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 по курсу "Анализ алгоритмов"

Тема Алгоритмы сортировки

Студент Ивахненко Д.А.

Группа ИУ7-56Б

Преподаватель Волкова Л.Л.

Москва — 2021 г.

Содержание

Введение

Алгоритм сортировки — это алгоритм для упорядочивания элементов в массиве каких-либо данных. Другими словами, это перегруппировка данных по какому-либо ключу.

В основном алгоритмы сортировки являются не конечной целью, а промежуточной - отсортированные данные намного легче обрабатывать. Например, чтобы удалить дубликаты в массиве, будет разумносначала отсортировать массив. Также намного легче производить поиск какой-либо информации в заранее отсортированном массиве. Ассимптотическая сложность при поиске в отсортированном массиве составляет $O(\log n)$ против $O(n)$ в произвольном массиве данных.

Каждый алгоритм сортировки имеет свои особенности реализации, однако в конечном итоге эту задачу можно разбить на следующие шаги:

1. Сравнение пары элементов массива. Для этого нужно указать способ сравнения (бинарное отношение) для двух элементов, например, учесть природу данных или задать ключ в случае, если записи в массиве имеют несколько полей;

2. Перестановка двух элементов, которые могут располагаться как последовательно друг за другом, так и находиться в разных частях массива;

3. Алгоритм должен завершать свою работу, когда массив становится полностью упорядоченным.

Алгоритмы сортировки оцениваются по скорости выполнения и эффективности использования памяти. Обычно асимптотическая сложность алгоритмов сортировок массивов размера n лежит в диапазоне от $O(n)$ до $O(n^2)$

Целью работы является анализ производительности алгоритмов сортировки. Для достижения поставленной цели необходимо выполнить следующие задачи:

- рассмотреть алгоритмы сортировки пузырьком, вставками и выбором;

- сравнить алгоритмы, выявить их достоинства и недостатки;

- разработать схемы выбранных алгоритмов;

- определить модель вычислений трудоемкости;
- на основе теоретических расчетов и выбранной модели вычислений провести сравнительный анализ трудоёмкости алгоритмов;
- определить средства для реализации алгоритмов;
- реализовать данные алгоритмы сортировки;
- выделить классы эквивалентности для сортировки массива;
- на основе выделенных классов эквивалентности разработать функциональные тесты;
- экспериментально провести сравнительный анализ производительности алгоритмов.

1 Аналитическая часть

В данном разделе будут описаны ключевые идеи каждого из выбранных алгоритмов сортировки.

1.1 Сортировка пузырьком

Сортировка пузырьком является одним из простейших алгоритмов сортировки данных.

Пусть имеется массив для сортировки размерностью N. В классической версии массив обходится слева направо. Алгоритм состоит из повторяющихся проходов по этому массиву - за каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется перестановка элементов так, чтобы пара стала упорядоченной.[1]

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. В результате каждого прохода по внутреннему циклу определяет ся очередной максимум подмассива, этот максимум помещается в конец рядом с предыдущим наибольшим элементом, вычисленным на преды дущем проходе, а наименьший элемент ближе к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма).

1.2 Сортировка выбором

Сортировка выбором — это, возможно, самый простой в реализации алгоритм сортировки. Как и в большинстве других подобных алгоритмов, в его основе лежит операция сравнения. Сравнивая каждый элемент с каждым, и в случае необходимости производя обмен, метод приводит последовательность к необходимому упорядоченному виду.

Шаги алгоритма[2]:

1. Находится номер минимального значения в текущем списке.

2. Производится обмен этого значения со знаением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции).

3. Сортируется хвост списка, исключив из рассмотрения уже отсортированные элементы.

1.3 Сортировка вставками

Идеи алгоритма сортировки вставками следующие[2]:

1. Массив делится на две части — отсортированную и неотсортированную.

2. Из неотсортированной части извлекается любой элемент.

