Assignment 4

Algorithm Design and Analysis

Question 1 (15 points)

Suppose we assign n persons to n jobs. Let C_{ij} be the cost of assigning the ith person to the jth job. Use a greedy approach to write an algorithm that finds an assignment that minimizes the total cost of assigning all n persons to all n jobs. Analyze your algorithm and show the time complexity using big O notation. You can either explain your algorithm or write a pseudocode for your algorithm.

The answer is in filename "VoorBenjamin_A4Q1_Nov13'24_COP4531-03.txt." I have copy-pasted the Notepad++ file below for your convenience, but it's not perfect.

```
COMMENT Benjamin Voor, Algorithm Design & Analysis, Assignment 4,
Question 1.
COMMENT each index 'i' corresponds to person 'i', and each value
corresponds to job 'j'.
COMMENT goal: return 'i' minimum values from each array 'j'.


BEGIN
INPUT integer array costs[n][n]
DECLARE OUTPUT integer array persons_with_jobs[n]
DECLARE integer person := 0
        COMMENT 'person' is the same as 'i'
REPEAT
COMMENT iterate through persons in the first for-loop
        DECLARE integer job := 0
        COMMENT 'job' is the same as 'j'
        REPEAT
        COMMENT iterate through jobs in the second for-loop
                IF persons_with_jobs[person] IS GREATER THAN costs[k][l]
                        ASSIGN persons_with_jobs[person] := costs[k][l]
        ENDIF
        ASSIGN job := job + 1
    UNTIL
        job EQUALS n
    ENDUNTIL
    ASSIGN person := person + 1
UNTIL
    person EQUALS to n
ENDUNTIL
RETURN integer array persons_with_jobs[n]
END
```

O(n^2) for the two for-loops, each ‘n’ times

Question 2 (30 points)

Suppose we have the following jobs, deadlines, and profits:

Job	*Deadline*	*Profit*
1	2	30
2	1	35
3	2	25
4	1	40

When we say that job 1 has a deadline of 2, we mean that job 1 can start at time 1 or time 2. There is no time 0. Because job 2 has a deadline of 1, that job can start only at time 1. The possible schedules and total profits are as follows:

Schedule	*Total Profit*
[1, 3]	30 + 25 = 55
[2, 1]	35 + 30 = 65
[2, 3]	35 + 25 = 60
[3, 1]	25 + 30 = 55
[4, 1]	40 + 30 = 70
[4, 3]	40 + 25 = 65

Impossible schedules have not been listed. For example, schedule [1, 2] is not possible, and is therefore not listed, because job 1 would start first at time 1 and take one unit of time to finish, causing job 2 to start at time 2. However, the deadline for job 2 is time 1. Schedule [1, 3], for example, is possible because job 1 is started before its deadline, and job 3 is started at its deadline. We see that schedule [4, 1] is optimal with a total profit of 70.

a) Instead of trying all possible ways and returning the best one, design a greedy algorithm to output the optimal solution. (20 points)

```
# If the language is not specified to be pseudocode,
might I give Python a try?
def main():
    deadline: int = 0
    profit: int = 1
    k: int = 0
    table = [[2,30],[1,35],[2,25],[1,40]] # 2D array
    for count_job, job in enumerate(table):
        if k < job[deadline]:
            k = job[deadline]
    answer = [0 for i in range(k)]

    for count_job, job in enumerate(table):
        if answer[job[deadline]-1] < job[profit]:
            answer[job[deadline]-1] = job[profit]
    return answer

if __name__ == '__main__':
    print(main())
    print(sum(main()))
```

b) Compare the time complexity of the brute force algorithm (trying all possible ways and choosing the best one) with your greedy approach. (10 points)

Initializing the 2D table could be considered O(n^2), if this is included in the algorithm analysis.

The brute-force solution would be O(n!), as every job is compared to every single other job. In the example provided, four jobs yielded six calculations, and wouldn't you know it, but 6 = 4!

Determining the size of the "answer" array 'k' was O(n), because I had to iterate through each deadline to find the answer. I did not bother sorting the 2D array, because I don't know how to do that.

The greedy approach was O(n), because the for-loop ran through every job, deciding which job with that particular deadline had the most profit.

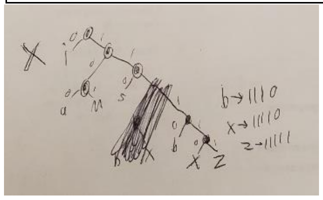
The time complexity was O(n)+O(n)=O(n).

The greedy approach only coincidentally chose the optimal solution in this case. I think that over greater data points with greater space between deadlines, the greedy approach would fail to yield the optimal solution.

Question 3 (15 points)

Use Huffman's algorithm to construct an optimal binary encoding using a prefix code tree for the following letters.
letter: frequency
a:12, b:7, z:2, x:5, s:9, m:10, i:18

i -> 0, a -> 100, m -> 101, s -> 110, b -> 1110, x -> 11110, z -> 11111



Question 4 (20 points)

You are given a set of items, each with a weight and a value. Your goal is to maximize the total value of items included in a knapsack with a fixed capacity. You can take a fraction of an item if needed.

Input:

- An array of items, where each item is represented by a weight and a value.
- The maximum capacity of the knapsack.

Output:

- The maximum total value that can be obtained by selecting items optimally.
- The fraction of each item selected.

Write a program or Pseudocode to solve the fractional knapsack problem using a greedy algorithm. You need to properly define all notations and variables that you use in the algorithm. At the end, you need to output the maximum total value and the fraction of each item selected.

Constraints: The input array of items will have at least one item. The weight and value of each item are positive integers. The maximum capacity of the knapsack is a positive integer.

The greedy approach in the 0/1 knapsack problem is not the optimal solution. However, the greedy approach in the fractional knapsack problem IS the optimal solution.

Question 5 (20 points)

You are tasked with solving the Fractional Knapsack problem using a greedy algorithm. You are given a set of items, each with a weight (Wi) and a value (Vi), and a knapsack with a fixed capacity (C). Your goal is to maximize the total value of items included in the knapsack while staying within its capacity.

Items (weight and value): [(5, 50), (10, 60), (20, 140), (15, 60), (25, 120)]. The knapsack capacity is 30. What's the maximum total value? Explain/Show your answer.

The answer is provided in the file named "Question4and5.py"