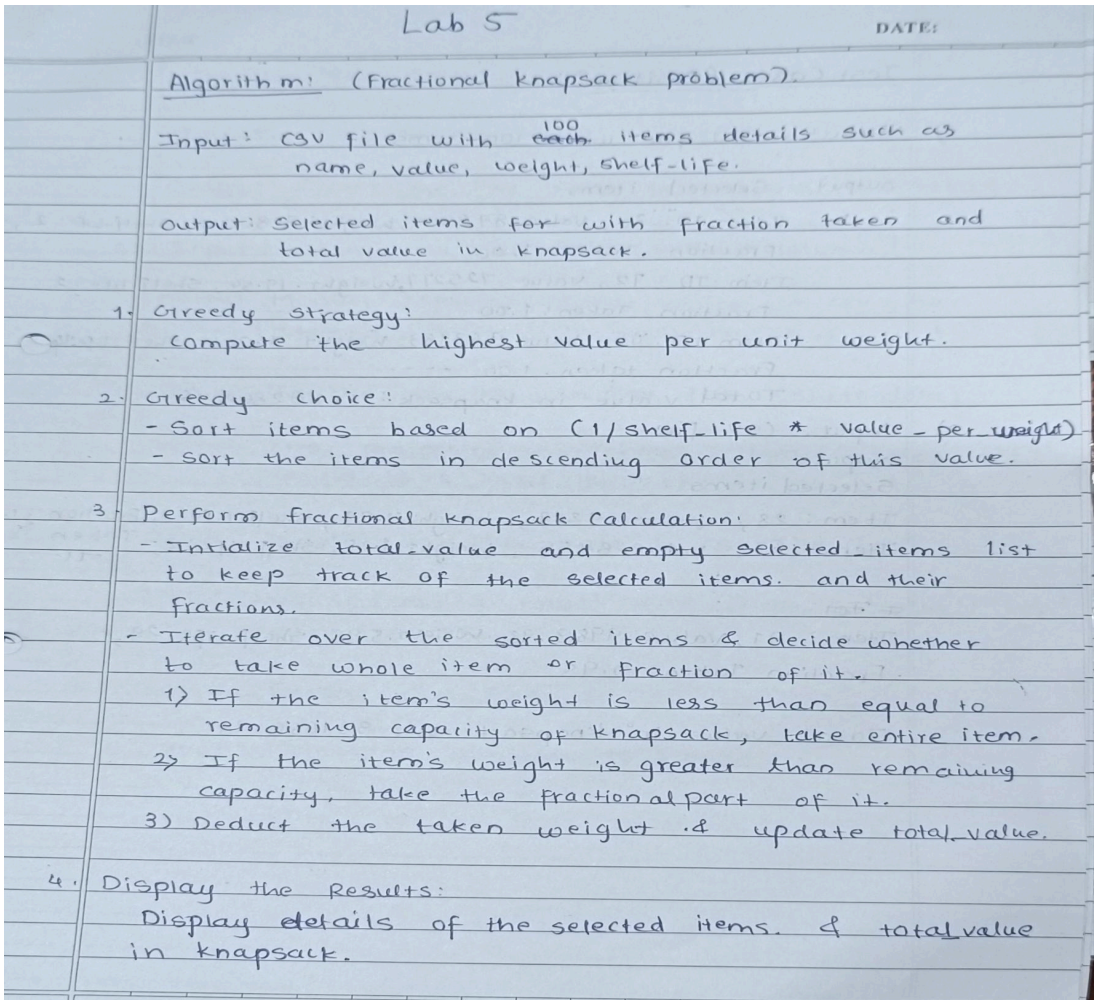
Name: Bhakti Anil Baraf
ID: 241071407
B.Tech SY computer engineering

Experiment 5

Experiment task-1:
Consider a XYZ courier company. They receive 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 shipped earlier. Consider a list of 100 such items and the capacity of a transport vehicle is 200 tons. Implement Algorithms for fractional knapsack problems.

Algorithm:



Lab 5 DATE:

Algorithm: (Fractional knapsack problem).

Input: CSV file with 100 items details such as name, value, weight, shelf-life.

Output: Selected items for with fraction taken and total value in knapsack.

1. Greedy strategy:
   Compute the highest value per unit weight.

2. Greedy choice:
   - Sort items based on (1/shelf_life * value_per_weight)
   - sort the items in descending order of this value.

3. Perform fractional knapsack calculation:
   - Intialize total_value and empty selected_items list to keep track of the selected items. and their fractions.
   - Iterate over the sorted items & decide whether to take whole item or fraction of it.
     1) If the item's weight is less than equal to remaining capacity of knapsack, take entire item.
     2) If the item's weight is greater than remaining capacity, take the fractional part of it.
     3) Deduct the taken weight & update total_value.

4. Display the Results:
   Display details of the selected items. & total value in knapsack.

Fractional knapsack. DATE:

Pseudocode:

```
class item:
    def __init__(name, value, weight, shelf_life)
        self.name = name
        self.value = value
        self.weight = weight
        Self.shelf_life = shelf_life
        self.value_per_weight = value/weight
        if weight != 0
        else ~~weight~~ 0

def read_items_from_csv(file_path):
    items = []
    try:
        open file_path as csvfile
        create reader from csv.DictReader(csvfile)
        for row in reader:
            try:
                name = row['name']
                value = convert_to_float(row['value'])
                weight = float(row['weight'])
                shelf_life = ~~float~~ int(row['shelf_life'])
                append name, value, weight, shelf_life to items
            except ValueError:
                print "Error: Invalid data in row"
    except FileNotFoundError:
        print "File Not Found"
    return items
```

