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 Алгоритмы сортировки

Студент Прянишников А. Н.

Группа ИУ7-55Б

Оценка (баллы) _____

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Содержание

Введение

Алгоритмы сортировки используются в широком наборе задач. Сортировка встречается повсеместно: например, в интернет-магазине нужно упорядочить товары по возрастанию цены. В компьютерной графике, в базах данных, в файловых системах - везде требуется сортировка.

Сортировки требует больших затрат по времени и памяти, поэтому существуют различные алгоритмы со своими преимуществами. С распространением BigData технологий выбор оптимального решения стал особенно актуальным, так как требуется быстро работать с огромными массивами данных.

Актуальность работы заключается в том, что нужно оптимизировать траты системных ресурсов на сортировку.

Целью данной работы является разработка программы, которая реализует три алгоритма сортировки.

Для достижения поставленной цели необходимо выполнить следующее:

- рассмотреть существующие алгоритмы сортировки;
- привести схемы реализации рассматриваемых алгоритмов;
- определить средства программной реализации;
- реализовать рассматриваемые алгоритмы;
- протестировать разработанное ПО;
- провести модульное тестирование всех реализаций алгоритмов;
- оценить реализацию алгоритмов по времени и памяти.

1 Аналитический раздел

В данном разделе будут рассмотрены три алгоритма сортировки. Будут проанализированы их преимущества и недостатки. Будет произведена общая формализация задачи.

1.1 Формализация задачи

Пусть A - массив, содержащий N чисел. Массив считается отсортированным по возрастанию, если для каждого элемента массива выполняется следующее соотношение 1.1:

$$\forall i, j \in [1, N] : i \leq j; A_i \leq A_j \tag{1.1}$$

Массив считается отсортированным по убыванию, если для каждого элемента массива выполняется следующее соотношение 1.2:

$$\forall i, j \in [1, N] : i \geq j; A_i \geq A_j \tag{1.2}$$

Для выполнения цели работы требуется написать алгоритм, который для произвольных данных сортирует по возрастанию массив.

1.2 Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву [1]. 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, если ещё не находится там.

Недостаток алгоритма состоит в том, что наименьшие элементы, находящиеся в конце массива, будут за каждую итерацию смещаться к правильному положению лишь на одну позицию. Алгоритм эффективен лишь на небольших массивах, для практических задач не используется.

К преимуществам алгоритма стоит отнести простоту реализации и отсутствие дополнительных затрат на память.

1.3 Сортировка вставками

Сортировка вставками - алгоритм сортировки, в котором элементы входной последовательности рассматриваются по одному, и каждый новый поступивший элемент размещается в подходящем месте среди ранее упорядоченных элементов[2].

В начальный момент последовательность отсортированных элементов пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию до тех пор, пока набор данных не будет исчерпан.

Для того, чтобы оптимизировать затраты на память, отсортированная последовательность будет храниться в начале массива. То есть на каждой итерации элемент ставится на правильное место относительно тех элементов, которые рассматривались на предыдущих итерациях.

К преимуществам можно отности то, что в случае удачного расположения элементов алгоритм будет работать за линейное время.

1.4 Сортировка выбором

Сортировка выбором – сортировка, на каждой итерации которого алгоритм ищет минимальный элемент в массиве и располагает его сразу на правильное место в отсортированном массиве. [3]

Алгоритм можно разбить на три шага:

1. Найти индекс минимального элемента в массиве.

2. Произвести обмен минимального элемента в массиве с первым неотсортированным по известным индексам.

3. Проделать те же операции для ещё не отсортированных элементов в массиве.

Чтобы обеспечить устойчивость алгоритма, необходимо минимальный элемент непосредственно вставлять в первую неотсортированную позицию, не меняя порядок остальных элементов.

1.5 Вывод

Были рассмотрены три алгоритма сортировок, проведена формализация задачи, проанализированы преимущества и недостатки алгоритмов.

2 Конструкторский раздел

В этом разделе будут приведены схемы реализации алгоритмов, а также способы тестирования и выделения классов эквивалентности.

2.1 Схемы алгоритмов

На рисунке 2.1 будет приведена схема реализации алгоритма сортировки пузырьком. На рисунке 2.2 будет приведена схема реализации алгоритма сортировки вставками. На рисунке 2.3 будет приведена схема реализации алгоритма сортировки выбором.