3. Элемент вставляется в отсортированную часть массива, сохраняя ее упорядоченность.

4. Так происходит до тех пор, пока массив не будет полностью отсортирован.

Поскольку первая часть массива всегда отсортирована, в ней достаточно быстро можно найти своё место для элемента, извлеченного из неотсортированной части.

Метод выбора очередного элемента из исходного массива произволен, однако обычно (и с целью получения устойчивого алгоритма сортировки), элементы вставляются по порядку их появления во входном массиве.

1.4 Вывод

В данном разделе были рассмотрены ключевые идеи выбранных алгоритмов сортировки данных: пузырьком, выбором и вставками.

2 Конструкторская часть

В данном разделе представлены схемы алгоритмов, описанных в аналитическом разделе, а также будет подсчитана их трудоемкость.

2.1 Трудоемкость алгоритмов

2.1.1 Модель вычислений

Для последующего вычисления трудоемкости введём модель вычислений:

1. Трудоемкость базовых операций: Пусть операции из (2.1) имеют единичную трудоемкость.

$$+, -, =, ==, !=, <, >, <=, >=, [], +=, -=, <<, >> \tag{2.1}$$

   Пусть операции из (2.2) имеют трудоемкость 2.

$$/, *, \% \tag{2.2}$$

2. Трудоемкость условного оператора “if условие then A else B” рассчитывается, как (2.3).

$$f_{if} = f_{\text{условия}} + \begin{cases} f_A, & \text{если условие выполняется,} \\ f_B, & \text{иначе.} \end{cases} \tag{2.3}$$

3. Трудоемкость цикла рассчитывается по C-подобной модели, то есть “for (i = 0; i < N; i += 1)”, как (2.4).

$$f_{for} = f_{\text{инициализации}} + f_{\text{сравнения}} + N(f_{\text{тела}} + f_{\text{инкремента}} + f_{\text{сравнения}}) \tag{2.4}$$

4. Трудоемкость вызова функции равна 0.

5. Трудоемкость внутреннего цикла с переменным пределом рассчитывается, как (2.5).

$$f = f_{\text{служебное}} + Q_{\text{сравнения}} \cdot f_{\text{на 1 сравнение}} \tag{2.5}$$

$f_{\text{служебное}}$ – затраты на содержание внешнего цикла, инициализацию и стартовое сравнение вложенного цикла.

$Q_{\text{сравнения}}$ – количество сравнений.

$f_{\text{на 1 сравнение}}$ – затраты на содержание 1 сравнения, то есть условия инкремента и сравнения вложенного цикла.

2.1.2 Алгоритм сортировки пузырьком

Размер исходного массива принимается за N.

Для рассчета трудоемкости внутреннего цикла с переменным пределом используется формула 2.5.

$f_{\text{на 1 сравнение}}$ определяется выражением 2.6.

$$f_{\text{на 1 сравнение}} = 7 + \begin{cases} 0, & \text{в лучшем случае} \\ 9, & \text{в худшем случае} \end{cases} \tag{2.6}$$

$Q_{\text{сравнения}}$ определяется выражением 2.7.

$$Q_{\text{сравнения}} = \frac{N(N-1)}{2} = \frac{N^2}{2} - \frac{N}{2} \tag{2.7}$$

$f_{\text{служебное}}$ определяется выражением 2.8.

$$f_{\text{служебное}} = 3 + (N-1) \cdot (3+3) = 6N - 3 \tag{2.8}$$

Трудоемкость в лучшем случае (отсортированный массив) описывается выражением 2.9

$$f_{\text{bubble-best}} = (6N-3) + (\frac{N^2}{2} - \frac{N}{2}) \cdot 7 = 4.5N^2 + 2.5N - 3 \approx 4.5N^2 \tag{2.9}$$

Трудоемкость в худшем случае (отсортированный в обратном порядке

массив) описывается выражением 2.10

$$f_{\text{bubble-worst}} = (6N - 3) + (\frac{N^2}{2} - \frac{N}{2}) \cdot 16 = 8N^2 - 2N - 3 \approx 8N^2 \qquad (2.10)$$

2.1.3 Алгоритм сортировки выбором

Размер исходного массива принимается за N.