DATE:

```
def fractional_knapsack(items, capacity):
    try:
        if capacity <= 0:
            raise ValueError
        Sort items by (1/shelf_life) * value_per_weight
        in descending order.
        total_value = 0
        selected_items = []
        for item in items:
            if capacity <= 0:
                break
            if item.weight <= capacity:
                total_value += item.value
                capacity -= item.weight
                append (item, 1) to selected_items
            else:
                fraction = capacity / item.weight
                total_value += item.value * fraction
                append (item, fraction) to selected_items
                capacity = 0
        print "Selected items:"
        for item, fraction in selected_items:
            print item.name, item.value, item.weight,
                item.shelf_life, fraction
        print "Total value in knapsack: ", total_value
        return total_value,
    except ValueError :
        print "Error".
```

DATE:

```
def main():
    file_path = 'items.csv'
    items = read_items_from_csv(file_path)
    if items is not empty:
        set capacity = 200
        fractional_knapsack (items, capacity)
    else:
        print "No items to process"
```

Code:

```
import csv

class Item:
    def __init__(self, name, value, weight, shelf_life):
        self.name = name
        self.value = value
        self.weight = weight
        self.shelf_life = shelf_life
        self.value_per_weight = value / weight if weight != 0 else 0

# Function to perform the fractional knapsack algorithm
def fractional_knapsack(items, capacity):
    try:
        if capacity <= 0:
            raise ValueError("The capacity of the knapsack must be greater than zero.")

        # Sort items by the ratio of (1 / shelf_life) * value_per_weight in descending
order
        items.sort(key=lambda x: (1 / x.shelf_life) * x.value_per_weight,
reverse=True)

        total_value = 0.0
        selected_items = []  # List to store selected items and fractions

        for item in items:
            if capacity <= 0:
                break

            # Check if the whole item can be taken
```

```
            if item.weight <= capacity:
                total_value += item.value
                capacity -= item.weight
                selected_items.append((item, 1))  # 1 indicates full item taken
            else:
                # Take the fractional part of the item
                fraction = capacity / item.weight
                total_value += item.value * fraction
                selected_items.append((item, fraction))  # Store item with fraction taken
                capacity = 0

        # Display all data of selected items
        print("\nSelected items:")
        for item, fraction in selected_items:
            print(f"Item: {item.name}, Value: {item.value}, Weight: {item.weight},
Shelf Life: {item.shelf_life}, Fraction Taken: {fraction:.2f}")

        print(f"\nTotal value in knapsack: {total_value:.2f}")
        return total_value

    except ValueError as e:
        print(f"Error: {e}")
    except Exception as e:
        print(f"Unexpected error: {e}")

# Function to read items from a CSV file
def read_items_from_csv(file_path):
    items = []
    try:
```

```
        with open(file_path, newline='') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                try:
                    name = row['name']
                    value = float(row['value'])
                    weight = float(row['weight'])
                    shelf_life = int(row['shelf_life'])
                    items.append(Item(name, value, weight, shelf_life))
                except ValueError as e:
                    print(f"Error processing row {row}: {e}")
                except KeyError as e:
                    print(f"Missing expected column {e} in CSV row: {row}")
    except FileNotFoundError:
        print(f"Error: The file '{file_path}' was not found.")
    except Exception as e:
        print(f"Unexpected error reading the file: {e}")
    return items

# Path to the CSV file (adjust the path as needed)
csv_file_path = 'items.csv'

# Read items from the CSV file
items = read_items_from_csv(csv_file_path)

if items:  # Proceed only if items were successfully read
    # Capacity of the transport vehicle
    capacity = 200  # in tonnes
```

```
    # Execute the algorithm
    fractional_knapsack(items, capacity)
else:
    print("No items available to process.")
```

Output:

```
Selected items:
Item ID: 20, Value: 878565.9598471712, Weight: 7.860125762871064, Shelf Life: 2, Fraction Taken: 1.00
Item ID: 72, Value: 725227.5941088903, Weight: 19.544131940933987, Shelf Life: 2, Fraction Taken: 1.00
Item ID: 85, Value: 963983.7773183603, Weight: 7.316543806249521, Shelf Life: 14, Fraction Taken: 1.00
Item ID: 75, Value: 649017.5729480961, Weight: 21.363332107068228, Shelf Life: 4, Fraction Taken: 1.00
Item ID: 16, Value: 146145.73470453333, Weight: 10.214107678630837, Shelf Life: 2, Fraction Taken: 1.00
Item ID: 58, Value: 572625.5173021168, Weight: 45.324108496557194, Shelf Life: 2, Fraction Taken: 1.00
Item ID: 13, Value: 694028.3607214413, Weight: 39.70716560086756, Shelf Life: 3, Fraction Taken: 1.00
Item ID: 44, Value: 666866.7514169753, Weight: 38.99980123443719, Shelf Life: 3, Fraction Taken: 1.00
Item ID: 51, Value: 715057.4291340605, Weight: 33.503169042969056, Shelf Life: 4, Fraction Taken: 0.29

Total value in knapsack: 5502862.46
```

```
Item: Item_54, Value: 9528.0, Weight: 6.03, Shelf Life: 22, Fraction Taken: 1.00
Item: Item_37, Value: 7428.55, Weight: 6.96, Shelf Life: 13, Fraction Taken: 1.00
Item: Item_45, Value: 2135.51, Weight: 1.12, Shelf Life: 25, Fraction Taken: 1.00
Item: Item_37, Value: 7428.55, Weight: 6.96, Shelf Life: 13, Fraction Taken: 1.00
Item: Item_37, Value: 7428.55, Weight: 6.96, Shelf Life: 13, Fraction Taken: 1.00
Item: Item_45, Value: 2135.51, Weight: 1.12, Shelf Life: 25, Fraction Taken: 1.00
Item: Item_52, Value: 5797.1, Weight: 5.21, Shelf Life: 15, Fraction Taken: 1.00
Item: Item_54, Value: 9528.0, Weight: 6.03, Shelf Life: 22, Fraction Taken: 1.00
Item: Item_70, Value: 9043.79, Weight: 4.25, Shelf Life: 30, Fraction Taken: 1.00
Item: Item_10, Value: 3367.2, Weight: 4.31, Shelf Life: 12, Fraction Taken: 1.00
Item: Item_30, Value: 7504.44, Weight: 7.3, Shelf Life: 16, Fraction Taken: 1.00
Item: Item_63, Value: 4589.94, Weight: 3.59, Shelf Life: 21, Fraction Taken: 1.00
Item: Item_31, Value: 9983.98, Weight: 5.98, Shelf Life: 29, Fraction Taken: 0.95

Total value in knapsack: 331660.18
```

