**Dynamic Array**

A dynamic array is a data structure that allows you to resize its size during runtime.

Dynamic arrays provide flexibility compared to static arrays with a fixed size.

It starts with an initial size, and as elements are added, it dynamically allocates more memory if

needed. If elements are removed, it may shrink in size to save memory.

**Key Features of Dynamic Array**

Add Element

Delete Element

Resize of Array Size

**Add element**

Add element at the end if the array size is not enough then extend the size of the array and add an element at the end of the original array as well as given index. Doing all that copying takes O(n) time, where n is the number of elements in our array. That’s an expensive cost for an append. In a

fixed-length array, appends only take O(1) time. But appends take O(n) time only when we insert into

a full array. And that is pretty rare, especially if we double the size of the array every time we run out

of space. So in most cases appending is still O(1) time, and sometimes it’s O(n) time.

**Delete Element**

Delete an element from array, default remove() method delete an element from end, simply store

zero at last index and you also delete element at specific index by calling removeAt(i) method where

i is index. removeAt(i) method shift all right element in the left side from the given index.

**Resize of Array Size**

When the array has null/zero data (aside from an element added by you) at the right side of the array, meaning it has unused memory, the method shrinkSize() can free up the extra memory. When all

space is consumed, and an additional element is to be added, then the underlying fixed-size array

needs to increase in size. Typically resizing is expensive because you have to allocate a bigger array

and copy over all of the elements from the array you have outgrown before we can finally append

our item.

**Memory Leak**

A memory leak in a dynamic array occurs when memory is allocated using new but the corresponding delete is missing, leading to unreleased memory. To prevent memory leaks:

* Always pair new with delete .
* Consider using smart pointers or standard containers for automatic memory management, reducing the risk of memory leaks.