2-1. Arrays and strings

#_area/coding_test/leetcode/2_arrays_and_strings

In terms of algorithm problems, arrays (1D) and strings are very similar: they both represent an ordered group of elements. Most algorithm problems will include either an array or string as part of the input, so it's important to be comfortable with the basic operations and to learn the most common patterns.
알고리즘 문제에서 (1차원) 배열과 문자열은 매우 유사합니다. 둘 다 순서가 있는 요소들의 집합을 나타내기 때문이죠. 대부분의 알고리즘 문제에서는 배열이나 문자열(혹은 둘 다)이 입력에 포함되므로, 기본적인 연산에 익숙해지고 자주 쓰이는 패턴들을 익히는 것이 중요합니다.

"Array" can mean something different between languages. For example, Python primarily uses "lists" instead of arrays which are extremely lenient. Initialization is as easy as `arr = []`, and you don't need to worry about the type of data you store in the list or the size of the list. Other languages like C++ require you to specify the size and data type of the array during initialization, but also have support for lists (like `std::vector` in C++).
언어마다 "배열(Array)"이라는 단어가 가리키는 대상이 조금씩 다를 수 있습니다. 예를 들어 파이썬은 주로 "리스트"를 사용하며, 이는 매우 유연합니다. `arr = []`처럼 간단히 초기화할 수 있고, 리스트에 저장되는 데이터의 타입이나 크기에 대해 미리 신경 쓸 필요가 없습니다. 반면 C++ 같은 언어에서는 배열을 초기화할 때 크기와 데이터 타입을 지정해야 하며, 리스트에 해당하는 기능(`std::vector` 등)을 별도로 제공합니다.

> Technically, an array can't be resized. A dynamic array, or list, can be. In the context of algorithm problems, usually when people talk about arrays, they are referring to dynamic arrays. **In this entire course, we will be talking about dynamic arrays/lists, but we will just use the word "array".**
> 엄밀히 말하면, 배열은 크기 변경이 불가능합니다. 동적 배열(Dynamic Array) 또는 리스트는 크기를 변경할 수 있죠. 알고리즘 문제 풀이 맥락에서 사람들이 "배열"이라고 말할 때는 주로 동적 배열을 의미합니다. **이 강의 전체에서도 동적 배열/리스트를 다루면서, 편의상 "배열"이라고 부르겠습니다.**

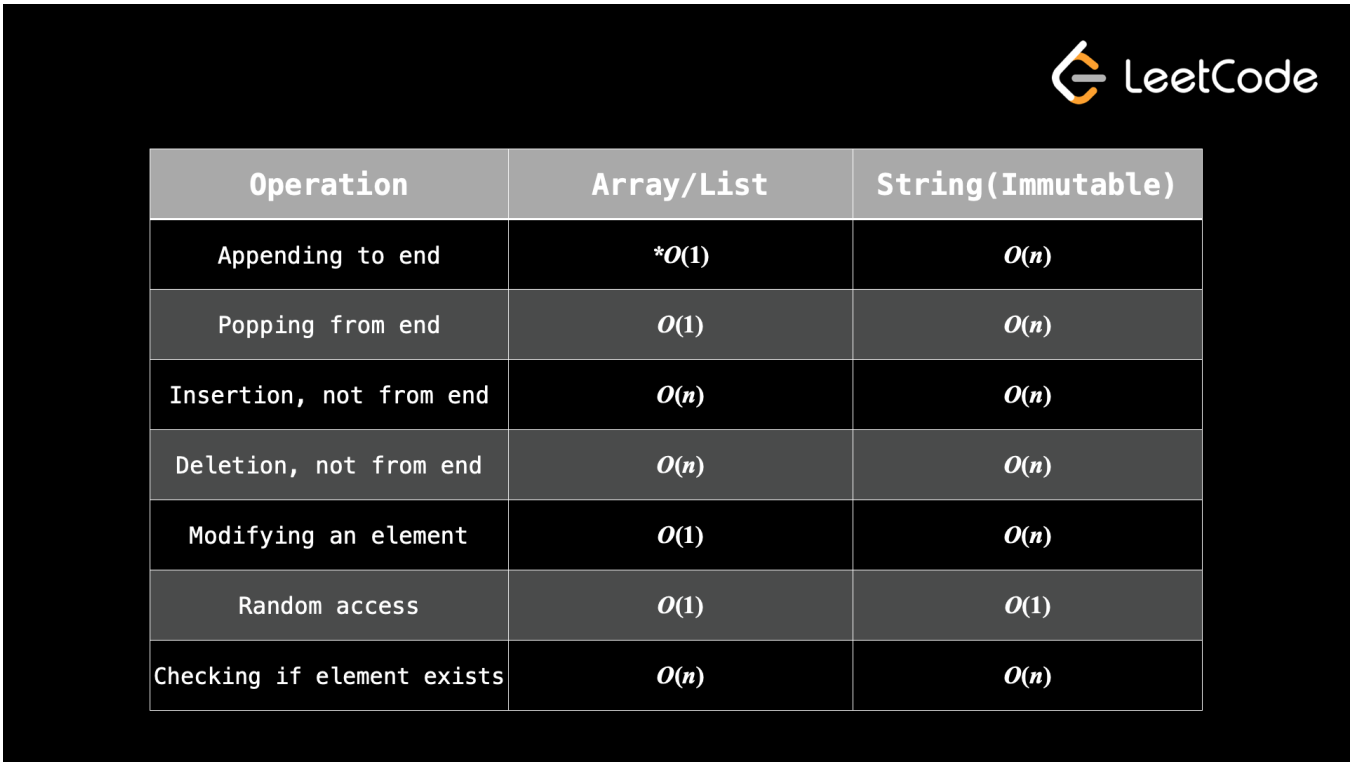
Similarly, strings are implemented differently between languages. In Python and Java, they are immutable. In C++, they are mutable. It's important to know the details behind arrays and strings for the language you plan on using in interviews. We don't have time to go through all the different implementations for each language, so please research it for your chosen language if you aren't already familiar.
문자열도 언어마다 구현이 다릅니다. 파이썬과 자바에서 문자열은 불변(Immutable)인 반면, C++에서는 변경이 가능합니다(Mutable). 인터뷰에서 사용하려는 언어가 있다면, 그 언어에서 배열과 문자열이 어떻게 동작하는지 미리 알아