Test Cases:

1. Positive Test Cases:

```
Item: Item_54, Value: 9528.0, Weight: 6.03, Shelf Life: 22, Fraction Taken: 1.00
Item: Item_37, Value: 7428.55, Weight: 6.96, Shelf Life: 13, Fraction Taken: 1.00
Item: Item_45, Value: 2135.51, Weight: 1.12, Shelf Life: 25, Fraction Taken: 1.00
Item: Item_37, Value: 7428.55, Weight: 6.96, Shelf Life: 13, Fraction Taken: 1.00
Item: Item_37, Value: 7428.55, Weight: 6.96, Shelf Life: 13, Fraction Taken: 1.00
Item: Item_45, Value: 2135.51, Weight: 1.12, Shelf Life: 25, Fraction Taken: 1.00
Item: Item_52, Value: 5797.1, Weight: 5.21, Shelf Life: 15, Fraction Taken: 1.00
Item: Item_54, Value: 9528.0, Weight: 6.03, Shelf Life: 22, Fraction Taken: 1.00
Item: Item_70, Value: 9043.79, Weight: 4.25, Shelf Life: 30, Fraction Taken: 1.00
Item: Item_10, Value: 3367.2, Weight: 4.31, Shelf Life: 12, Fraction Taken: 1.00
Item: Item_30, Value: 7504.44, Weight: 7.3, Shelf Life: 16, Fraction Taken: 1.00
Item: Item_63, Value: 4589.94, Weight: 3.59, Shelf Life: 21, Fraction Taken: 1.00
Item: Item_31, Value: 9983.98, Weight: 5.98, Shelf Life: 29, Fraction Taken: 0.95

Total value in knapsack: 331660.18
```

DATE:

Test Cases (Positive):

1) Input : CSV file with 100 items.

output : Selected items:

Item ID: 20, Value: 878565, Weight: 7.86012, Shelf-Life: 2,
Fraction Taken: 1.00.

Item ID : 72, Value: 725277, Weight: 19.54, Shelf-life: 2,
Fraction Taken: 1.00

Item ID: 85, Value: 963983, Weight: 7.3165, Shelf-life: 14
Fraction taken: 1.00 ----.

Total value in knapsack : 5502862.46.

2) Input. CSV file.

output:

Selected items:

Item: 28, Value: 9263, Weight: 1.99, Shelf-Life: 1, Fraction : 1.00
Item: 67, Value: 8455, Weight: 1.25, Shelf-life: 2, Taken

:

~~#~~ Item :

Item: 31, Value: 9983.98, Weight: 5.98, Shelf-Life: 29,
Fraction Taken: 0.95

Total value in knapsack is : 331660.18

```
Selected items:
Item ID: 20, Value: 878565.9598471712, Weight: 7.860125762871064, Shelf Life: 2, Fraction Taken: 1.00
Item ID: 72, Value: 725227.5941088903, Weight: 19.544131940933987, Shelf Life: 2, Fraction Taken: 1.00
Item ID: 85, Value: 963983.7773183603, Weight: 7.316543806249521, Shelf Life: 14, Fraction Taken: 1.00
Item ID: 75, Value: 649017.5729480961, Weight: 21.363332107068228, Shelf Life: 4, Fraction Taken: 1.00
Item ID: 16, Value: 146145.73470453333, Weight: 10.214107678630837, Shelf Life: 2, Fraction Taken: 1.00
Item ID: 58, Value: 572625.5173021168, Weight: 45.324108496557194, Shelf Life: 2, Fraction Taken: 1.00
Item ID: 13, Value: 694028.3607214413, Weight: 39.70716560086756, Shelf Life: 3, Fraction Taken: 1.00
Item ID: 44, Value: 666866.7514169753, Weight: 38.99980123443719, Shelf Life: 3, Fraction Taken: 1.00
Item ID: 51, Value: 715057.4291340605, Weight: 33.503169042969056, Shelf Life: 4, Fraction Taken: 0.29

Total value in knapsack: 5502862.46
```

```
Selected items:
Item: Item_28, Value: 9263.65, Weight: 1.99, Shelf Life: 1, Fraction Taken: 1.00
Item: Item_67, Value: 8455.57, Weight: 1.25, Shelf Life: 2, Fraction Taken: 1.00
Item: Item_72, Value: 9097.55, Weight: 1.29, Shelf Life: 4, Fraction Taken: 1.00
Item: Item_66, Value: 6633.94, Weight: 4.11, Shelf Life: 2, Fraction Taken: 1.00
Item: Item_60, Value: 4763.17, Weight: 7.55, Shelf Life: 1, Fraction Taken: 1.00
Item: Item_51, Value: 8329.48, Weight: 1.23, Shelf Life: 11, Fraction Taken: 1.00
Item: Item_58, Value: 5576.15, Weight: 9.13, Shelf Life: 1, Fraction Taken: 1.00
Item: Item_50, Value: 9146.7, Weight: 1.68, Shelf Life: 9, Fraction Taken: 1.00
Item: Item_85, Value: 8017.07, Weight: 1.71, Shelf Life: 10, Fraction Taken: 1.00
Item: Item_13, Value: 9719.55, Weight: 5.25, Shelf Life: 4, Fraction Taken: 1.00
Item: Item_20, Value: 9270.26, Weight: 1.24, Shelf Life: 17, Fraction Taken: 1.00
Item: Item_38, Value: 5847.7, Weight: 1.32, Shelf Life: 12, Fraction Taken: 1.00
Item: Item_57, Value: 7056.57, Weight: 5.25, Shelf Life: 4, Fraction Taken: 1.00
Item: Item_41, Value: 9582.11, Weight: 3.17, Shelf Life: 10, Fraction Taken: 1.00
Item: Item_92, Value: 7147.43, Weight: 8.03, Shelf Life: 3, Fraction Taken: 1.00
Item: Item_88, Value: 5586.28, Weight: 9.52, Shelf Life: 2, Fraction Taken: 1.00
Item: Item_25, Value: 6784.73, Weight: 1.67, Shelf Life: 18, Fraction Taken: 1.00
Item: Item_93, Value: 8216.16, Weight: 3.47, Shelf Life: 11, Fraction Taken: 1.00
Item: Item_33, Value: 4871.96, Weight: 5.77, Shelf Life: 4, Fraction Taken: 1.00
```

