Санкт-Петербургский Национальный Исследовательский Университет Информационных Технологий, Механики и Оптики

Факультет ИКТ

**Лабораторная работа №2**

Выполнили

Новиков Н. В.

Ходакова М. А.

Борисевич А. В.

Проверил: Мусаев А. А.

Санкт-Петербург,

2023

**Содержание**

[ВВЕДЕНИЕ 3](#_Toc147776784)

[1. 4](#_Toc147776785)

[2. BUBLE SORT (сортировка пузырьком). 4](#_Toc147776786)

[3. Алгоритмы различной сложности 5](#_Toc147776787)

[**3.1 O(3n)** 5](#_Toc147776788)

[**3.2 O(nlogn)** 5](#_Toc147776789)

[ЗАКЛЮЧЕНИЕ 6](#_Toc147776790)

[СПИСОК ИСТОЧНИКОВ 7](#_Toc147776791)

# ВВЕДЕНИЕ

Цель лабораторной работы №2 состояла в знакомстве с временной сложностью. Для её достижения были поставлены следующие задачи:

1. Построить зависимость между количеством элементом и количеством шагов для алгоритмов различной сложности;
2. Создать программу, которая сортирует пузырьком;
3. Создать алгоритмы разной сложности

# 1. Построение зависимости

Обозначим за N количество шагов программы. Вычислительная сложность алгоритма — константная, те , количество шагов не зависит от количества элементов.

Вычислительная сложность алгоритма — логарифмическая, количество шагов растет логарифмически с увеличением размера входного массива, те

Вычислительная сложность алгоритма — квадратичная, где количество шагов программы квадратично зависит от размера входного массива (). Вычислительная сложность алгоритма — экспоненциальная, где .

Расположим алгоритмы в порядке возрастания сложности. В общем случае имеем:

.

Однако стоит учитывать, что сложность означает лишь то, что количество шагов программы не зависит от количества элементов, и алгоритм сложности может потребовать большего количества шагов, чем, например, .

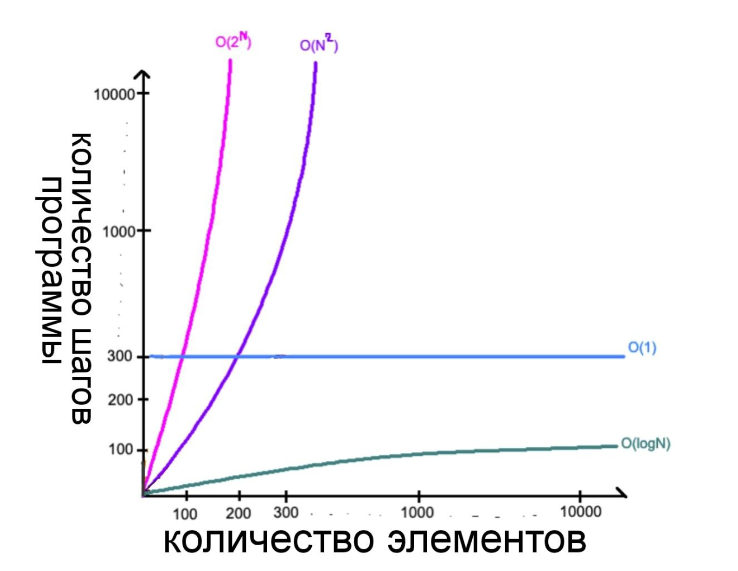


Рисунок 1 - Зависимость

# 2. BUBLE SORT (сортировка пузырьком).

Алгоритм состоит в повторяющихся проходах по сортируемому массиву (Рисунок 2). На каждой итерации последовательно сравниваются соседние элементы, и, если порядок в паре неверный, то элементы меняют местами. Создадим список элементов, которые стоят в невозрастающем порядке. Внешний цикл отвечает за количество раз, пройденных по списку (длина списка-1, единица отнимается, так как когда цикл будет идти l-1 раз, гарантируется, что самый левый элемент будет являться наименьшим). Внутренний цикл выполняется количество раз, равное (длина списка – текущий round – 1). Поскольку справа остаются уже отсортированные элементы, из длины списка вычитается не только 1, но и round (в данном случае показывает, сколько элементов справа отсортировано). Если число с большим индексом меньше текущего, то элементы меняются местами. Когда внешний цикл закончит работу, список будет отсортирован и выведен. Сложность данного алгоритма сортировки равна O(n^2).В данной сортировке выполняются всего два различных вида операции: сравнение элементов и их обмен. Поэтому время всего алгоритма T=T1+T2, где T1 — время,

