Arrays

Arrays allow you to store data in *contiguous space in memory*

Note 0.1

Pros:

- Arrays are flexible in what they can store (primitives, reference types, etc)
- *Constant time access* when the index is known
  - Accessing when index is not known (searching) -> O(n)

Cons:

- If you run out of space you need to resize the array which is O(n)

ArrayLists

ArrayLists are backed by arrays, and are **contiguous**, which means you cannot have null spaces between data elements. This causes us to need to shift data to fill up the null spaces after remove operations.

ArrayList Big O

Theorem 0.1

Adding

Adding to Front: **O(n)** -> need to shift elements over to make space to add

Adding to Back: **amortized O(1)***. There is no need to shift but its amortized because every n operations, you need to perform an O(n) operation by resizing

- Amortized: when an "expensive" operation occurs infrequently so we can "average" it over the runtimes

Removing

Removing from the Front: O(n) -> must shift elements to fill the empty space

Removing from the back: O(1) -> simple set to null

Adding and Removing at a Given Index: O(n) -> shift data around the index

Accessing at a given index: O(1) -> arraylist backed by array

Pros and Cons

Note 0.2

Pros

- Data elements are stored contiguously
- **Dynamic Memory** - even though we resize the backing array behind the scenes, we consider ArrayLists to be dynamic

Note 0.3

Cons

- Cannot store primitives
- Still needs O(n) operations for resizing