2. Negative Test Cases:

```
PS C:\Users\Bhakti\Documents\Python> & c:/Users/Bhakti/Documents/Python/.venv/Scri
Error: The file 'items_sa.csv' was not found.
No items available to process.
PS C:\Users\Bhakti\Documents\Python>
```

```
PS C:\Users\Bhakti\Documents\Python> & c:/Users/Bhakti/Documents/Python/.venv/Scripts/python.exe c:/Users/Bhakti/Documents/Python/daa_lab5.py
Missing expected column 'name' in CSV row: {'Item_name': 'Item_1', 'value': '1767.97', 'weight': '6.45', 'shelf_life': '28'}
Missing expected column 'name' in CSV row: {'Item_name': 'Item_2', 'value': '7068.19', 'weight': '9.48', 'shelf_life': '24'}
Missing expected column 'name' in CSV row: {'Item_name': 'Item_3', 'value': '1711.49', 'weight': '6.5', 'shelf_life': '19'}
Missing expected column 'name' in CSV row: {'Item_name': 'Item_4', 'value': '9784.67', 'weight': '8.57', 'shelf_life': '29'}
```

```
Total value in knapsack: 331660.18
PS C:\Users\Bhakti\Documents\Python> & c:/Users/Bhakti/Documents/Python/.venv/Scripts/python.exe c:/Users/Bhakti/Documents/Pyt
Error processing row {'name': 'Item_1', 'value': '1767.97', 'weight': '6.45', 'shelf_life': 'abc'}: invalid literal for int()
```

```
PS C:\Users\Bhakti\Documents\Python> & c:/Users/Bhakti/Documents/Python/.venv
No items available to process.
PS C:\Users\Bhakti\Documents\Python>
```

Improve Code Search Error Share Code Link

Negative Test Cases:

1) Input: 'item_sa.csv' file (Invalid file)

Output: Error: The file 'items_sa.csv' was not found.

2) Input: Wrong csv file (file without column)

output: Missing expected column 'name' in csv row:
{'Item': '1', 'Value': '1767', 'weight': 6.45, ... }

3) Input: Invalid row data in csv file. (shelf data)

output: Error processing row {'name': 'item_1', 'value': '1767', 'weight': '6.45', 'shelf_life': 'abc'}.

