|  |
| --- |
| Experiment No. 5 |
| Fractional Knapsack using Greedy Method |
| Date of Performance: |
| Date of Submission: |

**Experiment No. 5**

**Title:** Fraction Knapsack

**Aim:** To study and implement Fraction Knapsack Algorithm

**Objective:** To introduce Greedy based algorithms

**Theory:**

Greedy method or technique is used to solve Optimization problems. A solution that can be maximized or minimized is called Optimal Solution.

The knapsack problem or rucksack problem is a problem in combinatorial optimization: Given a set of items, each with a mass and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed size knapsack and must fill it with the most valuable items. The most common problem being solved is the 0-1 knapsack problem, which restricts the number xi of copies of each kind of item to zero or one.

In Knapsack problem we are given:1) n objects 2) Knapsack with capacity m, 3) An object i is associated with profit Wi , 4) An object i is associated with profit Pi , 5) when an object i is placed in knapsack we get profit Pi Xi .

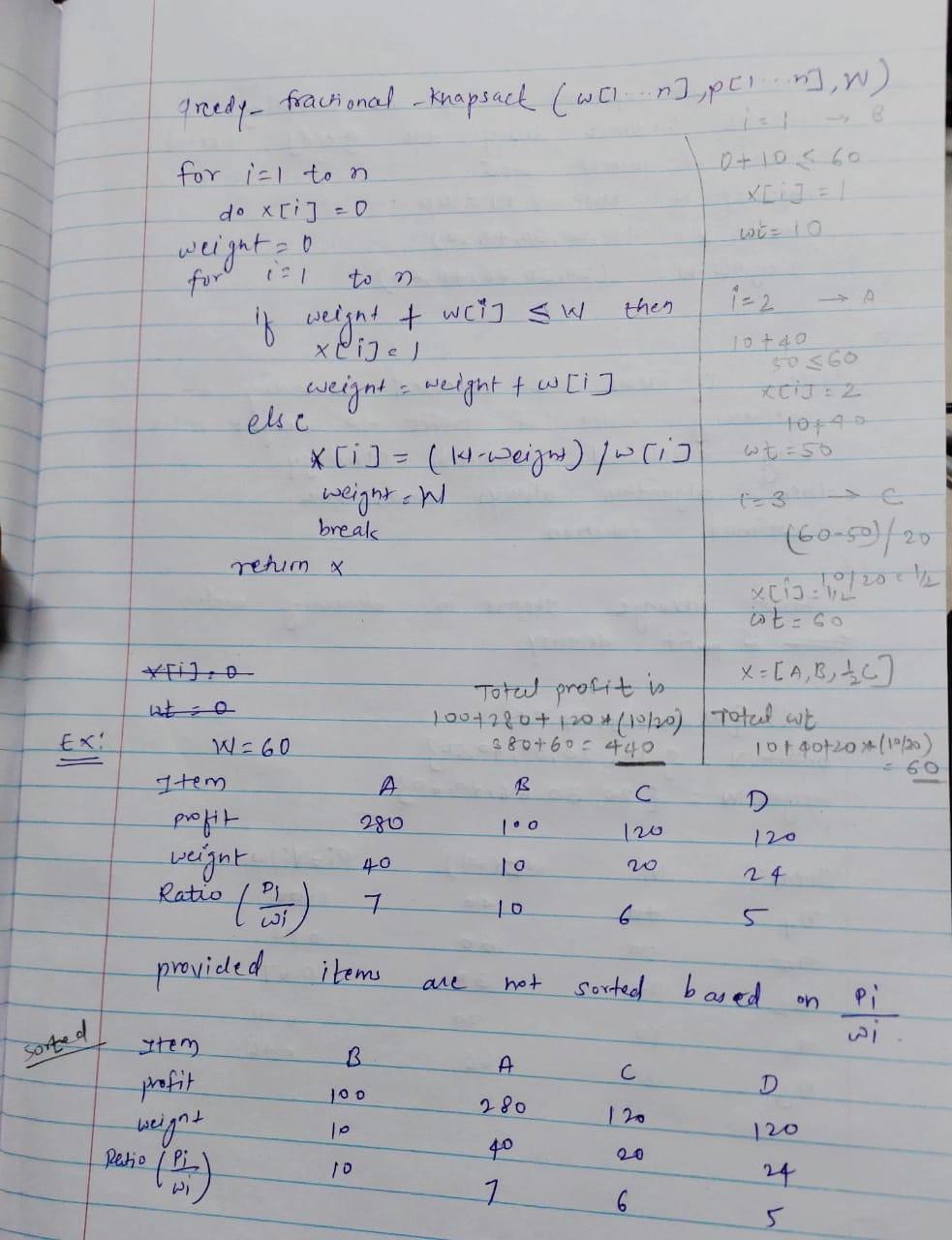
Here objects can be broken into pieces (Xi Values) The Objective of Knapsack problem is to maximize the profit.