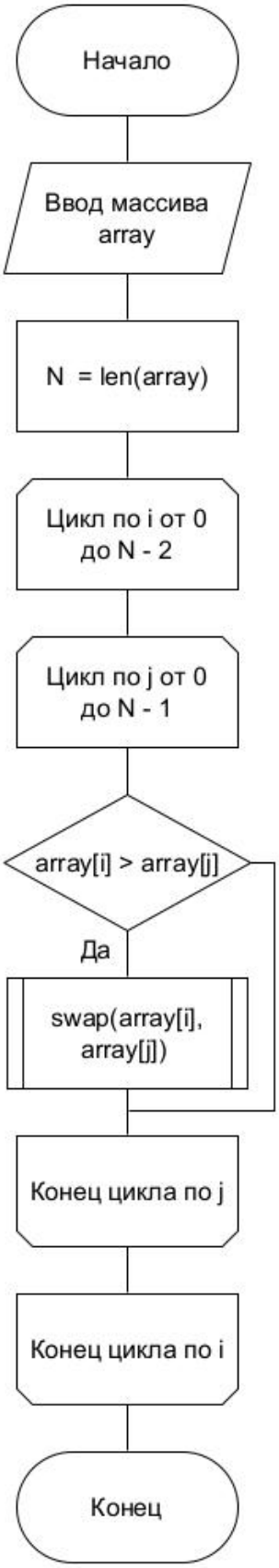

Рисунок. 2.1: Схема алгоритма сортировки пузырьком

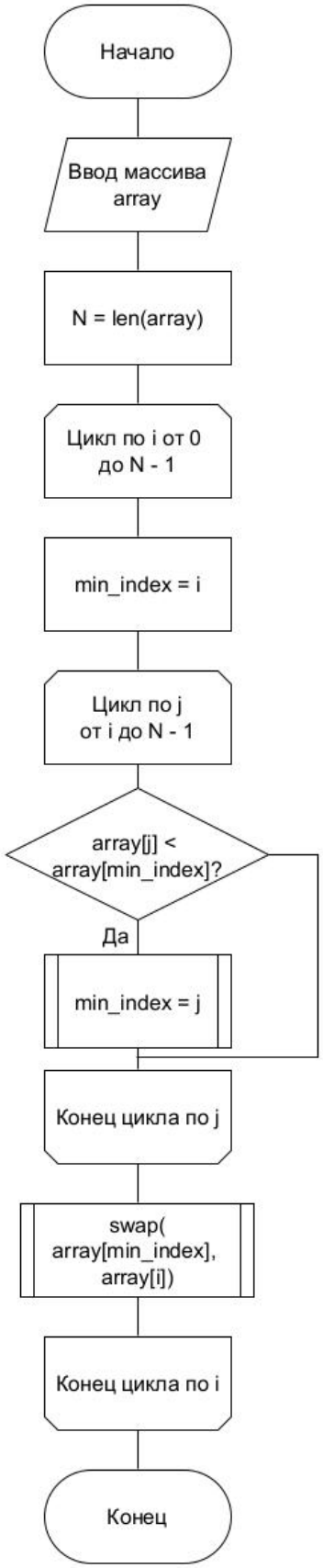

Рисунок. 2.2: Схема алгоритма сортировки выбором

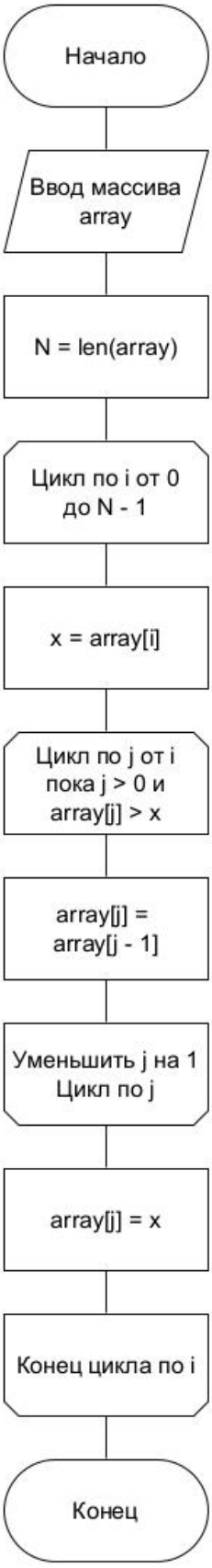

Рисунок. 2.3: Схема алгоритма сортировки вставками

2.2 Типы данных для алгоритмов

Тестирование алгоритмов будет производиться на целых числах, которые могут быть и отрицательными, и равны нулю. Несмотря на это, сами реализации алгоритмов универсальны и предназначены для любых численных типов данных.

Размер массива может быть произвольным.

2.3 Способ тестирования

Тестирование программы будет производиться методом чёрного ящика. Такой подход выбран, так как от реализаций алгоритмов требуется в первую очередь правильность работы. Сама по себе реализация не требует тестировки, так как в точности повторяет теоретические принципы, сформированные в аналитическом разделе.

В качестве классов эквивалентности были выбраны следующие сущности:

- массив из случайных элементов;
- отсортированный по возрастанию массив;
- отсортированный по убыванию массив;
- массив, состоящий из одинаковых элементов;
- массив нулевой длины;
- массив единичной длины;
- массив, в котором несколько одинаковых элементов;s

Для корректного сравнения требуется для каждой реализации сортировки сравнить полученный на выходе массив с эталонным результатом массива.

2.4 Вывод

Были приведены схемы алгоритмов, рассмотрены типы данных для алгоритмов, а также описан способ тестирования.

3 Технологический раздел

В этом разделе будут сформированы требования к ПО, приведены листинги кода и результаты тестирования. Также будет произведена оценка трудоёмкости алгоритмов.

