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»

Лабораторная работа №3 по курсу "Анализ алгоритмов"

Тема Алгоритмы сортировки

Студент Мишина Е.В.

Группа ИУ7-54Б

Оценка (баллы) _____

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

Москва — 2020 г.

Оглавление

Введение

На данный момент существует огромное количество вариаций сортировок. Эти алгоритмы необходимо уметь сравнивать, чтобы выбирать наилучшие подходящие в конкретном случае.

Эти алгоритмы оцениваются по:

- времени быстродействия.

Целью данной лабораторной работы является изучение применений алгоритмов сортировки, обучение расчету трудоемкости алгоритмов.

1 | Аналитическая часть

1.1 Описание алгоритмов

Сортировка массива — одна из самых популярных операций над массивом. Алгоритмы реализуют упорядочивание элементов в списке. В случае, когда элемент списка имеет несколько полей, поле, служащее критерием порядка, называется ключом сортировки

Область применения:

- физика,
- математика,
- экономика,
- итд.

1.1.1 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, очередной наибольший элемент ставится на своё место в конец неотсортированного массива. Таким образом наибольшие элементы "всплывают"как пузырек.

1.1.2 Сортировка вставками

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

1.1.3 Сортировка выбором

Алгоритм сортировки выбором заключается в поиске на необработанном срезе массива или списка минимального значения и в дальнейшем обмене этого значения с первым элементом необработанного среза. На следующем шаге необработанный срез уменьшается на один элемент[1].

Вывод

В данном разделе было дано описание сортировок пузырьком, вставками и выбором.

2 | Конструкторская часть

Ввод:

- на вход подается массив, состоящий из чисел;

Вывод:

- программа выводит отсортированные каждым из алгоритмов массивы;
- в режиме замера ресурсов программа выводит средние время, затраченное каждым алгоритмом.