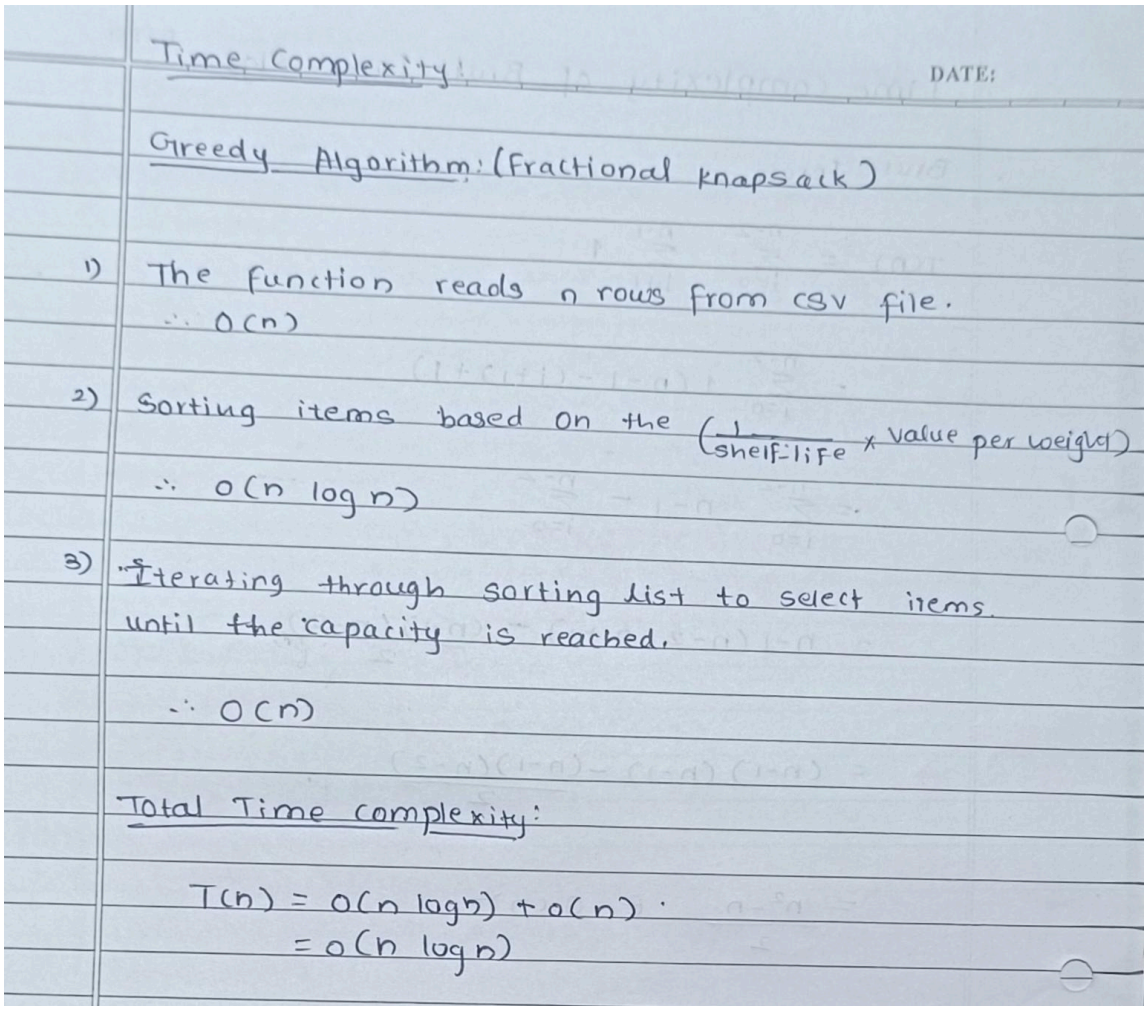
4) Input: Invalid ~~re~~ value data in csv file.

Output: Error processing row {'name': 'item_1', 'value': 'fgh', 'weight': '6.45', 'shelf_life': 2 }.

5) Input: Empty csv file.

Output: No items available to process.

Time Complexity (Fractional Knapsack Problem):



Time Complexity:

DATE:

Greedy Algorithm: (Fractional knapsack)

1) The function reads n rows from csv file.
   ∴ O(n)

2) Sorting items based on the $\left(\frac{1}{\text{shelf-life}} * \text{value per weight}\right)$
   ∴ O(n log n)

3) Iterating through sorting list to select items until the capacity is reached.
   ∴ O(n)

Total Time complexity:

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

Experiment task-2:

Download books from the website in html, text, doc, and pdf format. Compress these books using Hoffman coding technique. Find the compression ratio.

Algorithm:

DATE:

Algorithm: (Huffman code)

1. Input: Text data (~~format~~ In format of file:
   .txt, .html, .docx, .pdf).

   Output: Original Size.
   Compressed Size
   Compressed Ratio.

1. Take input data as a file (.txt, html, docx, pdf)
2. Traverse ~~the count of~~ through the text and count the frequency of each character.
3. Build Huffman tree:
   - Create nodes for each unique character & its frequency.
   - using min-heap (Priority Queue) ensure that the lowest frequency node is at the root.
     Do this for all the nodes.
   - Merge nodes:
     - pop two nodes with lowest frequency.
     - Create new node whose frequency is sum of 2 nodes.
     - The two nodes becomes left & right children of new node.
     - Insert new node back into heap.

DATE:

4. Generate Huffman codes:
   - Traverse from root of Huffman tree.
     - Assign '0' for left branch & '1' for right branch.
     - Assign string as Huffman code for each char at the leaf node.
   - A dictionary mapping each char to its Huffman code.

5. Compress the text:
   - Replace each char in original text with its corresponding Huffman code. i.e. 0's & 1's.

6. Calculate Compression Ratio:
   - Calculate original size (i.e. no. of bits required to represent the text).
     original size = len(text) * 8
   - Calculate compressed size (i.e. length of compressed text in bits).
     compressed size = len(compressed_text)
   - Calculate Ratio.
     Compression Ratio = original_size / compressed_size.

7. Display all the above calculated size.

Huffman coding

DATE:

Pseudocode:

```
class node:
    def __init__(char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None.

    def __lt__(other):
        self.freq < other.freq.

def build_huffman_tree(text):
    freq = defaultdict(int)
    for char in text:
        freq[char] += 1

    heap = [Node(char, f) for char, f in freq.items()]
    heapq.heapify(heap). # convert heap into priority Q.

    while heap size > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = Node(None, left.freq + right.freq)
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)

    return root node of heap.
```

DATE:

```
def generate_codes(node, prefix="", codebook=None):
    if codebook is None:
        codebook = {}
    if node is not None:
        if node.char is not None:
            codebook[node.char] = prefix
        generate_codes(node.left, prefix + "0", codebook)
        generate_codes(node.right, prefix + "1", codebook)
    return codebooks.

def compress(text, codebook):
    return "".join(codebook[char] for char in text)

def calculate_compression_ratio(original_size, compressed_size):
    return original size/compressed_size.

def extract_text_from_pdf(pdf_file):
    extract data using pyPDF2 library.

def extract_text_from_docx(docx_file):
    create instance of docx_file.
    text =
    for para in document.paragraphs:
        text=paragraph.text + "\n"
    return text.

def extract_text_from_html(html_file):
    extract html data using Beautiful Soup library

def extract_text_from_text(txt_file):
    open txt file in text mode as file.
    return file.read()
```


