Selection Sort	
What is Selection Sort?	It is a Simple and efficient algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the List
How does Selection Sort work?	You first start with an Unsorted List [64,25,12,22,11] - It will first check the 0th element, 64, 64 is larger than 25, 25 is larger than 12, 12 is smaller than 22, an 12 is larger than 11. Since 11 is the smallest value found in the list starting from 64, 64 will swap locations with 11. The list is now [11,25,12,22,64] - The second step is to look at the 1st element of the list, 25, 25 is larger than 12, 12 is smaller than 22, 12 is smaller than 64, making 12 the next smallest element. The list is now [11,12,25,22,64] - This will continue until the list is completely sorted. The list will end up as [11,12,22,25,64]
Advantages and Disadv-antages	Advantages - It is simple and easy - It works well with small datasets Disadvantages - It has O(N^2) Time Complexity - It does not do well with large datasets - It does not preserve the order of items with equal keys, meaning it is not stable
Time Complexity	O(N^2)
Auxillary Space	O(1)

Disjoint Set	
What is a Disjoint Set?	Two sets are called disjoint sets if they don't have any elements in common, therefore the intersection of the sets is a null set
What is a Disjoint Set Data Structure?	A Disjoint Set Data Structure Stores non-overlapping or disjoint subsets of elements.
What does this Data Structure Support?	This supports - Adding new sets to the disjoint set - Merging disjoint sets to a single disjoint set using a Union Operation - Finding Representation of a disjoint set using the Find operation - The ability to check if two sets are disjoint or not

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Disjoint Set (cont)

An Example	We are given 10 individuals: a, b, c, d, e, f, g, h, i, j The following relationships are found among them: a <-> b b <-> d c <-> f c <-> i j <-> e g <-> j Given the information of whether an individual is a friend of another or not we can divide them into 4 sub groups. g1 = { a, b, d} g2 = {c, f, i} g3 = {e, g, j} g4 = {h}
Data Structures that are used within the Disjoint Set Data Structure	An Array: There is an array of integers that we call Parent[]. If there are N items. The i'th elemth of the array represents the i'th item. More specifically, the i'th element of the Parent[] array is the parent of the i'th item. These relationships create one or more virtual trees. The other Data Sructure is a Tree representing the Disjoint Set: If two elements are in the same tree, then they are in the same Disjoint Set. The root node (The topmost node) of each tree is called the representative of the set. There is always one single Unique representative of eachset. A simple rule to help identify a representative is if "i" is the representative of a set, the Parent[i] = i. If "i" isn't the representative, it can be found by traveling up the tree.

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Disjoint Set (cont)	
Operations this data structure has	Find: This can be implemented by recursively traversing the Parent[] array until we hit a node that is the parent itself. Union: This will take 2 elements and it will find the representatives of their sets using the find operation (previously mentioned), and it will finaly put either on of the trees under the root node of the other tree. This effectively merges the trees and sets
Time Complexity	Creating Single Item Sets: O(N) Union and Set Operations: O(Log N) Overall: O(N +Log N)
Space Complexity	O(N)

Greedy Algorithm	
What is the Greedy Algorithm?	The Greedy Algorithm is an Algorithm paradigm that builds up a solution piece by piece. This will always choose the next piece that offers the most obvious and immediate benefit. This does not consider the overall optimal result.
Key Features	- Works in a top-down approach and will not reverse a previous decision - Always goes for the local best choice to produce the global best result
How to determine if the Greedy Algorithm Should be used	1. Greedy Choice Property: If an optimal solution to the problem can be found by choosing the best choice at each step without reconsidering the previous step chosen. 2. Optimal Substructure: If the oprimal overall solution corresponds to the optimal solution to the subproblems
Advantages and Disadvantages	Advantages: - The Algorithm is easier to describe - It can preform better than other algorithm when placed in the correct situation Disadvantages: - Doesn't always produce the optimal solution

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Kruskal's Minimum Spanning Tree (MST) Algorithm

What is a Minimum Spanning Tree?	A Minimum Spanning Tree/Minimum Weight Spanning Tree for a weighted, connected, undirected graph is a spanning tree with a weight less than or equal to the weight of every other spanning tree
What is a Spanning Tree?	A spanning tree is a subset of a given graph which has all the vertices covered with the minimum possible number of edges. A spanning tree does not have any cycles and cannot be disconnected
Steps to finding MST using the Kruskal Algorithm	1. Sort all the edges in non-decreasing order by their weight 2. Pick the smallest edge and check if it forms a cycle with the spanning tree formed so far. If it doesn't keep the edge and add it to the spanning tree, if it does, discard the edge. 3. Repeat step 2 until there are (vertices - 1) edges in the spanning tree
Time Complexity	O(E *LogE) or Log(E* LogV)
Auxiliary Space	O(V + E) V is the number of vertices within the given graph E is the number of edges within the given graph

