**Algorithms**

**Pass by Value?**

When we change the variable value, the actual value will remain Same

Like in case of “Number, String, Boolean”

**Pass by Reference?**

When we change the variable value, the actual value will also Change

Like in case of “Array, Object”

Sorting:

**01: Bubble Sort O(n^2)**

* Run two loops
* If **a[ j ] > a[ j+1 ]** then replace that

**02: Selection**

* Run Two Loops
* Store value of first loop each time
* Each time the second loop will be like that
  + // j=1->len // j=2->len // j=3->len
* If a[ j ] < a[ mainElement ] then mainElement = j
* When the second loop completed check that condition
  + If mainElement != i then swap them like arr[mainElement]⬄ arr[i]

**03: Insertion**

Picking One value and set it into its exact position

* Two Loops
* **Second loop is a reverse loop**

**04: Merge**

The Concept behind Merge Sort Algorithm is, take two sorted array and merge them and if we have only one unsorted array then divide them using recursion technique.

* Two Functions
* Recursion

**05: Quick**

The Concept behind the Algorithm is we select a value and place them in its correct position like all the elements in left hand side is less than the value and all the elements on right hand side are greater than the selected value.

**06: Counting Sort**

**07: Radix Sort**

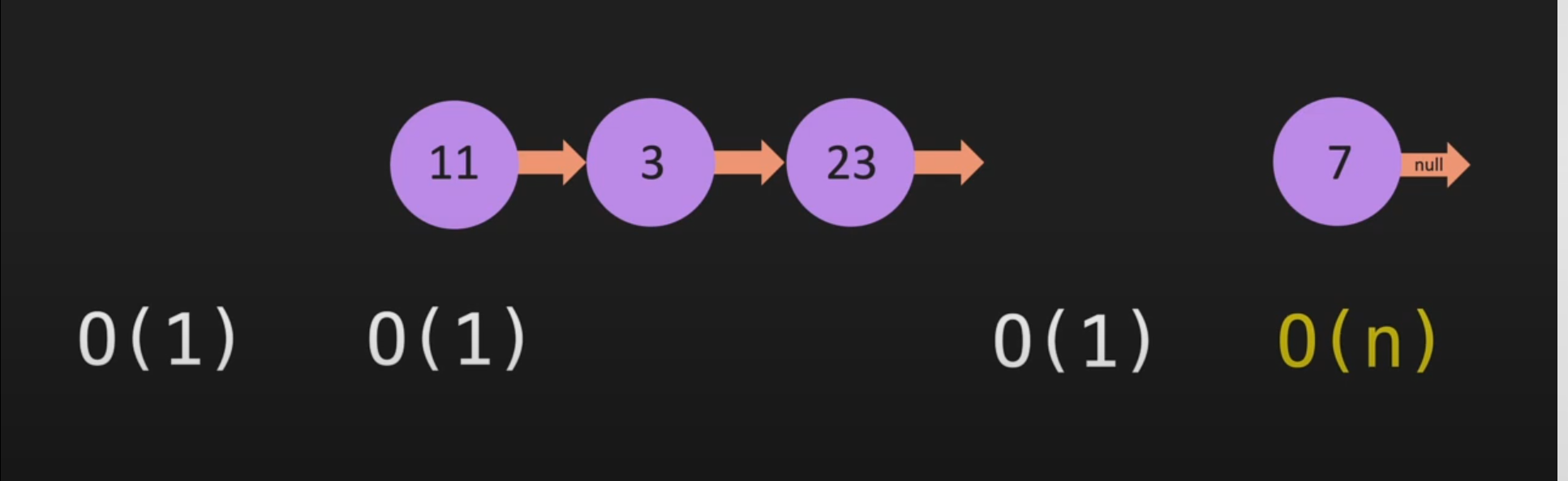
Stack And Queues: [Link](https://www.youtube.com/watch?v=--W233f-hCI&list=PLdPTfo6Ung1DKV0IUPsoRCZUXokM04nXf&index=1)

**01: Stack => LIFO**

* Data Structures that we can use with Stack
  + Array
  + Linked List
* If you are implementing Stack with Array then use Pop() Push() “O(1)” and do not use shift and unshift because then the index will always change and the complexity will be order of O(n)

**02: Queues => FIFO**

Data Structures that we can use

* Array
* Linked list
  + 

Searching:

**01: Linear**

**02: Binary:**

Can use for sorted array

Recursion: [Link](https://youtu.be/qz7UsK-Dz-A)