3.1 Средства реализации программного обеспечения

В качестве языка программирования выбран Python 3.9, так как имеется опыт разработки проектов на этом языке. Для замера процессорного времени используется функция process_time_ns из библиотеки time. В её результат не включается время, когда процессор не выполняет задачу [4].

3.2 Требования к ПО

Ниже будет представлен список требований к разрабатываемому программному обеспечению.

Требования к входным данным:

- массив должен состоять из целых чисел;
- длина массива может быть нулевой.

Требования к выводу:

- программа должна выводить массив после сортировки по возрастанию для каждого реализованного алгоритма.

3.3 Листинг кода

На листингах 1-3 приведены реализации различных алгоритмов сортировки по возрастанию.

Листинг 3.1: Реализация алгоритма сортировки пузырьком

```
def bubble_sort(array):
    N = len(array)
    for i in range(N - 1):
        for j in range(N - i - 1):
            if array[j] > array[j + 1]:
                array[j], array[j + 1] = \
                    array[j + 1], array[j]
    return array
```

Листинг 3.2: Реализация алгоритма сортировки выбором

```
def insertion_sort(array):
    for i in range(1, len(array)):
        key = array[i]
        j = i - 1
        while j >=0 and key < array[j]:
            array[j + 1] = array[j]
            j -= 1
        array[j + 1] = key
    return array
```

Листинг 3.3: Реализация алгоритма сортировки выбором

```
def select_sort(array):
    N = len(array)
    for i in range(N):
        minIndex = i
        for j in range(i, N):
            if (array[j] < array[minIndex]):
                minIndex = j
        array[i], array[minIndex] = \
            array[minIndex], array[i]

    return array
```

3.4 Функциональные тесты

В таблице 3.1 приведены результаты функциональных тестов. В первом столбце – исходный массив. Во втором столбце – правильный результат сортировки по возрастанию. В столбцах 3-5 – результаты сортировки реализаций алгоритмов.

Таблица 3.1: Результаты функциональных тестов

Массив	Ожид.Результат	BubbleSort	SelectSort	InsertSort
1 5 3 0 2 -2 6	-2 0 1 2 3 5 6	-2 0 1 2 3 5 6	-2 0 1 2 3 5 6	-2 0 1 2 3 5 6
-1 0 1 2 3 4	-1 0 1 2 3 4	-1 0 1 2 3 4	-1 0 1 2 3 4	-1 0 1 2 3 4
3 2 1 0 -1 -2	-2 -1 0 1 2 3	-2 -1 0 1 2 3	-2 -1 0 1 2 3	-2 -1 0 1 2 3
*	*	*	*	*
1	1	1	1	1
1 1 1 1 1	1 1 1 1 1	1 1 1 1 1	1 1 1 1 1	1 1 1 1 1
1 -1 1 -1	-1 -1 1 1	-1 -1 1 1	-1 -1 1 1	-1 -1 1 1

* – пустой массив.

Все тесты пройдены успешно.

3.5 Оценка трудоёмкости

Введем модель трудоемкости для оценки алгоритмов:

1. Для базовых операции: +, -, *, /, =, ==, <=, >=, !=, +=, [], ++ – трудоёмкость равна 1.

2. Трудоёмкость условия *if УСЛОВИЕ then A else B* будет подсчитана по формуле 3.1:

$$F = F_{\text{условия}} + \begin{cases} F_A & \text{, если условие выполняется} \\ F_B & \text{, иначе} \end{cases} \tag{3.1}$$

.

3. Трудоймкость цикла **for** будет подсчитана по формуле 3.2:

$$F_{for} = F_{\text{инициализации}} + F_{\text{сравнения}} + N(F_{\text{тела}} + F_{\text{инициализации}} + F_{\text{сравнения}}) \tag{3.2}$$

4. Трудоёмкость вызова функции равна 0.

3.5.1 Вычисление трудоёмкости сортировки пузырьком

Лучший случай: Массив отсортирован; не произошло ни одного обмена за 1 проход:
Трудоемкость: $1 + (N-1) + (N-1) * ((N-2) * (1+4)) = N + (N-1) * (5 * N - 10) = 5 * N^2 - 4 * N + 10 = O(n^2)$

Худший случай: Массив отсортирован в обратном порядке; в каждом случае происходил обмен:
Трудоемкость: $1 + (N-1) + (N-1) * ((N-2) * (1+4+5)) = N + (N-1) * (10 * N - 20) = 10 * N^2 - 9 * N + 20 = O(n^2)$

3.5.2 Вычисление трудоёмкости сортировки вставками

Лучший случай: Массив отсортирован; внутренние циклы состоят из одной итерации:
Трудоемкость: $1 + (N-1) * (13) = 13 * N - 12 = O(n)$

Худший случай: Массив отсортирован в обратном порядке; во всех внутренних циклах будет j итераций:
Трудоемкость: $1 + (N-1) * (N-1)/2 * (13) = 6.5 * N^2 - 13N - 5.5 = O(n^2)$

