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 курсу "Анализ алгоритмов"

Тема Трудоемкость сортировок

Студент Козлова И.В.

Группа ИУ7-52Б

Оценка (баллы)

Преподаватель Волкова Л.Л.

Москва — 2021 г.

Оглавление

Введение

Одной из важнейших процедур обработки структурированной информации является сортировка.

Сортировка - это процесс перегруппировки заданной последовательности (кортежа) объектов в некотором определеннном порядке. Такой опредленный порядок позволяет, в некоторых случаях, эффективнее и удобнее работать с заданной последовательностью. В частности, одной из целей сортировки является облегчение задачи поиска элемента в отсортированном множестве.

Алгоритмы сортировки используются практически в любой программой системе. Целью алгоритмов сортирови является упорядочение последовательности элементов данных. Поиск элемента в последовательности отсортированных данных занимает время, пропорциональное логарифму количеству элементов в последовательности, а поиск элемента в последовательности не отсортированных данных занимает время, пропорциональное воличеству элементов в последовательности, то есть намного больше. Существует множество различных методов сортировки данных. Однако любой алгоритм сортировки можно разбить на три основные части:

- сравнение, определяющее упорядочность пары элементов;
- перестановка, меняющая местами пару элементов;
- собственно сортирующий алгоритм, который осуществляет сравнение и перестановку элементов данных до тех пор, пока все эти элементы не будут упорядочены.

Одной из важнейшей характеристикой любого алгоритма сортировки является скорость его работы, которая определяется функциональной зависимостью среднего времени сортировки последовательностей элементов данных, определенной длины, от этой длины.

Задачи данной лабораторной:

- изучить и реализовать три алгоритма сортировки: шейкер, вставками, выбором;

- провести сравнительный анализ трудоемкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- провести сравнительный анализ алгоритмов на основе экспериментальных данных, а именно по времени;
- описать и обосновать полученные результаты в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В этом разделе будут представлены описания алгоритмов сортировки перемешиванием, вставками и выбором.

1.1 Сортировка перемешиванием

Сортировка перемешиванием [1] — это разновидность сортировки пузырьком. Отличие в том, что данная сортировка в рамках одной итерации проходит по массиву в обоих направлениях (слева направо и справа налево), тогда как сортировка пузырьком - только в одном направлении (слева направо).

Общие идеи алгоритма:

- обход массива слева направо, аналогично пузырьковой - сравнение соседних элементов, меняя их местами, если левое значение больше правого;
- обход массива в обратном направлении (справа налево), начиная с элемента, который находится перед последним отсортированным, то есть на этом этапе элементы также сравниваются между собой и меняются местами, чтобы наименьшее значение всегда было слева.

1.2 Сортировка вставками

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

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помешается 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 входным данным алгоритма.

1.3 Сортировка выбором

Сортировка выбором [3] - алгоритм, основанный на сравнение каждого элемента с каждым, и в случае необходимости производя обмен.

Шаги алгоритма.

1. Находим номер минимального значения в текущем массиве;
2. Производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции);
3. Далее сортируем “хвост” массива, исключив из рассмотрения уже отсортированные элементы.

Для реализации устойчивости алгоритма необходимо в пункте 2 минимальный элемент непосредственно вставлять в первую неотсортированную позицию, не меняя порядок остальных элементов, что может привести к резкому увеличению числа обменов.

Вывод

В данной работе стоит задача реализации 3 алгоритмов сортировки, а именно: перемешиванием, вставками и выбором. Необходимо оценить теоретическую оценку алгоритмов и проверить ее экспериментально.

2 Конструкторская часть

В этом разделе будут приведены схемы алгоритмов и вычисления трудоемкости данных алгоритмов.

2.1 Разработка алгоритмов

На рисунках 2.1, 2.2 и 2.3 представлены схемы алгоритмов сортировки пузырьком, выбором и вставками соответственно.

