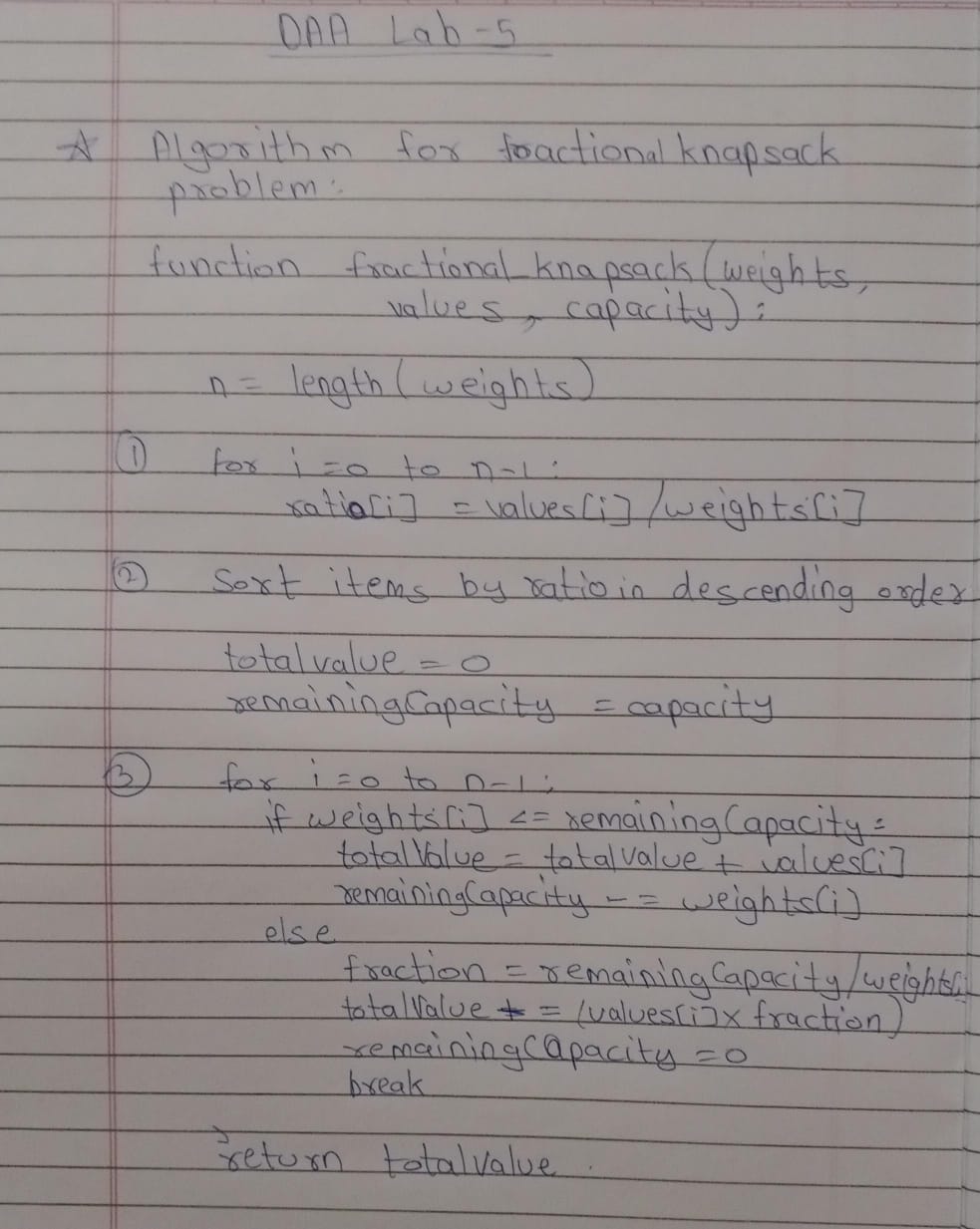