3.5.3 Вычисление трудоёмкости сортировки выбором

Лучший случай: Массив отсортирован; во внутреннем цикле нет присваиваний:
Трудоемкость: $1 + (N-1) * ((N-1) * 6) = 6N^2 - 7N + 5 = O(n^2)$

Худший случай: Массив отсортирован в обратном порядке; во внутреннем цикле будут присваивания на каждой итерации:
Трудоемкость: $1 + (N-1) * ((N-1) * 7) = 7N^2 - 8N + 6 = O(n^2)$

3.6 Вывод

Были сформированы требования к ПО, приведены листинги коды. Были вычислены трудоёмкости алгоритмов и проведены функциональные тесты. Все алгоритмы справились с тестированием.

4 Исследовательская часть

В этом разделе будет приведена демострация работы программы, а также исследовано время работы алгоритмов.

4.1 Демонстрация работы программы

На рисунке 4.1 показана демонстрация работы программы.

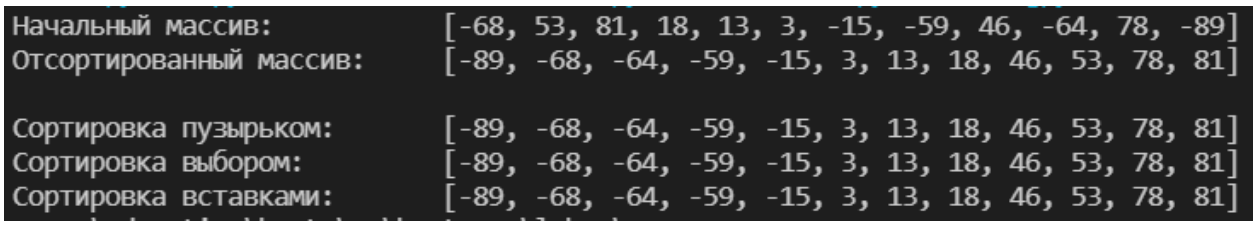

Рисунок. 4.1: Демонстрация работы программы

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование, следующие:

- операционная система: Windows 10;
- память: 8 GB;
- процессор: Intel Core i5-1135G7 @ 2.40GHz [5].

Во время тестирования ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.3 Время работы программы

Для замера времени работы алгоритмов рассматривалось три случая:

1. Массив, состоящий из случайных чисел.

2. Отсортированный массив.

3. Массив, сортированный в обратном порядке.

4.3.1 Сравнение времени работы алгоритмов на массиве случайных чисел

В таблице 4.1 приведены результаты тестирования работы алгоритмов на массиве случайных чисел. В первом столбце указан размер массива, а в следующих трёх – время работы для каждой реализации алгоритмов сортировки.

Таблица 4.1: Результаты тестов на массиве случайных чисел

Размер массива	BubbleSort	SelectSort	InsertSort
1	7812	9375	4687
5	20312	21875	12500
10	50000	67187	37500
20	201562	154687	106250
50	1284375	832812	795312
75	2660937	1798437	1478125
100	4468750	3156250	2515625
250	29375000	17578125	17109375
500	117187500	68906250	60937500
1000	476406250	276718750	257343750

На рисунке 4.2 изображены графики времени работы алгоритмов на массиве случайных чисел. График был построен в логарифмической шкале, так как сортировка вставками в сотни раз быстрее, чем остальные реализации, поэтому в обычной шкале график напоминает горизонтальную линию около нуля.