DATE:

```
def main():
    print "Enter file path:"
    read file_path

    if file_path does not exist:
        print "File does not exist"

    file_extension = os.path.splittext(file_path)[1].lower()

    check file_extension and accordingly extract text.
    using defined function. into "text".
    text.
    root = build_huffman_tree(text)
    codebook = generate_codes(root)
    compressed_text = compress(text, codebook)

    original_size = len(text) * 8
    compressed_size = len(compressed_text)
    compression_ratio = calculate_compression_ratio(
                        original_size, compressed_size)
```

Code:

```
import heapq
from collections import defaultdict
import PyPDF2
from docx import Document
import requests
from bs4 import BeautifulSoup
import os

# Class to represent a node in the Huffman Tree
class Node:
    def __init__(self, char, freq):
        self.char = char
        self.freq = freq
        self.left = None
        self.right = None

    # Compare nodes for the priority queue (min-heap)
    def __lt__(self, other):
        return self.freq < other.freq

# Function to build the Huffman Tree
def build_huffman_tree(text):
    # Count the frequency of each character in the text
    freq = defaultdict(int)
    for char in text:
        freq[char] += 1

    # Create a priority queue (min-heap) from the frequencies
    heap = [Node(char, f) for char, f in freq.items()]
```

```
    heapq.heapify(heap)

    # Build the Huffman Tree
    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)

        merged = Node(None, left.freq + right.freq)
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)

    return heap[0]

# Function to generate the Huffman codes
def generate_codes(node, prefix="", codebook=None):
    if codebook is None:
        codebook = {}

    if node is not None:
        if node.char is not None:
            codebook[node.char] = prefix
        generate_codes(node.left, prefix + "0", codebook)
        generate_codes(node.right, prefix + "1", codebook)

    return codebook

# Function to compress the text using the generated Huffman codes
def compress(text, codebook):
```

```
    return ''.join(codebook[char] for char in text)

# Function to calculate the compression ratio
def calculate_compression_ratio(original_size, compressed_size):
    return original_size / compressed_size

# Function to extract text from a PDF file
def extract_text_from_pdf(pdf_file):
    with open(pdf_file, 'rb') as file:
        reader = PyPDF2.PdfReader(file)
        text = ""
        for page in reader.pages:
            text += page.extract_text()
    return text

# Function to extract text from a DOCX file
def extract_text_from_docx(docx_file):
    doc = Document(docx_file)
    text = ""
    for para in doc.paragraphs:
        text += para.text + "\n"
    return text

# Function to extract text from an HTML file
def extract_text_from_html(html_file):
    with open(html_file, 'r', encoding='utf-8') as file:
        soup = BeautifulSoup(file, 'html.parser')
        text = soup.get_text(separator=' ', strip=True)
    return text
```

```
# Function to read a plain text file
def extract_text_from_txt(txt_file):
    with open(txt_file, 'r', encoding='utf-8') as file:
        text = file.read()
    return text

# Main function
def main():
        # Take input file name from user
        file_path = input("Enter the file path (html, txt, docx, or pdf): ").strip()

        # Check if the file exists
        if not os.path.exists(file_path):
            print("The file does not exist.")
            return

        if os.path.getsize(file_path) == 0:
            print("Error: The file is empty.")
            return

        # Step 1: Detect the file type by its extension and extract text accordingly
        file_extension = os.path.splitext(file_path)[1].lower()

        text = ""
        if file_extension == ".txt":
            text = extract_text_from_txt(file_path)
        elif file_extension == ".html":
            text = extract_text_from_html(file_path)
```

