■ Data Structures & Algorithms in Python (Jovian – Notes + Code)

Lesson 1: Binary Search, Linked Lists & Complexity

Linear vs Binary Search

Linear Search: O(N)
Binary Search: O(log N)

Linked Lists: Node-based, sequential access.

```
# Linear Search
def linear_search(arr, target):
    for i, val in enumerate(arr):
        if val == target:
            return i
    return -1

# Binary Search
def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1
```

Lesson 2: Trees, BST & Recursion

Binary Search Trees: Left < Root < Right.

Traversals: Inorder, Preorder, Postorder.
Recursion naturally used for tree problems.

```
class BSTNode:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

def insert(root, key):
    if root is None:
        return BSTNode(key)
    if key < root.key:
        root.left = insert(root.left, key)
    else:
        root.right = insert(root.right, key)
    return root

def inorder(root):
    if root:
        inorder(root.left)
        print(root.key, end=" ")
        inorder(root.right)
```

Lesson 3: Sorting & Divide & Conquer

Sorting algorithms: Bubble O(N^2), Merge O(N log N), Quick O(N log N) avg.

Divide & Conquer: Break -> Solve -> Combine.

```
def merge_sort(arr):
    if len(arr) <= 1:
        return arr
    mid = len(arr)//2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)

def merge(left, right):
    result, i, j = [], 0, 0
    while i < len(left) and j < len(right):
        if left[i] < right[j]:
            result.append(left[i]); i += 1
        else:
            result.append(right[j]); j += 1
    result.extend(left[i:]); result.extend(right[j:])
    return result
```

Lesson 4: Recursion & Dynamic Programming

DP avoids recomputation via memoization/tabulation.

Problems: LCS, 0/1 Knapsack.

```
def lcs(X, Y):
    m, n = len(X), len(Y)
    dp = [[0]*(n+1) for _ in range(m+1)]
    for i in range(1, m+1):
        for j in range(1, n+1):
            if X[i-1] == Y[j-1]:
                dp[i][j] = dp[i-1][j-1] + 1
            else:
                dp[i][j] = max(dp[i-1][j], dp[i][j-1])
    return dp[m][n]
```

Lesson 5: Graph Algorithms

Graphs: Represent with adjacency list/matrix.

BFS O(V+E), DFS O(V+E), Dijkstra O((V+E) log V).

```
from collections import deque

def bfs(graph, start):
    visited = set()
    queue = deque([start])
    while queue:
        node = queue.popleft()
        if node not in visited:
            print(node, end=" ")
            visited.add(node)
            queue.extend(graph[node] - visited)
```