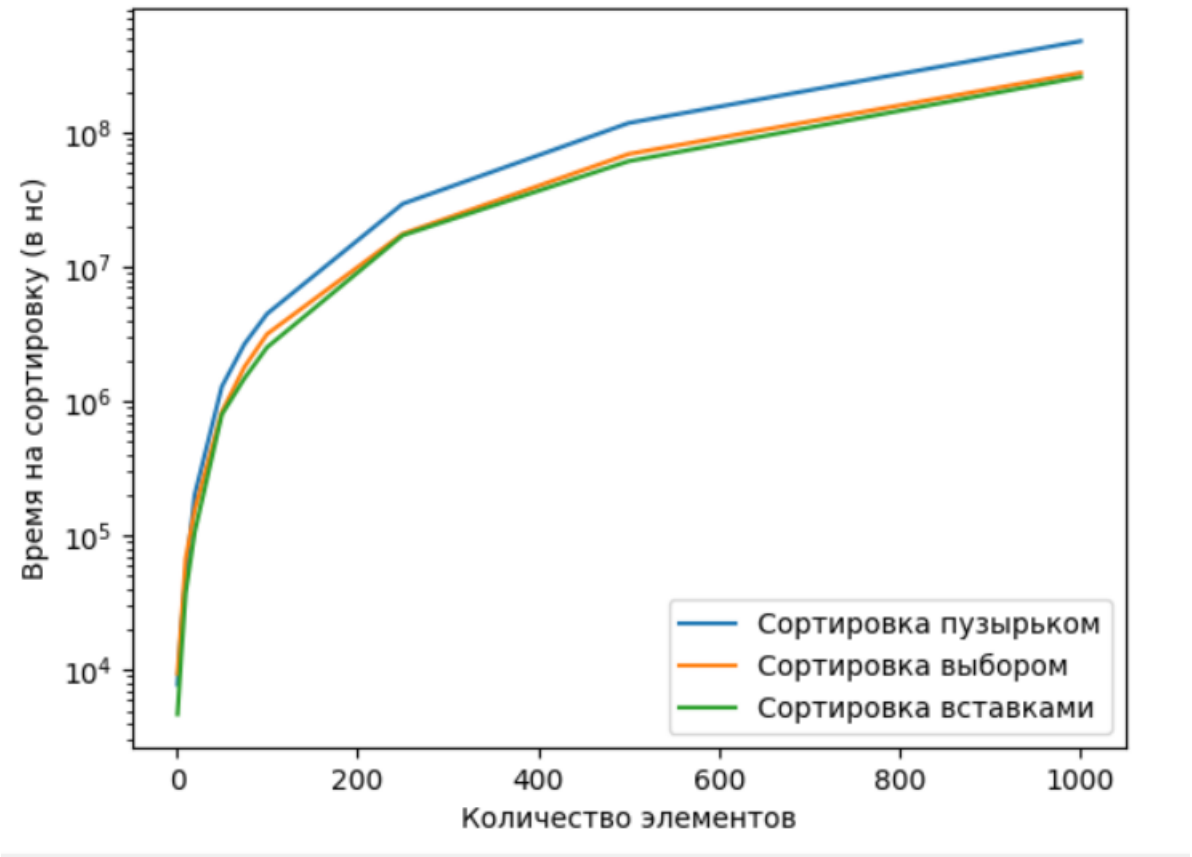

Рисунок. 4.2: График зависимости времени сортировки от количества элементов в массиве для массива случайных чисел

Также на рисунке 4.3 приведено сравнение сортировки пузырьком и сортировки выбором, так как по порядку они схожи.

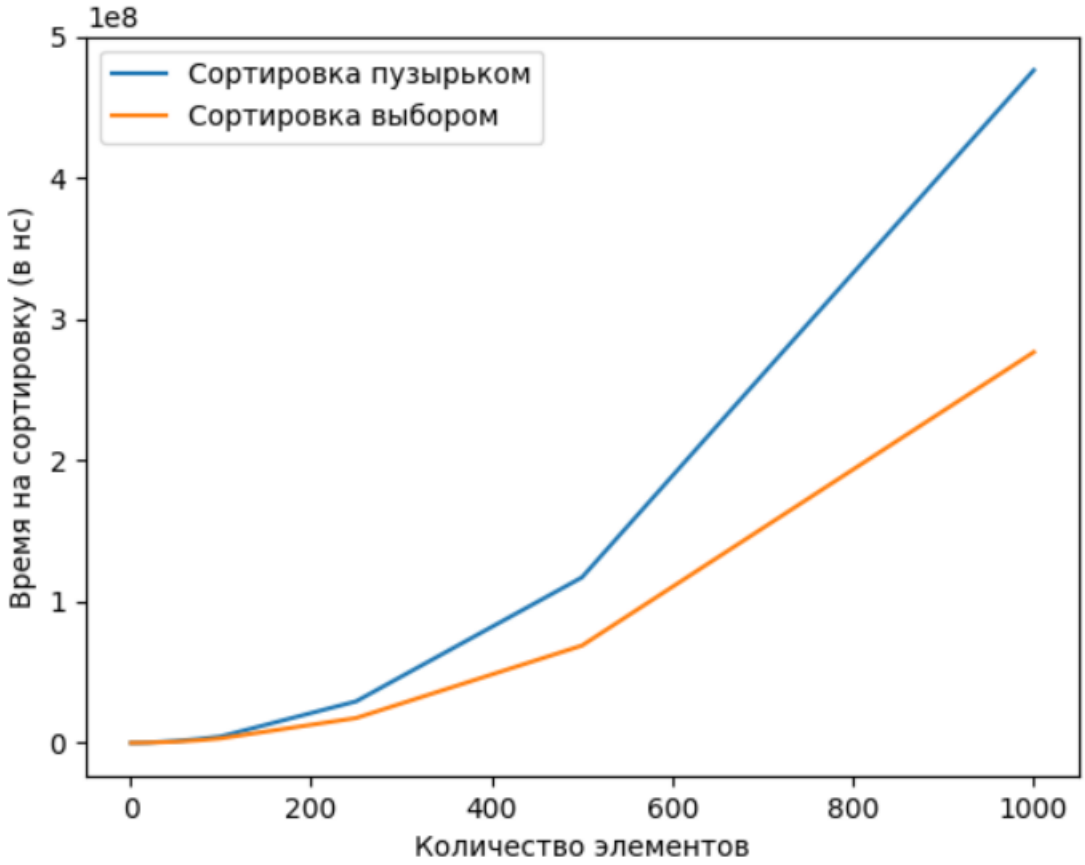

Рисунок. 4.3: График зависимости времени сортировки от количества элементов в массиве для массива случайных чисел

4.3.2 Сравнение времени работы алгоритмов на отсортированном массиве

В таблице 4.2 приведены результаты тестирования работы алгоритмов на отсортированном массиве. В первом столбце указан размер массива, а в следующих трёх – время работы для каждой реализации алгоритмов сортировки.

