# Лабораторная работа 3.1. Работа с массивами данных. Вычислительная сложность алгоритмов

**Цель работы**: познакомиться с основными понятиями вычислительной сложности алгоритмов, научиться оценивать сложность алгоритмов и операций над структурами данных. Освоить применение асимптотических нотаций и исследовать производительность различных алгоритмов на практике.

**Задачи работы**:

* + - Изучить основные понятия вычислительной сложности алгоритмов и асимптотических нотаций (O-нотация).
    - Понять принципы оценки вычислительной сложности алгоритмов.
    - Ознакомиться с вычислительными характеристиками операций для различных структур данных (списков, кортежей, множеств, словарей).
    - Изучить законы сложения и умножения при работе с O-нотацией.
    - Провести сравнение производительности алгоритмов на практике и научиться выбирать более эффективные алгоритмы для обработки массивов данных.
    - Решить практические задачи, связанные с анализом и оптимизацией вычислительной сложности.
* **Примечание**
* При выполнении заданий используйте заготовки решений: -> [Репозиторий](https://gitverse.ru/yuripetrov/python-course-tasks).
* Изучить теоретические основы раздела[Вычислительная сложность алгоритмов](https://www.yuripetrov.ru/edu/python/ch_06.html)

<https://www.yuripetrov.ru/edu/python/ch_06_01.html>

**Теоретические основы**

**3.2.1** Основные понятия

- **Вычислительная сложность —** это мера затрат ресурсов (времени и памяти) для выполнения алгоритма.

- **Временная сложность** описывает, как изменяется время выполнения алгоритма в зависимости от размера входных данных.

- **Пространственная сложность** оценивает объем памяти, необходимый алгоритму.

Пример: алгоритм сортировки списка размером \(n\) может иметь временную сложность \(O(n^2)\) или \(O(n \log n)\) в зависимости от метода сортировки.

**3.2.2** Асимптотические нотации

Асимптотические нотации используются для описания поведения алгоритма при увеличении размера входных данных \(n\).

**3.2.2.1** Верхняя оценка и O-нотация

- O-нотация (Big-O) показывает верхнюю границу роста функции. Например:

- (O(1)) — постоянное время выполнения (независимо от \(n\)).

- (O(n)) — линейное время выполнения (пропорционально \(n\)).

- (O(n^2)) — квадратичное время выполнения.

Пример:

# O(n): перебор всех элементов массива

def linear\_search(arr, x):

for i in arr:

if i == x:

return True

return False

**3.2.3** Оценка сложности алгоритмов

**3.2.3.1** Операции над структурами данных

**3.2.3.1.1** Список и кортеж

Список:

- Доступ по индексу: (O(1)).

- Вставка или удаление в конец: (O(1)).

- Вставка или удаление в начало или середину: (O(n)).

Кортеж:

- Доступ по индексу: (O(1)).

- Кортеж неизменяем, поэтому операции изменения недоступны.

**3.2.3.1.2** Множество

Множество обеспечивает быстрый доступ и изменение:

- Проверка наличия элемента: (O(1)).

- Добавление и удаление: (O(1)\).

**3.2.3.1.3** Словарь

**Словарь** — структура данных "ключ-значение":

- Доступ к значению по ключу: (O(1)).

- Добавление/удаление элемента: (O(1)).

**3.2.3.2** Закон сложения и умножения для O-нотации

1. Закон сложения: если алгоритм состоит из двух независимых частей с разной сложностью, итоговая сложность равна наибольшей из них.

Пример: (O(n) + O(n^2) = O(n^2))

2. Закон умножения: если части алгоритма выполняются последовательно, итоговая сложность — произведение сложностей частей.

Пример: (O(n) \* O(log n) = O(n\*log(n))

**3.2.4** Сравнение производительности работы алгоритмов

Сравнение проводится для алгоритмов с разной сложностью на одном и том же наборе данных. Например:

- Линейный поиск ((O(n)\)) и бинарный поиск ((O(log n))).

- Различные методы сортировки: пузырьковая сортировка ((O(n^2))) и сортировка слиянием ((O(n log n))).

