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 (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 Программная инженерия**

ОТЧЕТ

по лабораторной работе № 3

Название: Алгоритмы сортировки

Дисциплина: Анализ алгоритмов

Студент	ИУ7-51Б		О.А. Тюрин
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Л.Л. Волкова
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2020

Содержание

Введение

На данный момент существует огромной количество сортировок, именно поэтому их необходимо уметь сравнивать для того, чтобы выбрать наиболее подходящий для конкретного случая алгоритм.

Цель работы: оценить трудоемкость алгоритмов сортировки массива и получить практический навык реализации алгоритмов.

Задачи работы:

1. изучить алгоритмы сортировки массива: пузырьком, выбором и *quick sort*;
2. оценить трудоемкость данных алгоритмов сортировки;
3. реализовать данные алгоритмы сортировки массива на одном из языков программирования;
4. сравнить рассматриваемые алгоритмы сортировки между собой.

Алгоритмы сортировки оцениваются по следующим критериям:

- затрачиваемое время;
- затрачиваемая память.

1 Аналитическая часть

Сортировка - это процесс упорядочивания наборов данных одного типа по возрастанию или убыванию значения какого-либо признака.
Области применения сортировки:

- анализ данных;
- физика;
- экономика.

1.1 Описание алгоритмов

В данном разделе будут описан каждый исследуемый алгоритм.

1.2 Сортировка пузырьком

Алгоритм проходит по массиву n-1 раз или до тех пор, пока массив не будет полностью отсортирован. В каждом проходе элементы попарно сравниваются и, при необходимости, меняются местами. При каждом проходе алгоритма по внутреннему циклу, очереднй наибольший элемент ставится на своё место в конец неотсортированного массива. Таким образом наибольшие элементы "всплывают"как пузырек.

1.3 Сортировка выбором

Берётся первый элемент массива, далее происходит трассировка оставшегося массива в поисках наименьшего числа. Когда мы пробегаем весь массив, если меньшее число найдено — меняем его местами с первым элементом. После этого берётся второй элемент, и все повторяется, пока для каждого элемента мы не будет пройдена оставшаяся часть массива.

1.4 Быстрая сортировка

Массив разбивается на два (возможно пустых) подмассива. Таких, что в одном подмассиве каждый элемент меньше либо равен опорному, и при этом не превышает любой элемент второго подмассива. Опорный элемент вычислыется в ходе процедуры разбиения. Подмассивы сортируются с помощью рекурсивного вызова процедуры быстрой сортировки. Поскольку подмассивы сортируются на месте, для их объединения не трубуются никакие действия.

Вывод

В данном разделе было приведено описание работы каждого из алгоритмов.

2 Конструкторская часть

Требования к вводу: на вход подаются размерность массива и сам массив чисел.

Требования к программе::

- корректная сортировка массива;
- программа должна корректно обрабатывать массивы нулевой размерности.

2.1 Разработка алгоритмов

В данном разделе представлены схемы реализуемых алгоритмов.

На рисунке 2.1 представлена схема алгоритма сортировки пузырьком.