Для рассчета трудоемкости внутреннего цикла с переменным пределом также, как и для алгоритма сортировки пузырьком, используется формула 2.5.

$f_{\text{на 1 сравнение}}$ определяется выражением 2.11.

$$f_{\text{на 1 сравнение}} = 5 + \begin{cases} 0, & \text{в лучшем случае} \\ 1, & \text{в худшем случае} \end{cases} \qquad (2.11)$$

$Q_{\text{сравнения}}$ определяется выражением 2.12.

$$Q_{\text{сравнения}} = \frac{N(N-1)}{2} = \frac{N^2}{2} - \frac{N}{2} \qquad (2.12)$$

$f_{\text{служебное}}$ определяется выражением 2.13.

$$f_{\text{служебное}} = 3 + (N - 1) \cdot (3 + 4) = 7N - 4 \qquad (2.13)$$

Трудоемкость в лучшем случае (отсортированный массив) описывается выражением 2.14

$$f_{\text{select-best}} = (7N - 4) + (\frac{N^2}{2} - \frac{N}{2}) \cdot 5 = 2.5N^2 + 4.5N - 4 \approx 2.5N^2 \qquad (2.14)$$

Трудоемкость в худшем случае (отсортированный в обратном порядке массив) описывается выражением 2.15

$$f_{\text{select-worst}} = (7N - 4) + (\frac{N^2}{2} - \frac{N}{2}) \cdot 6 = 3N^2 + 4N - 4 \approx 3N^2 \qquad (2.15)$$

2.2 Схемы алгоритмов

На рисунке 2.1 представлена схема алгоритма сортировки пузырьком.

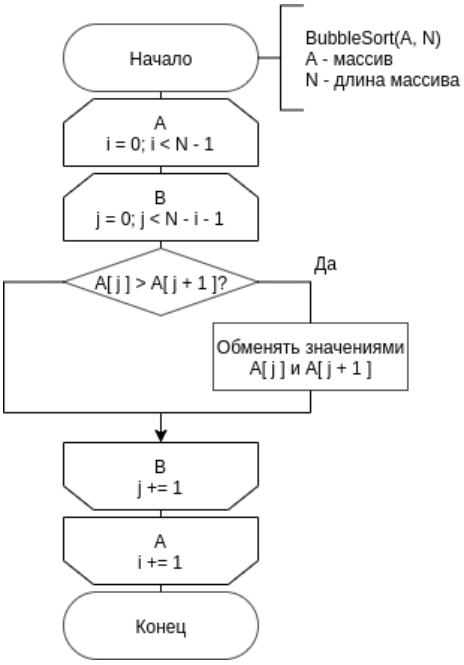

Рисунок 2.1 – Схема алгоритма сортировки пузырьком

На рисунке 2.2 представлена схема алгоритма сортировки выбором.

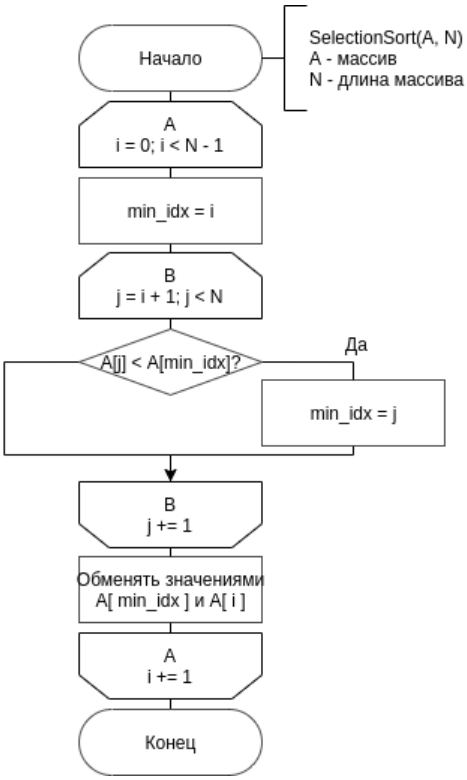

Рисунок 2.2 – Схема алгоритма сортировки выбором

На рисунке 2.3 представлена схема алгоритма сортировки вставками.

