Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Факультет Инфокоммуникационных Технологий

**Лабораторная работа №3**

Выполнили

Надери М.Ш.

Боженко М.А.

Адрат О.А.

Проверил

Мусаев А.А.

Санкт-Петербург

2022 г.

**Введение**

Целью выполнения лабораторной работы является ознакомление с основами изучения сложности алгоритмов. В рамках выполнения лабораторной работы поставлены задачи:

* Написание программы для пузырьковой сортировки. И сравнение сложности алгоритма с функцией sort().
* Создание алгоритмов для соответствующих сложностей.
* Отображение зависимости между количеством элементов и количеством шагов для алгоритмов с приведенными сложностями.

**Задание 1**

Ввод с клавиатуры строки различных чисел, вводимых через пробел. После перехода на новую строку начинается основная программа. Создаём два массива с введенными значениями. Проходимся в цикле с 0 до числа на 1 меньше длины массива (переменная «о»), внутри которого идем по элементам массива до элемента на «о» меньшего последнего элемента (чтобы с каждой итерацией проходить на один элемент меньше).

Далее сравниванием текущий элемент массива со следующим, если текущий элемент больше, меняем их местами.

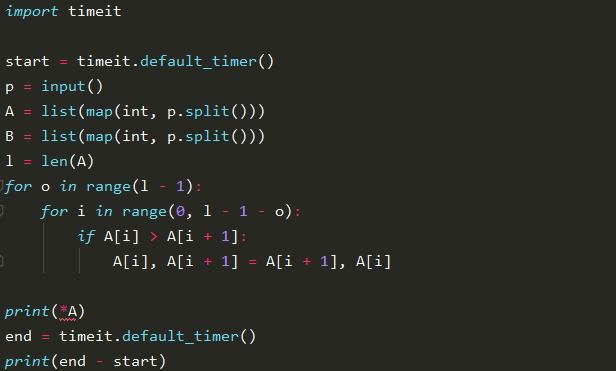


Рисунок 1 ‒ Пузырьковая сортировка

Выводим получившийся массив и время, за которое работает алгоритм.

Далее аналогично подсчитываем время для функции sort(), выводим получившийся массив и время, за которое работает функция.

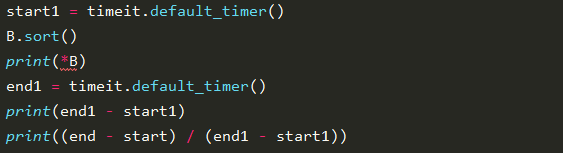


Рисунок 2 ‒ Функция sort() и сравнение времени

Итоговый вид вывода программы:

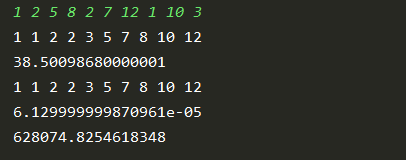


Рисунок 3 ‒ Вывод

**Задание 2**

1. O(3n)

Для алгоритма данной сложности выбран вариант из 3 строк с разными операциями.

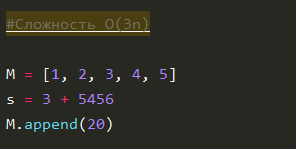


Рисунок 4 ‒ Пример алгоритма

1. O(nlogn)

Для этой сложности был выбран алгоритм сортировки слиянием.

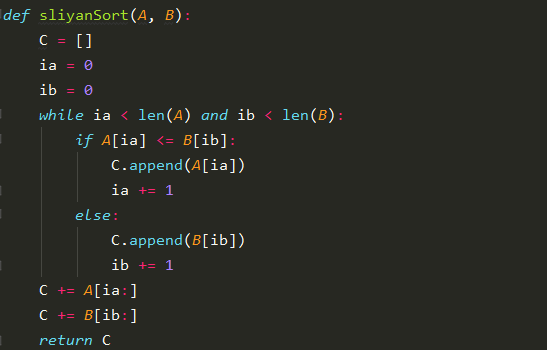


Рисунок 5 ‒ Пример алгоритма

Сам алгоритм дает сложность logn. Поэтому далее в функции main() мы вызываем его n раз.

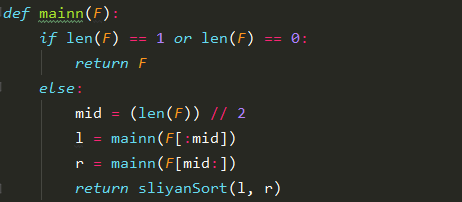


Рисунок 6 ‒ Пример алгоритма

1. O(n!)

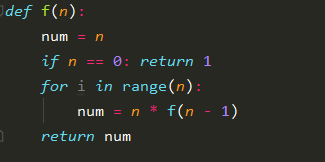


Рисунок 7 ‒ Пример алгоритма

Для алгоритма сложности n! выбран алгоритм подсчёта факториала числа.