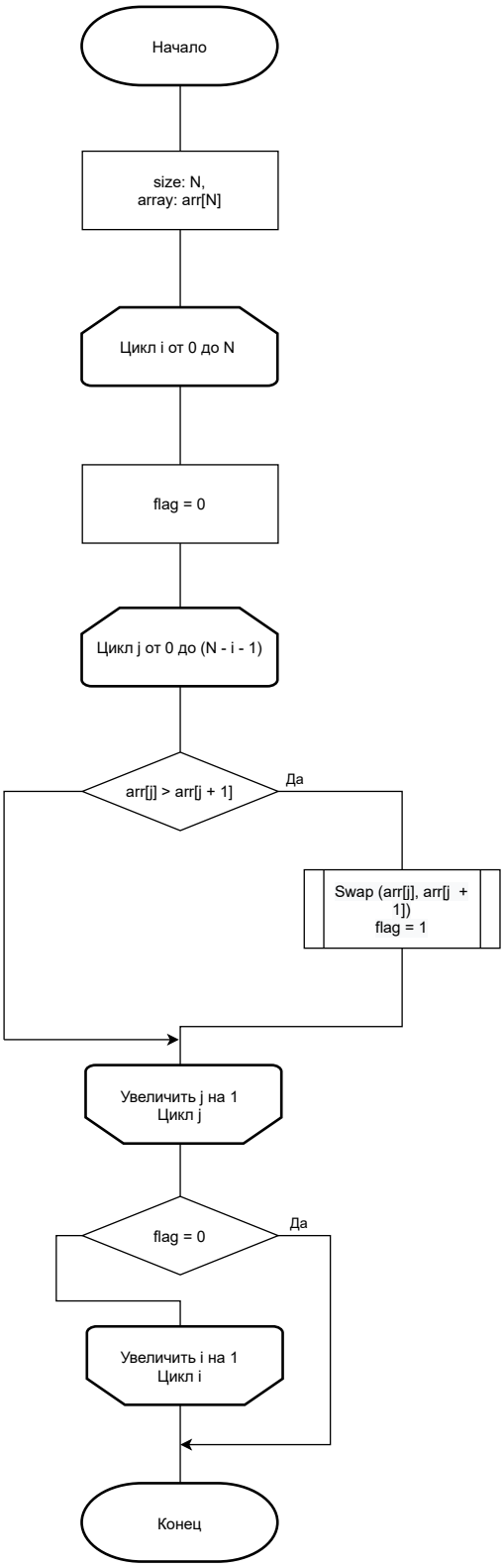


Рис. 2.1: Схема алгоритма сортировки пузырьком

На рисунке 2.2 представлена схема алгоритма сортировки выбором.

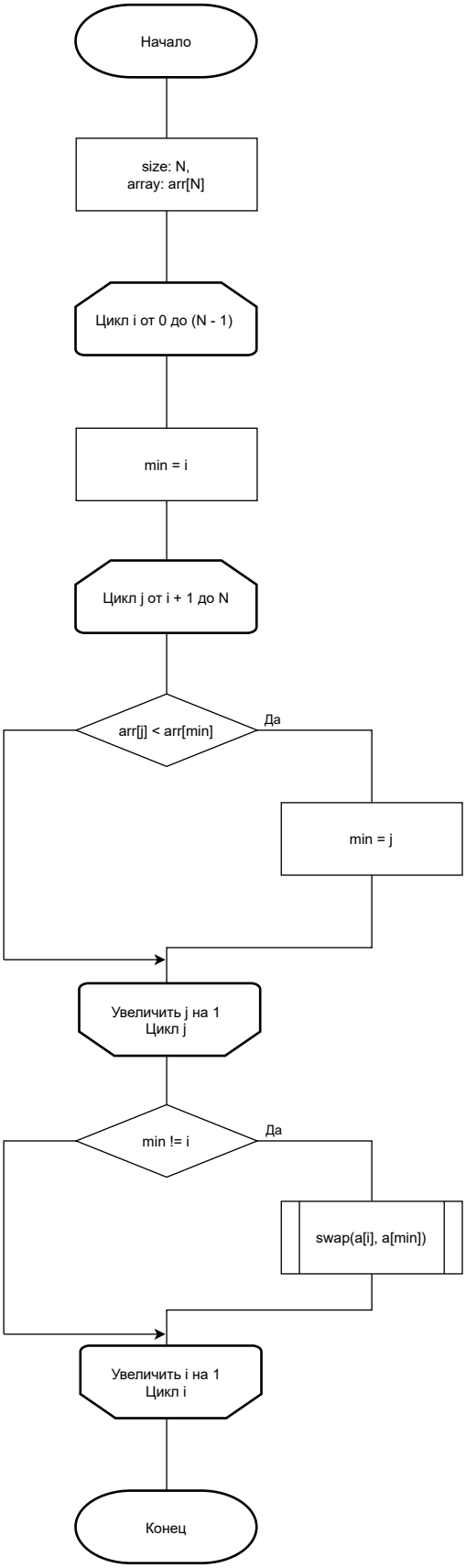


Рис. 2.2: Схема алгоритма сортировки Выбором

На рисунке 2.3 представлена схема алгоритма быстрой сортировки.