Таблица 4.2: Результаты тестов на отсортированном массиве

Размер массива	BubbleSort	SelectSort	InsertSort
1	7812	9375	4687
5	17187	23437	12500
10	42187	54687	15625
20	137500	160937	31250
50	795312	801562	71875
75	1682812	1715625	107812
100	2984375	2937500	140625
250	18828125	18359375	390625
500	72968750	70937500	625000
1000	295625000	275937500	1562500

На рисунке 4.4 изображены графики времени работы алгоритмов на отсортированном массиве. График был построен в логарифмической шкале, так как сортировка вставками в сотни раз быстрее, чем остальные реализации, поэтому в обычной шкале график напоминает горизонтальную линию около нуля.

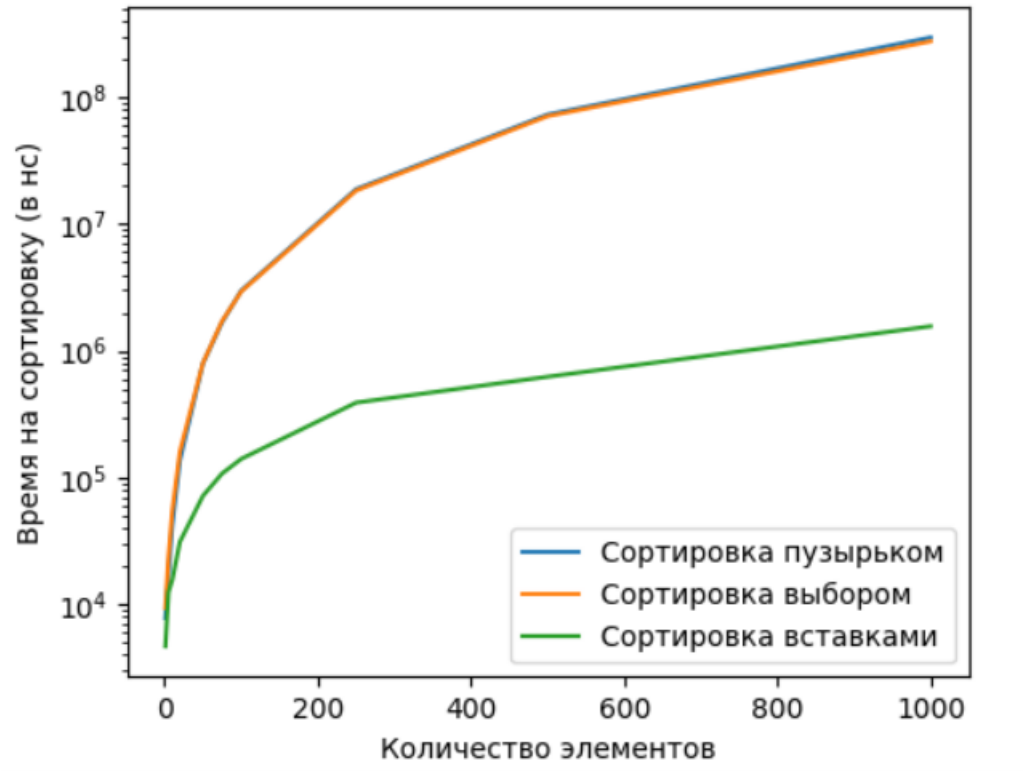

Рисунок. 4.4: График зависимости времени сортировки от количества элементов в массиве для отсортированного массива

Также на рисунке 4.5 приведено сравнение сортировки пузырьком и сортировки выбором, так как по порядку они схожи.

