МГТУ им. Баумана

Лабораторная работа №3

По курсу: "Анализ алгоритмов"

Алгоритмы сортировки

Работу выполнил: студент группы ИУ7-53Б
Наместник Анастасия

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2020

Оглавление

Введение

Сортировкой или упорядочением массива называется расположение его элементов по возрастанию (или убыванию). Для решения задачи сортировки было разработано множество алгоритмов, различающихся по трудоемкости и эффективности в связи с затрачиваемыми ими ресурсами ЭВМ. К таким алгоритмам относятся сортировка пузырьком, сортировка вставками, сортировка выбором, сортировка Шелла, сортировка расческой, быстрая сортировка и многие другие. Критерии оценки эффективности этих алгоритмов могут включать следующие параметры.

1. Количество шагов алгоритма, необходимых для упорядочения.
2. Количество сравнений элементов.
3. Количество перестановок, выполняемых при сортировке.

В рамках этой лабораторной работы будут рассмотрены следующие алгоритмы сортировки: сортировка пузырьком, сортировка вставками и быстрая сортировка.

Целью данной лабораторной работы является изучение, реализация и сравнительный анализ алгоритмов сортировки массивов.

В данной лабораторной работе требуется решить пять задач.

1. Изучить и программно реализовать алгоритм сортировки пузырьком.
2. Изучить и программно реализовать алгоритм сортировки вставками.
3. Изучить и программно реализовать алгоритм быстрой сортировки.
4. Оценить трудоемкость вышеперечисленных алгоритмов.

5. Сделать сравнительный анализ по затрачиваемым ресурсам (времени) компьютера на реализацию каждого рассмотренного алгоритма.

1 | Аналитическая часть

1.1 Описание алгоритмов

Первые прототипы современных алгоритмов сортировки появились еще в XIX веке [1], и впоследствии сортировка стала одной из наиболее распространенных операций обработки массивов данных. Благодаря ей, ускоряется процесс решения таких задач, как:

1. Поиск заданного элемента в массиве.
2. Определение, имеются ли пропущенные элементы в массиве.
3. Сравнение двух массивов.

Рассмотрим некоторые популярные алгоритмы сортировки массивов.

1.2 Сортировка пузырьком

Сортировка пузырьком — один из самых известных алгоритмов сортировки. Здесь нужно последовательно сравнивать значения соседних элементов и менять числа местами, если предыдущее оказывается больше последующего. Таким образом элементы с большими значениями оказываются в конце списка, а с меньшими остаются в начале.

Идея сортировки "пузырьком"состоит в том, что все соседние элементы массива попарно сравниваются друг с другом и меняются местами в том случае, если предшествующий элемент больше (меньше) последующего. Затем процесс повторяется. Сортировка заканчивается, когда перестановок в массиве не будет.

Особенность данного алгоритма заключается в следующем: после первого завершения внутреннего цикла максимальный элемент массива размеров в N элементов всегда находится на N-ой позиции. При втором проходе, следующий по значению максимальный элемент находится на N-1 месте. И так далее. Таким образом, на каждом следующем проходе число обрабатываемых элементов уменьшается на 1 и нет необходимости «обходить» весь массив от начала до конца каждый раз.

Этот алгоритм считается учебным и почти не применяется на практике из-за низкой эффективности: он медленно работает на тестах, в которых маленькие элементы (их называют «черепахами») стоят в конце массива. Однако на нём основаны многие другие методы, например, шейкерная сортировка и сортировка расчёской.

1.3 Сортировка вставками

Сортировка вставками - алгоритм сортировки, в 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 помещается 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 выходным данным алгоритма. Основной цикл алгоритма начинается не с 0-го элемента а с 1-го, потому что элемент до 1-го элемента будет считаться отсортированной последовательностью (массив, состоящий из одного элемента, является отсортированным), и уже относительно этого элемента с номером 0 будут вставляться все остальные.

На рисунке 1.1 представлен пример сортировки вставками.

Таким образом, суть сортировки вставками заключается в следующем.

1. Перебираются элементы в неотсортированной части массива.

2. Каждый элемент вставляется в отсортированную часть массива на то место, где он должен находиться.

