The five requirements of any data structure

- How to **Access** (one item / all items)
- How to **Insert** (at end / at position)
- How to **Delete** (from end / from position)
- How to **Find** (if exists / what location)
- How to **Sort** (sort in place / created sorted version)

Data Types: Something that describes data on how it can be used and what it can be

Primitive Type: is something that doesn't rely on the data inside of them:

1. short/long int – Whole numbers
2. Boolean – true or false = 1 single (1 or 0)
3. Chars – Uses 8 bits (C++)
4. Signed/Unsigned – positive/negative values (Signed) and double positive values (Unsigned)
5. Float/double – decimal

1 - 1 or 0 = Single bit

16 - 1s or 0 = bits

Data structures: give us organisation, storage, and access

Storing Data Consecutively: (One after another without interruption) in data structures we risk overwriting and losing precious data

Pointers: We will create an address that will point to where the structure or sometimes part of the structure of memory. We use a reference to find where in the memory we call strings and other data types implemented with data structures reference types.

Reference types: reference their specific value from an address of where the item is stored rather than direct access to the data itself. If the address changes the value the variable represents also changes:

1. String: since we allocate the space for strings

Memory Management:

1. C++ (Manage pointers, memory, and data)
2. Java and Python (Memory management handled for you)

Array: is a collection of elements, where each item is identified by an index or key

Valid indices: are like calling an index 5 of a limit 4 array

HELLO

H = Index 0, E = index 1, L = index 2, L = index 3, O = index 4

Array: store, organised and access our data

Multidimensional array is a way of adding columns to data with rows (0,0) (Row, Column)

Array ~ Jagged Array: Each array within the data structure can be of a different length

Java and C++: Basic arrays cannot be resized (Immutable)

Ruby, JavaScript: Basic arrays can be resized (Dynamic array or mutable arrays)

Mutable arrays: can slow down performance drastically

Sorting functions: Comparator to sort: A comparison of data to order objects like people's firstName or age

Big O Notation Array

0(1) Time = Constant time meaning during of algorithm no matter size

0(n) Linear search = Worst case scenario

Search

Insertion

Deletion

Sorting – Insertion sort, merge sort, heap sort, bubble sort, bucket sort, radix sort

Linked Lists: The elements in a linked list are linked with pointers so they don't share contiguous (touching) locations. Linear data structure.

Info: Each link is a pointer or holds an address to a memory location. We use the "next" to locate the address of memory like in train cars. We use the first train car next function to see where the second train

car is located in memory. Train cars are nodes. Pointer points to nodes.

Add – Back and front easiest

Access – We point the pointers until we get what we want

Delete – we find the item and update the next pointer

Search – We must traverse (Travel across or through) through the entire list

Things to consider when creating linked list: Is head pointing to NULL? If yes then we create a new head node else we reset the head node with new given data and use the next function to point to the previous head node

Linked list ~ Doubly Linked List: By adding a previous pointer to linked list

Java a list is not a data type

Python list is a data type

ArrayList: Has behaviour of a list on surface. Stored as an array under the hood, it favours direct access and won't provide great performance from insertion and deletions since it's like an array

Swift, Ruby, and JavaScript: No built-in linked list

Python: List are resizable arrays and no bulti-in linked list

C++: has a list container in the standard template library

Big O Notation Linked List

Access – 0(n)

Update – 0(n)

Insert – 0(1) since we don't care about order if so

Search – 0(n)

Delete – 0(n)

Sorting – Merge sort is preferred, quick sort and heapsort not ideal since random access

Stacks: Ordered series of objects. We push and pop object from a stack. Last in, First Out (LIFO), like in a card game such as uno. Are good to reverse things, keeping track of state and is like linked list. Can make a loop that removes everything on the stack.

Push, pop, peek (look at what is on top without removing it), contain, isempty (check whether stack is empty or not) and isfull (checks whether stack is full or not)

Can be used to keep track of every step if needed like error checking. Runtime stack keeps track of variables.

Stacks ~ Queues: First in, first out. Can be implemented with dynamic arrays or linked list.

Enqueue – When we add an item to the list

Dequeue – when we remove an item from the list

It is like having people standing in line. No indexing with queues. Applications like a printer, files communication lines, disk and taps. Multi-threading and concurrency situations: A CPU's execution of taking in multiple instruction sequences at the same time and making sure we take them in that order.

C# and python have a queue class.

put () = enqueue and get () dequeue for python.

Python talks lot about threading so thread synchronisation

C++ there is a queue container in standard template library push_back() and pop_front()

Priority Queues: Each element has a priority associated with it. Any item can be enqueued to the list but when we dequeue the one with the highest priority will be dequeued first. Elements with the same priority will be labelled as first in first out (Stacks) (Custom Comparing function for priority)

Java and C++ have priority queues but both require you to implement your custom comparing function for priority

DEQUEK or double-ended queue: It's like having a stack and a queue at the same time. We can add or remove from either end but can't remove within the queue.

