательности, затрачивая примерно одинаковое время на одну длину последовательности. Когда последовательность упорядочена сортировка вставками работает в 190 раз быстрее сортировки пузырьком, сортировка выбором и вставками выдает примерно равные показатели.

Список использованной литературы

[1] Кормен Т.Х., Лейзерсон Ч.И., Алгоритмы: Построение и анализ, год выпуска 2019, тираж 1328, 700 страниц.

[2] Наследование в Python [Электронный ресурс] Режим доступа: https://younglinux.info/oopython/inheritance. Дата обращения: 13.09.2021

[3] Гасфилд, Строки, деревья и последовательности в алгоритмах. Информатика и вычислительная биология. Невский Диалект БВХ-Петербург, год выпуска 2003, типаж 900, 653 страницы.

[4] Вычисление процессорного времени выполенения программы [Электронный ресурс] Режим доступа: https://www.tutorialspoint.com/python/time_clock.htm. Дата обращения: 13.09.2021