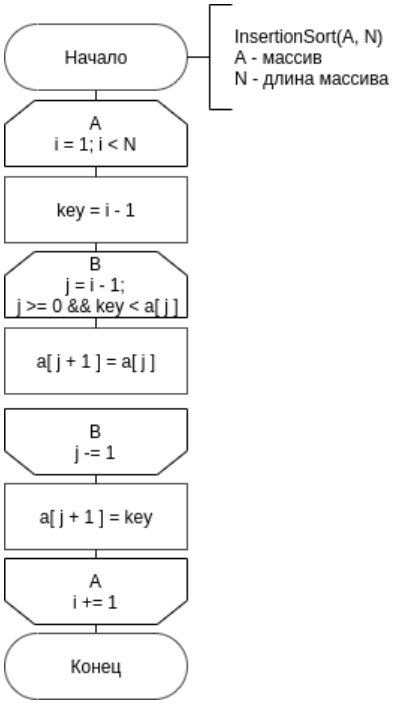

Рисунок 2.3 – Схема алгоритма сортировки выбором

2.3 Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы трех алгоритмов сортировки. На основе введенной модели вычислений оценена трудоемкость алгоритма сортировки пузырьком и алгоритма сортировки выбором в лучшем и худшем случаях.

Из преведнных оценок трудоемкостей алгоритмов следует, что алгорим сортировки выбором имеет меньшую трудоемкость, чем алгоритм сортировки пузырьком. Для лучшего и худшего случаев соответственно в 1.8 и 2.7 раза.

3 Технологическая часть

В данном разделе приведны требования к программному обеспечению, средства реализации и листинги кода

3.1 Требования к ПО

К программе предъявляется следующие требования:

- предоставить возможность работы в двух режимах: проведения эксперимента и ручного тестирования;
- в режиме ручного тестирования предоставить пользователю как самостоятельно осуществлять ввод двух массива, который необходимо отсортировать, так и использовать автозаполнение псевдослучайными числами от 0 до 99;
- результатом работы программы в ручном режиме должны являвляться три массива - результаты сортировки исходного массива тремя сортировами: пузырьком, выбором и вставками;
- в режиме проведения эксперимента результатом работы программы является результат эксперимента в текстовом виде, а также построенные графики в виде файла с изображением;
- программа должна корректно обрабатывать любые действия пользователя, в том числе, например, ввод массива длиной 0.

3.2 Средства реализации

Для реализации алгоритмов перемножения матриц был выбран язык программирования Python3[3], что обусловлено простотой и скоростью написания программ, а также наличием встроенных библиотек для построения графиков функций и тестирования. В качестве среды разработки был выбран `PyCharm`[4], как наиболее популярная IDE для Python3.

3.3 Функциональное тестирование

При разработке функционалых тестов были выделены следующие классы эквивалентности:

- отсортированный массив;
- массив, отсортированный в обратном порядке;
- массив из нечетного количества элементов;
- массив из четного количество элементов;
- массив из одного элемента;
- пустой массив;
- произвольный массив.

В соответствии с данными классами эквивалентности были разработаны тесты, представленные в таблице 3.1.