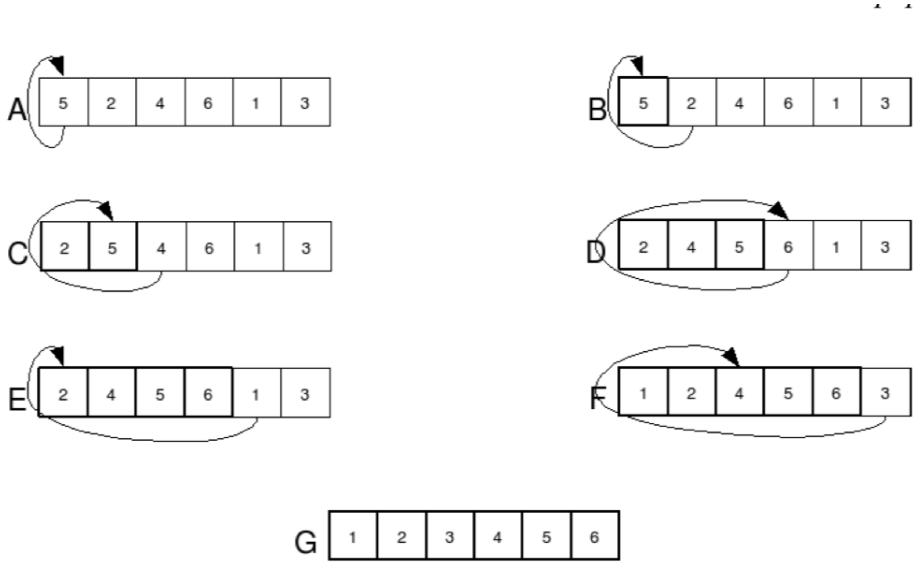

Рис. 1.1: Пример сортировки вставками

1.4 Быстрая сортировка

Быстрая сортировка, часто называемая qsort (по имени в стандартной библиотеке языка Си) — один из самых быстрых известных универсальных алгоритмов сортировки массивов.

Общая идея алгоритма состоит в следующем.

1. Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.

2. Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие».

3. Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

Таким образом, алгоритм сводится к выполнению следующих трех шагов.

1. Выбрать элемент из массива. Назовём его опорным.
2. Разбиение: перераспределение элементов в массиве таким образом, что элементы меньше опорного помещаются перед ним, а больше или равные после.
3. Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один элемент или отсутствуют элементы.

1.5 Вывод

Были рассмотрены основные теоретические сведения о трех распространенных алгоритмах сортировки массивов: сортировка пузырьком, сортировка вставками и быстрая сортировка, - которые в дальнейшем потребуются при разработке программного продукта.

2 | Конструкторская часть

В данном разделе будут представлены схемы алгоритмов сортировки массивов.

2.1 Сортировка пузырьком

На рисунке 2.1 представлена схема алгоритма сортировки пузырьком.

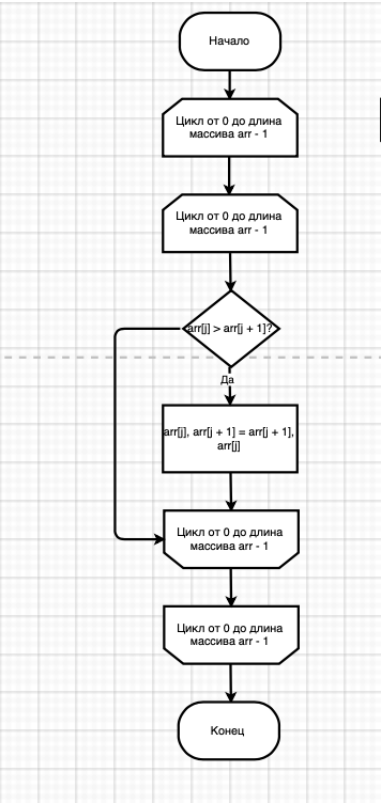

Рис 2.1: Схема алгоритма сортировки пузырьком

На листинге 2.1 представлен псевдокод алгоритма.

Листинг 2.1: Псевдокод алгоритма сортировки пузырьком

```
function bubbleSort(a):
  for i = 0 to n - 2
    for j = 0 to n - 2
      if a[j] > a[j + 1]
        swap(a[j], a[j + 1])
```

2.2 Сортировка вставками

На рисунке 2.2 представлена схема алгоритма сортировки вставками.