I think there are two types of recursions

1. Reverse
2. Forward
3. We can use multiple recursions inside a function

Standard Types of recursions

1. Linear Recursion (Only one fun ())
2. Tree Recursion (multiple fun ())
3. How to trace Tree recursion?
   1. BFS
   2. DFS

**01: Factorial**

**02: Fibonacci**

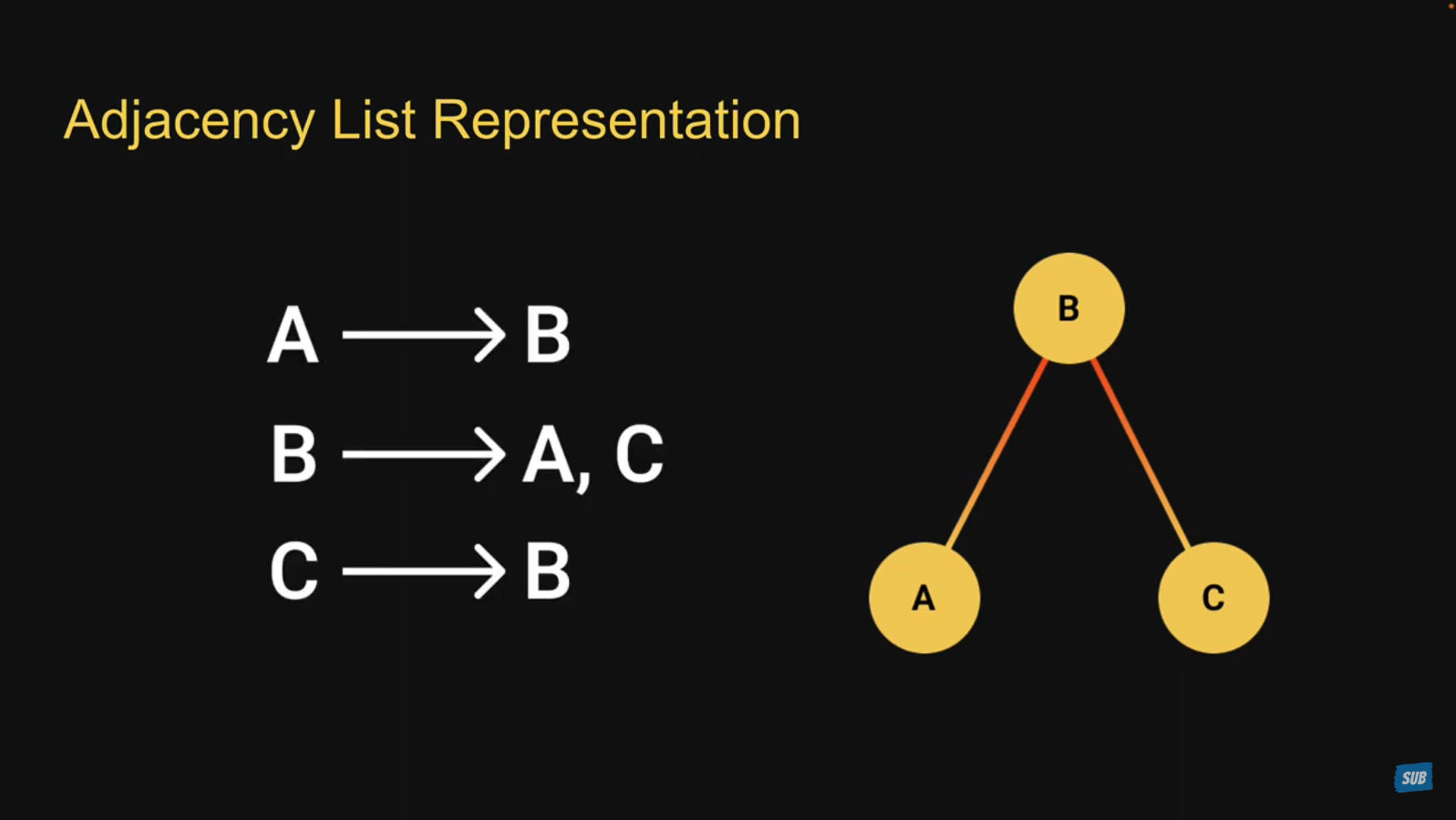
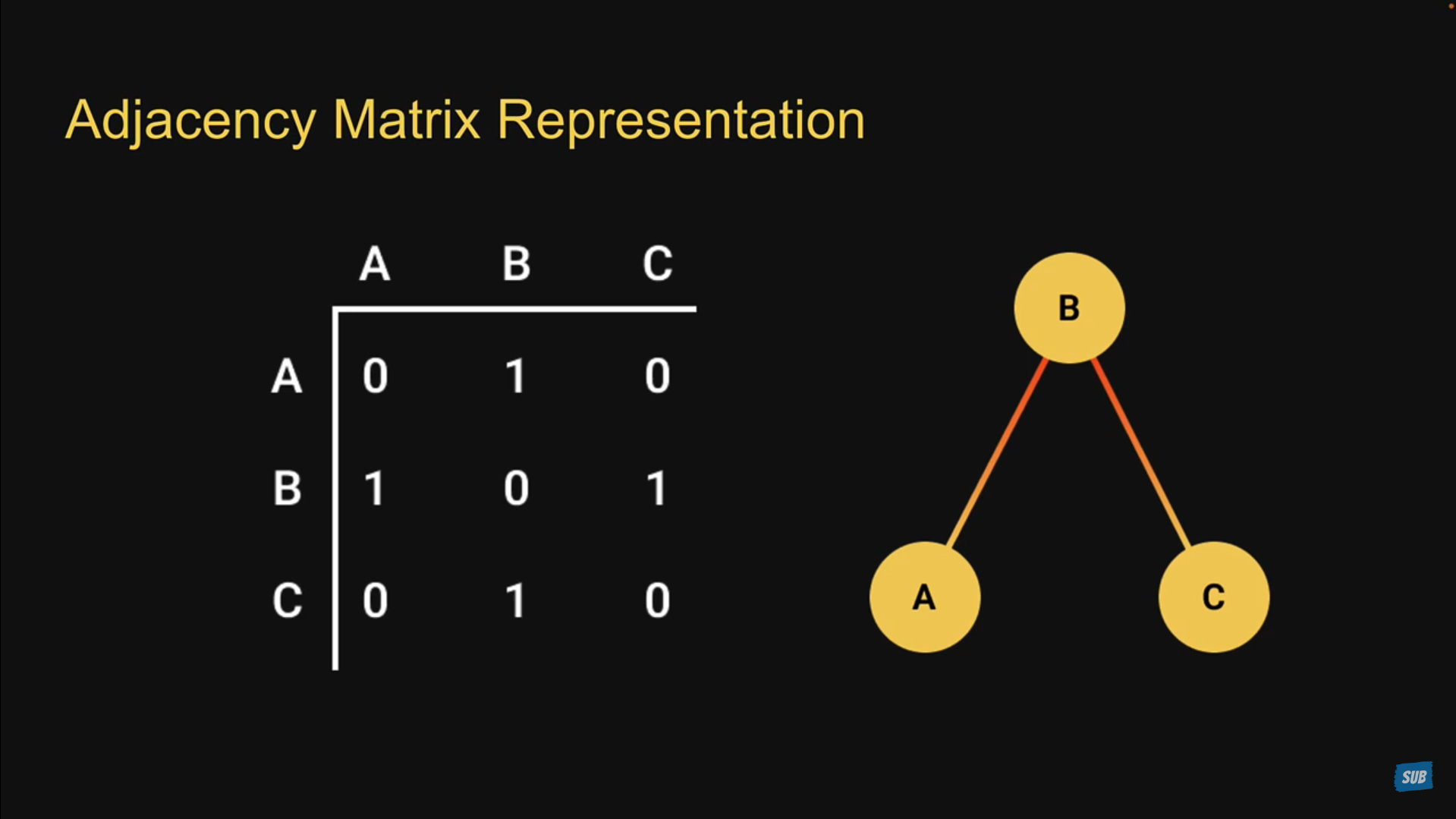
**03: Memorization**