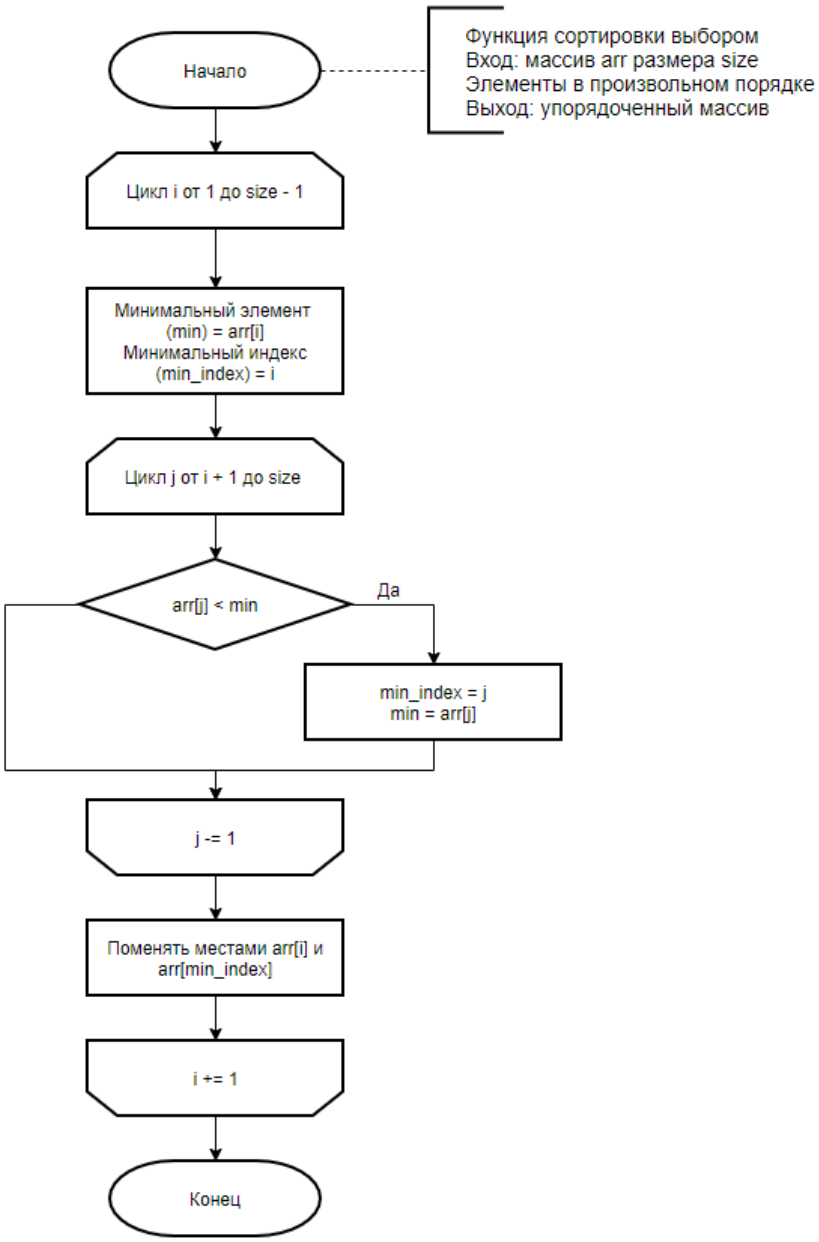

Рисунок 2.1 – Схема алгоритма сортировки выбором

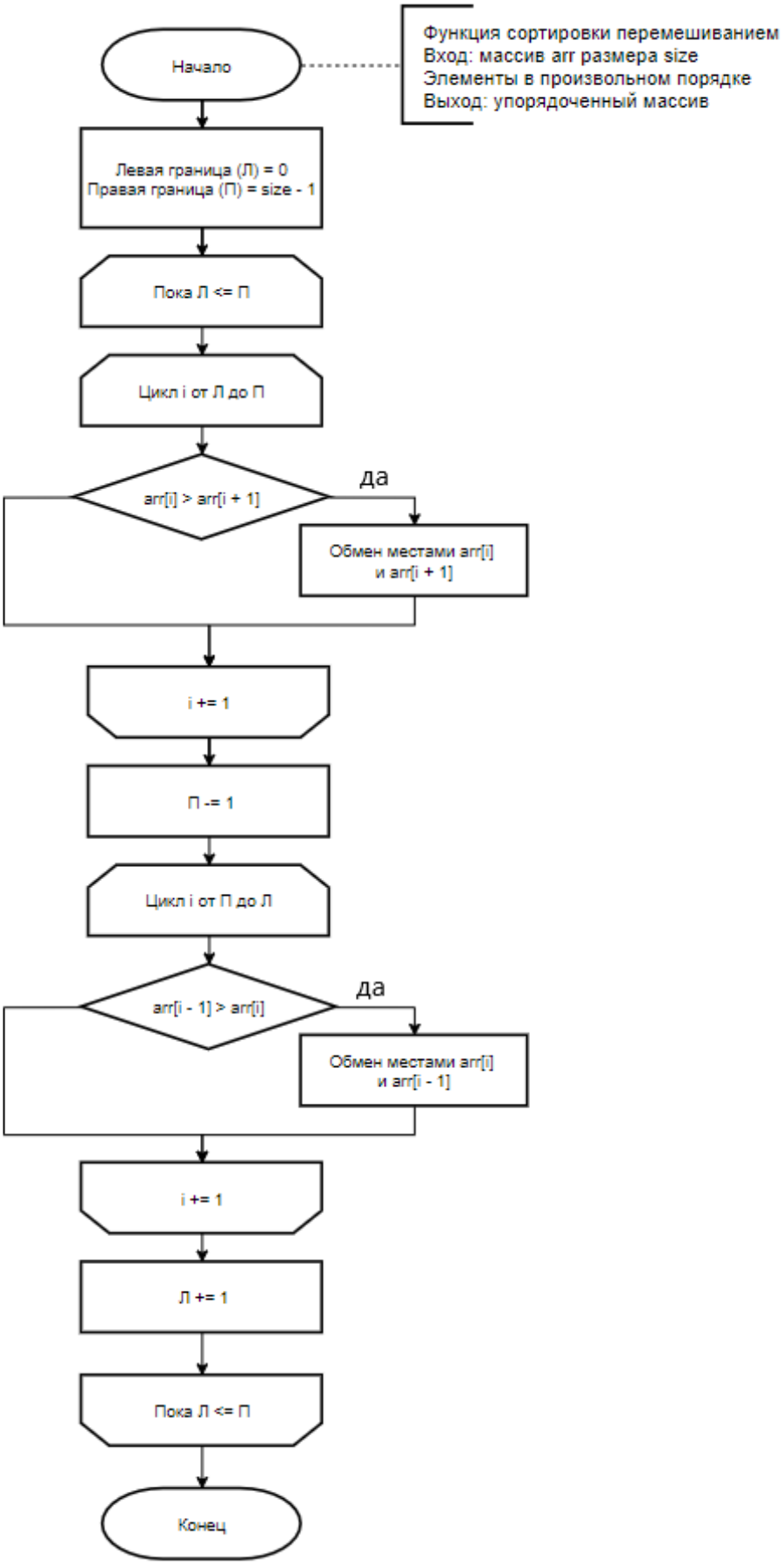

Рисунок 2.2 – Схема алгоритма сортировки перемешиванием

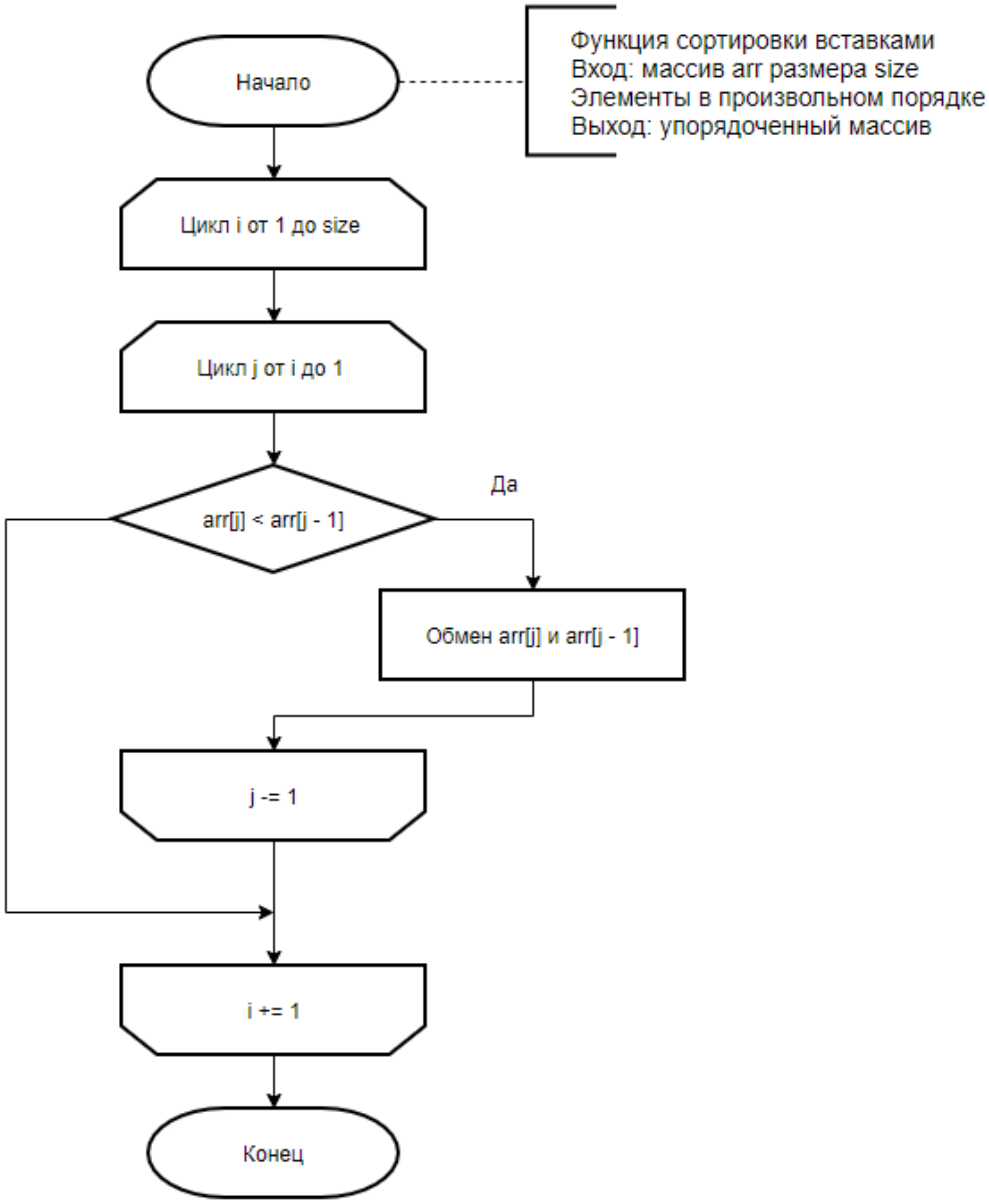

Рисунок 2.3 – Схема алгоритма сортировки вставками

2.2 Модель вычислений (оценки трудоемкости)

Для последующего вычисления трудоемкости необходимо ввести модель вычислений:

1. операции из списка (2.1) имеют трудоемкость 1;

$$+, -, /, *, \%, =, + =, - =, * =, / =, \% =, ==, ! =, <, >, <=, >=, [], ++, --$$ (2.1)

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

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

3. трудоемкость цикла рассчитывается, как (2.3);

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

4. трудоемкость вызова функции равна 0.

2.3 Трудоёмкость алгоритм

Обозначим во всех последующих вычислениях размер массивов как N.

2.3.1 Алгоритм сортировки перемешиванием

- Трудоёмкость сравнения внешнего цикла $WHILE(swap == True)$, которая равна (2.4):

$$f_{outer} = 1 + 2 \cdot (N - 1) \quad (2.4)$$

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

$$f_{inner} = 5(N - 1) + \frac{2 \cdot (N - 1)}{2} \cdot (3 + f_{if}) \quad (2.5)$$

- Трудоёмкость условия во внутреннем цикле, которая равна (2.6):

$$f_{if} = 4 + \begin{cases} 0, & \text{л.с.} \\ 9, & \text{х.с.} \end{cases} \tag{2.6}$$

Трудоёмкость в лучшем случае (2.7):

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

Трудоёмкость в худшем случае (2.8):

