PRACTICAL - 6

AIM : To perform fractional knapsack algorithm using greedy approach

PSEUDO CODE :

KnapsackDP (Array values, Array weights, int n, int capacity)

Step 1: Create a 2D array dp[n+1][capacity+1]
Step 2: Initialize dp[0][w] = 0 for all w from 0 to capacity
Step 3: for i from 1 to n, do steps 4-6
Step 4: for w from 0 to capacity, do steps 5-6
Step 5: if weights[i-1] <= w, then do step 6
Step 6: dp[i][w] = max(values[i-1] + dp[i-1][w - weights[i-1]], dp[i-1][w])
Step 7: Set max_value = dp[n][capacity]
Step 8: Set w = capacity
Step 9: Create an empty array selected_items
Step 10: for i from n to 1, do steps 11-13
Step 11: if dp[i][w] is not equal to dp[i-1][w], then do step 12
Step 12: Add i to selected_items
Step 13: if dp[i][w] is not equal to dp[i-1][w], then subtract weights[i-1] from w
Step 14: Print "Maximum value:", max_value
Step 15: Print "Selected items:", selected_items

CODE :

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
  int weight;
  int value;
  double ratio;
} Item;
int compareItems(const void* a, const void* b) {
  Item* itemA = (Item*)a;
  Item* itemB = (Item*)b;
  return itemB->ratio - itemA->ratio;
}
```

```
void fractionalKnapsack(int capacity, Item items[], int n) {
  qsort(items, n, sizeof(Item), compareItems); // Sort items based on ratios
    double totalValue = 0.0;
  printf("Selected Items and Fractions:\n");
  for (int i = 0; i < n; i++) {
     if (capacity == 0) {
        break;
     }
     int quantity = (capacity < items[i].weight) ? capacity : items[i].weight;
     double fraction = (double)quantity / items[i].weight;
     totalValue += fraction * items[i].value;
     capacity -= quantity;
     printf("Item %d: %.2lf of weight, Profit: %.2lf\n", i + 1, fraction, fraction * items[i].value);
  }
  printf("Maximum value that can be obtained: %.2lf\n", totalValue);
}
void main() {
  printf("Aabhas Kumar Jha - A2305221279\n\n");
  int capacity, n;
  printf("Enter the knapsack capacity: ");
  scanf("%d", &capacity);
  printf("Enter the number of items: ");
  scanf("%d", &n);
  Item items[n];
  for (int i = 0; i < n; i++) {
     printf("Enter the weight and value of item %d: ", i + 1);
     scanf("%d %d", &items[i].weight, &items[i].value);
     items[i].ratio = (double)items[i].value / items[i].weight;
  }
  fractionalKnapsack(capacity, items, n);
}
```

OUTPUT :

```
Aabhas Kumar Jha - A2305221279

Enter the knapsack capacity: 15
Enter the number of items: 6
Enter the weight and value of item 1: 5 10
Enter the weight and value of item 2: 3 40
Enter the weight and value of item 3: 6 30
Enter the weight and value of item 4: 2 50
Enter the weight and value of item 5: 1 60
Enter the weight and value of item 6: 4 20
Selected Items and Fractions:
Item 1: 1.00 of weight, Profit: 60.00
Item 2: 1.00 of weight, Profit: 50.00
Item 3: 1.00 of weight, Profit: 40.00
Item 4: 1.00 of weight, Profit: 30.00
Item 5: 0.75 of weight, Profit: 15.00
Maximum value that can be obtained: 195.00
```

COMPLEXITY : O(nlog n)

PRACTICAL - 6

AIM : To perform fractional knapsack algorithm using greedy approach

PSEUDO CODE :

KnapsackDP (Array values, Array weights, int n, int capacity)

Step 1: Create a 2D array dp[n+1][capacity+1]
Step 2: Initialize dp[0][w] = 0 for all w from 0 to capacity
Step 3: for i from 1 to n, do steps 4-6
Step 4: for w from 0 to capacity, do steps 5-6
Step 5: if weights[i-1] <= w, then do step 6
Step 6: dp[i][w] = max(values[i-1] + dp[i-1][w - weights[i-1]], dp[i-1][w])
Step 7: Set max_value = dp[n][capacity]
Step 8: Set w = capacity
Step 9: Create an empty array selected_items
Step 10: for i from n to 1, do steps 11-13
Step 11: if dp[i][w] is not equal to dp[i-1][w], then do step 12
Step 12: Add i to selected_items
Step 13: if dp[i][w] is not equal to dp[i-1][w], then subtract weights[i-1] from w
Step 14: Print "Maximum value:", max_value
Step 15: Print "Selected items:", selected_items

CODE :

```
#include <stdio.h>
#include <stdlib.h>
typedef struct {
  int weight;
  int value;
  double ratio;
} Item;
int compareItems(const void* a, const void* b) {
  Item* itemA = (Item*)a;
  Item* itemB = (Item*)b;
  return itemB->ratio - itemA->ratio;
}
```

```
void fractionalKnapsack(int capacity, Item items[], int n) {
  qsort(items, n, sizeof(Item), compareItems); // Sort items based on ratios
    double totalValue = 0.0;
  printf("Selected Items and Fractions:\n");
  for (int i = 0; i < n; i++) {
     if (capacity == 0) {
        break;
     }
     int quantity = (capacity < items[i].weight) ? capacity : items[i].weight;
     double fraction = (double)quantity / items[i].weight;
     totalValue += fraction * items[i].value;
     capacity -= quantity;
     printf("Item %d: %.2lf of weight, Profit: %.2lf\n", i + 1, fraction, fraction * items[i].value);
  }
  printf("Maximum value that can be obtained: %.2lf\n", totalValue);
}
void main() {
  printf("Rishita Chaubey - A2305221265\n\n");
  int capacity, n;
  printf("Enter the knapsack capacity: ");
  scanf("%d", &capacity);
  printf("Enter the number of items: ");
  scanf("%d", &n);
  Item items[n];
  for (int i = 0; i < n; i++) {
     printf("Enter the weight and value of item %d: ", i + 1);
     scanf("%d %d", &items[i].weight, &items[i].value);
     items[i].ratio = (double)items[i].value / items[i].weight;
  }
  fractionalKnapsack(capacity, items, n);
}
```

OUTPUT :

```
Rishita Chaubey - A2305221265

Enter the knapsack capacity: 15
Enter the number of items: 6
Enter the weight and value of item 1: 5 10
Enter the weight and value of item 2: 3 40
Enter the weight and value of item 3: 6 30
Enter the weight and value of item 4: 2 50
Enter the weight and value of item 5: 1 60
Enter the weight and value of item 6: 4 20
Selected Items and Fractions:
Item 1: 1.00 of weight, Profit: 60.00
Item 2: 1.00 of weight, Profit: 50.00
Item 3: 1.00 of weight, Profit: 40.00
Item 4: 1.00 of weight, Profit: 30.00
Item 5: 0.75 of weight, Profit: 15.00
Maximum value that can be obtained: 195.00
```

COMPLEXITY : O(nlog n)