DAA LAB-05:

(A) FRACTIONAL KNAPSACK

(1.) Algorithm

// This function calculates p/w ratio for each item
// Input : Dataset array of form [val, wt, shelf life]
// Output : None.

```
def prow_ratio (items) :
    • for item in items :
        val, wt, shelflife = item[0], item[1], item[2]

        If wt != 0 :
            ratio = val/wt
        else
            ratio = 0
        item.append (ratio)

    • items.sort( key=lambda item : (item[2], -item[-1]))
```

We will sort the items based on shelf life first and then price/weight ratio.

PAGE NO.:
DATE / /

d

```
// This function calculates benefit by filling knapsack.
// Input: Dataset array and capacity of knapsack.
// Output: Max Benefit

def benefit (items[], weight):
    → benefit = 0

    → for ind, item in enumerate(items):
        If item[1] <= weight :
            weight -= item[1]
            benefit += item[1] * item[-1]

        else :
            benefit += weight * item[-1]
            weight = 0

        if weight == 0 :
            break.

    → return benefit.
```

(2.) Testcases.

Positive TCs:

Different Testcases with valid value of all fields in input.

Display maximum benefit along with the items added in the knapsack.

Negative TCs:

1. Negative Price.

For eg. items[3] = [-100, 25, 10]

Display ⇒ Price can't be negative

2. Weight negative

For eg. items[7] = [1000, -3, 4]

Display ⇒ weight can't be negative

3. Negative shelf life.

For eg. items[6] = [400, 8, -19]

Display ⇒ shelf life can't be negative.

4. Dataset Empty

For eg. items = []

Display ⇒ Dataset is empty

PAGE NO.:
DATE / /

3. Time Complexity.

BRUTE FORCE:

To solve by brute force we need to consider all the subsets of items. We know the number of subsets of a set = 2^n.

However since this is fractional knapsack, we can take fractions of items.

∴ Time complexity = $O(K^n)$ where, K is the number of fractions of an item

GREEDY:

By solving by greedy approach we have 3 functions in the program:

check() :- Traverses through the array once and checks if all values are valid.

∴ Time Complexity - $O(n)$

prow-ratio() :- Iterates through the array and calculates p/w ratio → $O(n)$

Sorts the array based on shelf life and p/w ratio → $O(n \log n)$

Benefit() :- Iterates through the array only once → $O(n)$

∴ Tc = $n + n\log n + n + n = 3n + n\log n$

∴ $\boxed{Tc = O(n \log n)}$