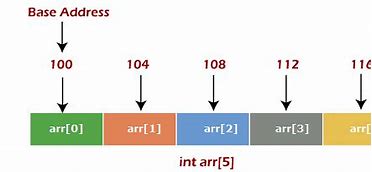
**Data Structures**

**Array**

Stores things in order. Has quick lookups by index.



**Operations of array:**

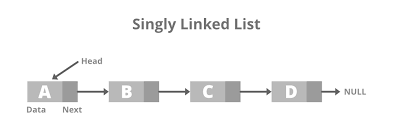
* Traverse − print all the array elements one by one.
* Insertion − Adds an element at the given index.
* Deletion − Deletes an element at the given index.
* Search − Searches an element using the given index or by the value.
* Update − Updates an element at the given index.

**[Dynamic Array](https://www.interviewcake.com/concept/dynamic-array)**

[An array that automatically grows as you add more items.](https://www.interviewcake.com/concept/dynamic-array)

**Linked List**

Also stores things in order. Faster insertions and deletions than arrays, but slower lookups (you have to "walk down" the whole list)



**Operations of linked list:**

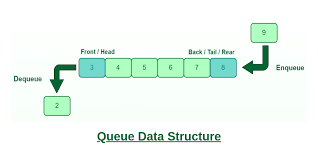
* Insertion: For the addition of nodes at any selected position.
* Traversal: To access all nodes one by one.
* Deletion: For removal of nodes at any selected position.

**Queue**

Like the line outside a busy restaurant. "First come, first served."

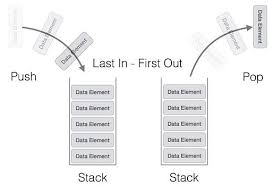
**operations of queue:**

* enqueue (adding elements to the rear of the queue)
* dequeue (removing elements from the front of the queue)



**Stack**

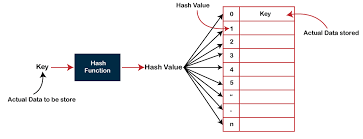
Like a stack of dirty plates in the sink. The first one you take off the top is the last one you put down.



* push : Adds an element to the top of the stack.
* pop : Removes the topmost element from the stack.
* isEmpty : Checks whether the stack is empty.
* isFull : Checks whether the stack is full.
* top : Displays the topmost element of the stack.

**Hash Table**

Like an array, except instead of indices you can set arbitrary keys for each value.

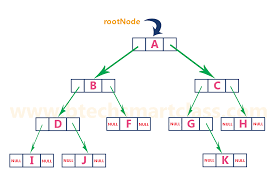


**operations of hashtable:**

search, insert, and delete.

**Tree**

Good for storing hierarchies. Each node can have "child" nodes.



**operations of tree in data structure:**

* Insert − Inserts an element in a tree.
* Pre-order Traversal − Traverses a tree in a pre-order manner.
* In-order Traversal − Traverses a tree in an in-order manner.
* Post-order Traversal − Traverses a tree in a post-order manner.

**Binary Search Tree**

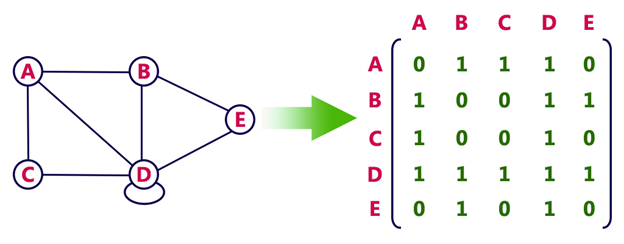
Everything in the left subtree is smaller than the current node, everything in the right subtree is larger. 𝑂(lg⁡𝑛)*O*(lg*n*) lookups, but only if the tree is balanced!

**operations of binary search tree:**

* Create: creates an empty tree.
* Insert: insert a node in the tree.
* Search: Searches for a node in the tree.
* Delete: deletes a node from the tree.
* Inorder: in-order traversal of the tree.
* Preorder: pre-order traversal of the tree.
* Postorder: post-order traversal of the tree.

**Graph**

Good for storing networks, geography, social relationships, etc.

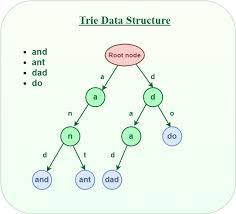


**operations of graph:**

* Add a vertex to the graph:
* This operation adds a new vertex to the graph.
* Add an edge to the graph:
* This operation adds a new edge to the graph, connecting two existing vertices.
* Remove a vertex from the graph:
* This operation removes a vertex from the graph, along with all edges connected to it.
* Remove an edge from the graph:
* This operation removes an edge from the graph, disconnecting the two vertices it was connected to.
* Find a path between two vertices:
* This operation finds a sequence of edges that connects two vertices in the graph.
* Find the shortest path between two vertices:
* This operation finds the path between two vertices that has the smallest total weight.

**Trie**

Stores a set of strings in a big tree of characters. Good for lookups by prefix. *Sometimes* saves space.



**operations of trie:**

insertion, deletion, and lookup of a string key