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 **09.03.01 Информатика и вычислительная техника**

О Т Ч Е Т

по лабораторной работе № 3

Название: Алгоритмы сортировки

Дисциплина: Анализ алгоритмов

Студент	ИУ7-54Б		Елгин И.Ю.
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			Волкова Л.Л.
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2021

Содержание

Введение

Алгоритмы сортировки часто применяются в практике программирования. В том числе в областях связанных с математикой, физикой, компьютерной графикой и т.д.

На сегодняшний день существует не одна вариация алгоритмов сортировки. Все они различаются по скорости сортировки и по объему необходимой памяти. Цель данной лабораторной работы - обучение расчету трудоемкости алгоритмов Для достижения поставленой цели необходимо осуществить следующие задачи:

1. Разработать и реализовать алгоритмы трёх различных сортировок;
2. Провести тестирование алгоритмов по времени на данных являющихся худшим, лучшим и обычным случаем для данного алгоритма;
3. По результатам проделанной работы сделать выводы и написать отчёт.

1 Аналитическая часть

Выберем три алгоритма сортировок для реализации.

1.1 Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов.

В данную сортировку может быть добавлен флаг показывающий происходила ли перестановка при очередном проходе. Если перестановки не происходило то массив отсортирован и мы можем завершить алгоритм.

1.2 Сортировка вставками

На каждом шаге выбирается один из элементов неотсортированной части массива (максимальный/минимальный) и помещается на нужную позицию в отсортированную часть массива.

При этом часть массива от позичии вставки до позиции взятия элемента сдвикается на 1 .

1.3 Сортировка шейкером

Алгоритм данной сортировки схож с алгоритмом сортировки пузырьком. Мы проходимся по масиву сравниваем два соседних элемента, если порядок в паре неверный, выполняется обмен элементов. Однако в данном алгоритме мы чередуеми проходы по масиву в одну и в другую сторону.

1.4 Вывод

В качестве алгоритмов сортировок для реализации были выбраны сортировка пузырьком, сортировка вставками и сортировка шейкером.

2 | Конструкторская часть

2.1 Схемы алгоритмов

В данном разделе будут рассмотрены схемы алгоритмов пузырьком с флагом (2.1), сортировки вставками (2.2), сортировки шейкером (2.3).

