Active Directory:

# BFS used to traverse the graph/tree because for every group and subgroups have to check if some user/leaf node exits. Also because if user found in a group then algorithm can directly return True.
# For this algorithm basically checking if some user/leaf node is a descendant of some group if it is return True else return False. Therefore, bfs used for level order traversing.
# O(m*n) Time Complexity. Where m refers to a node/ group and n refers to its groups and users
# O(m) Space Time Complexity because extra space used to store groups/nodes as BFS uses a Queue data structure.

LRU-Cache:

# Hash table and doubly linked list used to implement LRU-Cache
# O(1) Time Complexity for both put and get functions
# O(n) Space Complexity
# head of linked list acts as a LRU(least recently used)
# tail of linked list acts as a MRU(most recently used)
# hashtable key holds node as its value
# Doubly Linked list is used to have a O(1) removal of a node
# When ever LRU Cache reaches its limit due to put function, the LRU cache referenced by the head has to removed from the cache.Therefore, linked list is used.
# Whenever get function called that cache/Key-Value(node) if present already then identified using the

key, and value updated. Also it becomes now most recently used cache therefore add it to the end of the linked list. The hash table identifies the cache/key-value object using the key and then using doubly linked list we can simply remove that object/node from the cache/linked list and add it to the end because it isn't getting deleted anytime soon.

<u>Huffman-Coding:</u>

# Encoding Time complexity:
# Worst case time complexity for encoding will be O(n^2) because using linked list to determine highest
# priority, and in worst case appending would take O(n), and in best case O(1). Also since, have to
# iterate over n items.Therefore, worst case O(n^2) and best case O(n) to build the tree. Also since
# traversing through all the nodes in the tree it is also gonna take O(n). Thus
# Worst case: O(n^2), Best case:O(n), and Average case: O(n) (I believe)
# Space Time Complexity: O(1) because in the worst case there will be some constant nodes added to the linked.Since, there is a limit of number of characters. O(52)

# Decoding Time complexity:
# Average case: O(number of bits) because traversing through the root to the leaf node for all bits.

# Most frequent character is represented by fewer bits.
# Linked list acts as a priority queue in this algorithm
# head of linked list refers to the highest priority node.
# appending to a sorted list worst case:O(n), best case: O(1).
#frequency table a hash table used to key track of frequency/occurrence of some character
#DFS used to traverse because dfs is a depth first search or keeps traversing till leaf node encountered and then backtracks.

<u>Finding-Files:</u>
# Backtracking Algorithm or DFS used to solve the problem as have to check all possible combinations.
# O(V) Time complexity as all vertices/nodes checked
#O(1) Space Time Complexity

<u>Union-Intersection LinkedList:</u>
# The time complexity for Union is O(n)
# The time complexity for Intersection is O(n)
# For Finding Union between two linked lists first getting rid of duplicates using Hash table, from within those lists and then traversing the longest linked list and removing duplicates from it by using smallest linked list's hash table, and then once duplicates removed connecting the longest linked list to the smallest/other linked list.
# For Finding Intersection between two linked lists kept track of duplicates/common values in a list/array, and then created a new linked list iterating through those common

values.
# O(n) Worst case Space Time complexity: since storing nodes in a hash table and then also using external space to store duplicates/common values.

<u>Block-Chain:</u>
# O(1) Time complexity for adding to the linked list
# O(n) Space Complexity. Since, storing every block in a hash table, which is being referred by its Hash.
# hash table used where block's hash acts as a key, and block as its value
# Every block’s previous hash refers to the current head’s hash or previous block’s hash, and head of the linked list gets updated with the new block/latest block.
# Hash’ keys used to refer a block from a hash table, so that can traverse through the block chain.