2.1 Схемы алгоритмов

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

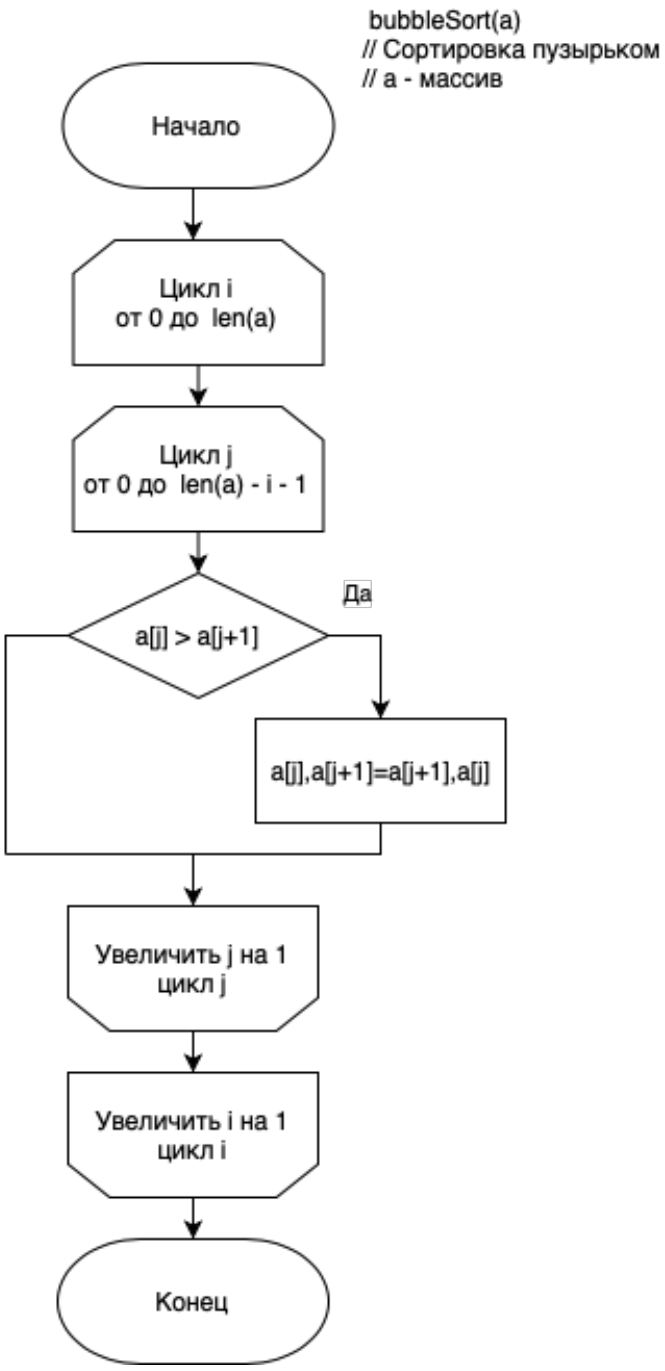

Рис. 2.1: Схема алгоритма сортировки пузырьком

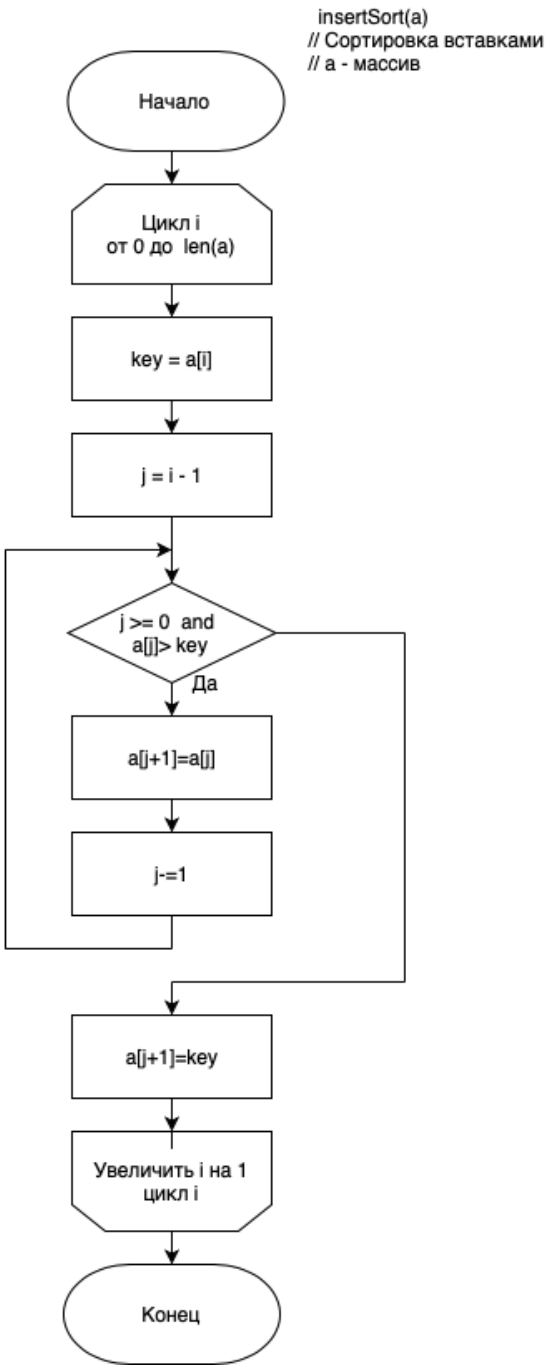

Рис. 2.2: Схема алгоритма сортировки вставками

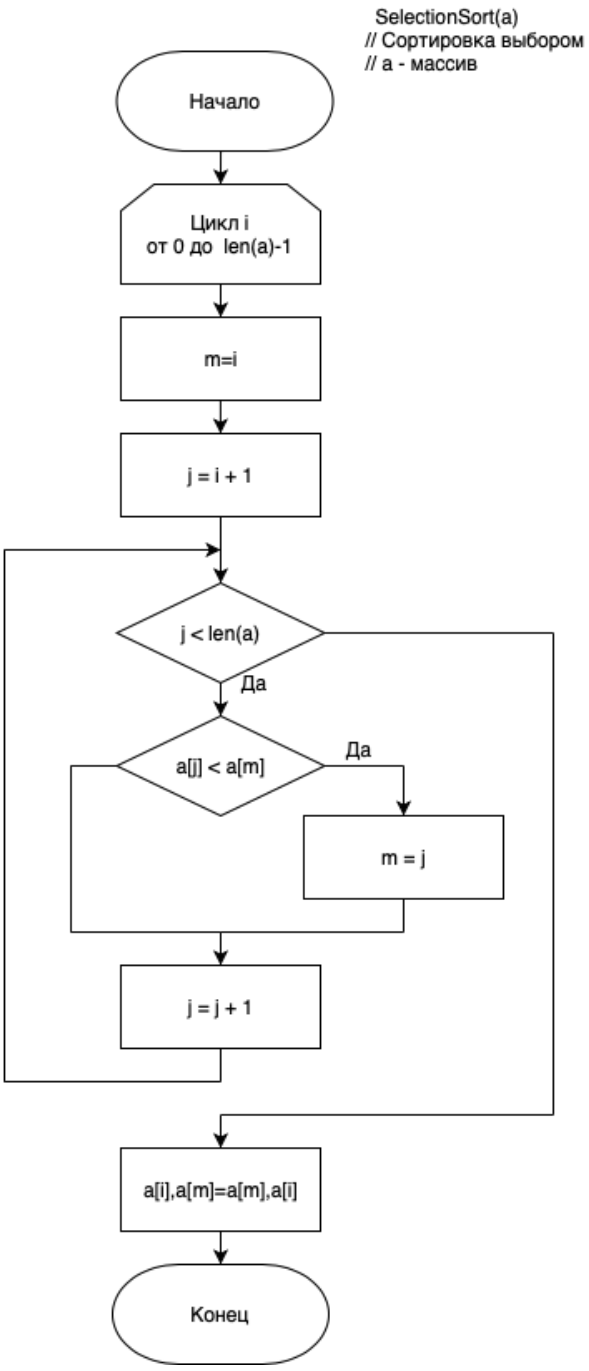

Рис. 2.3: Схема алгоритма сортировки выбором

2.2 Трудоемкость алгоритмов

Введем модель трудоемкости для оценки алгоритмов:

1. базовые операции стоимостью 1 — +, -, *, /, =, ==, <=, >=, !=, +=, [], ++, – получение полей класса

2. оценка трудоемкости цикла: Fц = a + N*(a + F(тела)), где a - условие цикла

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

Далее будет приведены оценки трудоемкости алгоритмов. Построчная оценка трудоемкости сортировки пузырьком с флагом (Табл. 2.1), вставками (Табл. 2.2), выбором (Табл. 2.3).

2.2.1 Сортировка вставками

Табл. 2.1 Построчная оценка веса

Код	Вес
for i in range(1, len(a)):	2
key = a[i]	2
j = i-1	2
while (j >= 0 and a[j] > key):	4
a[j+1] = a[j]	4
j-= 1	2
a[j+1] = key	3

Лучший случай: отсортированный массив. При этом все внутренние циклы состоят всего из одной итерации.
Трудоемкость: $T(n) = 2+(n-1)*(2+2+3) = n*7-7+2 = 7*n-5 = O(n)$

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: $T(n) = 2 + (n-1) * (2 + 2 + n * (4 + 2) + 3) = 2 + (n-1) * (7 + 6 * n) = 2 + 7 * n + 6 * n^2 - 7 - 6 * n = 6 * n^2 + n - 5 = O(n^2)$

2.2.2 Сортировка пузырьком

Табл. 2.2 Построчная оценка веса

Код	Вес
for i in range(len(a)):	2
for j in range(len(a) - i - 1):	4
if a[j] > a[j + 1]:	4
a[j], a[j + 1] = a[j + 1], a[j]	7

