Description:

C program that solves the Fractional Knapsack problem, and reports execution time using clock().

Source Code:

```
#include <stdio.h>
#include <stdlib.h>
#include<time.h>
// Structure to represent an item
typedef struct {
 int itemId;
 int weight;
 int profit;
 float pByw; // Profit-to-weight ratio
} Item;
// Comparison function for sorting items by profit-to-weight ratio in descending order
int compareItems(const void *a, const void *b) {
 Item *itemA = (Item *)a;
 Item *itemB = (Item *)b;
 if (itemA->pByw < itemB->pByw) return 1;
 if (itemA->pByw > itemB->pByw) return -1;
 return 0;
}
// Function to solve the Fractional Knapsack problem
float fractionalKnapsack(Item *items, int n, int capacity) {
 // Sort items based on profit-to-weight ratio in descending order
```

```
 qsort(items, n, sizeof(Item), compareItems);
 float totalProfit = 0.0;
 int currentWeight = 0;
 for (int i = 0; i < n; i++) {
 // If the current item can be taken completely
 if (currentWeight + items[i].weight <= capacity) {
 currentWeight += items[i].weight;
 totalProfit += items[i].profit;
 printf("Added item %d (Weight: %d, Profit: %d) completely. Current weight: %d, Total  profit:
%.2f\n",
 items[i].itemId, items[i].weight, items[i].profit, currentWeight, totalProfit);  } else {
 // Take a fraction of the current item
 float remainingCapacity = capacity - currentWeight;
 float fraction = remainingCapacity / items[i].weight;
 totalProfit += fraction * items[i].profit;
 currentWeight += remainingCapacity; // Knapsack is now full
 printf("Added %.2f%% of item %d (Weight: %d, Profit: %d). Current weight: %d, Total  profit:
%.2f\n",
 fraction * 100, items[i].itemId, items[i].weight, items[i].profit, currentWeight,  totalProfit);
 break; // Knapsack is full, no more items can be added
 }
 }
 return totalProfit;
}
int main() {
    clock_t start, end;
    double cpu_time_used;
```

```
 int n, knapsackCapacity;
 printf("Enter the number of items: ");
 scanf("%d", &n);
 Item *items = (Item *)malloc(sizeof(Item) * n);
 if (items == NULL) {
 printf("Memory allocation failed.\n");
 return 1;
 }
 printf("Enter itemId, weight, and profit for each item:\n");  for (int i = 0; i < n; i++) {
 printf("Item %d: ", i + 1);
 scanf("%d %d %d", &items[i].itemId, &items[i].weight, &items[i].profit);  items[i].pByw =
(float)items[i].profit / items[i].weight;  }
 printf("Enter the knapsack capacity: ");
 scanf("%d", &knapsackCapacity);
 start=clock();
 float maxProfit = fractionalKnapsack(items, n, knapsackCapacity);  printf("\nMaximum profit for
the given capacity: %.2f\n", maxProfit);
 free(items);
    end = clock();


   cpu_time_used = ((double)(end - start) / CLOCKS_PER_SEC);
   printf("Execution time: %f seconds\n", cpu_time_used);
 return 0;
}
```

Output:

```
Enter the number of items: 2
Enter itemId, weight, and profit for each item:
Item 1: 2
3
4
Item 2: 5
6
7
Enter the knapsack capacity: 9
Added item 2 (Weight: 3, Profit: 4) completely. Current weight: 3, Total  profit: 4.00
Added item 5 (Weight: 6, Profit: 7) completely. Current weight: 9, Total  profit: 11.00

Maximum profit for the given capacity: 11.00
Execution time: 0.000047 seconds
```