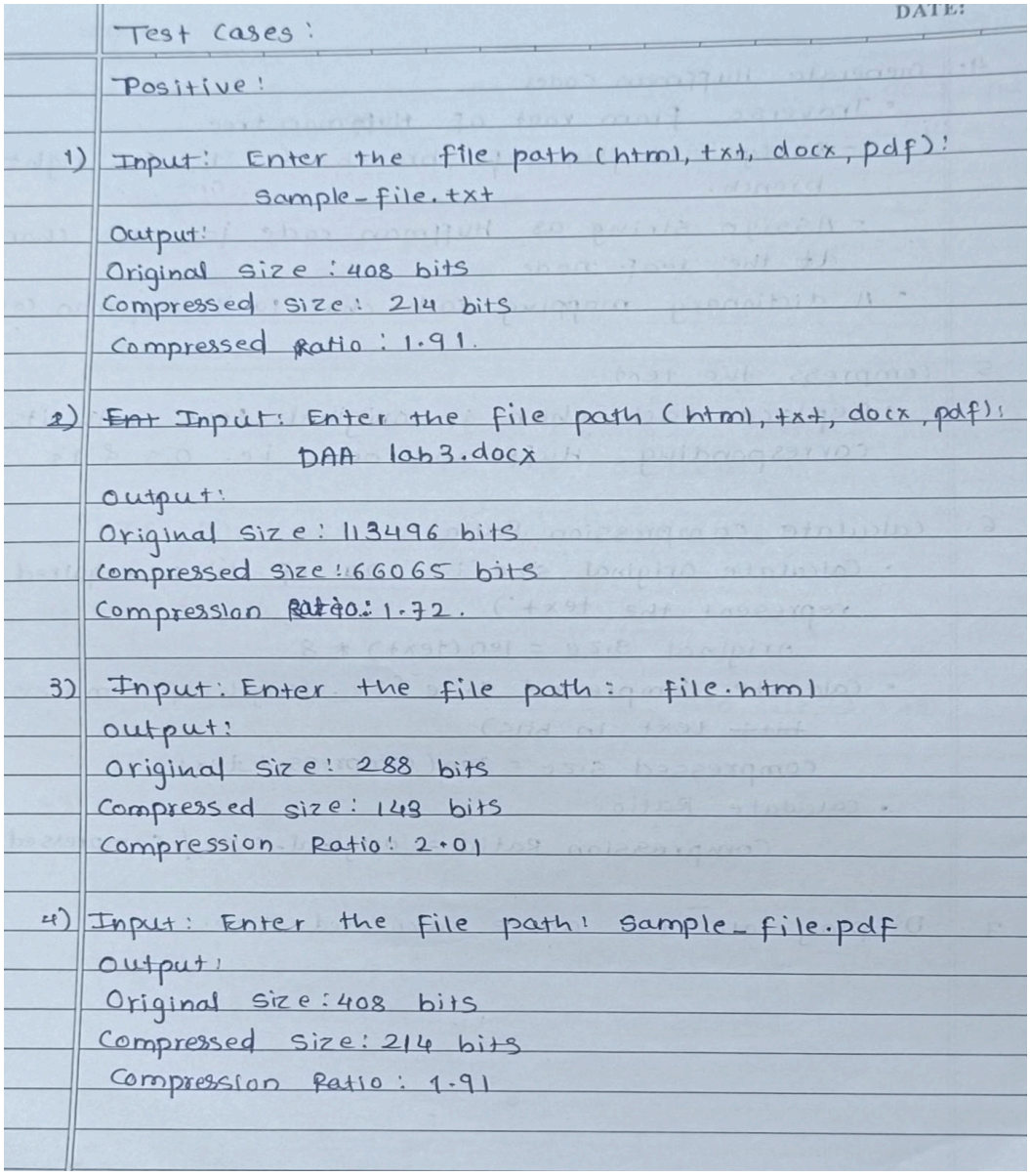
```
    elif file_extension == ".docx":
        text = extract_text_from_docx(file_path)
    elif file_extension == ".pdf":
        text = extract_text_from_pdf(file_path)
    else:
        print("Unsupported file format. Please provide a valid HTML, DOCX, TXT, or PDF file.")
        return


    # Step 2: Apply Huffman coding to compress the text
    root = build_huffman_tree(text)
    codebook = generate_codes(root)
    compressed_text = compress(text, codebook)

    # Step 3: Calculate the compression ratio
    original_size = len(text) * 8  # Original size in bits (assuming 8 bits per character)
    compressed_size = len(compressed_text)  # Compressed size in bits

    compression_ratio = calculate_compression_ratio(original_size, compressed_size)
    print(f"Original Size: {original_size} bits")
    print(f"Compressed Size: {compressed_size} bits")
    print(f"Compression Ratio: {compression_ratio:.2f}")

if __name__ == "__main__":
    main()
```

Output:

```
PS C:\Users\Bhakti\Documents\Python> & c:/Users/Bhakti/Documents/Python/.venv/Scripts/python.exe c:/U
Enter the file path (html, txt, docx, or pdf): book.txt
Original Size: 10048 bits
Compressed Size: 6009 bits
Compression Ratio: 1.67
PS C:\Users\Bhakti\Documents\Python>
```

Test Cases:

1. **Positive Test Cases:**



Test cases:

Positive:

1) Input: Enter the file path (html, txt, docx, pdf):
sample-file.txt
Output:
Original size : 408 bits
Compressed size: 214 bits
Compressed Ratio : 1.91.

2) Input: Enter the file path (html, txt, docx, pdf):
DAA lab3.docx
Output:
Original size: 113496 bits
Compressed size: 66065 bits
Compression Ratio: 1.72.

3) Input: Enter the file path: file.html
output:
Original size: 288 bits
Compressed size: 143 bits
Compression Ratio: 2.01

4) Input: Enter the file path: sample-file.pdf
Output:
Original size: 408 bits
Compressed size: 214 bits
Compression Ratio: 1.91

```
PS C:\Users\Bhakti\Documents\Python> & c:/Users/Bhakti/Documents/Pyt
Enter the file path (html, txt, docx, or pdf): book.txt
Original Size: 10048 bits
Compressed Size: 6009 bits
Compression Ratio: 1.67
PS C:\Users\Bhakti\Documents\Python>
```

```
PS C:\Users\Bhakti\Documents\Python> & c:/Users/Bhakti/Documents/Python/.venv/Scripts/python.exe c:/Us
Enter the file path (html, txt, docx, or pdf): DAA prac.txt
Original Size: 3184 bits
Compressed Size: 1716 bits
Compression Ratio: 1.86
PS C:\Users\Bhakti\Documents\Python>
```

```
PS C:\Users\Bhakti\Documents\Python> & c:/Users/Bhakti/Documents/Python/.venv/
Enter the file path (html, txt, docx, or pdf): file.html
Original Size: 288 bits
Compressed Size: 143 bits
Compression Ratio: 2.01
PS C:\Users\Bhakti\Documents\Python>
```

```
PS C:\Users\Bhakti\Documents\Python> & c:/Users/Bhakti/Documents/Python/.venv/Scripts/python.exe
Enter the file path (html, txt, docx, or pdf): sample_file.txt
Original Size: 408 bits
Compressed Size: 214 bits
Compression Ratio: 1.91
PS C:\Users\Bhakti\Documents\Python>
```

```
Enter the file path (html, txt, docx, or pdf): DAA Lab 3.docx
Original Size: 113496 bits
Compressed Size: 66065 bits
Compression Ratio: 1.72
PS C:\Users\Bhakti\Documents\Python>
```

2. Negative Test Cases:

5) Input: Enter the ~~pa~~ file path (html, txt, docx, pdf):
book.txt
Output:
Original Size: 10048 bits
Compressed Size: 6009 bits
Compression Ratio: 1.67.