**Ход работы**

**Оценка сложности алгоритмов**

1. Задача на линейный поиск.

Реализуйте алгоритм линейного поиска в массиве. Определите его временную сложность.

def linear\_search(arr, x):

for i in arr:

if i == x:

return True

return False

2. Задача на бинарный поиск.

Реализуйте алгоритм бинарного поиска. Сравните его производительность с линейным поиском.

def binary\_search(arr, x):

left, right = 0, len(arr) - 1

while left <= right:

mid = (left + right) // 2

if arr[mid] == x:

return True

elif arr[mid] < x:

left = mid + 1

else:

right = mid - 1

return False

**Сравнение сложности алгоритмов**

1. Сравнение сортировок.

Напишите функцию для сортировки списка пузырьковым методом (\(O(n^2)\)) и слиянием ((O(n log n))). Замерьте время выполнения каждого метода для списка длиной 100, 1000 и 10000.

Пример:

import time

def bubble\_sort(arr):

n = len(arr)

for i in range(n):

for j in range(0, n-i-1):

if arr[j] > arr[j+1]:

arr[j], arr[j+1] = arr[j+1], arr[j]

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2

left = arr[:mid]

right = arr[mid:]

merge\_sort(left)

merge\_sort(right)

i = j = k = 0

while i < len(left) and j < len(right):

if left[i] < right[j]:

arr[k] = left[i]

i += 1

else:

arr[k] = right[j]

j += 1

k += 1

while i < len(left):

arr[k] = left[i]

i += 1

k += 1

while j < len(right):

arr[k] = right[j]

j += 1

k += 1

**Контрольные вопросы**

* + - Что такое вычислительная сложность алгоритма?
    - Объясните разницу между временной и пространственной сложностью.
    - Как работает O-нотация? Какие существуют виды сложности (например, (O(1)), (O(n)), \(O(n^2)))?
    - Почему для эффективного алгоритма важно учитывать сложность операций над структурами данных?
    - В чём преимущество использования множества и словаря перед списком?

**Варианты заданий**

## [Тема Оценка сложности алгоритмов](https://www.yuripetrov.ru/edu/python/ch_06_02.html#id6)

Для задач 3.2.1-3.2.7 ответьте на следующие вопросы:

1. Что выполняет приведенная функция?
2. Какова вычислительная сложность алгоритма (наихудшая оценка, O-нотацию)?

Ответы необходимо указать в строке документации с пометкой «Алгоритм» и «Сложность» соответственно. Сложность - один из вариантов:

O(1), O(N), O(log(N)), O(N log(N)) или O(N^2).

считая, что N - размер входных данных.

**№№ 3.2.1 - 3.2.7**

См. заготовки. [Репозиторий](https://gitverse.ru/yuripetrov/python-course-tasks).

## [Тема Сравнение сложности алгоритмов](https://www.yuripetrov.ru/edu/python/ch_06_02.html#id7)[¶](https://www.yuripetrov.ru/edu/python/ch_06_02.html#id5)

Для каждой задачи ниже:

1. Реализуйте требуемые способы решения.
2. Укажите наихудшую вычислительную (асимптотическую) сложность каждого из способов в строке документации.
3. Сравните способы решения, сопоставив их асимптотические и временные сложности (устно).

**№ 3.2.8**

Дан список случайных чисел. Определить значения минимального и максимального элементов списка.

**№ 3.2.9**

По вылету из аэропорта осуществляется паспортный контроль: сотрудник пограничной службы выполняет проверку в специальном приложении.

Для каждого паспорта в приложении хранится (количество записей большое):

* номер: 7-значное число;
* отметка о допуске: может / не может пересекать границу.

Предполагается, что проверка осуществляется путем ввода номера паспорта, после чего приложение должно выдать отметку о допуске (отсутствие паспорта в приложении аналогично отсутствие возможности вылета).

Реализуйте функцию проверки допуска к вылету.

**№ 3.2.10**

Дана информация о клиентах банка (идентификационный номер и сумма активов). Реализуйте поиск топ-3 клиентов по размеру активов.