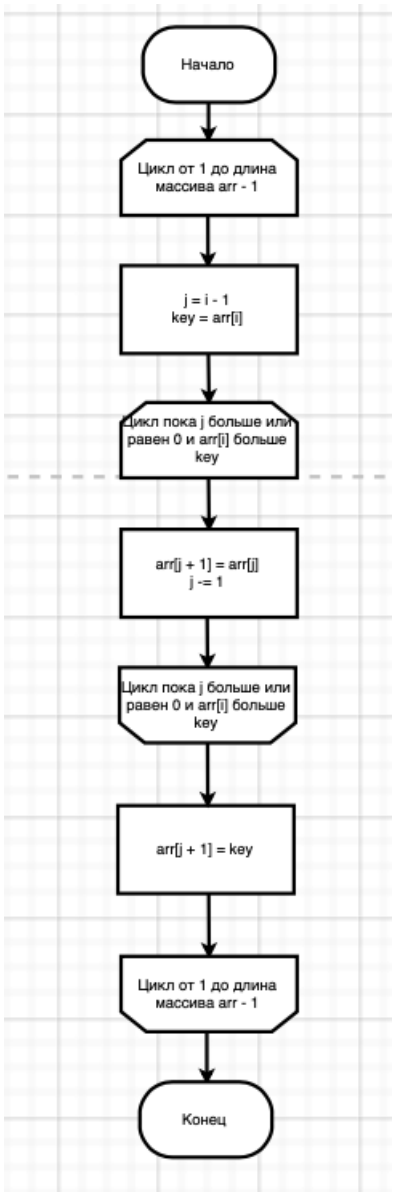

Рис 2.2: Схема алгоритма сортировки вставками

На листинге 2.2 представлен псевдокод алгоритма, где A — сортируемый массив, a low и high — соответственно, нижняя и верхняя границы

сортируемого участка этого массива (предполагается, что неописанная функция partition возвращает индекс опорного элемента).

Листинг 2.2: Псевдокод алгоритма быстрой сортировки

```
1 algorithm quicksort(A, low, high) is
2     if low < high then
3         p := partition(A, low, high)
4         quicksort(A, low, p - 1)
5         quicksort(A, p + 1, high)
```

2.3 Быстрая сортировка

На рисунках 2.3 - 2.4 - 2.5 представлена схема алгоритма быстрой сортировки.

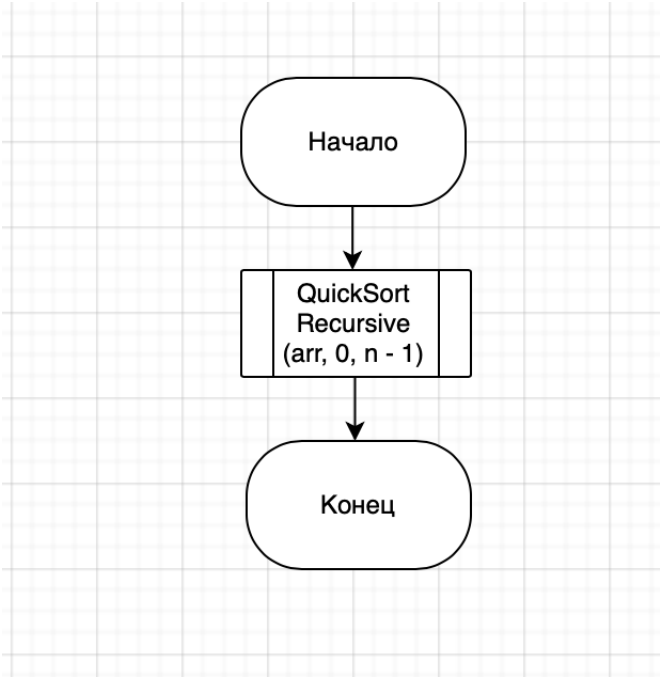

Рис 2.3: Схема алгоритма быстрой сортировки (часть 1)