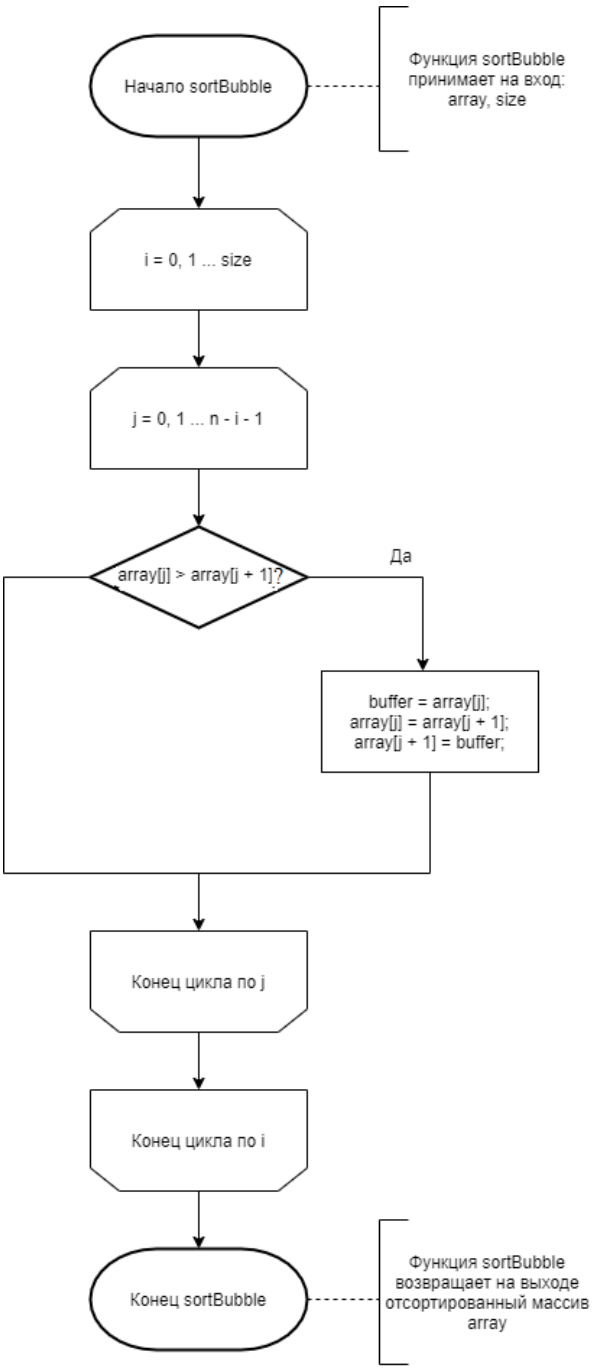

Рис. 2.1: Схема алгоритма сортировки пузырьком

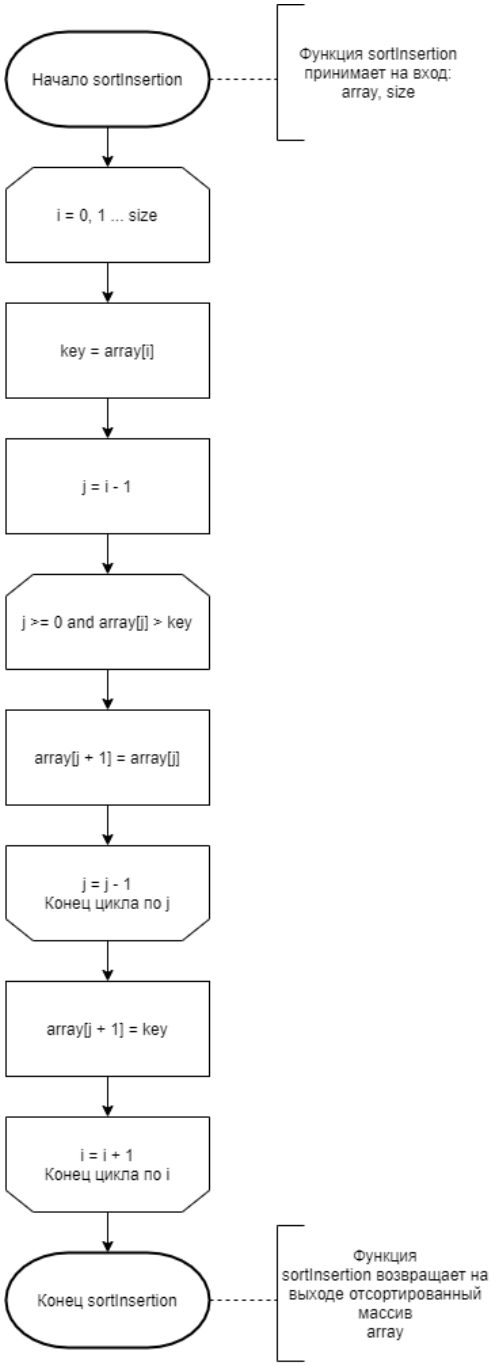

Рис. 2.2: Схема алгоритма сортировки вставками

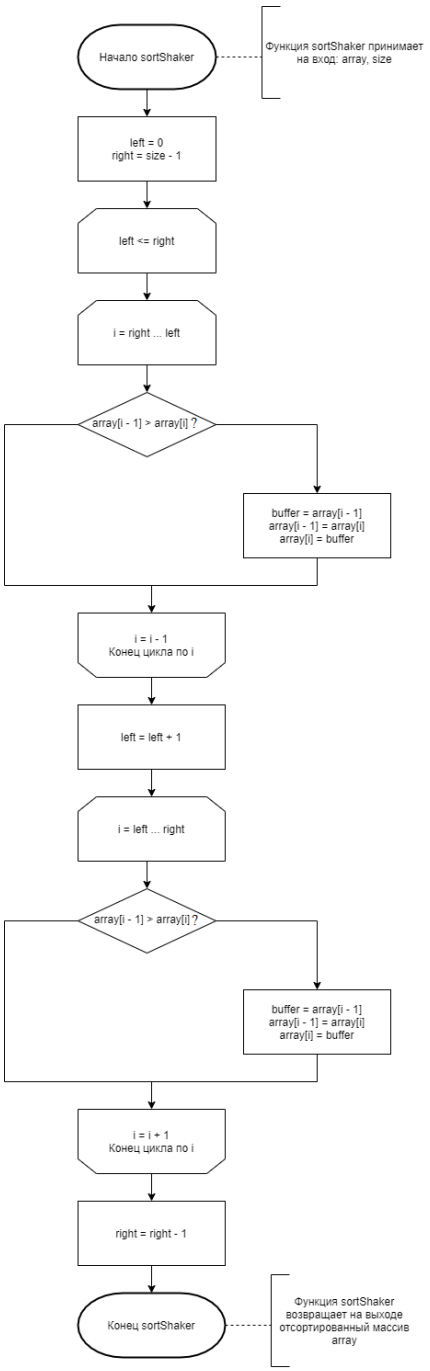

Рис. 2.3: Схема алгоритма сортировки шейкером

2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

1. базовые операции стоимостью 1: $+, -, =, ==, <=, >=, !=, +=, -=$, $[], ++, --, and, !, or$ получение полей класса
2. базовые операции стоимостью 2: $*, /, /=, *=$
3. оценка трудоемкости цикла:

$$F = a + N * (a + F)$$

, где a - условие цикла

4. стоимость условного перехода возьмем за 0, стоимость вычисления условия остаётся.

Далее будет приведены оценки трудоемкости алгоритмов.

2.2.1 Сортировка вставками

Лучший случай: отсортированный массив. При этом все внутренние циклы состоят всего из одной итерации.
Трудоемкость:

$$T(n) = 4 + 1 + 2n * (2 + 2 + 3) = 2n * 7 = 14n + 1 = O(n) \qquad (2.1)$$

Худший случай: массив отсортирован в обратном нужному порядке. Каждый новый элемент сравнивается со всеми в отсортированной последовательности. Все внутренние циклы будут состоять из j итераций.
Трудоемкость:

$$T(n) = 1 + n * (2 + 2 + 4n * (4 + 1) + 3) = 2n * n + 7n + 1 = O(n^2) \quad (2.2)$$

2.2.2 Сортировка пузырьком

Лучший случай: Массив отсортирован; не произошло ни одного обмена за 1 проход -> выходим из цикла
Трудоемкость:

$$1 + 1 + 1 + 1 + 2n * 5 + 2 + 1 = 10n + 7 = O(n) \tag{2.3}$$

Худший случай: Массив отсортирован в обратном порядке; в каждом случае происходил обмен
Трудоемкость:

$$1 + 1 + 2n * (1 + 1 + (n - 1)/2 * (5 + 1 + 8)) = 2 + 14n^2 - 10n = O(n^2) \tag{2.4}$$

2.2.3 Сортировка шейкером

Лучший случай: Массив отсортирован; не произошло ни одного обмена за 1 проход -> выходим из цикла
Трудоемкость:

$$1 + 2n * (1 + 8 + 5) = 1 + 28n = O(n) \tag{2.5}$$

Худший случай: Массив отсортирован в обратном порядке; в каждом случае происходил обмен
Трудоемкость:

$$1 + 1 + 2n * (1 + 1 + 2n * (4 + 5)) = 2 + 4n + 36n * n = O(n^2) \tag{2.6}$$

2.3 Вывод

Сортировка пузырьком: лучший - $O(n)$, худший - $O(n^2)$
Сортировка вставками: лучший - $O(n)$, худший - $O(n^2)$
Сортировка шейкером: лучший - $O(n)$, худший - $O(n^2)$

3 Технологическая часть

3.1 Выбор ЯП

В качестве языком программирования мною выбран язык Python, так как я имею опыт работы с данным языком программирования, Pyton позволяет удобно работать с масивами.

Время работы алгоритмов было замерено с помощью функции process_time() из библиотеки time [4].

3.2 Описание структуры ПО

Программа состоит из одного файла main.py в котором находятся функции алгоритмов сортировок.

3.3 Сведения о модулях программы

Листинг 3.1: Сортировка пузырьком

```
def BubbleSort(array, n):
    fl = 1
    for i in range(0, n - 1):
        if fl :
            fl = 0
            for j in range(1, n - i):
                if (array[j-1] > array[j]):
                    fl = 1
                    array[j], array[j - 1] = array[j - 1],
                        array[j]
        else:
            break;
```

Листинг 3.2: Сортировка вставками

```
def InsertionSort(array, n):
    for i in range(0, n - 1):
        b = array[i]
        j = i - 1
        while j >=0 and b < array[j] :
            array[j+1] = array[j]
            j -= 1
        array[j+1] = b
```

Листинг 3.3: Сортировка шейкером

```
def ShakerSort(array, n):
    f = 1
    s = 0
    e = n - 1
    while f:
        swapped = False
        for i in range(s, e):
            if (array[i] > array[i + 1]) :
                array[i], array[i + 1] = array[i + 1],
                    array[i]
                f = 1
        if not(f):
            break
        f = 0
        e -= 1
        for i in range(e, s, -1):
            if (array[i] > array[i + 1]):
                array[i], array[i + 1] = array[i + 1],
                    array[i]
                f = 1
        s = s + 1
```

3.4 Вывод

В качестве языка программирования для реализации алгоритмов сортировки был выбран Python. На данном языке програмирования были написаны функции сортировок и функция тестирования сортировок по времени.