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

2.3.2 Алгоритм сортировки выбором

Трудоёмкость алгоритма сортировки выбором состоит из:

- Трудоёмкость сравнения, инкремента внешнего цикла, а также зависимых только от него операций, по $i \in [1..N)$, которая равна (2.9):

$$f_{outer} = 2 + 12(N - 1) \tag{2.9}$$

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

$$f_{inner} = 2(N - 1) + \frac{N \cdot (N - 1)}{2} \cdot f_{if} \tag{2.10}$$

- Трудоёмкость условия во внутреннем цикле, которая равна (2.11):

$$f_{if} = 3 + \begin{cases} 0, & \text{л.с.} \\ 3, & \text{х.с.} \end{cases} \tag{2.11}$$

Трудоёмкость в лучшем случае (2.12):

$$f_{best} = -12 + 12.5N + \frac{3}{2}N^2 \approx \frac{3}{2}N^2 = O(N^2) \tag{2.12}$$

Трудоёмкость в худшем случае (2.13):

$$f_{worst} = -12 + 11N + 3N^2 \approx 3N^2 = O(N^2) \tag{2.13}$$

2.3.3 Алгоритм сортировки вставками

Трудоёмкость сортировки вставками может быть рассчитана таким же образом, что и трудоёмкости алгоритмов выше. Асимптотическая трудоёмкость алгоритма сортировки вставками $O(N^2)$ в худшем случае и $O(N)$ — в лучшем [2].

Вывод

Были разработаны схемы всех трех алгоритмов сортировки. Для каждого из них были рассчитаны и оценены лучшие и худшие случаи.

3 Технологическая часть

В данном разделе будут приведены требования к программному обеспечению, средства реализации и листинги кода.

3.1 Требования к ПО

К программе предъявляется ряд требований:

- на вход подаётся массив сравнимых элементов (целые числа);
- на выходе — тот же массив, но в отсортированном порядке.

3.2 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран ЯП Python [4].

Данный язык достаточно удобен и гибок в использовании.

Время работы алгоритмов было замерено с помощью функции process_time() из библиотеки time [5]

3.3 Сведения о модулях программы

Программа состоит из двух модулей:

1. main.py - главный файл программы, в котором располагаются коды всех алгоритмов и меню;
2. test.py - файл с замерами времени для графического изображения результата.

3.4 Листинг кода

В листингах 3.1, 3.2, 3.3 представлены реализации алгоритмов сортировок (перемешиванием, вставками и выбором).

Листинг 3.1 – Алгоритм сортировки выбором

```
def selection_sort(arr):
    for i in range(0, len(arr) - 1):
        min = i
        for j in range(i + 1, len(arr)):
            if arr[j] < arr[min]:
                min = j
        arr[i], arr[min] = arr[min], arr[i]
    return arr
```

Листинг 3.2 – Алгоритм сортировки вставками

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        j = i - 1
        key = arr[i]

        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr
```

Листинг 3.3 – Алгоритм сортировки перемешиванием (или сортировка шейкер)

```
def shaker_sort(arr):
    length = len(arr)
    swapped = True
    start_index = 0
    end_index = length - 1

    while (swapped == True):
        swapped = False

        for i in range(start_index, end_index):
            if (arr[i] > arr[i + 1]):
```