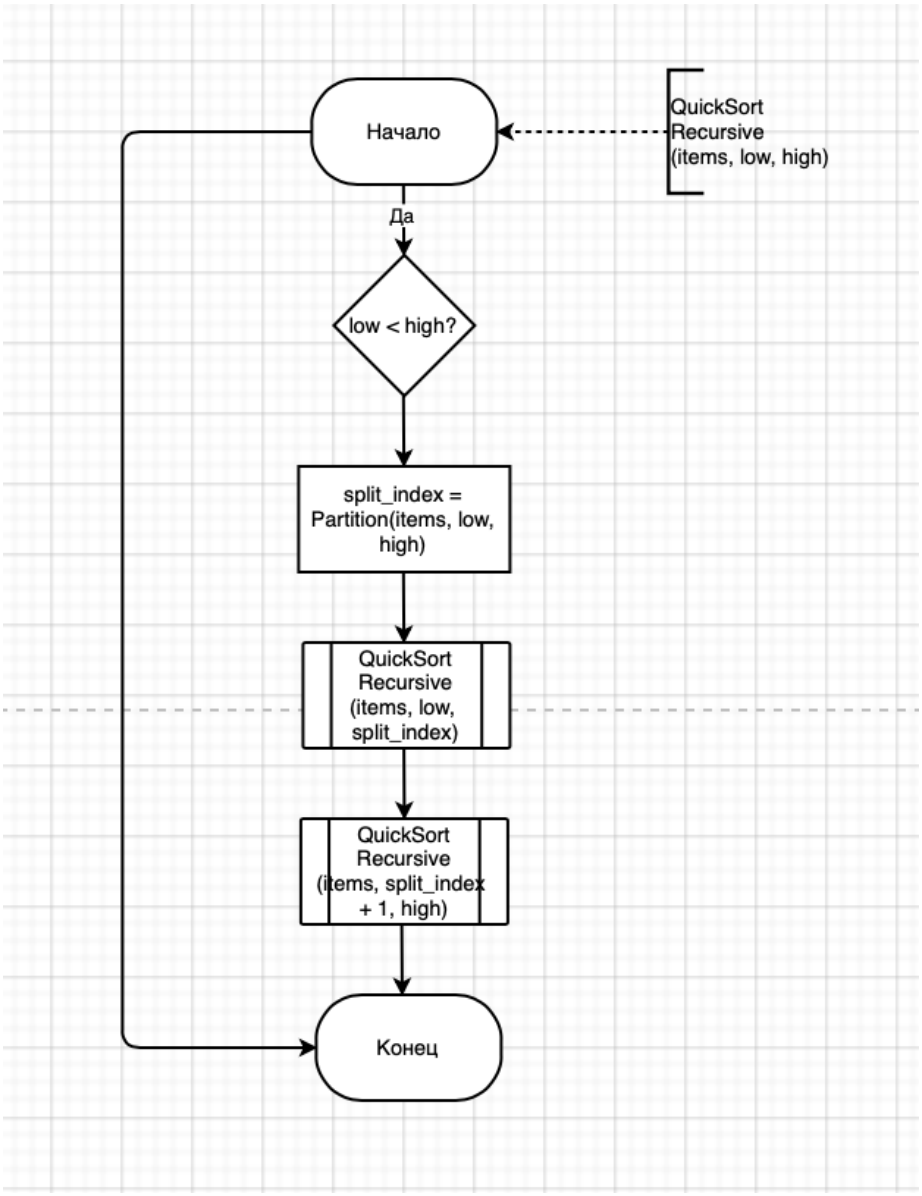

Рис 2.4: Схема алгоритма быстрой сортировки (часть 2)