4 Исследовательская часть

Был проведен замер времени работы каждой из сортировок на худших лучших и случайных данных.

4.1 Постановка эксперемента

Лучшим случаем для всех трёх сортировок будет массив отсортированный в нужном порядке, так как он совпадает с результатом и никаких действий над ним производить не нужно.

Худшим случаем для сортировки пузырьком является обратно отсортированный масив так как для полной отсортировки каждый эллемент должен проделать путь от в i элементов где i позиция элементов. А за один проход на место будет перемещатся только один элемент.

Худшим случаем для сортировки вставками является обратно отсортированный масив [3].

Худшим случаем для сортировки шейкером является обратно отсортированный масив. По анологии с сортировкой пузырьком, только здесь будут чередоваться перемещения наибольшего и наименьшего элемента.

4.1.1 Сортировка пузырьком

В таблице 4.1 представлен результат работы сортировки на массивох разных длин, для худщих лучших и случайных данных.

len	Отсортированный, сек.	Обратный порядок, сек.	Случайный,сек
200	0.0	0.0015225	0.0
400	0.0	0.003125	0.0015625
600	0.0	0.0046875	0.003125
800	0.0	0.0078125	0.0046875
1000	0.0015625	0.0125	0.0109375

Таблица 4.1: Время работы сортировки пузырьком

4.1.2 Сортировка вставками

В таблице 4.2 представлен результат работы сортировки на массивох разных длин, для худщих лучших и случайных данных.

len	Отсортированный, сек.	Обратный порядок, сек.	Случайный,сек
200	0.0	0.0015625	0.0
400	0.0	0.0015625	0.0015625
600	0.0	0.00625	0.0031250
800	0.0	0.0140625	0.0046750
1000	0.0	0.0352521	0.0.0076875

Таблица 4.2: Время работы сортировки вставками

4.1.3 Сортировка шейкером

В таблице 4.3 представлен результат работы сортировки на массивох разных длин, для худщих лучших и случайных данных.

len	Отсортированный, сек.	Обратный порядок, сек.	Случайный,сек
200	0.0	0.0015625	0.0015625
400	0.0	0.0046875	0.0015625
600	0.0	0.0125000	0.009375
800	0.0015625	0.0.0203125	0.0140625
1000	0.0015625	0.034375	0.0203225

Таблица 4.3: Время работы сортировки шейкером

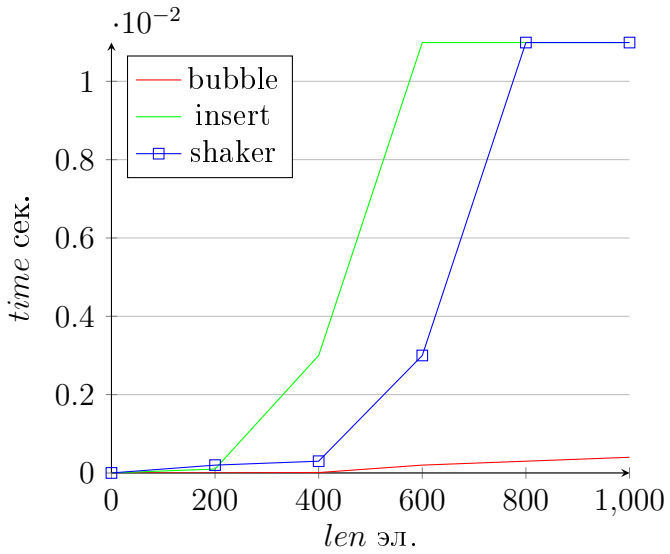

Рис. 4.1: Сравнение сортировки уже отсортированных массивов

4.1.4 Графики времени сортировок

На данных графиках представлены зависимости времени работы от размера массива для сортировок при лучших, хучших и случайных данных.