Таблица 3.1 – Тестирование функций

Входные данные	Ожидаемый результат	Реальный результат
$[-1, 0, 5, 9, 11]$	$[-1, 0, 5, 9, 11]$	$[-1, 0, 5, 9, 11]$
$[11, 9, 5, 0, -1]$	$[-1, 0, 5, 9, 11]$	$[-1, 0, 5, 9, 11]$
$[3, -1, 0, 7, 5]$	$[-1, 0, 3, 5, 7]$	$[-1, 0, 3, 5, 7]$
$[3, -1, 0, 7]$	$[-1, 0, 3, 7]$	$[-1, 0, 3, 7]$
$[7]$	$[7]$	$[7]$
$[]$	$[]$	$[]$

Все тесты пройдены успешно.

3.4 Реализация алгоритмов

В листинге 3.1 представлена реализация алгоритма сортировки пузырьком.

Листинг 3.1 – Реализация алгоритма сортировки пузырьком

```
def bubble_sort(a: ArrayInt, n: int) -> None:
    i = 0
    while i < n - 1:

        j = 0
        while j < n - i - 1:
            if a[j] > a[j + 1]:
                a[j], a[j + 1] = a[j + 1], a[j]
            j += 1

        i += 1
```

В листинге 3.2 представлена реализация алгоритма сортировки вставками.

Листинг 3.2 – Реализация алгоритма сортировки вставками

```
def insertion_sort(a: ArrayInt, n: int) -> None:
    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
```

В листинге 3.3 представлена реализация алгоритма сортировки выбором.

Листинг 3.3 – Реализация алгоритма сортировки выбором

```
def selection_sort(a: ArrayInt, n: int) -> None:
    i = 0
    while i < n - 1:
        min_idx = i

        j = i + 1
        while j < n:
            if a[j] < a[min_idx]:
                min_idx = j
            j += 1

        a[min_idx], a[i] = a[min_idx], a[i]

        i += 1
```

3.5 Вывод

В данном разделе были выделены классы эквивалентности для операции сортировки массива, на основе которых разработаны функциональные тесты для ПО, также были реализованы сами алгоритмы сортировки массива на языке Python3.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялся эксперимент:

- операционная система: Ubuntu[5] Linux x86_64;
- память: 16 GiB;
- процессор: AMD Ryzen™ 7 4700U[6].

4.2 Проведение эксперимента

Эксперимент проводился на упорядоченных, упорядоченных в обратном порядке, а также неупорядоченных массивах различных размеров, заполненных псевдослучайными числами из диапазона $[0; 999]$.

Для каждого размера массива при помощи встроенного встроенного модуля `timeit`[7] языка Python3 было произведено 100 замеров процессорного времени, после чего определено среднее время выполнения алгоритма.

Результаты для отсортированного массива представлены в табл. 4.1 и на рис. 4.1.

Таблица 4.1 – Время выполнения алгоритмов (в мс) для отсортированного массива

Размер	Пузырьком	Вставками	Выбором
10	0.0042	0.0011	0.0035
50	0.0938	0.0048	0.0673
100	0.3697	0.0098	0.2595
250	2.3094	0.0235	1.594
500	11.9433	0.0563	8.2343
750	28.4584	0.0897	19.362
1000	52.1071	0.1205	34.7