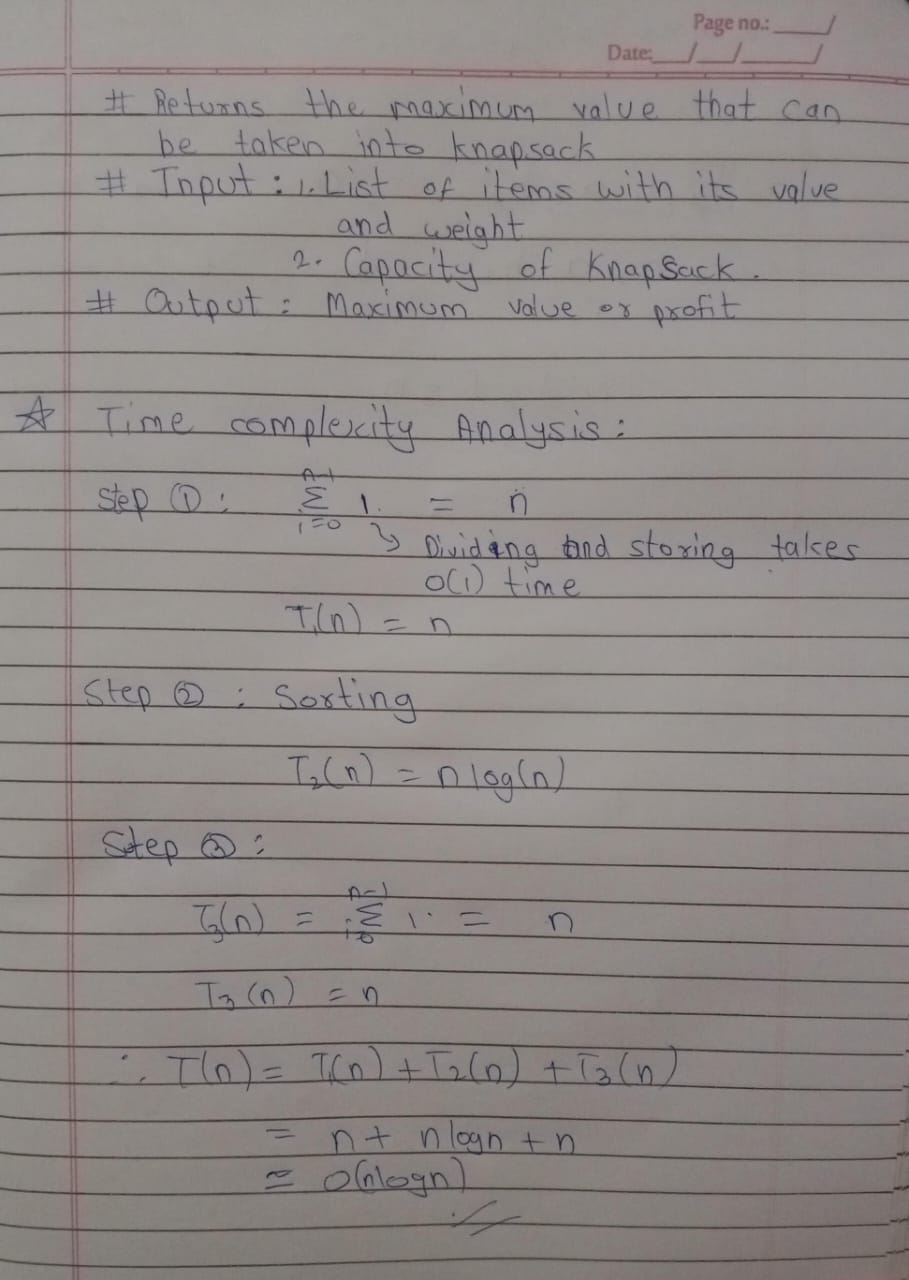
**DAA LAB-5**

