**Получение элемента, s[i]**

Алгоритмическая сложность получения элемента s[i] для строки или списка s в большинстве языков программирования составляет O(1). Это означает, что доступ к элементу происходит за постоянное время независимо от размера строки или списка.

В Python, например, строка или список реализованы таким образом, что каждый элемент имеет фиксированный адрес в памяти, и для доступа к конкретному индексу достаточно просто обратиться по этому адресу, что занимает константное время.

**Размер списка, len(s)**

Алгоритмическая сложность операции len(s) для списка или строки s в Python также составляет O(1). Это связано с тем, что длина списка или строки хранится как отдельное значение внутри структуры данных, поэтому её получение не требует обхода всех элементов. Таким образом, операция выполняется за постоянное время независимо от длины списка или строки.

**Получение среза, s[a:b]**

Алгоритмическая сложность получения среза s[a:b] для строки или списка s в Python составляет O(n). То есть она линейная относительно длины среза.

- При создании среза Python создаёт новый объект, который содержит элементы исходного списка или строки начиная с индекса a до индекса b. Для этого необходимо скопировать все элементы между этими индексами, а количество таких операций пропорционально длине среза (n).

**Размер множества, len(s)**

Алгоритмическая сложность операции len(s) для множества s в Python также составляет O(1). Как и в случае со списками и строками, размер множества хранится отдельно и доступен сразу при вызове функции len(), без необходимости перебора всех элементов.

**Добавление элемента, s.add(x)**

Алгоритмическая сложность добавления элемента в множество s с помощью метода add() в Python составляет O(1) в среднем случае. В худшем случае она может достигать O(n), где **n** — количество элементов в множестве.

Множества в Python реализованы на основе хеш-таблиц. В идеальном случае добавление нового элемента сводится к вычислению хеша и вставке значения в соответствующую ячейку таблицы. Это занимает константное время.

Однако, если возникает коллизия (когда два разных элемента попадают в одну и ту же ячейку), то может потребоваться проверка уникальности через сравнение элементов. В редких случаях, когда много элементов попадают в одни и те же ячейки, может возникнуть необходимость перебирать несколько элементов, что увеличивает время выполнения до O(n).

**Проверка наличия значения, x in/not in s**

Алгоритмическая сложность проверки наличия значения x in s или x not in s для множества s в Python составляет O(1) в среднем случае. Как и в случае с добавлением элемента, эта операция использует хеширование, чтобы быстро найти нужный элемент или убедиться в его отсутствии.

Операция in или not in проверяет наличие элемента в множестве путем вычисления хеша элемента и поиска соответствующего места в хеш-таблице. Если элемент найден, возвращается результат True (для in) или False (для not in), иначе — наоборот.

Как и в других операциях с хеш-таблицей, в худших случаях, связанных с коллизиями, время выполнения может возрастать до O(n).

**Перебор множества, for v in s:**

Алгоритмическая сложность перебора всех элементов множества s с использованием цикла for v in s: в Python составляет O(n), где **n** — количество элементов в множестве.

При использовании оператора for Python последовательно проходит по всем элементам множества. Поскольку множество неупорядочено, порядок итерации может отличаться от порядка вставки элементов, однако это не влияет на общую сложность. Время выполнения зависит исключительно от числа элементов в множестве, так как каждая итерация требует константного времени.

**Объединение (union), s \| t**

Алгоритмическая сложность объединения двух множеств s | t в Python составляет O(len(s) + len(t)). Это значит, что время выполнения операции объединения прямо пропорционально сумме размеров обоих множеств.

Операция объединения двух множеств включает следующие шаги:

1. Создается новое множество.

2. Все уникальные элементы из первого множества копируются в новое множество. Это занимает время O(len(s)).

3. Затем все уникальные элементы второго множества добавляются в новое множество. Если какой-то элемент уже присутствует в новом множестве, он пропускается (это одна из особенностей работы множеств). Этот шаг занимает время O(len(t)).

Суммарно, общее время выполнения этой операции составляет O(len(s) + len(t)), что является линейной сложностью относительно общего числа элементов в обоих множествах.