**Example:**

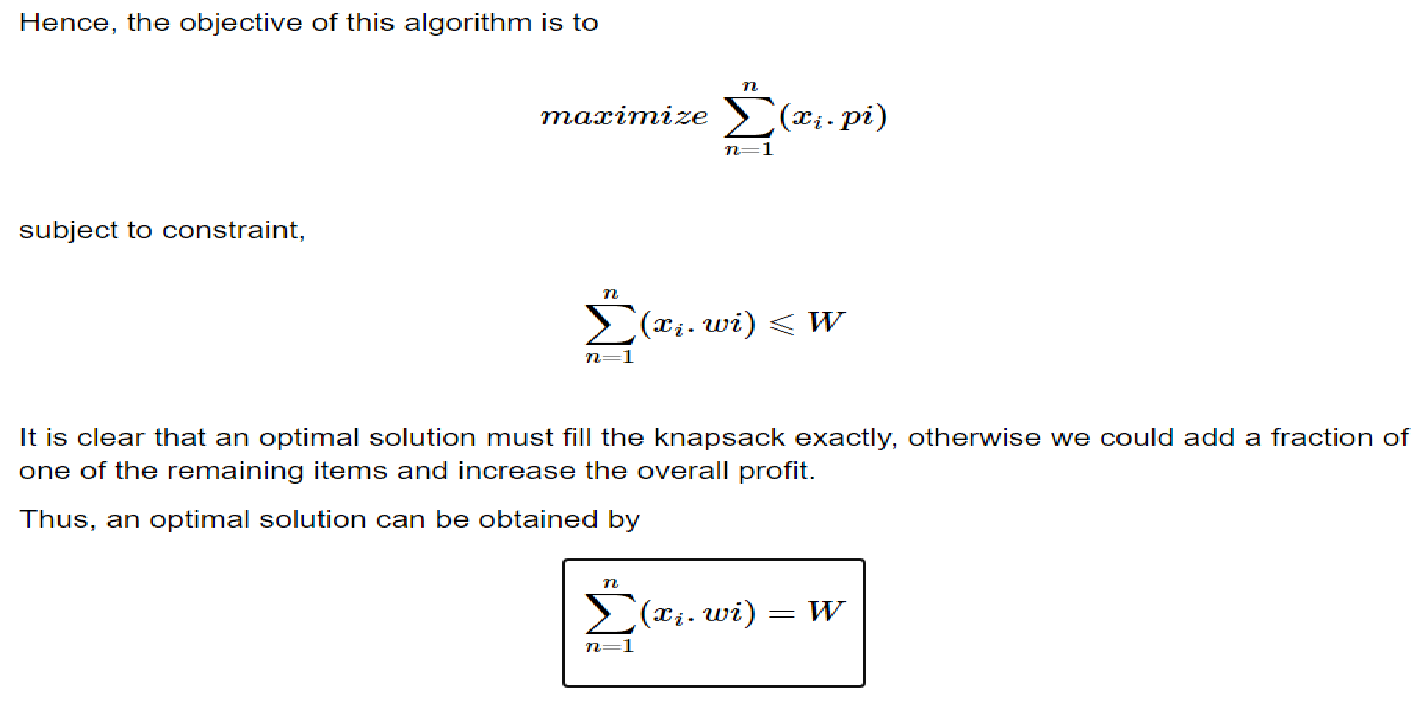
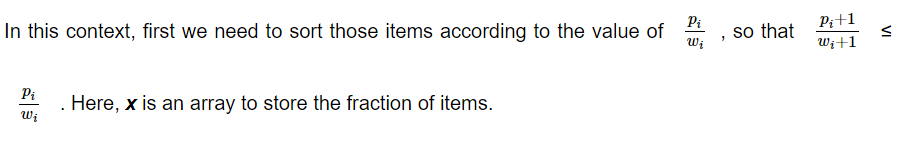
In this version of Knapsack problem, items can be broken into smaller pieces. So, the thief may take only a fraction *xi* of ith item.