**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.





CODE:

import csv

class Item:

def \_\_init\_\_(self, value, weight, shelf\_life):

*self*.value = value

*self*.weight = weight

*self*.shelf\_life = shelf\_life

*self*.priority = value / (weight \* shelf\_life)

def fractional\_knapsack(items, capacity):

items.sort(key=lambda x: x.priority, reverse=True)

total\_value = 0

for item in items:

if capacity <= 0:

break

if item.weight <= capacity:

total\_value += item.value

capacity -= item.weight

else:

fraction = capacity / item.weight

total\_value += item.value \* fraction

capacity = 0

return total\_value

def read\_items\_from\_csv(filename):

items = []

with open(filename, mode='r') as file:

reader = csv.reader(file)

next(reader)

for row in reader:

item\_id, value, weight, shelf\_life = row

item = Item(float(value), float(weight), float(shelf\_life))

items.append(item)

return items

if \_\_name\_\_ == "\_\_main\_\_":

items = read\_items\_from\_csv('items.csv')

capacity = 200

max\_value = fractional\_knapsack(items, capacity)

print(f"Maximum value in the knapsack = {max\_value:.2f}")

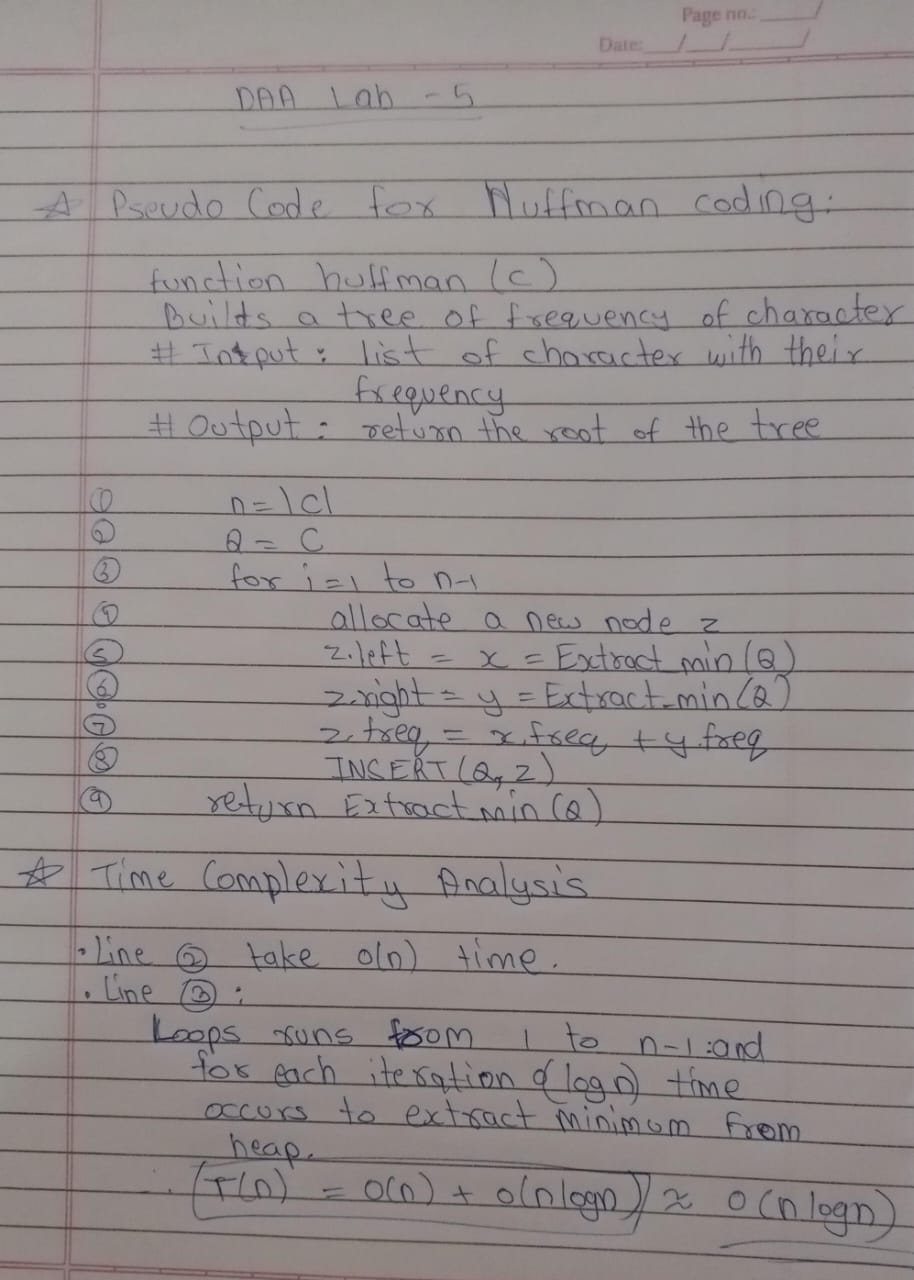
OUTPUT:



CONCLUSION:

This algorithm ensures that the maximum value is achieved within the given weight capacity of 200 tons. By prioritizing goods with shorter shelf lives and higher values, the company optimally utilizes its transport resources, minimizing the risk of valuable goods expiring before delivery. This fractional knapsack approach is effective in real-world logistics and helps companies manage resources efficiently, especially when handling perishable and high-value goods.