```
12                 arr[i], arr[i + 1] = arr[i + 1], arr[i]
13                 swapped = True
14
15         if (not (swapped)):
16             break
17
18         swapped = False
19         end_index = end_index - 1
20
21         for i in range(end_index - 1, start_index - 1, -1):
22             if (arr[i] > arr[i + 1]):
23                 arr[i], arr[i + 1] = arr[i + 1], arr[i]
24                 swapped = True
25
26         start_index = start_index + 1
27     return arr
```

3.5 Функциональные тесты

В таблице 3.1 приведены тесты для функций, реализующих алгоритмы сортировки. Тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Входной массив	Ожидаемый результат	Результат
$[1, 2, 3, 4]$	$[1, 2, 3, 4]$	$[1, 2, 3, 4]$
$[5, 4, 3, 2, 1]$	$[1, 2, 3, 4, 5]$	$[1, 2, 3, 4, 5]$
$[3, 2, -5, 0, 1]$	$[-5, 0, 0, 2, 3]$	$[-5, 0, 0, 2, 3]$
$[4]$	$[4]$	$[4]$
$[]$	$[]$	$[]$

Вывод

Были разработаны схемы всех трех алгоритмов сортировки. Для каждого из них были рассчитаны и оценены лучшие и худшие случаи.

4 Исследовательская часть

В данном разделе будут приведены примеры работы программ, постановка эксперимента и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование, следующие.

- Операционная система: Ubuntu 20.04.3 [6] Linux [7] x86_64.
- Память: 8 GiB.
- Процессор: 11th Gen Intel® Core™ i5-1135G7 @ 2.40GHz [8].

Тестирование проводилось на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Демонстрация работы программы

На рисунке 4.1 представлен результат работы программы.

4.3 Время выполнения алгоритмов

Алгоритмы тестировались при помощи функции process_time() из библиотеки time языка Python. Данная функция всегда возвращает значения времени, а имеено сумму системного и пользовательского процессорного времени текущего процессора, типа float в секундах.

Контрольная точка возвращаемого значения не определна, поэтому допустима только разница между результатами последовательных вызовов.