두는 것이 중요합니다. 여기서 모든 언어별 구현을 다 다룰 수는 없으니, 아직 익숙하지 않다면 따로 공부해 보시기 바랍니다.

> Mutable: a type of data that can be changed.
> Immutable: A type of data that cannot be changed. If you want to change something immutable, you will need to recreate the entire thing.
> Mutable: 변경 가능한 데이터 타입
> Immutable: 변경 불가능한 데이터 타입. 만약 불변 타입을 수정하고 싶다면 전체를 새로 만들어야 합니다.

Why should we care about something being mutable or immutable? If you have a mutable array `arr = ["a", "b", "c"]` and an immutable string `s = "abc"`, but you want to instead represent `"abd"`, you can easily do `arr[2] = "d"`, but you cannot do `s[2] = "d"`. As such, if you wanted the string `s = "abd"`, you would need to create it entirely from scratch. With such a small string, it's not a big deal. But sometimes you are dealing with strings with 100,000 characters, so creating new versions just to modify one character is very expensive (*O(n)*, where *n* is the size of the string).
왜 어떤 것이 Mutable한지, Immutable한지가 중요할까요? 예를 들어, 변경 가능한 배열 `arr = ["a", "b", "c"]`와 불변 문자열 `s = "abc"`가 있다고 합시다. 이제 `"abd"`로 바꾸고 싶다면, 배열에서는 손쉽게 `arr[2] = "d"`라고 하면 되지만, 불변 문자열 `s`에서는 `s[2] = "d"` 같은 작업이 불가능합니다. 따라서 `s = "abd"` 문자열을 얻으려면 전체 문자열을 새로 만들어야 합니다. 작은 문자열이라면 큰 문제가 되지 않지만, 때로는 100,000자 정도의 문자열을 다룰 때도 있으므로, 단 하나의 문자를 바꾸기 위해 전체를 복사해야 한다면 이는 비용이 매우 큽니다 (*O(n)*, 여기서 *n*은 문자열의 길이입니다).

As mentioned before, a majority of algorithm problems will involve an array or string. They are extremely versatile data structures and it's impossible to list all the relevant problem-solving techniques in one article. In the next few articles, we'll go over the most common techniques. But first, let's take a quick look at the time complexity of array and string operations.
앞서 말했듯이, 알고리즘 문제의 상당수는 배열이나 문자열을 다룹니다. 이들은 워낙 범용성이 높아서, 한 글 안에 모든 문제 해결 기법을 다 담을 수는 없습니다. 이어지는 글들에서 자주 쓰이는 주요 기법을 다루어 보겠습니다. 먼저, 배열과 문자열 연산의 시간 복잡도부터 간단히 살펴봅시다.



Appending to the end of a list is amortized O(1).

리스트의 끝에 추가하는 연산은 [상환 분석 시 O(1)]입니다.