Лучший случай: Массив отсортирован; не произошло ни одного обмена за 1 проход -> выходим из цикла
Трудоемкость: $2 + n * (4 + n * 4) = 2 + 4 * n + 4 * n^2 = O(n^2)$

Худший случай: Массив отсортирован в обратном порядке; в каждом случае происходил обмен
Трудоемкость: $2 + n * (4 + n * (4 + 7)) = 2 + 4 * n + 11 * n^2 = O(n^2)$

2.2.3 Сортировка выбором

Табл. 2.3 Построчная оценка веса

Код	Вес
for i in range(len(a) - 1):	3
m = i	1
j = i + 1	2
while j < len(a):	2
if a[j] < a[m]: :	3
m = j :	1
j = j + 1 :	2
a[i], a[m] = a[m], a[i] :	5

Лучший случай: Массив отсортирован; не произошло ни одного обмена за 1 проход -> выходим из цикла
Трудоемкость: $3 + n * (1 + 2 + 2 + n * 2 + 5) = 3 + 10 * n + 2 * n^2 = O(n^2)$

Худший случай: Массив отсортирован в обратном порядке; в каждом случае происходил обмен
Трудоемкость: $3 + n * (1 + 2 + 2 + n * (2 + 1) + 5) = 3 + 10 * n + 3 * n^2 = O(n^2)$

2.3 Вывод

Сортировка пузырьком: лучший - $2+4*n+4*n^2$, худший - $2+4*n+11*n^2$. Сравнивая коэффициенты при n^2, можно увидеть разницу 2.3 раза.

Сортировка вставками: лучший - $7 * n - 5$, худший - $6 * n^2 + n - 5$

Сортировка выбором: лучший - $3+10*n+2*n^2$, худший - $3+10*n+3*n^2$ Сравнивая коэффициенты при n^2, можно увидеть разницу 1.5 раза.

В данном разделе была приведена оценка трудоемкости алгоритмов сортировки пузырьком, вставками и выбором.

3 | Технологическая часть

3.1 Выбор ЯП

Для реализации программы был выбран Python[2] из-за наличия опыта разработки на данном языке программирования.

Для тестирования были выбрана операционная система - Windows 10. Процессор - 2,3 GHz Quad-Core Intel Core i5. Память - 8 GB 2133 MHz LPDDR3. Графическое ядро - Intel Iris Plus Graphics 655 1536 MB.

3.2 Реализация алгоритма

Листинг 3.1: Функция сортировки выбором

```
1
2     def SelectionSort(a):
3         for i in range(len(a) - 1):
4
5             m = i
6             j = i + 1
7
8             while j < len(a):
9                 if a[j] < a[m]:
10                    m = j
11                j = j + 1
12            a[i], a[m] = a[m], a[i]
13
14        return a
```

Листинг 3.2: Функция сортировки вставками

```
1
2     def insertSort(a):
3         for i in range(1, len(a)):
4             key = a[i]
5             j = i - 1
6             while j >= 0 and a[j] > key:
7                 a[j + 1] = a[j]
8                 j -= 1
9             a[j + 1] = key
10        return a
```

Листинг 3.3: Функция сортировки пузырьком

```
1
2     def bubbleSort(a):
3         for i in range(len(a)):
4             for j in range(len(a) - i - 1):
5                 if a[j] > a[j + 1]:
6                     a[j], a[j + 1] = a[j + 1], a[j]
7         return a
```

3.3 Вывод

В данном разделе были рассмотрены реализация алгоритмов сортировки пузырьком, вставками и выбором.

4 | Исследовательская часть

4.1 Сравнение на основе замеров времени работы алгоритмов

4.2 Постановка эксперимента

Будем измерять время работы алгоритмов на отсортированных, отсортированных в обратном порядке и случайно сгенерированных массивах. Для замера времени будем использовать функцию time.