Queues is like doubly linked list and if you need to remove something from the middle a priority queue can be helpful but if it's mostly middle of queue than front not good

Associative Array: Collection of key-value pairs (Key: Value = NewYork: Queens or reversed)

Flash card game. Front of flash card can be the key and the back of the flash card can be the value

Rules:

- Key-value pairs are bound together
- Each key must be unique

- Order isn't important so we don't use index
- Values are access with the key
- Values do need to be unique
- Abstract data type (a type (or class) for objects whose behaviour is defined by a set of values and a set of operations not implementation) like queues, stacks, arrays and linked list

Abstract Data Type vs Concrete Data Type

Abstract Data type (ADT) is a type (or class) for objects whose behaviour is defined by a set of values and a set of operations. Primitive values like int, float, char data types only with the knowledge that these data type can operate and be performed on without any idea of how they are implemented. So, a user only needs to know what a data type can do, but not how it will be implemented. Think of ADT as a black box which hides the inner structure and design of the data type. List, stack and queue are ADT

Concrete Data type (CDT) is absolutely defined. Only certain inputs and outputs can exist in this. Examples are Boolean, Integer, Floating Point, Text and more. Arrays, lists and trees are CDT.

Hashing: Data conversion process where we take raw data and mixing it together to form a smaller single piece of data. Raw data goes through a hash function. We can put characters, objects or numbers as the raw data.

It is like cooking where we use ingredients to cook but we can't reverse it like cooking. This may seem like encryption but with hashing you can't decrypt the hash value at all.

Application: A bank of usernames and passwords and a hacker has access to all this information. However, when we use the hash function, which can generate random characters that correlate to the original data. Fun Fact: That's why websites don't give you your current password you forgot when resetting the password because in their end its some random characters that even you won't be able to correlate toward figuring out your own forgotten password.

Anytime two inputs produce the same hash value then is a COLLISION. How to sort collision is

through ~ Separate Chaining: It affects performance because it is a linked listed which must be traversed in that specific slot to find what specific value we are looking for.

Hashing ~ Hash Table: Is an implementation of associative array abstract data structure (Bucket = key-pair)

Application: Hash Value % (Size of hash table) (Modulus) = reminder, which we use as the index. There is no linear search or traversing in a series of elements. We go straight to the element with the key. We use hash functions to create the hash table.

Python: Dictionaries

Hash Functions Language Implementation:

Java: hashCode Function

Python: GetHashCode Functiuon

JavaScript: Implement your own custom code or

Install node.js to get npm module needed

Hash Table Language Implementation:

Python: dict type and stored as an associative array or hash table internally

JavaScript: Use objects as associative arrays

Java: hash table and hash map collections

Big O Notation Hashing

Search: 0(1)

Insertion: 0(1)

Deletion: 0(1)

If you have collisions which will create linked list and may use more time = 0(n)

You can't sort with hashing base structures

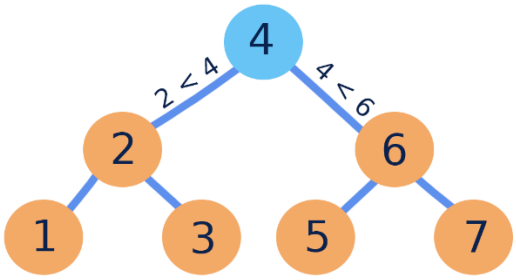
Sets: A collection of unique items, order doesn't matter, none of the elements are duplicated. Used to groups things with a common relationship like colours, clothing and more (No index or key) Works

with Membership. Are good for checking if a certain item exists and no duplicate values but that's it.

Python: has may functionality with sets.

Tree Data Structure: Like a linked list but has nodes that can link to many nodes. Root node is the beginning. Child node of the parent node. Child nodes with the same parents are siblings and a node with no children is a leaf.

Tree Data Structure ~ Binary search tree (BSTs): Left and right nodes and one parent node can have a max of two child nodes. Provides order where the left child node is less than parent and right child node must be greater than parent. Can have an imbalanced tree which can make it harder to insert or delete things within the tree. The tree must remain sorted to search faster.



In Order Traversal: 1 2 3 4 5 6 7

C++: Sets are implemented with BSTs.

Java: TreeMap implemented with a red-black tree

Python, JavaScript: third party implementation available

C#: sorted dictionaries are implemented with BSTs.

