Arrays

Static Arrays: A **static array** is a data structure with a fixed size, determined at the time of its creation. Unlike dynamic arrays, the size of a static array cannot be changed during runtime. Static arrays are straightforward to use and are typically stored in contiguous memory locations.

Dynamic Arrays: A **dynamic array** is a data structure that can grow or shrink in size during runtime, unlike a static array, which has a fixed size. Dynamic arrays provide flexibility and are widely used in programming when the exact size of the array is not known beforehand.

- **Extra Space for Growth:**
  When a dynamic array is created, it allocates a **static array** with some extra space (capacity) beyond the current number of elements.
  This extra space allows new elements to be added without needing to resize the array immediately.
- **Doubling Capacity**:
  When the array becomes full (i.e., the number of elements equals the capacity), the dynamic array:
  - Creates a **new static array** with double the capacity.
  - Copies all the elements from the old array to the new one.
  - Deallocates the old array to free memory.
- **Extra Space is a Trade-off**:
  The extra positions (unused space) are a trade-off for efficiency. They reduce the frequency of resizing operations, which can be costly.

The **time complexity** of operations in a dynamic array depends on whether resizing (or "recycling") happens
If the array has unused capacity, the new element is simply placed in the next available slot. This is a **constant time operation**: **O(1)**.

If the array is full, resizing is triggered:

1. A new array (usually double the size) is created.

2. All existing elements are copied to the new array.
3. The new element is added.

So the copying process takes **linear time**: **O(n)**, where nnn is the number of elements in the array.

Operation	Array/List	String(Immutable)
Appending to end	*$O(1)$	$O(n)$
Popping from end	$O(1)$	$O(n)$
Insertion, not from end	$O(n)$	$O(n)$
Deletion, not from end	$O(n)$	$O(n)$
Modifying an element	$O(1)$	$O(n)$
Random access	$O(1)$	$O(1)$
Checking if element exists	$O(n)$	$O(n)$

As Strings are immutable in python so every operation creates and new string first, cloning the actual and then perform certain operations on clone, so all operations have **O(n)**, except for randomly accessing any index of a string which have **O(1)**