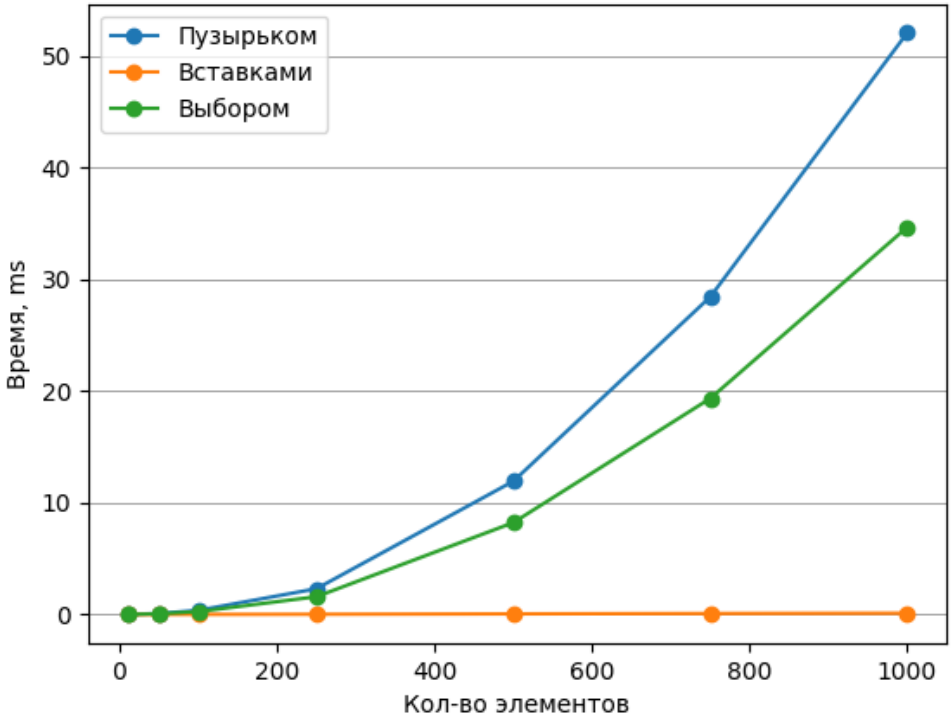

Рисунок 4.1 – Зависимость времени работы алгоритмов от кол-ва элементов для отсортированного массива

Результаты для отсортированного в обратном порядке массива приведены в табл. 4.2 и на рис. 4.2.

Таблица 4.2 – Время выполнения алгоритмов (в мс) для отсортированного в обратном порядке массива

Размер	Пузырьком	Вставками	Выбором
10	0.0069	0.0042	0.0036
50	0.1609	0.0872	0.0688
100	0.6399	0.3416	0.2671
250	4.0055	2.1369	1.6483
500	19.0076	8.9443	8.6054
750	46.0853	20.5732	20.2551
1000	81.7425	36.9837	36.4731

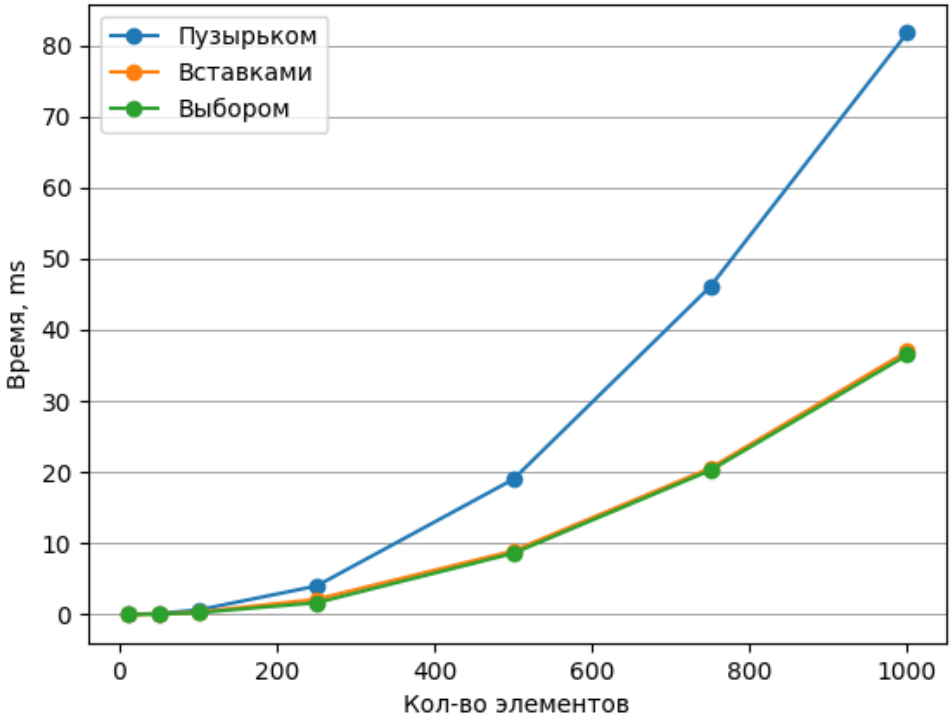

Рисунок 4.2 – Зависимость времени работы алгоритмов от кол-ва элементов при обратно отсортированном массиве