**04: Tail Recursion**

Graphs: [Link](https://www.youtube.com/watch?v=QzysBDwCRDY&list=PL8p2I9GklV47TMMnPzqnkCtSOS3ebr4O7&index=48)

A Graph is a non-linear data structure that consists of vertices (nodes) and edges.

There are two ways of representation of a graph

1. Adjacency List
   1. 
2. Adjacency Matrix
   1. 

There are two Ways of Search in a Graph

1. BFS
2. DFS
   1. Pre-order Traversal is a type of Depth First Search
   2. In-order Traversal is a type of Depth First Search
   3. Post-order Traversal is a type of Depth First Search

**03 Flood Fill**

Dynamic Programming:

[GPT Chat Link](https://chatgpt.com/share/b6b2168b-e4f3-4832-8ca3-386905050191)

**01: Knapsack Problem**

**02: Longest Common Subsequence**

Linked list:

**01: Single**

**02: Double**

Hash Table:

**01: With collision**

**02 Without collision**

Tree Traversal:

Tree is a subpart/small part of the Graph

1. BFS:
   1. Explores nodes level by level.
   2. Useful for finding the shortest path in unweighted graphs.
   3. Suitable for problems involving "levels" or distances from a source node.
2. DFS:
   1. Explores nodes branch by branch.
   2. Can be used for pathfinding and exploring all possible paths.
   3. Suitable for problems requiring backtracking, topological sorting, and cycle detection.
3. Binary Search

Two Pointers:

01: Simple two pointer

Backtracking:

01: General purpose

02: Simple Example

Binary Heaps:

[Link](https://youtu.be/HqPJF2L5h9U)

Union Find

Ad hoc / String manipulation:

This is simply doing operations on the String. If we cover all the topics of Yahoo Baba String manipulation then its enough.

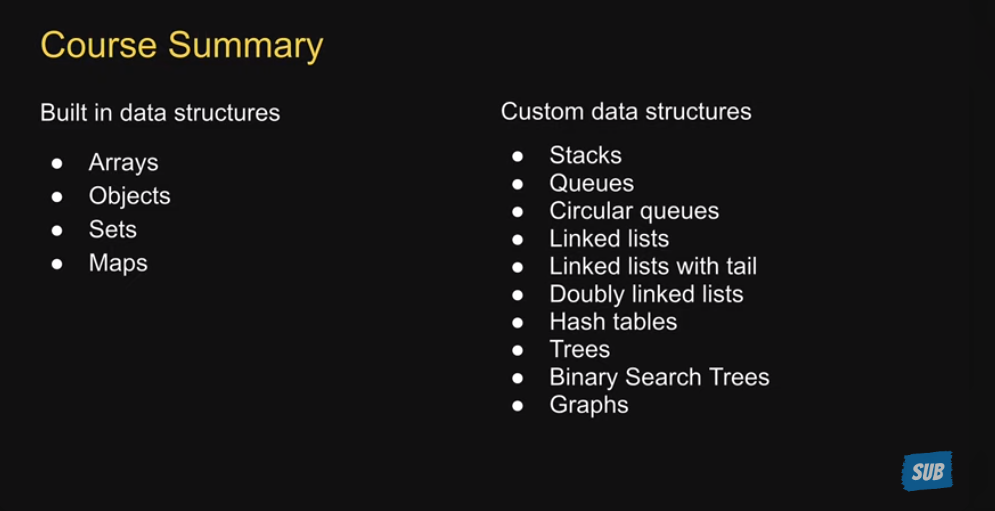
Greedy Algorithm:

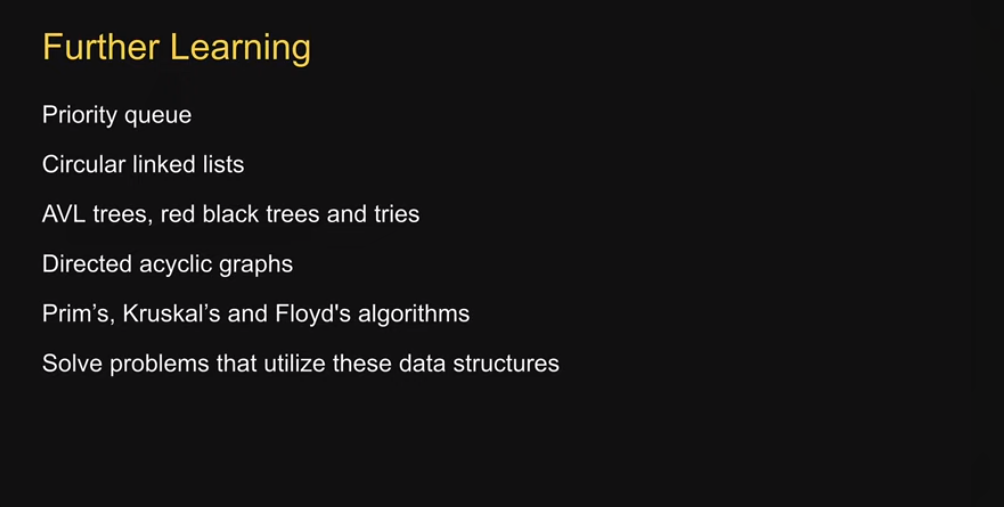
Trie:

A Trie (pronounced as "try") is a tree-like data structure that is used to efficiently store and retrieve keys in a dataset of strings. It is commonly used for tasks such as autocomplete and spell checking. Here's a step-by-step guide on how to implement a Trie in JavaScript:

Go to this URL for build your understanding

<https://youtu.be/kRYZmOZDlY0?list=PLC3y8-rFHvwjPxNAKvZpdnsr41E0fCMMP>





Segment Tree:

Memory updated

A Segment Tree is a versatile data structure that allows efficient querying and updating of array ranges. It's particularly useful for problems involving range queries and updates, like finding the sum, minimum, or maximum over a subrange of an array.