затрачиваемое на сравнение элементов, а T2 — время, за которое мы производим все необходимые обмены элементов. Так как в алгоритме меняться местами могут только соседние элементы, то каждый обмен уменьшает количество инверсий на единицу. Следовательно, количество обменов равно количеству инверсий в исходном массиве. Максимальное количество инверсий содержится в массиве, элементы которого отсортированы по убыванию. Количество инверсий в таком массиве n(n−1)^2. Получаем, что T2=O(n^2). Точное количество сравнений зависит от исходного массива. Известно, что худший случай равен n(n−1)^2, а лучший — n−1. Следовательно, T1=O(n^2). В итоге получаем

T=T1+T2=O(n^2)+O(n^2)=O(n^2).

Метод sort(). Сложность равна O(NlogN). Сложность пузырьковой сортировки больше, чем сложность метода sort().

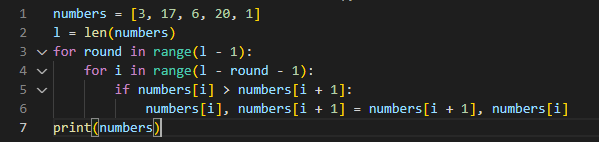


Рисунок 2 - Сортировка пузырьком

# 3. Алгоритмы различной сложности

## **3.1 O(3n)**

Для создания алгоритма сложностью O(3n) был создан цикл, который сначала перебирает все элементы массива, а потом в него был вложен внутренний цикл, который перебирает первые 3 элемента. Таким образом и получается сложность O(3n)

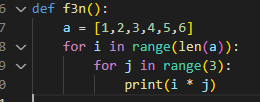


Рисунок 3 - O(3n)

## **3.2 O(nlogn)**

Для реализации сложности O(nlogn) была создана функция, которая принимает в себя массив и после выполняет над ним определённые операции (Рисунок 4). Так как каждый раз при вызове функции происходит сокращение массива пополам, то для достижения базового случае необходимо O(logn) шагов. В разбиении списка пополам занимает O(n), следовательно суммарная сложность алгоритма O(nlogn).

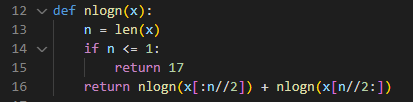


Рисунок 4 - O(nlogn)

**3.3 O(n!)**

Для реализации O(n!) была создана функция, которая принимает на вход пустой массив, элементы для перестановки и индекс начала сортировки. Функция вычисляет все возможные перестановки элементов массива (Рисунок 5.

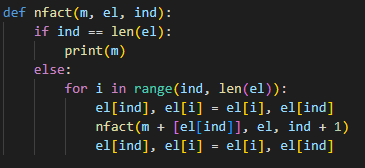


Рисунок 5 - O(n!)

**3.4 O(n^3)**

Функция O(n^3) представляет собой поиск корней уравнения перебирая все возможные комбинации. Каждый из циклов имеет сложность O(n), а их всего 3, следовательно сложность всей программы O(n^3) (Рисунок 5).

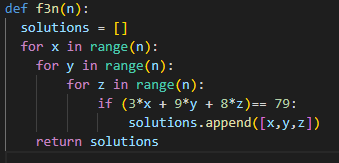


Рисунок 6 - O(n^3)

**3.5 O(3log(n))**

Данный алгоритм был реализован с помощью алгоритма бинарного поиска, который в свою очередь имеет сложность O(logn), а внутрь функции был помещен цикл, который делает обход первый 3 переменный. Следовательно общее время выполнения O(3logn) (Рисунок 7)

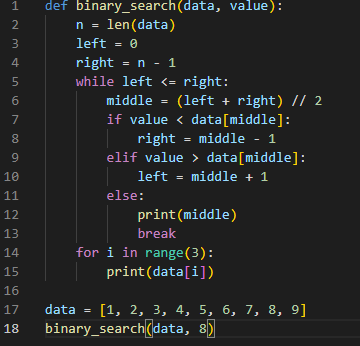


Рисунок 7 - O(3logn)

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы №2 были получены навыки работы с алгоритмами различной сложности.

# СПИСОК ИСТОЧНИКОВ

1. Википедия. Временная сложность алгоритма [Электронный ресурс] <https://ru.wikipedia.org/wiki/%D0%92%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%BB%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D1%8C_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B0> (Дата обращения 07.09.2023)