**Experiment task-2:** Download books from the website in html, text, doc, and pdf format. Compress these books using Huffman coding technique. Find the compression ratio.



CODE:

#include <iostream>

#include <fstream>

#include <queue>

#include <unordered\_map>

#include <vector>

using namespace std;

class Node {

public:

char ch;

int freq;

Node\* left;

Node\* right;

Node(char ch, int freq) : ch(ch), freq(freq), left(nullptr), right(nullptr) {}

};

class Compare {

public:

bool operator()(Node\* left, Node\* right) {

return left->freq > right->freq;

}

};

void encode(Node\* root, string str, unordered\_map<char, string>& huffmanCode) {

if (root == nullptr)

return;

if (!root->left && !root->right) {

huffmanCode[root->ch] = str;

}

encode(root->left, str + "0", huffmanCode);

encode(root->right, str + "1", huffmanCode);

}

Node\* buildHuffmanTree(string text) {

unordered\_map<char, int> freq;

for (char ch : text) {

freq[ch]++;

}

priority\_queue<Node\*, vector<Node\*>, Compare> pq;

for (auto pair : freq) {

pq.push(new Node(pair.first, pair.second));

}

while (pq.size() != 1) {

Node\* left = pq.top(); pq.pop();

Node\* right = pq.top(); pq.pop();

int sum = left->freq + right->freq;

Node\* sumNode = new Node('\0', sum);

sumNode->left = left;

sumNode->right = right;

pq.push(sumNode);

}

return pq.top();

}

string readFile(const string& filename) {

ifstream file(filename);

string content((istreambuf\_iterator<char>(file)), istreambuf\_iterator<char>());

file.close();

return content;

}

void writeCompressedFile(const string& filename, const string& compressedData) {

ofstream outFile(filename, ios::binary);

outFile << compressedData;

outFile.close();

}

int calculateCompressedSize(const string& text, const unordered\_map<char, string>& huffmanCode) {

int compressedSize = 0;

for (char ch : text) {

compressedSize += huffmanCode.at(ch).length();

}

return compressedSize;

}

int main() {

string filename = "test.txt";

string content = readFile(filename);

Node\* root = buildHuffmanTree(content);

unordered\_map<char, string> huffmanCode;

encode(root, "", huffmanCode);

string compressedData = "";

for (char ch : content) {

compressedData += huffmanCode[ch];

}

writeCompressedFile("compressed.txt", compressedData);

int originalSize = content.length() \* 8;

int compressedSize = calculateCompressedSize(content, huffmanCode);

double compressionRatio = (double)compressedSize / originalSize;

double compressionPercentage = 100 \* (1 - compressionRatio);

cout << "\nOriginal File Size: " << originalSize << " bits\n";

cout << "Compressed File Size: " << compressedSize << " bits\n";

cout << "Compression Ratio: " << compressionRatio << '\n';

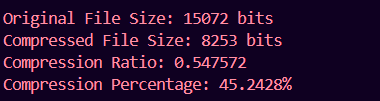
cout << "Compression Percentage: " << compressionPercentage << "%\n";

cout << "\nFile has been compressed and saved as 'compressed.bin'.\n";

return 0;

}

OUTPUT:



CONCLUSION:

In this experiment, we successfully extracted and compressed the content of a PDF file using Huffman encoding. We demonstrated the following:

1. Text Extraction: Utilized libraries like Poppler (C++) or PyMuPDF (Python) to effectively read text from PDF documents.
2. Huffman Compression: Implemented Huffman coding to efficiently compress the extracted text, reducing file size by assigning shorter codes to frequently occurring characters.
3. Performance Measurement: Quantified the efficiency of our compression method by comparing the original and compressed file sizes.

This experiment highlighted the effectiveness of Huffman encoding as a lossless compression technique and its practical applications in data management and storage optimization.