```
МЕНЮ:
        0. Выход
        1. Сортировка Шейкер
         2. Сортировка вставками
         3. Сортировка выбором
         4. Замер времени (длина слов от 1 до 10)
         Выбор: 4
Введите массив: 4 5 9 0 23 2 -4 1
Массив отсортированный сортировкой-Шейкер  [-4, 0, 1, 2, 4, 5, 9, 23]
Время в мкс (сортировкой-Шейкер)  6.71875
Массив отсортированный сортировкой вставками  [-4, 0, 1, 2, 4, 5, 9, 23]
Время в мкс (сортировкой вставками)  5.15625
Массив отсортированный сортировкой выбором  [-4, 0, 1, 2, 4, 5, 9, 23]
Время в мкс (сортировкой выбором) 5.9375
```

Рисунок 4.1 – Пример работы программы

Результаты замеров приведены в таблицах 4.1, 4.2 и 4.3.

На рисунках 4.2, 4.3 и 4.4, приведены графики зависимостей времени работы алгоритмов сортировки от размеров массивов на отсортированных, обратно отсортированных и случайных данных.

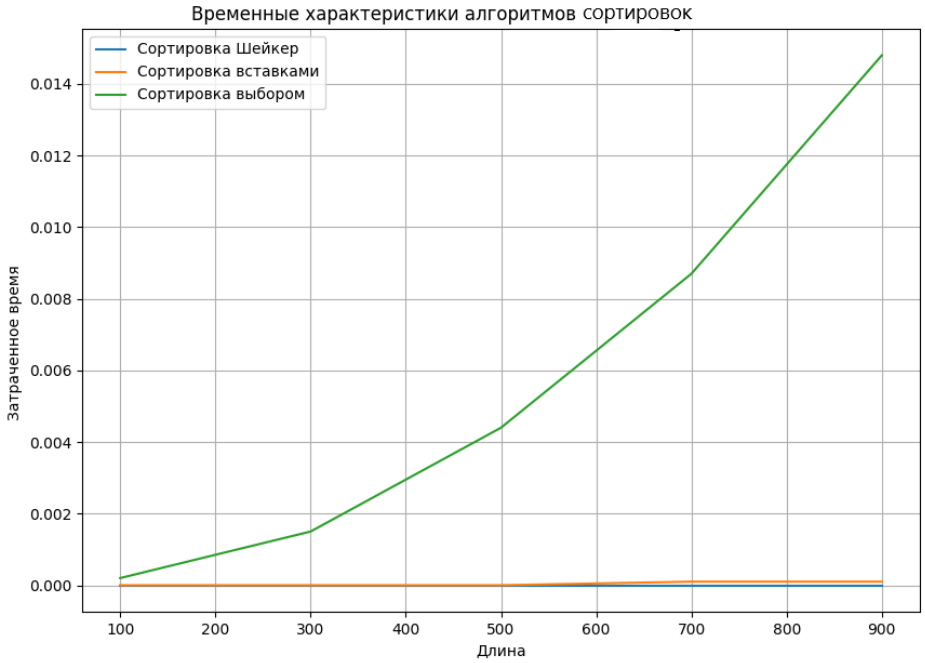

Рисунок 4.2 – Зависимость времени работы алгоритма сортировки от размера отсортированного массива (мск)

Таблица 4.1 – Время работы алгоритмов сортировки на отсортированных данных (мск)

Размер	Шейкером	Вставками	Выбором
100	5.2410	10.0906	258.0546
200	9.2891	23.5862	873.4441
300	15.6069	39.2295	1670.5604
400	21.3040	51.9447	2904.2619
500	27.7164	64.3753	4678.6021
600	28.7513	72.2639	6565.3837
1000	51.1101	96.3444	20440.3201
2000	168.7951	327.707139	120891.3995
2500	209.5140	394.2722	160569.3123

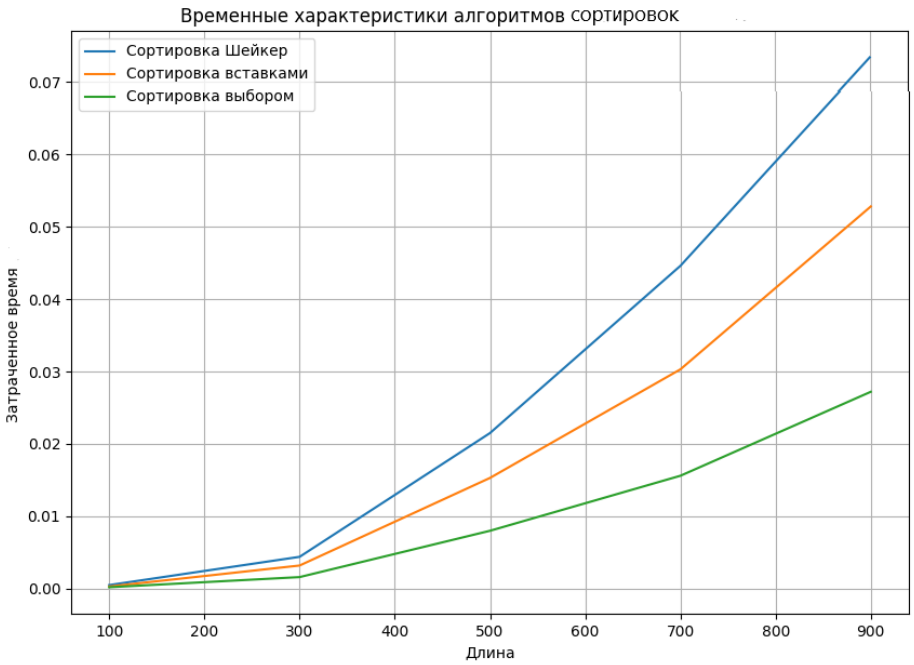

Рисунок 4.3 – Зависимость времени работы алгоритма сортировки от размера массива, отсортированного в обратном порядке (мск)

Таблица 4.2 – Время работы алгоритмов сортировки на обратно отсортированных данных (мск)

Размер	Шейкером	Вставками	Выбором
100	639.0698	363.9547	179.1664
200	2195.5410	1475.2803	735.1370
300	4819.4516	3400.9440	1677.5576
400	9230.5019	6217.0601	2988.1903
500	18682.5808	15383.5356	8015.5767
600	33239.3775	22488.2615	11754.5552
1000	942220.648	64293.5541	33109.7468
2000	370076.0900	255556.7835	131883.3811
2500	597217.8193	372790.2408	188786.7314

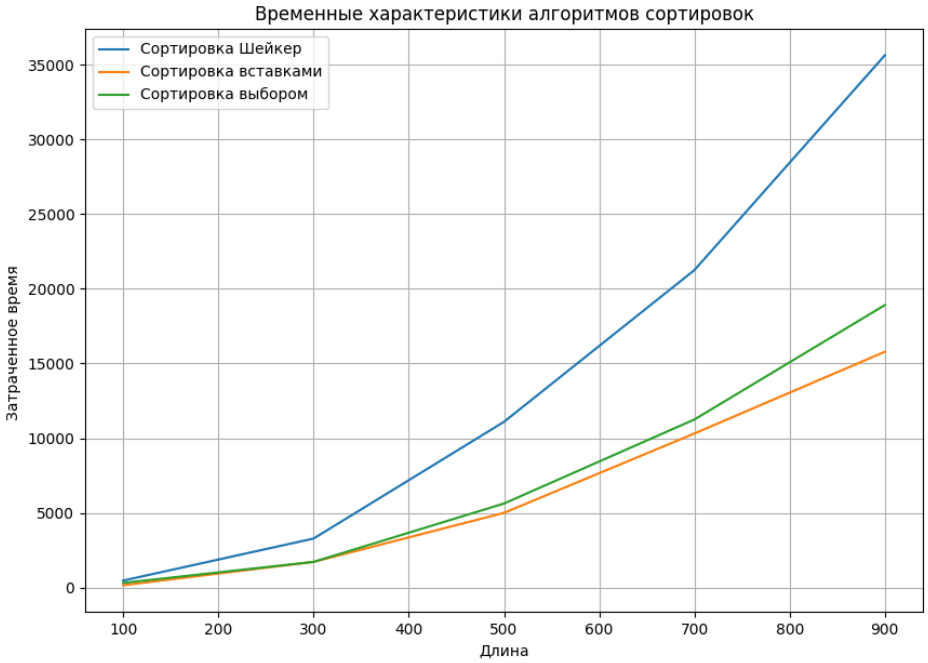

Рисунок 4.4 – Зависимость времени работы алгоритма сортировки от размера массива, заполненного в случайном порядке (мск)

Таблица 4.3 – Время работы алгоритмов сортировки на случайных данных (мск)

