EXPERIMENT 4

PRAKASH P. BISWAS

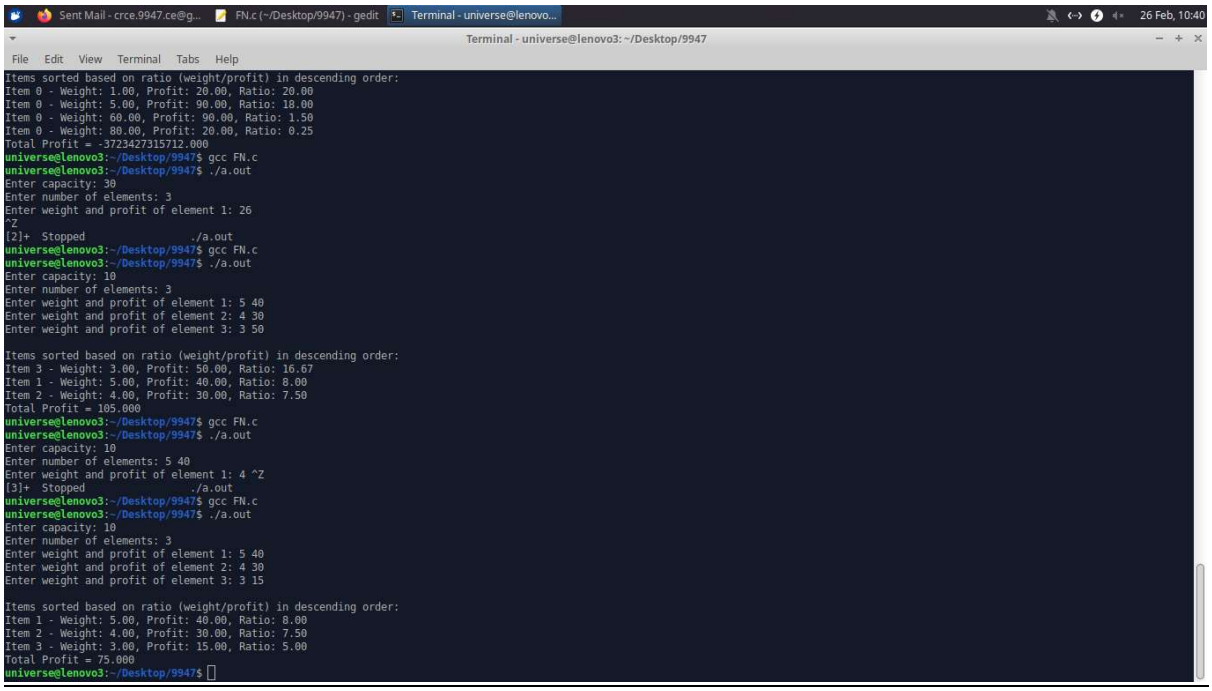
9947

COMPS-B

```c
#include <stdio.h>
typedef struct {
    int no;
    float weight, profit, ratio;
} Item;
void swap(Item *a, Item *b) {
    Item temp = *a;
    *a = *b;
    *b = temp;
}
int main() {
    int n, K;
    float tp = 0.0, cap;
    printf("Enter capacity: ");
    scanf("%f", &cap);
    printf("Enter number of elements: ");
    scanf("%d", &n);
    Item I[n];
    for (int i = 0; i < n; i++) {
        printf("Enter weight and profit of element %d: ", i + 1);
        scanf("%f %f", &I[i].weight, &I[i].profit);
        I[i].ratio = I[i].profit / I[i].weight;
        I[i].no = i + 1;
    }
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (I[j].ratio < I[j + 1].ratio) {
                swap(&I[j], &I[j + 1]);
            }
        }
    }
    printf("\nItems sorted based on ratio (weight/profit) in descending
order:\n");
    for (int i = 0; i < n; i++) {
        printf("Item %d - Weight: %.2f, Profit: %.2f, Ratio: %.2f\n", I[i].no,
I[i].weight, I[i].profit, I[i].ratio);
    }
    K = 0;
    while (cap > 0 && K < n) {
        if (I[K].weight <= cap) {
            I[K].ratio = 1;
            cap = cap - I[K].weight;
```

```
            tp = tp + I[K].ratio * I[K].profit;
            K++;
        } else {
            I[K].ratio = cap / I[K].weight;
            tp = tp + I[K].ratio * I[K].profit;
            cap = 0;
            break;
        }
    }
    printf("Total Profit = %.3f\n", tp);
    return 0;
}
```

OUTPUT:



POSTLAB:

9947
COMPS-B

The greedy strategy is a problem-solving approach that makes locally optimal choices at each stage of a problem with the hope of finding a global optimum. In other words, at each step, the algorithm makes the best possible decision given the current information, without considering the future consequences or looking ahead. The strategy is called "greedy" because it makes choices that seem best at the moment, hoping that these choices will lead to an overall optimal solution.

Here are the key characteristics of a greedy strategy in problem-solving:

Greedy Choice Property:

At each step, the algorithm makes the choice that appears to be the best at that particular moment.

This choice is often determined based on some locally optimal criteria.

Optimal Substructure:

A problem exhibits optimal substructure if an optimal solution to the overall problem can be constructed from optimal solutions of its subproblems.

Greedy algorithms typically work well when a problem has optimal substructure, meaning that the optimal solution to the problem can be built by combining optimal solutions to its subproblems.

No Backtracking:
Greedy algorithms make decisions that are never reconsidered. Once a decision is made, it is final and does not change based on future events or decisions.
Does Not Always Guarantee Global Optimality:
While greedy strategies are intuitive and computationally efficient, they do not always guarantee a globally optimal solution.
A locally optimal choice at each step does not necessarily lead to the best overall solution.
Examples of Greedy Algorithms:
Greedy algorithms are commonly used in a variety of problems, such as the fractional knapsack problem, Dijkstra's shortest path algorithm, Huffman coding, and interval scheduling.
Fractional Knapsack Problem Example:
In the context of the fractional knapsack problem, a greedy strategy involves selecting items based on their value-to-weight ratio in descending order.
The algorithm iteratively adds items to the knapsack, starting with the most valuable items first, until the capacity is reached.
It's important to note that the success of a greedy algorithm depends on the specific problem at hand. While greedy strategies work well for some problems, they may not always provide optimal solutions for all types of problems.