Результаты для неупорядоченного массива приведены в табл. 4.3 и на рис. 4.3.

Таблица 4.3 – Время выполнения алгоритмов (в мс) для неупорядоченного массива

Размер	Пузырьком	Вставками	Выбором
10	0.0058	0.0029	0.0036
50	0.1308	0.0477	0.0687
100	0.5152	0.1726	0.263
250	3.2178	1.0676	1.6146
500	15.8039	4.3968	8.3057
750	37.7369	10.8403	19.4612
1000	68.9718	19.3811	35.0158

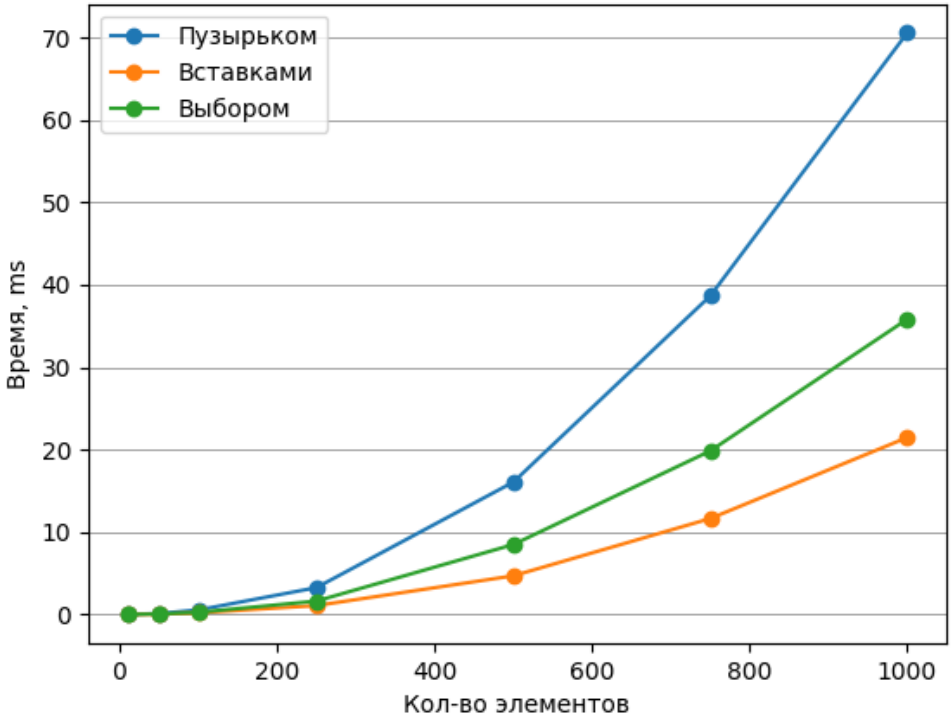

Рисунок 4.3 – Зависимость времени работы алгоритмов от кол-ва элементов при неупорядоченном массиве

4.3 Вывод

Алгоритмы сортировки пузырьком и выбором на отсортированных массивах работают медленнее, чем сортировка вставками. Так уже при длине массива, равной 10 элементам, сортировка вставками быстрее сортировки выбором в три раза, а сортировки пузырьком - в четыре. С ростом размера массива преимущество только увеличивается: на массивах с 1000 элементами сортировка вставками выполняется быстрее сортировки выбором в 300 раз, а сортировки пузырьком в 430 раз.