Размер	Шейкером	Вставками	Выбором
100	438.8057	186.5141	204.1938
200	1455.2835	692.1317	725.6267
300	3156.4853	1563.0922	1667.3059
400	5716.6090	4964.3020	4954.9819
500	14948.6332	7955.9345	8240.3997
600	21706.8825	11442.8451	11955.7538
1000	60019.2610	31373.1001	33769.7239
2000	250424.1895	130588.0997	141522.1266
2500	414524.0602	214781.0473	236668.4435

Вывод

Алгоритм сортировки вставками работает лучше остальных двух на случайных числах и уже отсортированных. Интерес представляет лишь первый случай, на котором сортировка вставками стабильно быстрее двух других алгоритмов.

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

- изучены и реализованы 3 алгоритма сортировки: перемешиванием, вставками, выбором;
- проведен сравнительный анализ трудоёмкости алгоритмов на основе теоретических расчетов и выбранной модели вычислений;
- проведен сравнительный анализ алгоритмов на основе экспериментальных данных;
- подготовлен отчет о лабораторной работе.

Поставленная цель достигнута.

Литература

[1] Шейкерная сортировка (перемешиванием) [Электронный ресурс]. Режим доступа: https://kvodo.ru/shaker-sort.html (дата обращения: 12.09.2021).

[2] Сортировка вставками [Электронный ресурс]. Режим доступа: https://kvodo.ru/sortirovka-vstavkami-2.html (дата обращения: 12.09.2021).

[3] Сортировка выбором [Электронный ресурс]. Режим доступа: https://kvodo.ru/sortirovka-vyiborom-2.html (дата обращения: 12.09.2021).

[4] Welcome to Python [Электронный ресурс]. Режим доступа: https://www.python.org (дата обращения: 04.09.2021).

[5] time — Time access and conversions [Электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html#functions (дата обращения: 04.09.2021).

[6] Ubuntu 20.04.3 LTS (Focal Fossa) [Электронный ресурс]. Режим доступа: https://releases.ubuntu.com/20.04/ (дата обращения: 04.09.2021).

[7] Linux – Википедия [Электронный ресурс]. Режим доступа: https://ru.wikipedia.org/wiki/Linux (дата обращения: 04.09.2021).

[8] Процессор Intel® Core™ i5-1135G7 [Электронный ресурс]. Режим доступа: https://www.intel.ru/content/www/ru/ru/products/sku/208658/intel-core-i51135g7-processor-8m-cache-up-to-4-20-ghz/specifications.html (дата обращения: 04.09.2021).