Here's an overview and a basic implementation of a Segment Tree in JavaScript:

Overview

* Structure: A binary tree where each node represents a segment (or range) of the array.
* Build Time: O(n)
* Query Time: O(log n)
* Update Time: O(log n)

Steps to Implement a Segment Tree

* Build the Segment Tree: Construct the tree from the given array.
* Query the Segment Tree: Find the sum, minimum, or maximum over a specific range.
* Update the Segment Tree: Update an element in the array and modify the tree accordingly.

Visiting Neighbors [Link](https://www.youtube.com/watch?v=Glp7THUpGow)

Fenwick Trees:

Fenwick Trees, also known as Binary Indexed Trees (BIT), are data structures that provide efficient methods for cumulative frequency tables. They are particularly useful for dynamic scenarios where frequent updates and prefix queries on an array are required.

**How Fenwick Tree Works**

A Fenwick Tree uses an internal array (often the same size as the input array) to store cumulative frequency information. Each element in the Fenwick Tree array represents a sum of elements in the input array in a particular range.

* Fenwick Tree Construction
* Initialize a tree array with zeros.
* Update the tree array with the values from the input array using the update operation.
* Fenwick Tree in JavaScript

Here’s a step-by-step implementation:

1. Initialize the Fenwick Tree:
2. Update Operation:
3. Prefix Sum Operation:

Bitmask:

Linear Data Structure

* Array
* Linked list
* Stack

Non-Linear Data Structure

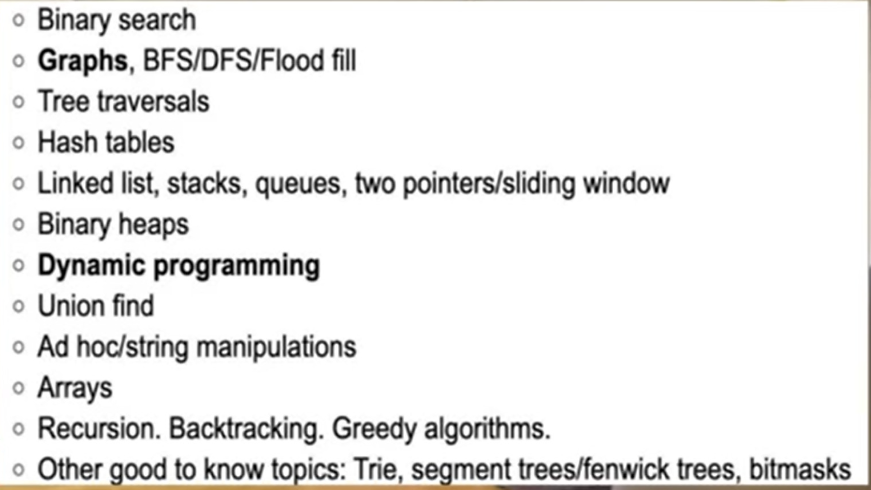
* Graphs

Time Complexity:

[Link](https://www.youtube.com/watch?v=W8yTh1I40DQ&list=PL8p2I9GklV47TMMnPzqnkCtSOS3ebr4O7&index=9)

// --- Important Resources

All the Algorithms which we should know before apply to Google

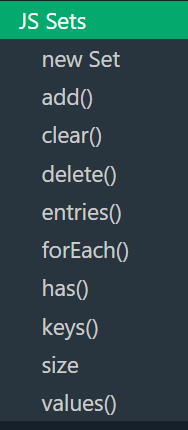


All Algorithms are present there with Animation:

~ <https://www.w3schools.com/dsa/index.php>

GPT Chat link:

~ <https://chatgpt.com/share/01b0c8ed-2a3d-448b-8e38-7f91c28cc1aa>



What is linear and non-linear?

**Linear** data structures are those in which elements form a sequence or a linear list. Each element is connected to its previous and next element in a sequential manner.

**Non-linear** data structures are those in which elements do not form a sequence or a linear list. Instead, they are organized in a hierarchical or interconnected manner.