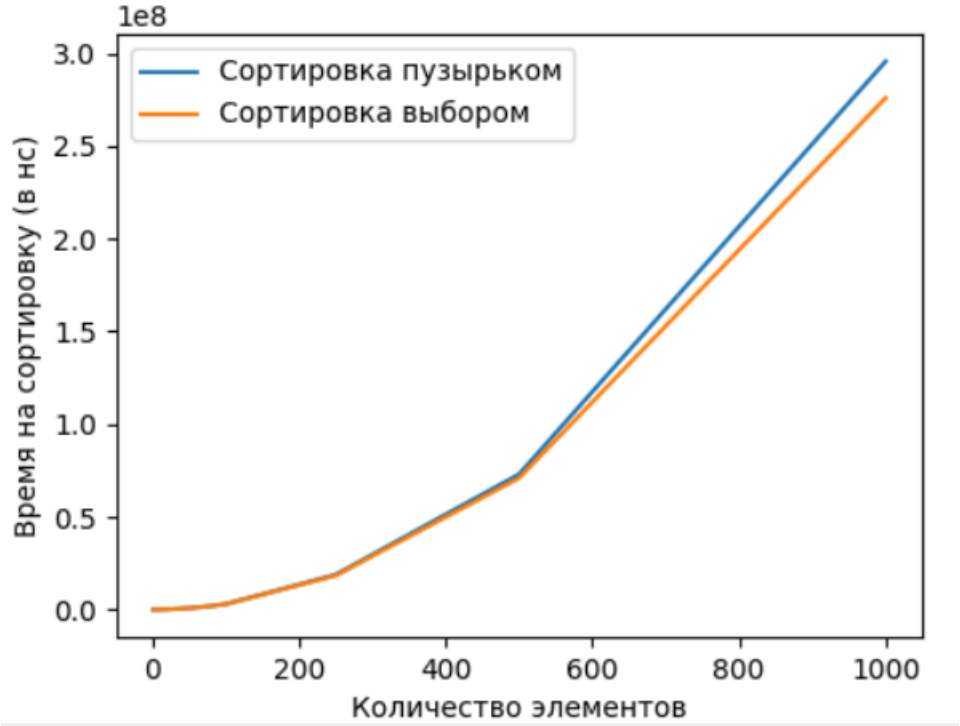

Рисунок. 4.5: График зависимости времени сортировки от количества элементов в массиве для массива случайных чисел

4.3.3 Сравнение времени работы алгоритмов на обратно сортированном массиве

В таблице 4.3 приведены результаты тестирования работы алгоритмов на массиве, сортированном в обратном порядке. В первом столбце указан размер массива, а в следующих трёх – время работы для каждой реализации алгоритмов сортировки.

Таблица 4.3: Результаты тестов массиве, отсортированном в обратном порядке

Размер массива	BubbleSort	SelectSort	InsertSort
1	6250	9375	4687
5	25000	26562	21875
10	75000	67187	68750
20	282812	179687	218750
50	1629687	954687	1240625
75	3625000	2104687	3003125
100	6687500	3734375	5015625
250	39453125	21796875	29921875
500	170468750	89218750	122812500
1000	671875000	342031250	523750000

На рисунке 4.6 изображены графики времени работы алгоритмов на массиве, отсортированном в обратном порядке. График был построен в логарифмической шкале, так как сортировка вставками в сотни раз быстрее, чем остальные реализации, поэтому в обычной шкале график напоминает горизонтальную линию около нуля.

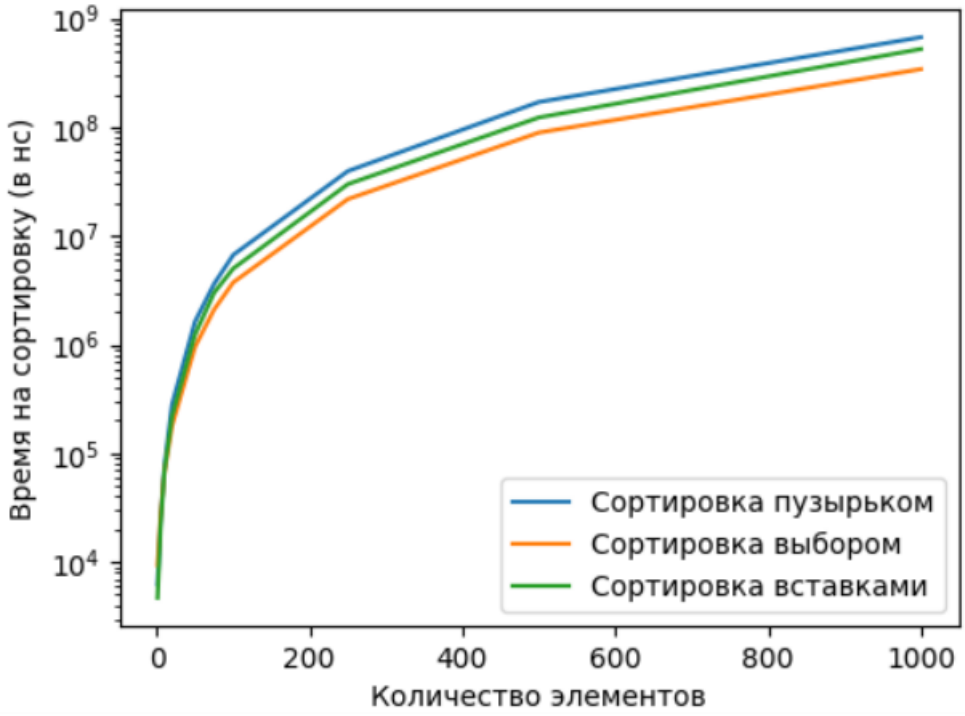

Рисунок. 4.6: График зависимости времени сортировки от количества элементов в массиве для массива случайных чисел

Также на рисунке 4.7 приведено сравнение сортировки пузырьком и сортировки выбором, так как по порядку они схожи.