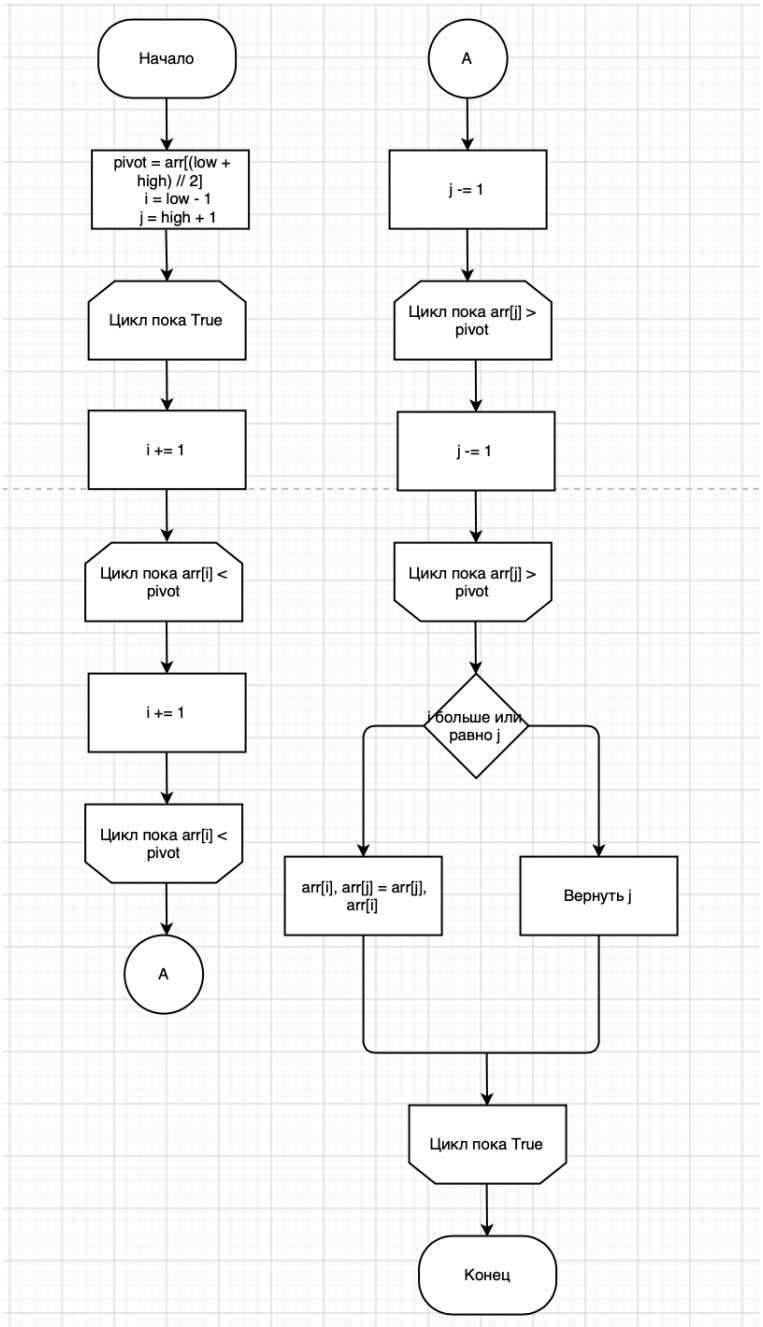

Рис 2.5: Схема алгоритма быстрой сортировки (часть 3)

2.4 Вывод

В данном разделе были рассмотрены схемы трех алгоритмов: стандартного алгоритма умножения матриц, алгоритма Винограда умножения матриц и алгоритма Винограда умножения матриц с оптимизациями. Из полученных данных можно отметить, что объем кода растет от стандартного алгоритма к оптимизированному алгоритму Винограда, что объясняется использованием дополнительных переменных и структур данных в алгоритме Винограда умножения матриц.

3 | Технологическая часть

3.1 Выбор языка программирования

В данной лабораторной работе использовался язык программирования - python (3.8.3) [2] в целях упрощения работы со структурами и визуализацией данных сравнительных анализов, а также из-за наличия опыта работы с данным языком программирования. В качестве интегрированной среды разработки использовался интерпретатор python - IDLE [3]. Для замеров процедурного времени использовалась функция process_time() библиотеки time [4]. Для генерации матрицы использовалась функция randint() модуля random [5].

3.2 Сведения о модулях программы

Программа состоит из следующих модулей:

- sort.py - главный файл программы;
- time_test.py - файл с замерами временнных характеристик.