1. O(n3)

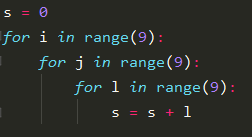


Рисунок 8 ‒ Пример алгоритма

Для этой сложности построен алгоритм с тремя циклами, вызванных внутри друг друга.

1. O(3log(n))

Данный алгоритм имеет сложность log(n), и вызывая его трижды, мы увеличиваем сложность в 3 раза.

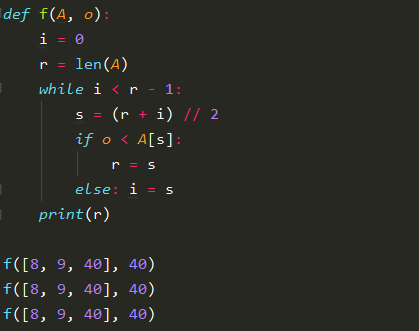


Рисунок 9 ‒ Пример алгоритма

**Задание 3**

Первый график отражает зависимость для функций сложностей O(1) и O(logn):

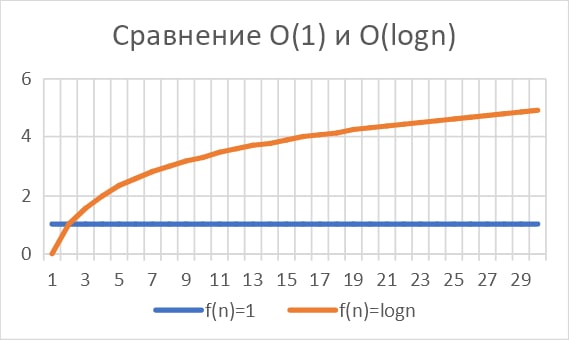


Рисунок 10 ‒ Сравнение O(1) и O(logn)

На втором графике отражены все 4 зависимости:

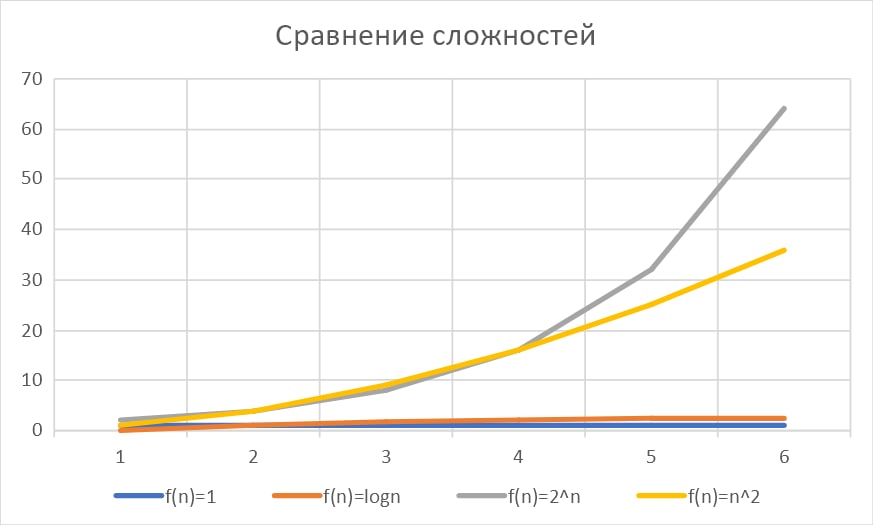


Рисунок 11 ‒ Сравнение сложностей

**Заключение**

Была реализована программа сортировки пузырьком, и проведено сравнение со встроенной функцией sort(). Встроенная функция оказалась быстрее составленного алгоритма. Были реализованы алгоритмы по данным в задании сложностям. А также сравнивались программы некоторых сложностей и построена зависимость между количеством переменных и количеством шагов.

Ссылка на GitHub:

<https://github.com/MariamNaderi/A-BD/tree/main/lab3>

<https://clck.ru/32iCb7>

# Список использованных источников

1. TechRocks: офиц. сайт.URL: <https://techrocks.ru/2020/07/15/big-o-explanation-for-newbies/> (Дата обращения 02.11.2022).
2. Notes Algoprog: офиц. сайт.URL: <https://notes.algoprog.ru/complexity/index.html> (Дата обращения: 02.11.2022)
3. Habr: офиц. сайт.URL: <https://habr.com/ru/post/104219/> (Дата обращения: 02.11.2022)
4. Tproger: офиц. сайт.URL: <https://tproger.ru/translations/data-structure-time-complexity-in-python/> (Дата обращения: 02.11.2022)