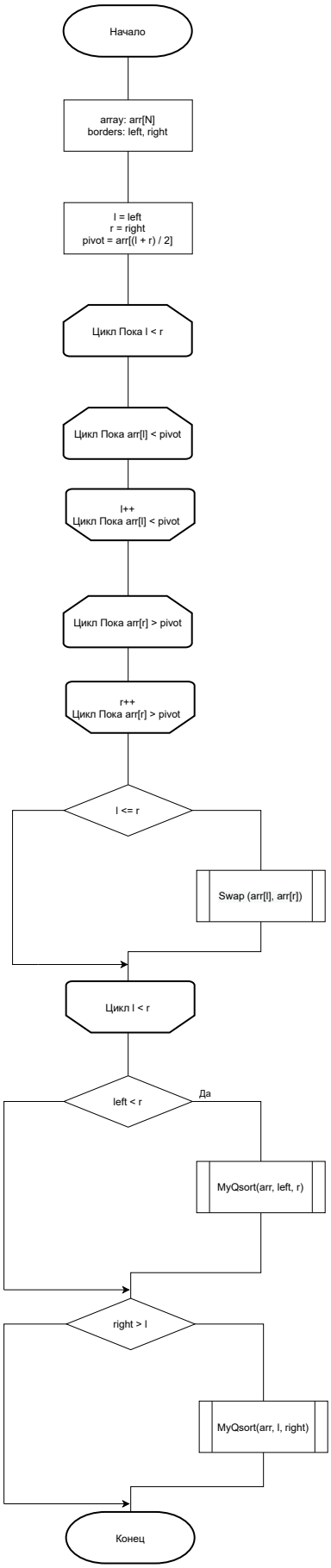


Рис. 2.3: Схема алгоритма быстрой сортировки

Вывод

В данном разделе были рассмотрены схемы реализуемых алгоритмов.

3 Технологическая часть

В данном разделе будет описана технологическая часть лабораторной работы: требования к ПО, листинг кода, сравнительный анализ всех алгоритмов.

3.1 Требования к программному обеспечению

Входные данные: размерность массива, сам массив
Выходные данные: отсортированный массив
Среда выполнения: Windows 10 x64

3.2 Средства реализации

Для выполнения данной работы был использован Python 3.9.0

3.3 Листинг кода

В данном разделе будет представлен листинг кода разработанных алгоритмов.

Ниже, на листинге 3.1, представлена реализация алгоритма сортировки пузырьком:

Листинг 3.1: Сортировка пузырьком

```
def bubble_sort(arr):
    if len(arr) <= 1:
        return

    for i in range(len(arr)):
        flag = 0;
        for j in range(len(arr) - 1):
            if arr[j] > arr[j + 1]:
                flag = 1
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
        if flag == 0:
            break

    return arr
```

Ниже, на листинге 3.2, представлена реализация алгоритма сортировки выбором:

Листинг 3.2: Сортировка выбором

```
def selection_sort(arr):
        if len(arr) <= 1:
                return

        for i in range(len(arr) - 1):
                min_ind = i
                for j in range(i + 1, len(arr)):
                        if arr[j] < arr[min_ind]:
                                min_ind = j
                if min_ind != i:
                        arr[i], arr[min_ind] = arr[min_ind], arr[i]

        return arr
```

Ниже, на листинге 3.3, представлена реализация алгоритма быстрой сортировки:

Листинг 3.3: Быстрая сортировка

```
def qsort(arr, left, right):
    if len(arr) <= 1:
        return

    l, r = left, right
    pivot = arr[int((l + r) / 2)]

    while (l <= r):
        while arr[l] < pivot:
            l += 1
        while arr[r] > pivot:
            r -= 1

        if (l <= r):
            arr[l], arr[r] = arr[r], arr[l]
            l, r = l + 1, r - 1

    if left < r:
        qsort(arr, left, r)
    if right > l:
        qsort(arr, l, right)
```

3.4 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

- базовые операции стоимостью 1: +, -, *, /, =, ==, <=, >=, !=, +=, [], получение полей класса (в том числе и геттеры);
- оценка трудоемкости цикла: $F_{cycle} = init + N * (statement + iteration + F_{body}) + statement$, где init - инициализация цилка, N - количество итераций цикла, iteration - действие в конце каждой итерации цикла, statement - выражение, описывающее условие цикла;
- стоимость условного перехода применим за 0, стоимость вычисления; условия остаётся.