Листинг 3.1: Подпрограмма алгоритма сортировки "пузырьком"

```
def BubbleSort(arr, n):
    for i in range(n - 1):
        for j in range(n - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
```

Листинг 3.2: Подпрограмма алгоритма сортировки вставками

```
def InsertSort(arr, n):
    for i in range(1, n):
        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: Подпрограмма быстрой сортировки

```
def QuickSort(arr, n):

    QuickSortRecursive(arr, 0, n - 1)

    return arr
```

Листинг 3.4: Подпрограмма сравнения элементов с выбранным опорным

```
def Partition(arr, low, high):
    pivot = arr[(low + high) // 2]
    i = low - 1
    j = high + 1
    while True:
        i += 1
        while arr[i] < pivot:
            i += 1

        j -= 1
        while arr[j] > pivot:
            j -= 1

        if i >= j:
            return j

         arr[i], arr[j] = arr[j], arr[i]
```

Листинг 3.5: Подпрограмма создания массива

```
def CreateArray(n):
    arr = []
    for i in range(n):
        arr.append(int(input()))
    return arr
```

Листинг 3.6: Подпрограмма создания массива с использованием функции randint модуля random [5]

```
def CreateArrayRandom(n):
    return [randint(0, 100) for i in range(n)]
```

3.3 Тесты

В данном разделе будут представлены тесты.

1. При массиве arr = [1, 3, 5, 2, 4] длиной n = 5 Сортировка пузырьком: 1 2 3 4 5 Сортировка вставками: 1 2 3 4 5 Быстрая сортировка: 1 2 3 4 5

2. При массиве arr = [10, 9, 8, 7, 6, 5, 1, 2, 3, 4] длиной n = 10 Сортировка "пузырьком": 1 2 3 4 5 6 7 8 9 10 Сортировка вставками: 2 3 4 5 6 7 8 9 10 Быстрая сортировка: 2 3 4 5 6 7 8 9 10

3. При массиве arr = [1] длиной n = 1 Сортировка "пузырьком": 1 Сортировка вставками: 1 Быстрая сортировка: 1

3.4 Вывод

В технологической части были представлены модули программы, листинги кода и тестов к программе, а также обусловлен выбор языка программирования и приведены использовавшиеся в ходе работы инструменты.

4 | Исследовательская часть

4.1 Результат замеров времени выполнения всех алгоритмов

Сравним временные показатели работы каждого из рассматриваемых четырех алгоритмов при длине массива от 5 до 20 чисел. Для этого установим количество итераций (повторений вызова процедуры) iter и воспользуемся библиотекой time для замера времени выполнения каждого алгоритма по iter раз. Возьмем iter = 100 и следующие массивы.

