Сложность алгоритмов

1. Что такое сложность алгоритмов?

Сложность алгоритма — это характеристика, показывающая, сколько ресурсов (времени и памяти) потребуется для выполнения алгоритма в зависимости от размера входных данных.

- **Временная сложность:** измеряет, сколько времени понадобится для выполнения алгоритма.
- **Пространственная сложность:** измеряет, сколько памяти потребуется для выполнения алгоритма.

2. Оценка сложности: O-большое (Big-O Notation)

Для описания сложности используется нотация "O-большое" (“Big-O”), которая показывает, как изменяется время выполнения или использование памяти в зависимости от роста объема данных nn.

Примеры сложности:

Сложность	**O-большое**	**Пример алгоритма**
Константная	O(1)	Доступ к элементу массива по индексу
Линейная	O(n)	Линейный поиск
Логарифмическая	O(log n)	Бинарный поиск
Квадратичная	O(n^2)	Сортировка пузырьком
Кубическая	O(n^3)	Трудоемкие вложенные циклы
Экспоненциальная	O(2^n)	Решение задачи перебором (например, рекурсивный перебор подмножеств)

3. Основные типы временной сложности

1. **O(1): Константная сложность**
    - Выполнение алгоритма не зависит от размера входных данных.
    - Пример: доступ к элементу массива по индексу.
2. **O(n): Линейная сложность**
    - Время выполнения растет линейно с увеличением объема данных.
    - Пример: поиск элемента в неотсортированном массиве (линейный поиск).
3. **O(log n): Логарифмическая сложность**
    - Время выполнения увеличивается медленно при росте объема данных.
    - Пример: бинарный поиск в отсортированном массиве.
4. **O(n^2): Квадратичная сложность**
    - Используется в алгоритмах с вложенными циклами.
    - Пример: сортировка пузырьком или поиск совпадений в двух массивах.

4. Почему важно учитывать сложность?

- **Эффективность:** Алгоритмы с меньшей сложностью быстрее обрабатывают большие объемы данных.
- **Реальная практика:** В задачах с большими массивами или сложными операциями выбор подходящего алгоритма позволяет экономить время и ресурсы.

5. Пример сравнения алгоритмов

- Задача: найти число в массиве.
    - **Линейный поиск (O(n)):** просматривает каждый элемент массива последовательно, время выполнения растет линейно.
    - **Бинарный поиск (O(log n)):** делит массив пополам на каждом шаге, что значительно быстрее для больших массивов, но требует предварительной сортировки.

6. Практическое задание

1. Приведите примеры алгоритмов с разной сложностью из вашей повседневной жизни.
2. Решите задачу: оцените временную сложность следующего кода:

```
array = [3, 1, 4, 1, 5, 9, 2]
sum = 0
for i in array:
    for j in array:
        sum += i * j
```

Подсказка: Обратите внимание на вложенные циклы.

7. Рекомендуемая литература

1. Томас Кормен и др. “Алгоритмы. Построение и анализ”.
2. Роберт Седжвик “Фундаментальные алгоритмы на C++” (или других языках программирования).
3. Онлайн-ресурсы: GeeksforGeeks, Khan Academy.