На обратно отсортированных массивах сортировка вставками сравнима по быстродействию с сортировкой выбором. При длине массива, равной 10 элементам, сортировка вставками медленнее сортировки выбором в 1.17 раз, при 500 элементах - 1.04, а при 1000 элементах всего лишь в 1.01 раз. При этом сортировка пузырьком медленнее остальных примерно в 2.2 раза.

Если рассматривать случай, более приближенный к реальности, когда массивы никак не упорядочены, то сортировка вставками снова оказывается

быстрее остальных. Она быстрее сортировки выбором и пузырьком в 1.2 и 2 раза соответственно при 10 элементах в массиве, в 1.9 и 3.6 раза - при 500 элементах, в 1.8 и 3.5 раза - при 1000 элементах.

Следовательно, можно сделать вывод, что сортировка вставками является наиболее эффективной сортировкой из рассмотренных, а сортировка пузырьком, наоборот, наименее эффективной.

Заключение

В ходе выполнения лабораторной работы была выполнены поставленные задачи, а именно:

- рассмотрены алгоритмы сортировки пузырьком, вставками и выбором;
- разработаны схемы выбранных алгоритмов;
- на основе теоретических расчетов и выбранной модели вычислений проведен сравнительный анализ трудоёмкости алгоритмов;
- реализованы выбранные алгоритмы сортировки;
- разработаны функциональные тесты для программы;
- экспериментально проведен сравнительный анализ быстродействия алгоритмов.

Экспериментально были установлены различия в производительности сортировок пузырьком, выбором и вставками. Так алгоритмы сортировки выбором и пузырьком на отсортированных массивах работают медленнее, чем сортировка вставками в 3 и 4 раза соответственно при 10 элементах, и в 300 и 430 раз соответственно при 1000 элементах.

На обратно отсортированных массивах сортировка вставками сравнима по быстродействию с сортировкой выбором. При длине массива 500 элементов разница в быстродействии 4%, а при 1000 - всего лишь в 1%. При этом сортировка пузырьком медленнее остальных примерно в 2.2 раза.

Однако на практике массивы никак не упорядочены. В таких случаях сортировка вставками снова оказывается быстрее остальных. Она быстрее сортировки выбором и пузырьком в 1.2 и 2 раза соответственно при 10 элементах, в 1.9 и 3.6 раза - при 500 элементах, в 1.8 и 3.5 раза - при 1000 элементах.

Следовательно, можно сделать вывод, что сортировка вставками является наиболее эффективной сортировкой из рассмотренных, а сортировка пузырьком, наоборот, наименее эффективной.

Литература

[1] Левитин А. В. Метод грубой силы: Пузырьковая сортировка // Алгоритмы. Введение в разработку и анализ. - М.: Вильямс, 2006. С. 144–146.

[2] Левитин А. В. Метод грубой силы: Сортировка выбором // Алгоритмы. Введение в разработку и анализ. - М.: Вильямс, 2006. С. 143–144.

[3] Python 3.9.7 documentation [Электронный ресурс]. Режим доступа: `https://docs.python.org/3/` (дата обращения: 28.09.2021).

[4] PyCharm: IDE для профессиональной разработки на Python [Электронный ресурс]. Режим доступа: `https://www.jetbrains.com/ru-ru/pycharm/` (дата обращения: 28.09.2021).

[5] Ubuntu 20.04.3 LTS [Электронный ресурс]. Режим доступа: `https://ubuntu.com/download/desktop` (дата обращения: 28.09.2020).

[6] Проессор AMD Ryzen™ 7 4700U [Электронный ресурс]. Режим доступа: `https://www.amd.com/ru/products/apu/amd-ryzen-7-4700u` (дата обращения: 28.09.2021).

[7] timeit — Measure execution time of small code snippets [Электронный ресурс]. Режим доступа: `https://docs.python.org/3/library/timeit.html` (дата обращения: 28.09.2021).