Exp_13\0_1Knapsack_DP_Greedy.c

```c
#include <stdio.h>
#include <stdlib.h>

// Structure to represent an item
typedef struct {
    int weight;
    int value;
    double ratio; // value-to-weight ratio for greedy approach
} Item;

// Function to find maximum of two integers
int max(int a, int b) {
    return (a > b) ? a : b;
}

// Dynamic Programming approach for 0/1 Knapsack
int knapsackDP(int weights[], int values[], int n, int capacity) {
    int dp[n + 1][capacity + 1];

    // Build table dp[][] in bottom-up manner
    for (int i = 0; i <= n; i++) {
        for (int w = 0; w <= capacity; w++) {
            if (i == 0 || w == 0)
                dp[i][w] = 0;
            else if (weights[i - 1] <= w)
                dp[i][w] = max(values[i - 1] + dp[i - 1][w - weights[i - 1]],
                               dp[i - 1][w]);
            else
                dp[i][w] = dp[i - 1][w];
        }
    }

    return dp[n][capacity];
}

// Comparison function for qsort (descending order of ratio)
int compare(const void *a, const void *b) {
    Item *item1 = (Item *)a;
    Item *item2 = (Item *)b;
    if (item2->ratio > item1->ratio)
        return 1;
    else if (item2->ratio < item1->ratio)
        return -1;
    return 0;
}

// Greedy approach for 0/1 Knapsack (approximation)
int knapsackGreedy(int weights[], int values[], int n, int capacity) {
    Item items[n];

    // Create items array with value-to-weight ratio
```

```c
    for (int i = 0; i < n; i++) {
        items[i].weight = weights[i];
        items[i].value = values[i];
        items[i].ratio = (double)values[i] / weights[i];
    }

    // Sort items by ratio in descending order
    qsort(items, n, sizeof(Item), compare);

    int totalValue = 0;
    int currentWeight = 0;

    // Pick items based on greedy choice
    for (int i = 0; i < n; i++) {
        if (currentWeight + items[i].weight <= capacity) {
            currentWeight += items[i].weight;
            totalValue += items[i].value;
        }
    }

    return totalValue;
}

int main() {
    int n, capacity;

    printf("Enter the number of items: ");
    scanf("%d", &n);

    int weights[n], values[n];

    printf("Enter the weights of items:\n");
    for (int i = 0; i < n; i++) {
        printf("Item %d: ", i + 1);
        scanf("%d", &weights[i]);
    }

    printf("Enter the values of items:\n");
    for (int i = 0; i < n; i++) {
        printf("Item %d: ", i + 1);
        scanf("%d", &values[i]);
    }

    printf("Enter the knapsack capacity: ");
    scanf("%d", &capacity);

    // Dynamic Programming approach
    int dpResult = knapsackDP(weights, values, n, capacity);
    printf("\n--- Dynamic Programming Approach ---\n");
    printf("Maximum value (DP): %d\n", dpResult);

    // Greedy approach
    int greedyResult = knapsackGreedy(weights, values, n, capacity);
    printf("\n--- Greedy Approach ---\n");
```

```c
    printf("Maximum value (Greedy): %d\n", greedyResult);

    printf("\nNote: Greedy approach may not always give optimal solution for 0/1 Knapsack.\n");

    return 0;
}
```