1. **Получение элемента, s[i]: сложность O (1)**

Алгоритм выполняется за константное время, т.е. доступ к элементу по индексу в списке происходит за постоянное время. В данном случае нам неважно сколько всего элементов, т.к. элементы списка хранятся последовательно. И тут у нас получается, что доступ к каждому элементу по индексу происходит на прямую, перебирать что-то и выполнять другие операции не надо.

Например, у нас есть строка *s = meow*, получается *s [3] = w* и операция совершится за O (1)

Размер списка, len(s): сложность O (1)

Эта функция не производит итерацию по структуре данных. Операция не зависит от размера структуры данных, поэтому опять же время у нас константа.

Например, у нас есть строка *s = meow*, получается *len(s) = 4* и операция совершится за O (1)

Получение среза, s[a:b]: сложность O (N) / O (b-a)

У нас создаётся новый объект, в который копируются элементы из исходного объекта, и он не изменяется. Эта операция занимает время, пропорциональное количеству копируемых элементов. Т.е. если длина среза равна N, то сложность как раз будет O (N). В худшем случае будет равно размеру исходного списка

Например у нас есть строка *s = meooow*, тогда вызов *s [1:4]* вернет *'eoo'*, что требует копирования 3 символов, и, следовательно, будет O (3) = O(N).

2. **Размер множества, len(s): сложность O (1)**

Информация о размере множества (количество элементов) хранится в структуре данных множества. Т.е. итерации по структуре данных нет. Доступ к этой информации осуществляется за постоянное время.

Например, для множества *s = {7,8,9,10}* вызов *len(s) = 4* за O (1).

Добавление элемента, s.add(x): сложность O (1) в среднем

Множества у нас реализованы с использованием хэш-таблиц. В результате просматривается есть ли переменная x в хэш-таблице, если нет, то добавляется в неё. Но в худшем случае сложность может увеличиться. Например, при коллизиях, когда два различных ключа хэшируются в один и тот же индекс массива. Это приводит к тому, что более одного элемента пытаются занять одну и ту же ячейку в хеш-таблице.

Например, для множества *s = {7,8,9,10}* вызов *s.add(6)* добавит элемент *6* за O(1) в среднем.

Проверка наличия значения, x in/not in s: сложность O (1) в среднем

Проверка наличия элемента в множестве также выполняется за O (1) в среднем благодаря хешированию. Опять же чекаем хэш-таблицу. В худшем случае, как и при добавлении, сложность может быть выше.

Например, для множества *s = {7,8,9,10}* вызов *7 in s* вернет True за O (1) в среднем.

Перебор множества, for v in s: Сложность O(N)

В этом цикле нас происходит итерирование, т.е. перебор всех элементов множества. Выполняется за O(N), где N - количество элементов в множестве.

Например, для множества *s = {7,8,9,10}* перебор элементов с помощью *for v in* s: займет O (4).

Объединение (union), s | t: Сложность O(len(s) + len(t)) или O (N+M)

У нас происходит объединение двух множеств, поэтому необходимо создание нового множества, в которое будут добавлены элементы из этих существующих. Поэтому у нас происходит итерация по всем элементам множества s, а затем добавление их в новое множество. То же самое у нас проделывается со вторым множеством t. Получается, что множество s у нас имеет длину в N элементов, а множество t содержит длину в M элементов. В результате сложность получается O (N + M), ну и типа O(len(s) + len(t))

Например, есть множества *s = {1, 2, 3}* и *t = {4, 5, 6, 7}*, затем с помощью функции *s | t* у нас создаётся новое множество *p = {1, 2, 3, 4, 5, 6, 7}*, которое создалось засчёт того, что пробежались по всем элементам обоих множеств. *len(s) = 3*, а *len(t) = 4*, получается что O(3 +4), а значит O(7).

Ой свою фотку вставила случайно