Big O Notation Binary Search Tree

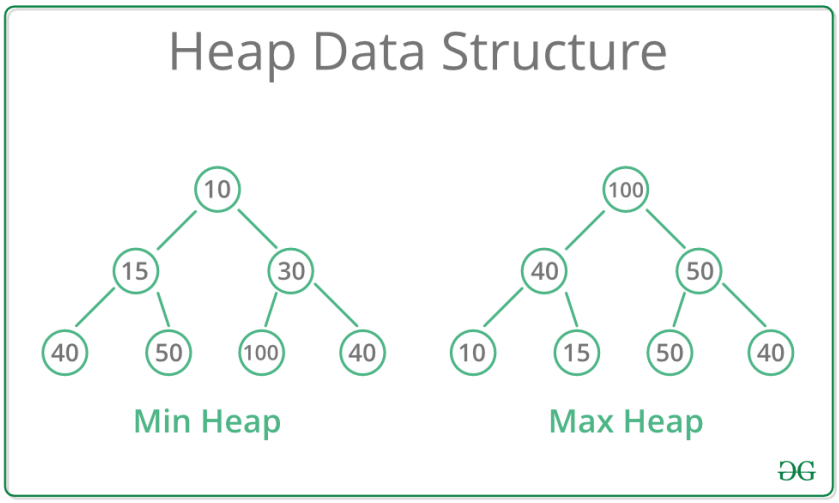
Binary Search Tree: maintain sorted order while staying fast for insertion, deletion, and accessing.

Balanced tree: 0(log(n))

Unbalanced tree: 0(n)

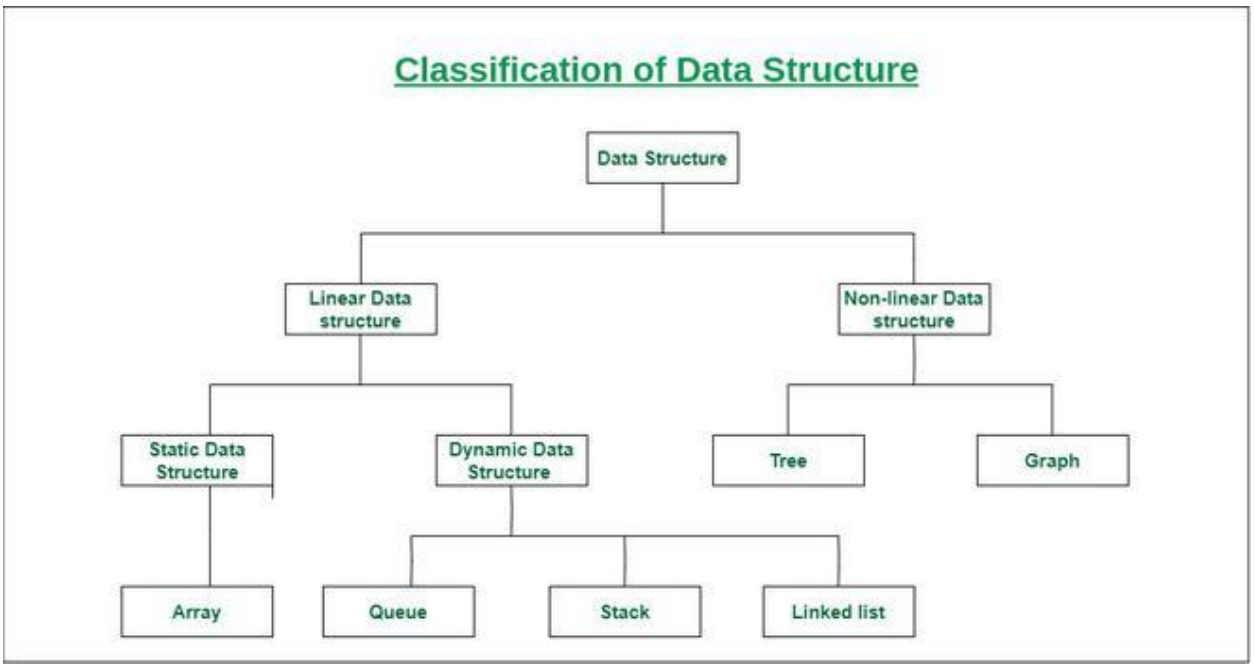
Heap: A data structure implemented as a binary tree. Collections of objects, Added top to bottom. Left to right. No issue with imbalanced tree. Min Heap, Max Heap. If a child node is greater or less

than its parent node depending on max or min heap. We make that child node the parent instead. Min heap bubbles up the lowest value and max heap bubbles up the high value. Heap isn't a fully sorted data structure. Priority queue are implemented by heaps.



Big O Notation Heaps

Heaps ~ Priority queue. Find min/man: 0(1) Insert 0(log(n)) Searching and deleting: 0(n) since we have to scam all items and no specific order



Static Data Structure vs Dynamic Data Structure

Static Data Structure: Fixed size like an array. The content of the data structure can be modified but without changing the memory space allocated to it.

Dynamic Data Structure: Have no fixed size like linked list, stacks, queues, tree and more.