3.4.1 Трудоёмкость алгоритма сортировки пузырьком

Рассмотри трудоёмкость алгоритма сортировки пузырьком.
Лучший случай: массив изначально отсортирован. Был сделан один проход, в ходе которого было установлено, что массив уже отсортирован

$$1 + 1 + 1 + 1 + 1 + (size - 1)(3 + 1 + 4) + 3 + 1 \tag{3.1}$$

$$8 * (size) + 1 = O(n) \tag{3.2}$$

Худший случай: массив отсортирован в обратном порядке

$$1 + 1 + size * (1 + 1 + 1 + 1 + (size - 1)(2 + 1 + 4 + 1 + 3) + 2) + 1 \tag{3.3}$$

$$11 * size * size - 5 * size + 3 = O(n^2) \tag{3.4}$$

3.4.2 Трудоёмкость алгоритма сортировки выбором

Рассмотри трудоёмкость алгоритма сортировки выбором.
Лучший случай: массив изначально отсортирован.

$$1 + 1 + (size - 1)(2 + 1 + 1 + 2 + (size - i - 1)(3) + 1 + 1) + 2 \tag{3.5}$$

$$3 * size * size + ... = O(n^2) \tag{3.6}$$

Как видно, даже в самом лучшем случае сложность квадратичная.

Худший случай: массив отсортирован в обратном порядке

$$1 + 1 + (size - 1)(2 + 1 + 1 + 2 + (size - i - 1)(3) + 1 + 3 + 1) + 2 \tag{3.7}$$

$$3 * size * size + ... = O(n^2) \tag{3.8}$$

3.4.3 Трудоёмкость алгоритма быстрой сортировки

Рассмотрим трудоёмкость алгоритма быстрой сортировки.
Лучший случай: Разбиение на подмассивы происходит таким образом, что их размерности совпадают, либо разлчиаются на 1. При таком раскладе глубина рекурсии будет равняться $log(size)$. В любом случае нам необходимо будет сравнить все элементы массива. Таким образом сложность равна:

$$O(size * log(size)) \tag{3.9}$$

Худший случай: Разбиение на подмассивы происходит таким образом, что их размерность одного из подмассивов равен 1 (т.е. на каждом из этапов в качестве pivot был выбран либо наименьший, либо наибольший из обрабатываемых элементов). При таком раскладе глубина рекурсии будет равняться n. В любом случае нам необходимо будет сравнить все элементы массива. Таким образом сложность равна:

$$O(n^2) \tag{3.10}$$

3.5 Описание тестирования

Были проведены следующие тривиальные тесты:

```
1 arr = [10, -1, 0 ,4, 5, 8]
2
3 Bubble sort: [10, -1, 0 ,4, 5, 8]
4
5 Selection sort: [10, -1, 0 ,4, 5, 8]
6
7 Quick sort: [10, -1, 0 ,4, 5, 8]
```

Также были проведены тесты на больших размерностях со случайными числами в качестве элементов. В качестве эталона для сравнения массивов была использована функция sorted.
Ниже, на листинге 3.4, представлен фрагмент кода тестирования корректной работы реализации алгоритмов

Листинг 3.4: Тестирование корректной работы алгоритмов

```
1  def test():
2      arr1 = [random.randint(1, 100) for i in range(100)]
3      arr4 = arr3 = arr2 = arr1
4
5      bubble_sort(arr1)
6      selection_sort(arr2)
7      qsort(arr3, 0, len(arr3) - 1)
8      sorted(arr4)
9
10     if arr1 == arr2 == arr3 == arr4:
11         print("OK")
```

Все тесты пройдены успешно.

Вывод

В данном разделе был представлен листинг реализованных алгоритмов, а также описание тестирование корректности их работы.

4 Экспериментальная часть

В данной части работы будут приведен анализ алгоритмов на основе эксперементальных данных.

