EXP1 Aim :- Consider a XYZ courier company. They reave different goods to transport to different cities. Company needs to ship the goods based on their life and value. Goods having less shelf life and high cost shall be shifted earlier. Consider list of 100 such items and capacity of transport vehicle is 200 tones. Implement Algorithm for fractional knapsack problem

Algorithm :-

1) Brute Force

```
Func^n brute_force (item, capacity)
    Set Max_value → 0
  for each combination of items
    {
      SET total_wt → 0
      SET total_value → 0
    }
  for each item in combination
    if item can add fully
    {
      total_wt + = item - wt
      total - value + = item - value
    }
  else
      frac - value = (item - value / item - wt) * remaing
      total_value + = fractional - value
      total_wt + = capacity
if total_wt <= capacity & total value > max-value
    max_value = total_value
return max-value
```

2) Fractional Knapsack.

// input : list of items object (wt, ID, ~~self life~~, value)
: capacity of knapsack
// output : max_value accomodations in knapsack.
: list of selected items.

func" fractional_knapsack (items, capacity)
Sorts items by ratio → v/w * (1/s-life) ~~→ efficiency~~ // O(n log
total_value → 0
selected items = [] // empty list.

for each item → items :
if capacity <=0
break; // knapsack is full.
if item_wt <= capacity.
total_value += item_value
[add item detail → selected_item[]]
capacity_item_wt
else
frac = capacity / item_wt.
face_value = item_value * face
total_value += face_value
[Add the frac item details to selection]
capacity 20 // full knapsack.
return. total_value, selected item.

★ Time complexity

1) Brute force

outer loop :- The no of combinations of n itemes (2^n)

inner loop :- The no of combinations of it runs for (n)

$T(n) = O(n2^n)$.

2) Fractional Knapsack

Generate all possible subset of given items

∴ Generation of subset = $O(2^n)$

For each subset algorithm runs for subset size $O(n)$

Ie $T(n) = O(n \cdot 2^n)$

2) Fractional Knapsack

Creating vector pairs = $O(n)$

Sorting created vector = $O(n \log n)$

Calculating answer using greedy = $O(n)$

$T(n) = O(n \log n + 2n)$

$= O(n \log n)$.

★ Test Cases

1) I/P : Total weight of all items

O/P : Sum of cost of all items.

2) I/P : Multiple item with same cost to life ratio.

O/P : Prioritize based on order in list

3) I/P : Provide an empty list of item

O/P : total value = 0.

4) I/P : set transport vehicle capacity to 0

O/P : total_value = 0

* Algorithm - Huffmann Coding

1) Calculate frequency
For each character in input. If character is in frequency-table increment frequency of character in frequency table

2) else
Add character to frequency table with frequency = 1

2) Initialize Priority queue
For each character & frequency in Frequency-table create a node with character & frequency. Add node to priority queue.

3) Build the huffmann tree
While (priority-queue has more than one node)
left-node = remove node with lowest frequency from priority queue.
right-node = remove node with next lowest frequency from priority queue.

Create new-node with :
frequency = left node.frequency + right node frequency
left = left-node
right = right-node.

Add new-node to priority queue
End while

```
Generate code (node, current-code)
{
    If node is NULL
        return
    else If node is a leaf (node character is not NULL)
        SET huffman-code (node.character) = current code

    else
        generate-code (node.left, current-code + "0")
        generate-code (node.right, current-code + "1")
```

* Time Complexity

① Brute force

- Generate binary string of lenght $k = 2^n$
- For each character 'm' total no. of subset will be,

$$\sum_{k=1}^{m} 2^k = 2^{m+1} - 2 \approx O(2^m)$$

comparing two code will invole nested loops adding n^2

- Also, for each ~~student~~ subset, we have $O(k^2)$ checks thus,

$$T.C = O\left(\sum_{k=1}^{m} \binom{m}{k} k^2\right)$$

largest check will occour at $m=k$. Thus $TC = O(2^n, n^2)$

② Greedy Approach

- Counting frequency of every letter using O(n)
- To build priority queue from each unique character K, will have

$$O(K \log K)$$

for merge, the loop run (K-1) times,

$$O(K)$$

overall

$$T.C = O(K \log K)$$

* Test Cases.

File Type	Original Text	Compressed
TxT	1016	441
DOCX	50216	21390
HTML	50480	21433
Pdf	2680	526
md	error - type not supported	
PDF (empty)	no-content extracted	
TxT (non existing)	no file	

Comprises ratio

0.43

0.43

0.42

0.43