Negative Test Cases:

1) Input: Enter the file path (...): sample_file.pdf
Output: Error: the file is empty.
If file doesn't contain any data

2) If File ~~path~~ ~~gi~~ doesn't exist.

Input: Enter the file path (...): ~~sam~~ file23.txt
Output: The file does not Exist.

3) If file given is not supported.

Input: Enter the file path (html, txt, docx, or pdf):
items.csv
Output: Unsupported file format. Please provide a valid HTML, DOCX, TXT or PDF File.

DATE:

4) Input: Enter the file path (html, docx, txt, pdf):
pd_lab4.py
Output:
Unsupported file format. Please provide a valid HTML, DOCX, TXT or PDF file.

5) Input: Enter the file path (...): sample_file
Output: The file doesn't exist.

```
PS C:\Users\Bhakti\Documents\Python> & c:/Users/Bhakti/Documents/
Enter the file path (html, txt, docx, or pdf): sample_file.pdf
Error: The file is empty.
PS C:\Users\Bhakti\Documents\Python>
```

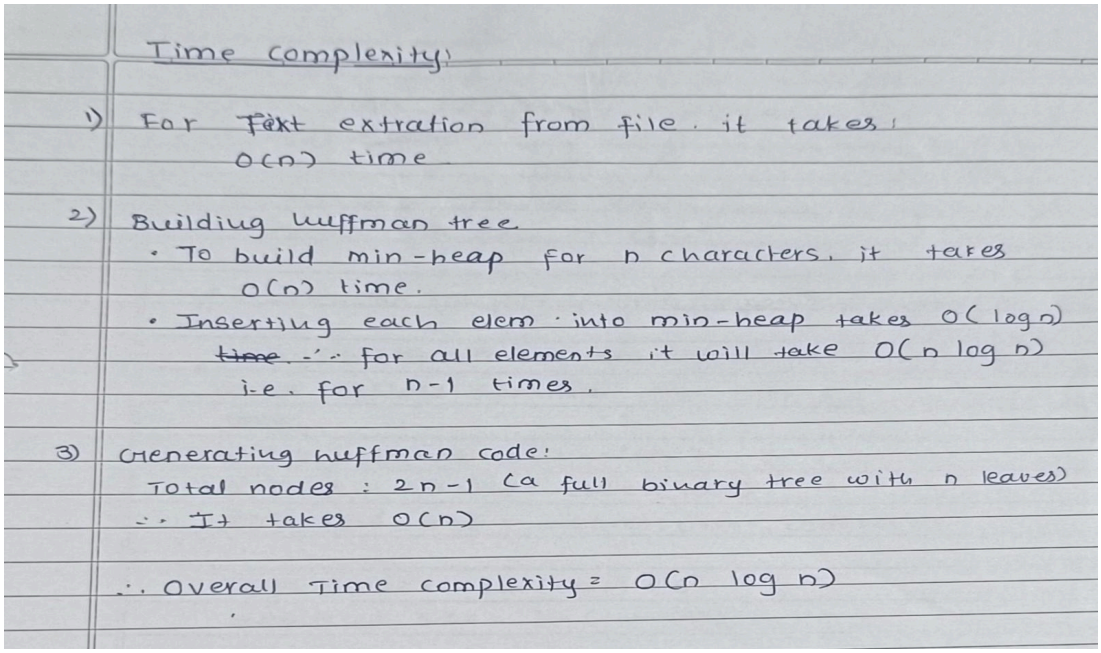
```
PS C:\Users\Bhakti\Documents\Python> & c:/Users/Bhakti/Documents/Python/.venv/Scripts/pyth
Enter the file path (html, txt, docx, or pdf): sample_file.csv
Error: Unsupported file format. Please provide a valid HTML, DOCX, TXT, or PDF file.
PS C:\Users\Bhakti\Documents\Python>
```

```
PS C:\Users\Bhakti\Documents\Python> & c:/Users/Bhakti/Documents/Python/.venv/Scripts/python.ex
Enter the file path (html, txt, docx, or pdf): pd_lab4.py
Unsupported file format. Please provide a valid HTML, DOCX, TXT, or PDF file.
PS C:\Users\Bhakti\Documents\Python>
```

```
PS C:\Users\Bhakti\Documents\Python> & c:/Users/Bhakti/Documents/Python/.venv/Scr
Enter the file path (html, txt, docx, or pdf): sample_file.ppt
Error: The file is empty.
PS C:\Users\Bhakti\Documents\Python>
```

```
PS C:\Users\Bhakti\Documents\Python> & c:/Users/Bhakti/Documents/Python/.venv/Scripts/p
Enter the file path (html, txt, docx, or pdf): sample_file
The file does not exist.
PS C:\Users\Bhakti\Documents\Python>
```

Time Complexity:



Time complexity:

1) For ~~Text~~ extration from file. it takes:
   O(n) time.

2) Building huffman tree.
   - To build min-heap for n characters, it takes O(n) time.
   - Inserting each elem into min-heap takes O(log n)
     ~~time~~. ∴ for all elements it will take O(n log n)
     i.e. for n-1 times.

3) Generating huffman code:
   Total nodes : 2n-1 (a full binary tree with n leaves)
   ∴ It takes O(n)

∴ Overall Time complexity = O(n log n)

Conclusion:

I learned about Fractional Knapsack Algorithms and Huffman coding.
How to implement fractional knapsack algo and Huffman coding.