4.1 Сравнительный анализ на основе замеров времени работы алгоритмов

Был проведён замер времени работы каждого из алгоритмов. Каждый эксперимент на каждой размерности был произведён 10 раз, затем бралось среднее арифметическое полученного результата.
Все эксперименты были проведены для размерностей, начиная со 100, заканчивая 1000 с шагом 100.
Ниже во всех таблицах время сортировки указано в миллисекундах.
Ниже, в таблице 4.1, представлена таблица замеров времени работы алгоритмов.
Ниже, на графиках 4.1.1 и 4.1.2, представлена графическая интерпретация

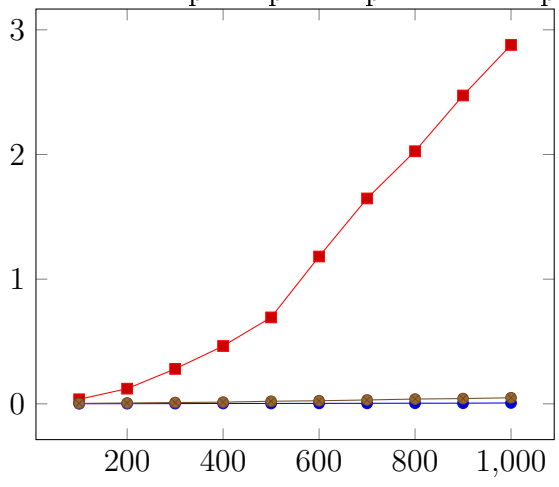
4.1.1 Эксперимент 1

Массив отсортирован

Таблица 4.1: Замеры времени работы алгоритмов

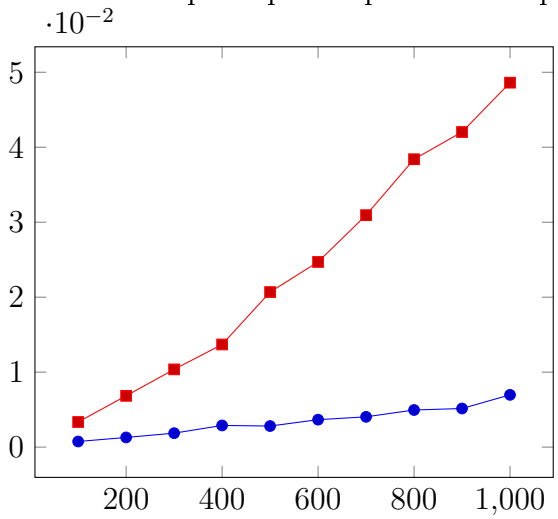
Размерность	Bubble	Selection	QSort
100	0.00075	0.0365	0.00335
200	0.00129	0.12175	0.00684
300	0.00186	0.28013	0.01037
400	0.00289	0.46373	0.01369
500	0.00281	0.69324	0.02068
600	0.00366	1.18182	0.0247
700	0.00404	1.64755	0.03094
800	0.00495	2.02626	0.0384
900	0.00515	2.47341	0.04203
1000	0.00697	2.87887	0.04861

График 4.1.1: Замеры вермени работы алгоритмов



Легенда:
Синий цвет - Сортировка пузырьком

Красный цвет - Сортировка выбором
Коричневый цвет - Быстрая сортировка
График 4.1.2: Замеры вермени работы алгоритмов



Легенда:
Синий цвет - Сортировка пузырьком
Красный цвет - Быстрая сортировка

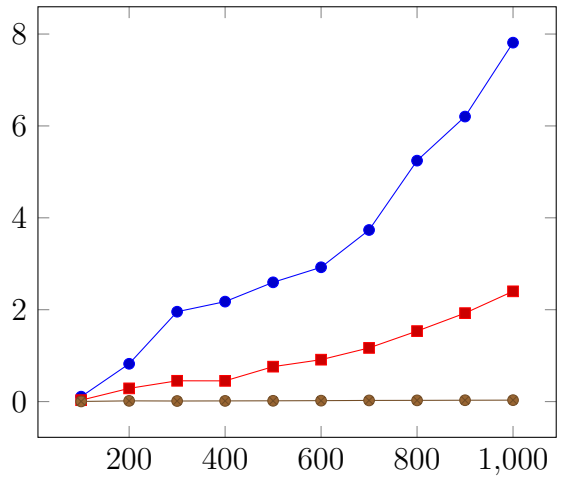
4.1.2 Эксперимент 2

Массив отсортирован в обратном порядке
Ниже, в таблице 4.2, представлена таблица замеров времени работы алгоритмов.
Ниже, на графике 4.2, представлена графическая интерпретация