4.2.1 Сортировка пузырьком

длина	Отсортированный, сек.	Обратный порядок, сек.	Случайный,сек
100	0.0000070596	0.0000114870	0.0000079632
200	0.0000169492	0.0000394011	0.0000273299
300	0.0000490808	0.0001106691	0.0000873494
400	0.0001789284	0.0003815031	0.0003695011
500	0.0004285693	0.0009476304	0.0008392596

4.2.2 Сортировка вставками

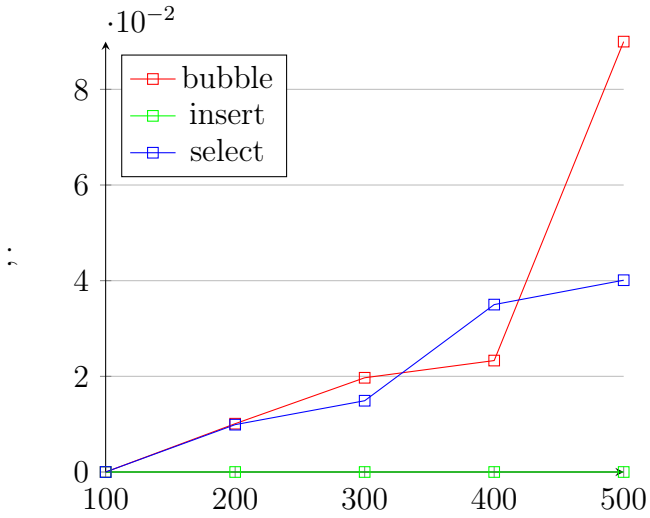

Рис. 4.1: Сравнение сортировки уже отсортированных массивов

длина	Отсортированный, сек.	Обратный порядок, сек.	Случайный,сек
100	0.0000001526	0.0000080109	0.0000035906
200	0.0000002813	0.0000264192	0.0000136399
300	0.0000011587	0.0000710797	0.0000406003
400	0.0000029778	0.0002423406	0.0001570511
500	0.0000045323	0.0004874802	0.0003791213

4.2.3 Сортировка выбором

длина	Отсортированный, сек.	Обратный порядок, сек.	Случайный,сек
100	0.0000061798	0.0000065613	0.0000066209
200	0.0000263286	0.0000256014	0.0000270915
300	0.0003964090	0.0003686976	0.0004604006
400	0.0008022571	0.0008208489	0.0008285213
500	0.0013114190	0.0013642597	0.0014996982