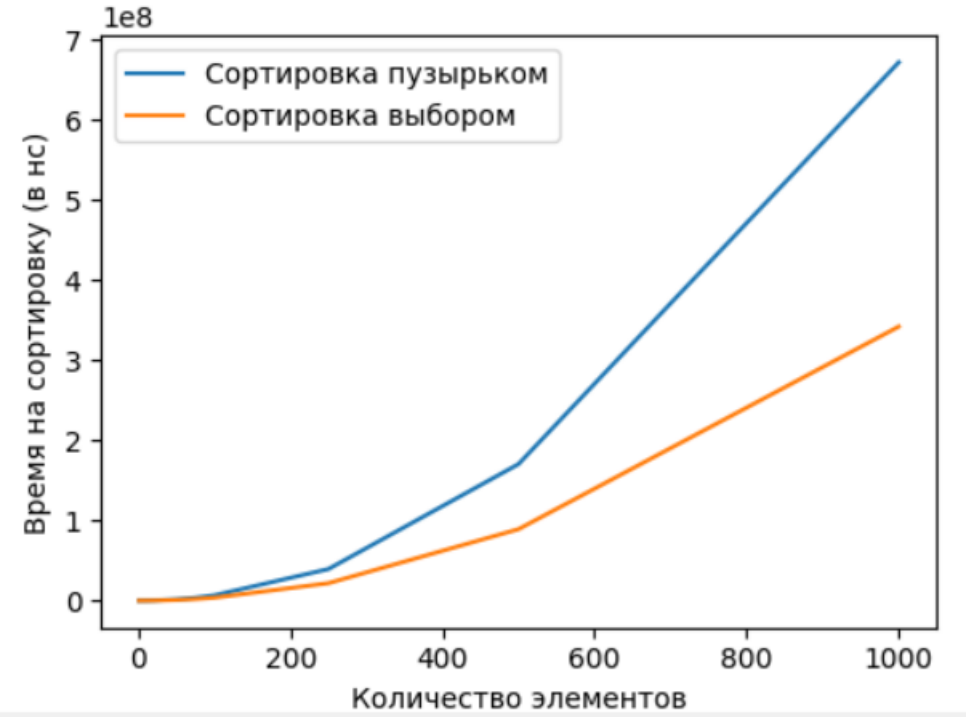

Рисунок. 4.7: График зависимости времени сортировки от количества элементов в массиве для массива случайных чисел

4.4 Вывод

Для массива, состоящих из случайных чисел, сортировка вставками показала лучшие результаты на всех N. Сортировка пузырьком показала самый медленный результат, выполняясь при $N = 1000$ почти в два раза дольше, чем остальные способы.

Для уже отсортированного массива сортировка вставками оказалась при $N = 1000$ в 250 раз быстрее, чем другие сортировки. Тем самым подтвердилось, что в лучшем случае сортировка вставками работает быстрее, чем остальные, так как её порядок - $O(N)$, а для других сортировок - $O(N^2)$. Результаты сортировки выбором почти не отличаются от результатов при массиве, состоящих из случайных чисел. Сортировка пузырьком происходила в полтора раза быстрее, чем при массиве случайных чисел, но и в этом случае она была самой медленной.

Для массива, сортированном в обратном порядке, сортировка выбором оказалась самым быстрым вариантом, превзойдя при $N = 1000$ в полтора раза сортировку вставками и в два раза - сортировку пузырьком.

Тем самым подтвердились следующие оценки трудоёмкости:

- сортировка вставками в лучшем случае работает на порядок быстрее, чем остальные способы;
- сортировка выбором оказалась быстрее всех при худшем случае, при оценке трудоёмкоси ей коэффициент перед N^2 был наименьшим.

5 Заключение

В ходе работы были рассмотрены три алгоритма сортировки: сортировка пузырьком, сортировка вставками и сортировками выбором. Были составлены схемы алгоритмов, подсчитана их трудоёмкость. Было реализовано работоспособное ПО, удалось провести анализ зависимости затрат по времени от количества элементов массива.

В лучшем случае сортировка вставками оказалась при $N = 1000$ на два порядка быстрее, чем остальные способы сортировки. Также сортировка вставками при всех N показала самое маленькое время для случая массива случайных чисел. В худшем случае сортировка выбором оказалась более эффективной при всех $N >= 10$. Сортировка пузырьком для всех N>10 при любом случае оказалась самой медленной. Дополнительных затрат на память для всех сортировок не потребовалось.

Литература

[1] Сортировка пузырьком. Режим доступа https://kvodo.ru/puzyirkovaya-sortirovka.html (дата обращения: 15.10.2021).

[2] Сортировка вставками. Режим доступа https://kvodo.ru/sortirovka-vstavkami-2.html (дата обращения: 15.10.2021).

[3] Сортировка выбором. Режим доступа https://kvodo.ru/sortirovka-vyiborom-2.html (дата обращения: 15.10.2021).

[4] time - Time access and conversions. Режим доступа: https://docs.python.org/3/library/time.html (дата обращения: 21.09.2021).

[5] Процессор Intel® Core™ i5-1135G7. 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: 15.10.2021).