Таблица 4.2: Замеры времени работы алгоритмов

Размерность	Bubble	Selection	QSort
100	0.10772	0.03127	0.00349
200	0.82298	0.28794	0.01623
300	1.95633	0.453	0.01292
400	1.48573	0.45165	0.01531
500	2.59618	1.1219	0.01746
600	2.92382	0.91247	0.02086
700	3.73524	1.16877	0.02568
800	5.24454	1.53452	0.02697
900	6.2039	1.92548	0.02995
1000	7.8126	2.40036	0.0319

График 4.2: Замеры вермени работы алгоритмов



Легенда:
Синий цвет - Сортировка пузырьком
Красный цвет - Сортировка выбором
Коричневый цвет - Быстрая сортировка

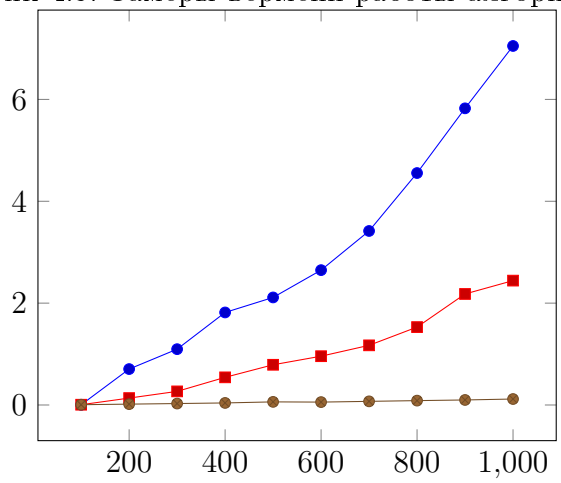
4.1.3 Эксперимент 3

Массив заполнен случайными значениями Ниже, в таблице 4.3, представлена таблица замеров времени работы алгоритмов.
Ниже, на графике 4.3, представлена графическая интерпретация

Таблица 4.3: Замеры времени работы алгоритмов

Размерность	Bubble	Selection	QSort
100	0.00438	0.00539	0.00563
200	0.70537	0.13577	0.01681
300	1.09619	0.26728	0.02833
400	1.81696	0.54301	0.04036
500	2.11177	0.78962	0.06112
600	2.64796	0.95799	0.05681
700	3.41683	1.1714	0.0715
800	4.55442	1.52958	0.08578
900	5.82487	2.17994	0.09873
1000	7.04937	2.44209	0.11722

График 4.3: Замеры вермени работы алгоритмов



Легенда:
Синий цвет - Сортировка пузырьком
Красный цвет - Сортировка выбором
Коричневый цвет - Быстрая сортировка

4.2 Сравнительный анализ на материале экспериментальных данных

В результате экспериментов было выявлено, что теоретичесие расчёты сложности верны. В лучшем случае сортировка пузырьком имеет линейное время, а сортировка выбором - квадратичное. Но в то же время в худшем случае оба алгоритма имеют одинаковую сложность - квадратичную. Однако сортировка пузырьком имеет больший коэфициент при квадрате, следовательно, и работает она медленнее. В то же время быстрая сортировка прекрасна показала себя во всех экспериментах.

Вывод

В данном разделе были представлены эксперименты, проведённые над реализованным алгоритмами.

Заключение

В ходе работы были изучены и реализованы следующие алгоритмы сортировки: пузырьковая сортировка, сортировка выбором и быстрая сортировка. Была дана теоретическая оценка всех рассматриваемых алгоритмов, было проведено тестирование и выполнено сравнение всех рассматриваемых алгоритмов. В ходе исследования была установлена зависимость времени выполнения алгоритмов от изначального заполнения массивов.

Список использованной литературы

[1] Основные виды сортировок и примеры их реализации. [электронный ресурс]. Режим доступа: https://academy.yandex.ru/posts/osnovnye-vidy-sortirovok-i-primery-ikh-realizatsii (Дата обращения: 16.10.2020)

[2] Описание алгоритмов сортировки и сравнение их производительности [электронный ресурс]. Режим доступа: https://habr.com/ru/post/335920/, свободный (Дата обращения: 16.10.2020)

[3] Официальный сайт Python, документация [электронный ресурс]. Режим доступа: https://docs.python.org/3/library/time.html, свободный (Дата обращения: 16.09.20)