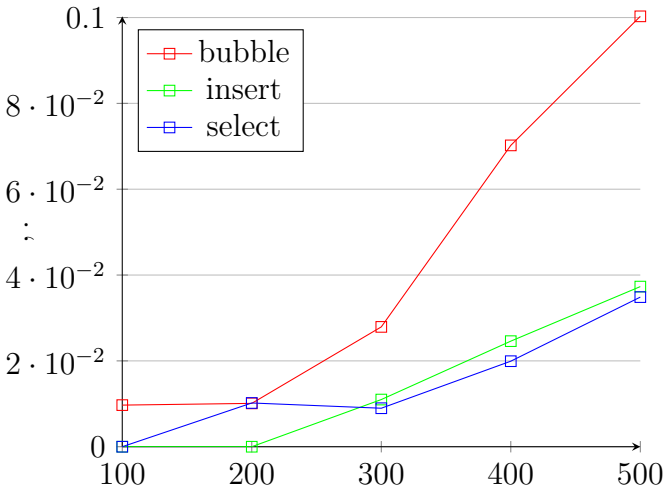

Рис. 4.2: Сравнение сортировки массивов, отсортированных в обратном порядке

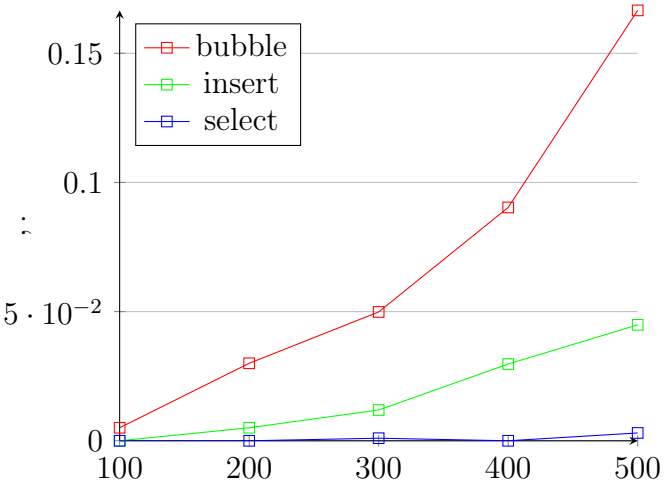

Рис. 4.3: Сравнение сортировки случайных массивов

4.3 Тестовые данные

Проведем тестирование программы. В столбцах "Ожидаемый результат"и "Полученный результат"4 числа соответсвуют результату сортировки пузырьком, выбором и вставкой.

Таблица 4.1: Таблица тестовых данных

№	массив	Ожидаемый результат	Полученный результат
1	[1,2,4,5]	[1,2,4,5], [1,2,4,5], [1,2,4,5]	[1,2,4,5], [1,2,4,5], [1,2,4,5]
2	[5, 4, 3, 2, 1]	[1,2,4,5], [1,2,4,5], [1,2,4,5]	[1,2,4,5], [1,2,4,5], [1,2,4,5]
3	[]	[],[],[]	[],[],[]
4	[1,5,4,3,2]	[1,2,4,5], [1,2,4,5], [1,2,4,5]	[1,2,4,5], [1,2,4,5], [1,2,4,5]
5	[0, 0, 1, 4, 2]	[0,0,1,2,4], [0,0,1,2,4], [0,0,1,2,4]	[0,0,1,2,4], [0,0,1,2,4], [0,0,1,2,4]

4.4 Вывод

Были протестированы алгоритмы сортировки на массивах размерами 100. . . 500 с шагом 100. Рассмотрены отсортированные, отсортированные в обратном порядке массивы и массивы со случайными значениями.

Эксперименты подтвердили подсчитанную ранее трудоемкость алгоритмов.

Заключение

Было дано описание алгоритмов сортировки пузырьком, вставками и выбором, были рассмотрены их схемы и реализации алгоритмов на языке программирования Python.

Была проведена оценка трудоемкости алгоритмов: Сортировка пузырьком: лучший - $2+4*n+4*n^2$, худший - $2+4*n+11*n^2$. Сравнивая коэффициенты при n^2, можно увидеть разницу 2.3 раза.

Сортировка вставками: лучший - $7*n-5$, худший - $6*n^2+n-5$

Сортировка выбором: лучший - $3+10*n+2*n^2$, худший - $3+10*n+3*n^2$
Сравнивая коэффициенты при n^2, можно увидеть разницу 1.5 раза.

В результате тестирования сортировок 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.

Эксперименты подтвердили подсчитанную ранее трудоемкость алгоритмов.

При сравнении времени работы алгоритмов можно сделать вывод, что при работе с массивами больших размеров сортировка пузырьком показывает наихудший результат. При сортировки случайных массивов лучшее время показывает сортировка выбором.

На основе полученных графиков тестов сортировка выбором выигрывает по времени при росте длины массива, следовательно сортировка выбором более применима в реальных проектах.

Таким образом, цель достугнута, все задачи выполнены.

Список использованных источников

1. Сортировка выбором [Электронный ресурс]. – Режим доступа: https://younglinux.info/alg
min. – Дата доступа: 01.11.2020.

2. Python [Электронный ресурс]. – Режим доступа: https://www.python.org.
– Дата доступа: 01.11.2020.

3. PyCharm [Электронный ресурс]. – Режим доступа: https://www.jetbrains.com/ru-
ru/pycharm. – Дата доступа: 01.11.2020.