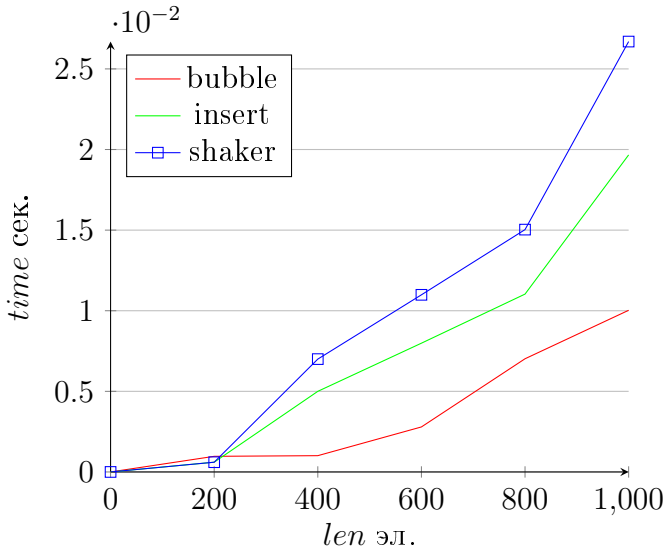

Рис. 4.2: Сравнение сортировки массивов, отсортированных в обратном порядке

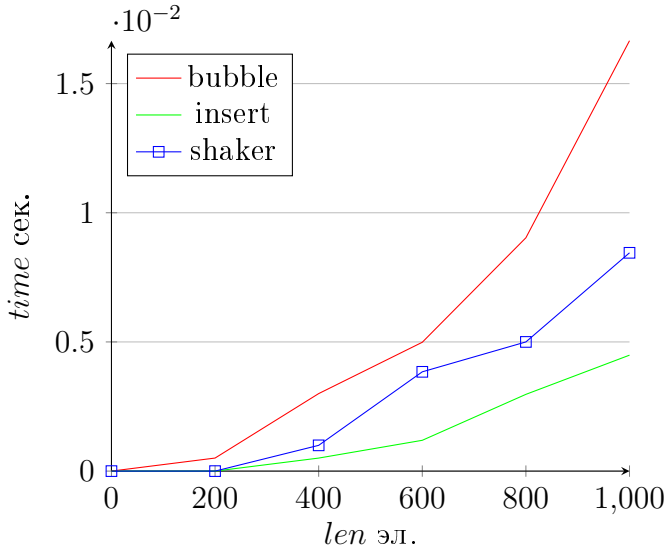

Рис. 4.3: Сравнение сортировки случайных массивов

4.1.5 Вывод

Были протестированы алгоритмы сортировки на массивах размерами 200...1000 с шагом 200. Рассмотрены отсортированные, отсортированные в обратном порядке массивы и массивы со случайными значениями.

В результате тестирования было получено, что лучшее время сортировки показывают на отсортированных массивах. Худшие значения сортировки показывают на обратно отсортированных массивах, причём чем больше размер такого массива, тем медленнее работают сортировки.

При сравнении времени работы алгоритмов можно сделать вывод, что на размерах массива до 400 элементов время работы алгоритмов приближено к 0 сек. При работе с массивами размерами более 400 элементов сортировка пузырьком показывает наихудший результат. При сортировки случайных массивов лучшее время показывает сортировка вставками.

Заключение

В ходе выполнения данной лабораторной работы были реализованы три алгоритма сортировки: сортировка пузырьком, сортировка вставками и быстрая сортировка. Был проведён анализ каждого алгоритма и измерено время работы алгоритмов для массивов разных размеров. Была оценена трудоёмкость алгоритмов.

Лучшее время работы все сортировки показывают на отсортированных массивах. Худшее время алгоритмы сортировки показали при работе с массивами, отсортированными в обратном порядке. Причём в худшем случае зависимость времени работы от размера массива квадратичная.

Был сделан вывод, что на размерах массива до 400 элементов время работы алгоритмов приближено к 0 сек. При работе с массивами размерами более 400 элементов сортировка пузырьком показывает наихудший результат. При сортировки случайных массивов лучшее время показывает сортировка вставками.

Литература

[1] Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Глава 7. Быстрая сортировка // Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 2-е изд. — М.: Вильямс, 2005.

[2] Левитин А. В. Глава 4. Метод декомпозиции: Быстрая сортировка // Алгоритмы. Введение в разработку и анализ — М.: Вильямс, 2006.

[3] Алгоритм сортировки вставками [Электронный ресурс]. – Режим доступа:$https://www.quora.com/What-is-the-worst-case-example-of-selection-sort-and-insertion-sort$ (дата обращения 10.10.21)

[4] Сортировка всавками [Электронный ресурс]. – Режим доступа:$https://studopedia.ru/9_97870_sortirovka-vstavkami.html$ (дата обращения 10.10.21)