Data Stuctures

Adrien Tremblay

Data Structure(DS)

A **data structure** is a way of organizing data so that it can be used effectively.

- Essential ingredient to creating.
- Help manage and organize data.
- Make code cleaner and easier to understand.
- Good coders now what data structure fits every given scenario.

Abstract Data Type (ADT)

An **Abstract Data Type** is an abstraction of a data structure which provides only the interface to which a data structure must adhere to.

- Language/implementation non-specific.

Examples:

Abstraction (ADT)	Implementation (DS)
List	Dynamic Array, Linked List
Queue	Linked List based queue, Array based queue, stack based queue
Map	tree Map, Hash Map
Vehicle	Golf Cart, Bike, Smart Car

Dynamic and Static arrays

Static Array

A **Static Array** is a fixed length container containing n elements **indexable** from range [0, n-1]

- Contiguous in memory (next to each other).
- Fixed size (cannot grow or shrink in size).

Used in

- Storing sequential data.
- Temporarily storing data.
- Buffers.
- In lookup tables.
- A lot of other places>

Complexity

Action	Static Array	Dynamic Array
Access	O(1)	O(1)
Search	O(n)	O(n)
Insertion	N/A	O(n)
Appending	N/A	O(1)
Deletion	N/A	O(n)

Dynamic Arrays

- Can grow or shink in size
- AKA Arraylists
- Typically implemented using a static array.
  - Create static array with non-zero size.
  - Add elements, keeping track of size.
  - If size goes over limit, create a new array of double the size and copy all elements over.

Linked Lists

A **Linked List** is a sequential list of nodes that hold data which point to other nodes.

- The last node always points to null.
- **Head**: first element
- **Tail**: last element

Used in

- Implementation of List, Queue, and Stack.
- Creating circular lists.
- Model real world things like trains.
- In hashtables, graphs.

Signly Linked vs Doubly Linked

- Singly only contains reference to the next node.
- Doubly also contains reference to the previous node.

Complexity

Action	Signly Linked	Doubly Linked
Search	O(n)	O(n)
Insert at head	O(1)	O(1)
Insert at tail	O(1)	O(1)
Remove at head	O(1)	O(1)
Remove at tail	O(1)	O(1)
Remove in middle	O(n)	O(n)