**EXPERIMENT 5**

**FRACTIONAL KNAPSACK PROBLEM**

**AIM: For following instance of the Knapsack Problem:** **n = 3 , m =20,**

**(p1, p2, p3) = (25, 24, 15) and (w1, w2, w3) = (18, 15, 10), Calculate the total profit.**

**PSEUDOCODE:**

**Knapsack algorithm**

**for i = 1 to n**

**Calculate Profit/Weight**

**Sort Objects in decreasing order of P/W ratio**

**for i = 1 to n**

**if (M > 0 and Wi ≤ M)**

**{**

**M = M – Wi;**

**P = P +Pi;**

**}**

**else break;**

**if (M > 0 )**

**P = P + Pi ( M / Wi ) ;**

**\*\*\*\*Solve the given example, and match theoretical output with practical output\*\*\*\***

**// C++ program to solve fractional Knapsack Problem**

**#include <bits/stdc++.h>**

**using namespace std;**

**// Structure for an item which stores weight and**

**// corresponding value of Item**

**struct Item {**

**int profit, weight;**

**// Constructor**

**Item(int profit, int weight)**

**{**

**this->profit = profit;**

**this->weight = weight;**

**}**

**};**

**// Comparison function to sort Item**

**// according to profit/weight ratio**

**static bool cmp(struct Item a, struct Item b)**

**{**

**double r1 = (double)a.profit / (double)a.weight;**

**double r2 = (double)b.profit / (double)b.weight;**

**return r1 > r2;**

**}**

**// Main greedy function to solve problem**

**double fractionalKnapsack(int W, struct Item arr[], int N)**

**{**

**// Sorting Item on basis of ratio**

**sort(arr, arr + N, cmp);**

**double finalvalue = 0.0;**

**// Looping through all items**

**for (int i = 0; i < N; i++) {**

**// If adding Item won't overflow,**

**// add it completely**

**if (arr[i].weight <= W) {**

**W -= arr[i].weight;**

**finalvalue += arr[i].profit;**

**}**

**// If we can't add current Item,**

**// add fractional part of it**

**else {**

**finalvalue**

**+= arr[i].profit**

**\* ((double)W / (double)arr[i].weight);**

**break;**

**}**

**}**

**// Returning final value**

**return finalvalue;**

**}**

**// Driver code**

**int main()**

**{**

**int W = 50;**

**Item arr[] = { { 60, 10 }, { 100, 20 }, { 120, 30 } };**

**int N = sizeof(arr) / sizeof(arr[0]);**

**// Function call**

**cout << fractionalKnapsack(W, arr, N);**

**return 0;**

**}**

**TIME COMPLEXITY: O(n log2 n)**