1. 1, 3, 2, 4, 5
2. 10, 9, 8, 7, 6, 5, 1, 2, 3, 4
3. 10, 1, 3, 2, 4, 5, 6, 8, 7, 9, 12, 11, 15, 13, 14
4. 1, 2, 3, 4, 5, 10, 9, 8, 7, 6, 5, 11, 13, 12, 14, 15, 20, 19, 16, 18, 17

Время выполнения алгоритмов сортировки массивов довольно мало, поэтому, чтобы оценить его, следует взять среднее арифметическое от времени, за которое процедура отработает установленные iter раз.

4.2 Случайная выборка

На рисунке 4.1 показана работа алгоритмов сортировки массивов при случайной выборке.

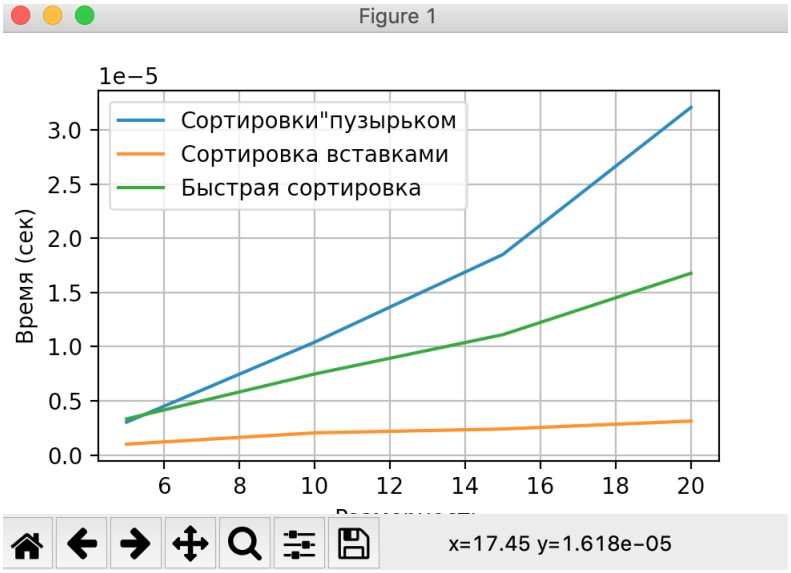

Рис 4.1: Сравнение времени выполнения алгоритмов (случайная выборка)

В качестве примера в таблице 4.1 представлены временные характеристики всех алгоритмов при размере массива от 5 до 20 чисел.

Сравнительная таблица времени выполнения (сек.) всех алгоритмов при случайной выборке

Разм. алгоритма	Пузырек	Вставки	Быстрая сортировка
5	0.00000302	0.00000107	0.0000035
10	0.00001044	0.0000019	0.0000082
15	0.00001979	0.0000025	0.0000118
20	0.00003753	0.0000040	0.0000172

4.3 Лучший случай

Лучшим случаем будем считать ситуацию, когда массив изначально отсортирован. Для сравнительного анализа возьмем следующие массивы.

1. 1, 3, 2, 4, 5
2. 1, 3, 2, 4, 5, 6, 7, 8, 9, 10
3. 1, 3, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15
4. 1, 3, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20

На рисунке 4.3 показана работа алгоритмов сортировки массивов при лучшем случае.

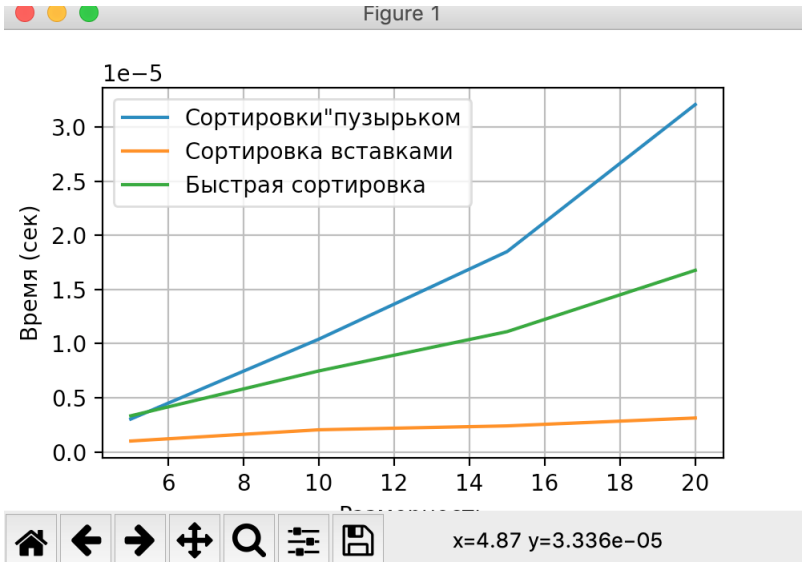

Рис 4.2: Сравнение времени выполнения алгоритмов (лучший случай)

В качестве примера в таблице 4.2 представлены временные характеристики всех алгоритмов при размере массива от 5 до 20 чисел.

Разм. алгоритма	Пузырек	Вставки	Быстрая сортировка
5	0.00000284	0.0000009	0.0000032
10	0.0000089	0.0000016	0.00000703
15	0.0000182	0.0000024	0.00001107
20	0.0000317	0.0000031	0.00001539

4.4 Худший случай

Худшим случаем будем считать ситуацию, когда массив изначально отсортирован в обратном порядке. Для сравнительного анализа возьмем следующие массивы.

1. 5, 4, 3, 2, 1
2. 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
3. 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1
4. 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

На рисунке 4.3 показана работа алгоритмов сортировки массивов при худшем случае.

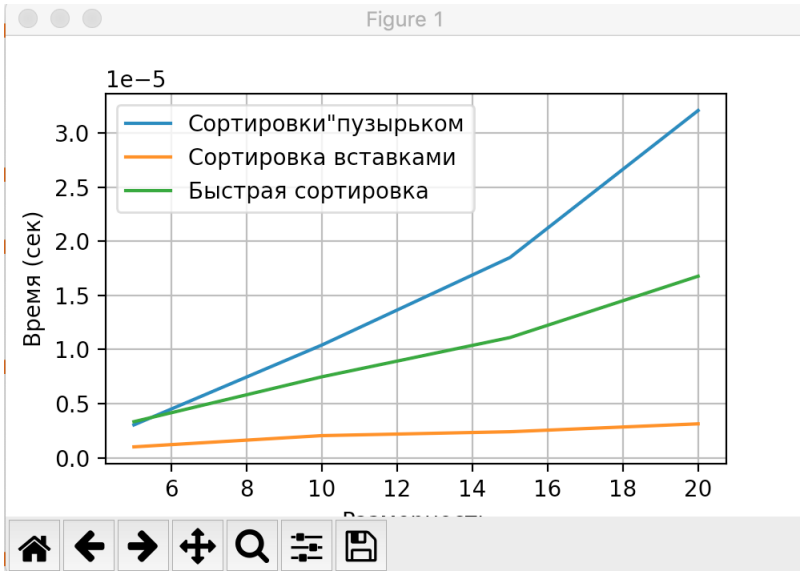

Рис 4.3: Сравнение времени выполнения алгоритмов (худший случай)

