**Question-1**

**b) Show how the algorithm works, step by step, for the given example in Table 1. The maximum weight, V, is 10 pounds. W (whole) indicates the items that cannot be subdivided, and F(fractional) stands for items whose fractional value can be taken**

Input:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Weight (in Pounds) | Cost (in dollar) | Type |
| 1 | 3 | 50 | W |
| 2 | 2 | 25 | W |
| 3 | 4 | 60 | W |
| 4 | 1 | 15 | W |
| 5 | 3 | 40 | W |
| 6 | 2 | 30 | W |
| 7 | 3 | 50 | F |
| 8 | 2 | 30 | F |
| 9 | 4 | 50 | F |
| 10 | 1 | 10 | F |

Maximum weight (V): 10 pounds

**Working:**

1. Begin by solving the 0/1 Knapsack problem for a set of objects, assuming that they cannot be divided. Keep track of the selected objects using a status variable.

2. After filling the knapsack with non-divisible objects, check if there's any remaining weight capacity.

3. For any remaining capacity, apply the Fractional Knapsack algorithm to maximize profit for objects that are divisible (indicated by type 'F'), making sure they haven't been taken in the 0/1 Knapsack phase.

4. Calculate the total profit by summing up the results of the 0/1 Knapsack and Fractional Knapsack steps.

This process combines 0/1 Knapsack with Fractional Knapsack to optimize profit while considering object divisibility.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **i , w** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
| **0** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0** | **0** | **0** | **0** |
| **10** | 0 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 | 10 |
| **4** | 0 | 15 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 | 25 |
| **2** | 0 | 0 | 15 | 25 | 40 | 50 | 50 | 50 | 50 | 50 | 50 |
| **8** | 0 | 15 | 30 | 45 | 55 | 70 | 80 | 80 | 80 | 80 | 80 |
| **6** | 0 | 15 | 30 | 45 | 60 | 75 | 85 | 100 | 110 | 110 | 110 |
| **5** | 0 | 15 | 30 | 45 | 60 | 75 | 85 | 100 | 115 | 125 | 140 |
| **9** | 0 | 15 | 30 | 45 | 60 | 75 | 85 | 100 | 115 | 125 | 140 |
| **7** | 0 | 15 | 30 | 50 | 65 | 80 | 95 | 110 | 125 | 135 | 150 |
| **1** | 0 | 15 | 30 | 50 | 65 | 80 | 100 | 115 | 130 | 145 | 160 |
| **3** | 0 | 15 | 30 | 50 | 65 | 80 | 100 | 115 | 130 | 145 | **160** |

After completing the 0/1 knapsack problem, all available weight has been utilized, making it unnecessary to proceed with a fractional knapsack. The maximum achievable cost remains at 160.

Let us take an example that involves both.

Example-2:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Weight (in Pounds) | Cost (in dollar) | Type |
| 1 | 3 | 50 | W |
| 2 | 2 | 30 | F |
| 3 | 4 | 20 | W |
| 4 | 5 | 40 | F |

Max weight = 6

In the provided input, when solving the 0/1 knapsack problem, we can fill the knapsack up to a weight of 5 pounds and achieve a total profit of 80 dollars by selecting items 1 and 2. However, there is still 1 pound of capacity remaining in the knapsack. To maximize the profit further, we can consider fractional knapsack principles and include a fraction of an item with the highest cost-to-weight ratio that has not been taken. In this case, item 4 has the highest cost-to-weight ratio, so we take 1 pound of item 4, which yields a profit of 8 dollars. As a result, the combined profit is the sum of the profits obtained from both the 0/1 knapsack (80 dollars) and the fractional knapsack (8 dollars), totaling 88 dollars.

**c) Prove the correctness of your algorithm.**

Proof:

1. The dynamic programming table dp[n][W] stores the optimal value for subproblems from 0 to n items and 0 to W weight limit. This is filled correctly based on the recursive optimal substructure property of knapsack.

2. After filling the DP table, the solution is reconstructed by tracing back from dp[n][W] to pick items that were selected in the optimal solution. This picks the maximum value set of whole items that fit within the weight limit.

3. The remaining weight capacity after picking whole items is correctly calculated.

4. The fractional items are sorted by value/weight ratio and inserted greedily based on this priority to maximize value picked while filling the remaining capacity. Fractional knapsack is proven to be optimal for this continuous relaxation.

5. The final output value is the sum of:

- Optimal value dp[n][W] achieved using whole items

- Optimal value achieved using fractional items for remaining capacity

6. Therefore, the final output is the maximum possible value that can be achieved using the given items within the weight limit. It combines optimal solutions for 0/1 knapsack (whole items) and fractional knapsack (fractional items) components.

7. By mathematical induction on the subproblems and greedy choice property, this combined approach yields the global optimal solution.

Thus, the algorithm is proven to return the maximum value packed within the weight limit.

**(d)  Compute the complexity of your algorithm.**

The time complexity of this algorithm is O(nW) where n is the number of items and W is the weight capacity.

Here is the analysis:

1. Sorting the items takes O(nlogn) time.

2. The dynamic programming array dp[n+1][W+1] is filled by nested loops from 0 to n and 0 to W. This takes O(nW) time.

3. Reconstructing the solution by tracing back the dp array takes O(nW) time in the worst case (when all items are selected).

4. For fractional items, sorting takes O(nlogn) time and the greedy algorithm takes O(n) time.

So overall, the asymptotic time complexity is dominated by the dynamic programming which takes O(nW) time.

The sorting steps take O(nlogn) which is smaller than O(nW) for n >= W.

So in summary, the overall time complexity is O(nW) where n is the number of items and W is the weight capacity. This is pseudopolynomial due to the dependence on W.

**(e)  Let the weight of all the items be W and the cost of all the items be C. Let n=m=K/2. Let the limit of the weight be V = (K/2)∗W. Prove that in this case, taking any subset of half the items will provide the maximum value.**

Proof:

The maximum weight that can be taken is V = (K/2) \* W.

According to our code, first, we need to consider all the items as in-divisible and solve the problem.

That means this problem becomes 0/1 Knapsack at the beginning. Taking K/2 items means taking exactly half of the total items. Since all the items have equal weight and equal profit, an item can be taken.

After picking exactly half items, the weight of the bag will become (K/2) \* W which is the capacity of the bag. No more items can be added.

Hence it is proved that taking any subset containing K/2 items out of K items will give a maximum profit of (K/2) \* C.

Proof-2:

This is a 0/1 knapsack problem initially where there are K identical items each with weight W and profit W. The knapsack capacity is V.

Since all items are identical, the optimal solution is to take the maximum number of items that fit in the knapsack. This will maximize both weight and profit.

The knapsack capacity V dictates the maximum number of items that can be taken. Specifically, the capacity V equals (the maximum number of items \* W).

Solving this:

V = (maximum number of items) \* W

V/W = maximum number of items

V/W = K/2 (since W is the same for each item)

Therefore, the maximum number of items that fit in the knapsack is K/2. Taking K/2 items maximizes both the weight (which is K/2 \* W) and profit (which is also K/2 \* W since weight = profit per item).

So in summary, the maximum weight that can be taken is V = (K/2) \* W.