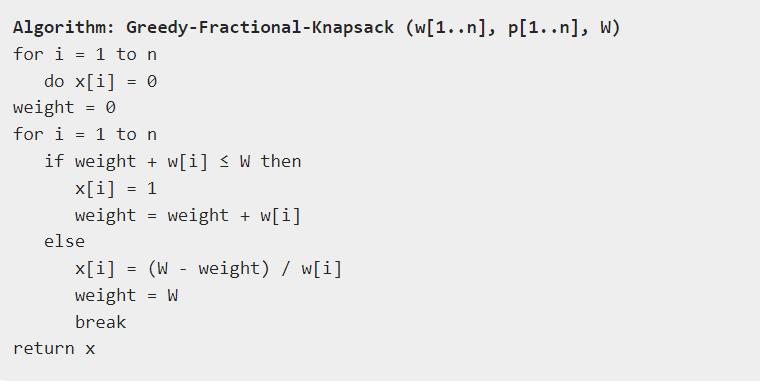
0⩽xi⩽1

The ith item contributes the weight xi.wi to the total weight in the knapsack and profit xi.pi to the total profit**.**

****

**Algorithm:**

****



**Implementation:**

#include <stdio.h>

int max(int a, int b) { return (a > b) ? a : b; }

int knapSack(int W, int wt[], int val[], int n)

{

// Base Case

if (n == 0 || W == 0)

return 0;

// If weight of the nth item is more than

// Knapsack capacity W, then this item cannot

// be included in the optimal solution

if (wt[n - 1] > W)

return knapSack(W, wt, val, n - 1);

// Return the maximum of two cases:

// (1) nth item included

// (2) not included

else

return max(

val[n - 1]

+ knapSack(W - wt[n - 1], wt, val, n - 1),

knapSack(W, wt, val, n - 1));

}

// Driver code

int main()

{

int profit[] = { 60, 100, 120 };

int weight[] = { 10, 20, 30 };

int W = 50;

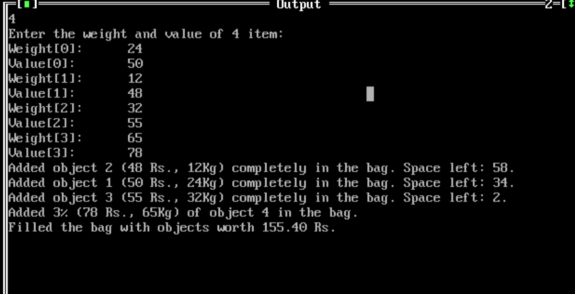
int n = sizeof(profit) / sizeof(profit[0]);

printf("%d", knapSack(W, weight, profit, n));

return 0;

}

**Output:**



**Conclusion:** experiment successfully implemented the fractional knapsack algorithm, efficiently allocating items based on their values and weights. By prioritizing fractional solutions, we optimized resource utilization, demonstrating the algorithm's practicality and effectiveness in real-world scenarios.