В качестве примера в таблице 4.3 представлены временные характеристики всех алгоритмов при размере массива от 5 до 20 чисел.

Разм. алгоритма	Пузырек	Вставки	Быстрая сортировка
5	0.00000284	0.0000099	0.0000032
10	0.0000090	0.0000017	0.00000722
15	0.0000189	0.0000024	0.00001109
20	0.0000326	0.0000031	0.00001541

4.5 Сравнительный анализ алгоритмов

Введем модель вычислений трудоемкости алгоритма. Пусть трудоемкость 1 у следующих базовых операций: +, -, *, /, =, ==, !=, <, <=, >, >=. Трудоемкость цикла: f цикла = f иниц + f сравн + N итер(f тела + fинкрем + fсравн).Трудоемкость условного перехода 1.

Алгоритм сортировки пузырьком обладает трудоемкостью 4.1.

$$N^2 * \begin{bmatrix} 4 \text{ л.с.} \\ 8.5 \text{ х.с} \end{bmatrix} + N * \begin{bmatrix} 3 \text{ л.с.} \\ -1.5 \text{ х.с} \end{bmatrix} - 4 \tag{4.1}$$

Трудоемкость квадратичная от размера массива.

Сортировка вставками в лучшем случае, если уже отсортированный массив: O(N). В худшем случае, если обратно отсортированный массив: $O(N^2)$.

Быстрая сортировка в лучшем случае: O(Nlog(N)).

В худшем случае: $O(N^2)$.

4.6 Вывод

Все алгоритмы в худшем случае обладают квадратичной сложностью. А в лучшем случае меньше всего сложность у алгоритма сортировки вставками.

Заключение

В ходе лабораторной работы были реализованы и проанализированы алгоритм сортировки массивов "пузырьком алгоритм сортировки массивов вставками и алгоритм быстрой сортировки.

В рамках выполнения работы решены следующие задачи.

1. Изучен и программно реализован алгоритм сортировки "пузырьком".
2. Изучен и программно реализован алгоритм сортировки вставками.
3. Изучен и программно реализован алгоритм быстрой сортировки.
4. Оценена трудоемкость вышеперечисленных алгоритмов.
5. Сделан сравнительный анализ по затрачиваемым ресурсам (времени) компьютера на реализацию каждого рассмотренного алгоритма.

Литература

[1] *Э., Кнут Д.* Искусство программирования. Том 3. Сортировка и поиск / Кнут Д. Э. — Вильямс, 2007. — P. 832.

[2] *Майкл, Доусон.* Python Programming for the Absolute Beginner, 3rd Edition / Доусон Майкл. — Прогресс книга, 2019. — P. 416.

[3] *Lutz, M.* The IDLE User Interface / M Lutz. — O'Reilly Media, 2013.

[4] Time. https://docs.python.org/3/library/time.html.

[5] random — Generate pseudo-random numbers. https://docs.python.org/3/library/random.html.