All Pairs Shortest Path - Floyd Warshall Algorithm

What is it?	It is an Algorithm to find the shortest distance between every pair of vertices in a given edge-weighted directed graph
What approach does it follow?	This algorithm follows the dynamic programming approach to find the shortest path
How to accomplish the task using the algorithm	Steps 1. Initialize a solution matrix

Single Source Shortest Path - Dijkstra's Algorithm

What is it?	An Algorithm used to find the shortest paths from the source to all vertices in the given graph

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Single Source Shortest Path - Dijkstra's Algorithm (cont)

What are the steps?	Steps 1. Create a sptSet (Shortest Path Tree Set). This will keep track of vertices included in the SPT whose minimum distance from the source is calculated and finalized. Initilize as empty 2. Assign distance values to all vertices in the input graph. Inilize all distance values as INFINITE except the source vertex which is 0, this is so it will be picked first. 3. While the sptSet doesn't include all vertices - Pick a vertex U that is not in the sptSet and has a minimum distance value - Include U into the sptSet - Update the distance values of all adjacent vertices of U -- To update, iterate through all adjacent vertices -- For every adjacent vertex of V, if the sum of the distance value of U (from source) and weight of edge U-V, is less than the distance value of V, then update the distance value of V. Note: Use a boolean array sptSet[] to represent the set of vertices in SPT. If a value of sptSet[V] is true, then vertex V is included in SPT
Time Complexity	O(V^2)
Auxillary Space	O(V)

Binary Search Trees

What are they?	Binary search trees are node-based binary tree data structures which have the following properties - The left subtrees of nodes contain only nodes with keys lesser than a node's key - The right subtrees of nodes contain only nodes with keys greater than a node's key - The left and right subtree each must also be a binary search tree {{nl)}} Note: There may be no duplicates within the tree
Inserting a node into a BST	Time Complexity: O(N) Auxillary Space: O(1)
Inorder Traversal	Time Complexity: O(N) Auxillary Space: O(1)
Preorder Traversal	Time Complexity: O(N) Auxillary Space: O(1)
Postorder Traversal	Time Complexity: O(N) Auxillary Space: O(1)
Level Order Traversal	Time Complexity: O(N) Auxillary Space: O(1)

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Binary Search Trees (cont)	
Print Nodes at Given Level	Time Complexity: O(N) Auxillary Space: O(1)
Print Nodes at Given Leaf	Time Complexity: O(N) Auxillary Space: O(1)
Print all non-Leaf nodes	Time Complexity: O(N) Auxillary Space: O(1)
Right View of BST	Time Complexity: O(N) Auxillary Space: O(1)
Left View of BST	Time Complexity: O(N) Auxillary Space: O(1)
Height of BST	Time Complexity: O(N) Auxillary Space: O(1)
Delete a Node of BST	Time Complexity: O(Log N) Auxillary Space: O(1)
Smallest Node of the BST	Time Complexity: O(Log N) Auxillary Space: O(1)
Total Number of Nodes in the BST	Time Complexity: O(N) Auxillary Space: O(1)
Delete a BST	Time Complexity: O(N) Auxillary Space: O(1)
Advantages and Disadvantages	Advantages

Binary Search Trees	
What are they?	Binary search trees are node-based binary tree data structures which have the following properties - The left subtrees of nodes contain only nodes with keys lesser than a node's key - The right subtrees of nodes contain only nodes with keys greater than a node's key - The left and right subtree each must also be a binary search tree {{nl)}} Note: There may be no duplicates within the tree
Inserting a node into a BST	Time Complexity: O(N) Auxillary Space: O(1)
Inorder Traversal	Time Complexity: O(N) Auxillary Space: O(1)
Preorder Traversal	Time Complexity: O(N) Auxillary Space: O(1)
Postorder Traversal	Time Complexity: O(N) Auxillary Space: O(1)
Level Order Traversal	Time Complexity: O(N) Auxillary Space: O(1)

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com

Binary Search Trees (cont)	
Print Nodes at Given Level	Time Complexity: O(N) Auxillary Space: O(1)
Print Nodes at Given Leaf	Time Complexity: O(N) Auxillary Space: O(1)
Print all non-Leaf nodes	Time Complexity: O(N) Auxillary Space: O(1)
Right View of BST	Time Complexity: O(N) Auxillary Space: O(1)
Left View of BST	Time Complexity: O(N) Auxillary Space: O(1)
Height of BST	Time Complexity: O(N) Auxillary Space: O(1)
Delete a Node of BST	Time Complexity: O(Log N) Auxillary Space: O(1)
Smallest Node of the BST	Time Complexity: O(Log N) Auxillary Space: O(1)
Total Number of Nodes in the BST	Time Complexity: O(N) Auxillary Space: O(1)
Delete a BST	Time Complexity: O(N) Auxillary Space: O(1)
Advantages and Disadvantages	Advantages - Fast Search - In Order Traversal - Space Efficient Disadvantages - Skewed Trees - Additional Time Required - Efficiency

Sponsored by **Readable.com**
Measure your website readability!
https://readable.com