DAY 20:

ASSIGNMENT 1:

Task 1: Knapsack Problem

Write a function int Knapsack(int W, int[] weights, int[] values) in Java that determines the maximum value of items that can fit into a knapsack with a capacity W. The function should handle up to 100 items. Find the optimal way to fill the knapsack with the given items to achieve the maximum total value. You must consider that you cannot break items, but have to include them whole.

ANSWER:

```
public class Knapsack {

    public static int knapsack(int W, int[] weights, int[] values) {
        int n = weights.length;
        int[] dp = new int[W + 1];

        // Build dp[] in bottom-up manner
        for (int i = 0; i < n; i++) {
            for (int w = W; w >= weights[i]; w--) {
                dp[w] = Math.max(dp[w], values[i] + dp[w - weights[i]]);
            }
        }

        return dp[W];
    }

    public static void main(String[] args) {
        int W = 50; // Example capacity
        int[] weights = {10, 20, 30}; // Example weights
        int[] values = {60, 100, 120}; // Example values
        System.out.println("Maximum value in Knapsack = " + knapsack(W, weights, values));
    }
}
```

Explanation:

1. Initialization:

- dp[w] will hold the maximum value that can be attained with weight w.

- Initialize dp[w] = 0 for all w because with zero capacity or no items, the maximum value is 0.

2. Filling the DP Array:

- Iterate over each item (from 0 to n-1).

- For each item, iterate over each possible weight from W down to the weight of the current item weights[i] in reverse order.

- Update dp[w] as the maximum of the current value dp[w] and the value of including the current item values[i] + dp[w - weights[i]].

3. Result:

- The result will be in dp[W], which represents the maximum value with n items and capacity W.