Through 0/1 Knapsack problem, analyze the greedy and dynamic programming approach for the same dataset.

#include <stdio.h>

#include <stdlib.h>

// Structure to store item information (weight and value)

typedef struct {

int value;

int weight;

} Item;

// Function for Greedy Approach (Approximated solution)

int greedyKnapsack(Item items[], int n, int capacity) {

// Calculate value-to-weight ratio for each item

float ratio[n];

for (int i = 0; i < n; i++) {

ratio[i] = (float)items[i].value / items[i].weight;

}

// Sort items based on the value-to-weight ratio in descending order

for (int i = 0; i < n-1; i++) {

for (int j = 0; j < n-i-1; j++) {

if (ratio[j] < ratio[j+1]) {

// Swap items and ratios

Item temp = items[j];

items[j] = items[j+1];

items[j+1] = temp;

float temp\_ratio = ratio[j];

ratio[j] = ratio[j+1];

ratio[j+1] = temp\_ratio;

}

}

}

int totalValue = 0;

int currentWeight = 0;

for (int i = 0; i < n; i++) {

if (currentWeight + items[i].weight <= capacity) {

// If the item can be added to the knapsack, add it

currentWeight += items[i].weight;

totalValue += items[i].value;

} else {

// If the item cannot be fully added, skip it

break;

}

}

return totalValue;

}

// Function for Dynamic Programming approach (Optimal solution)

int knapsackDP(Item items[], int n, int capacity) {

int dp[n+1][capacity+1];

// Build the DP table

for (int i = 0; i <= n; i++) {

for (int w = 0; w <= capacity; w++) {

if (i == 0 || w == 0) {

dp[i][w] = 0; // Base case: no items or zero capacity

} else if (items[i-1].weight <= w) {

// Include the item or exclude it, choose the maximum value

dp[i][w] = (items[i-1].value + dp[i-1][w - items[i-1].weight] > dp[i-1][w])

? items[i-1].value + dp[i-1][w - items[i-1].weight]

: dp[i-1][w];

} else {

// Exclude the item

dp[i][w] = dp[i-1][w];

}

}

}

return dp[n][capacity];

}

int main() {

// Set of items (value, weight)

Item items[] = {

{60, 10},

{100, 20},

{120, 30}

};

int n = sizeof(items) / sizeof(items[0]);

int capacity = 50;

// Greedy Knapsack

int greedyResult = greedyKnapsack(items, n, capacity);

printf("Greedy Approach: Maximum value in Knapsack = %d\n", greedyResult);

// Dynamic Programming Knapsack

int dpResult = knapsackDP(items, n, capacity);

printf("Dynamic Programming Approach: Maximum value in Knapsack = %d\n", dpResult);

return 0;

}