Assignment
CSE 2218: Data Structure and Algorithms II Laboratory
Trimester: **Fall 2023**

Course Teacher:
Md. Tarek Hasan
Lecturer, Department of Computer Science and Engineering (CSE)
United International University
Former Software Engineer, Samsung Research and Development Institute, Bangladesh
Departmental Profile Google Scholar LinkedIn

Any student found adopting unfair means will be expelled from the trimester/program as per UIU disciplinary rules.

You are required to create **three coding questions/problems along with algorithmic justification and solution of the problems**, each focusing on one of the topics we have covered in our recent classes - Dynamic Programming, Greedy Algorithms, and Divide and Conquer. The questions should be designed in a way that reflects the complexity and unique characteristics of each algorithmic paradigm.

Submission Guidelines:

Coding Questions:
Each question should be accompanied by a clear and concise problem statement.
Provide sample inputs and corresponding outputs to demonstrate the expected behavior of your solution.
As the examples of the coding questions, you can follow the questions of Class Performance 1 and 2. **In addition, one example is given for your reference at the end of this document.**
Sample Question of Class Performance 1
Sample Question of Class Performance 2

Algorithmic Justification:
In addition to solving the problems, you are required to write a brief explanation for each question, justifying why it falls under the category of Dynamic Programming, Greedy Algorithms, or Divide and Conquer.

Highlight the specific characteristics or properties of your questions that align with the chosen algorithmic paradigm.

Solution:
You have to provide a solution for each question that you create.

Finally, submit the assignment as one PDF. **The PDF should contain the following items in order: Coding Question 1 with sample input-output on Divide and Conquer Approach, Algorithmic Justification for Coding Question 1, solution of Coding Question 1, Coding Question 2 with sample input-output on Greedy Approach, Algorithmic Justification for Coding Question 2, solution of Coding Question 2, Coding Question 3 with sample input-output on Dynamic Programming Approach, Algorithmic Justification for Coding Question 3, solution of Coding Question 3.**

Conclusion:
- Prepare 3 programming questions with sample Input Output on 3 topics, Divide and Conquer, Greedy, and Dynamic Programming

- Write proper justifications for each question on why you think the question you prepared falls into the category of Divide and Conquer, Greedy, or Dynamic Programming
- Write the solution to each of the programming questions.

I encourage you to be creative in designing your questions while ensuring that they align with the principles we've discussed in class. Remember, this assignment is not only about solving problems but also about understanding and justifying your choices of algorithmic paradigms.

Feel free to reach out if you have any questions or need clarification. I look forward to reviewing your submissions.

Example

Question on Greedy Approach

Problem Statement: Imagine a scenario where a set of valuable items is stored in a warehouse, each item having a specific weight and value. There's a group of thieves eager to steal these items, and they possess knapsacks with uniform capacities. Your task is to figure out the minimum number of thieves needed to pilfer all the goods from the warehouse while ensuring each thief maximizes their profit. Keep in mind that items can be taken partially.
The input will specify the number of items n, followed by n lines, each detailing an item's weight and value. The last input line indicates the knapsack capacity carried by each thief. The output will showcase the items taken by each thief, including the weight of each item taken, followed by the total number of required thieves.

Sample Input Output:

Input	**Output**

4
10 200
15 150
12 360
8 200
8
	Thief 1: Item 3 (8 kg)
	
	Thief 2: Item 3 (4 kg), Item 4 (4 kg)
	
	Thief 3: Item 4 (4 kg), Item 1 (4 kg)
	
	Thief 4: Item 1 (6 kg), Item 2 (2 kg)
	
	Thief 5: Item 2 (8 kg)
	
	Thief 6: Item 2 (5 kg)

Justification: To find the optimal solution, we need to employ a greedy technique for each thief. Every thief wants to maximize their profit, hence they will prioritize the items that are the most valuable among the items while taking the items into their knapsack. This technique falls into the greedy strategy.

Solution:

```
#include<bits/stdc++.h>
using namespace std;
struct Item
```

```
{
    int itemID;
    int value;
    int weight;
};
bool compare(Item a, Item b)
{
    if(a.value/(double)a.weight>=b.value/(double)b.weight)
        return true;
    else
        return false;
}
void fractionalKnapsack(vector< Item > items, int capacity)
{
    sort(items.begin(), items.end(), compare);
    bool previousTheifLeft = true;
    int thiefCount = 0;
    int currentTheifCapacity;
    int itemCount = 0;

    for(Item &item : items)
    {
        itemCount++;
        while(item.weight)
        {
            if(previousTheifLeft)
            {
                previousTheifLeft = false;
                thiefCount++;
                currentTheifCapacity = capacity;
                if(thiefCount!=1) cout << endl << endl;
                cout << "Thief " << thiefCount << ": ";
            }
            if(currentTheifCapacity<item.weight)
            {
                item.weight -= currentTheifCapacity;
                printf("Item %d (%d kg)", item.itemID, currentTheifCapacity);
                currentTheifCapacity = 0;
                previousTheifLeft = true;
            }
            else
            {
                currentTheifCapacity -= item.weight;
                printf("Item %d (%d kg)", item.itemID, item.weight);
                item.weight = 0;
                if(currentTheifCapacity==0)
                    previousTheifLeft = true;
                else
                {
                    if(itemCount!=items.size())
                        cout << ", ";
```

```cpp
                }
            }
        }
    }
    cout << endl;
}
int main()
{
    // freopen("input.txt", "r", stdin);

    int numberOfItems;
    cin >> numberOfItems;
    vector< Item > items(numberOfItems);

    int id = 1;
    for(Item &item : items)
    {
        cin >> item.weight >> item.value;
        item.itemID = id++;
    }

    int capacity; cin >> capacity;
    fractionalKnapsack(items, capacity);

    return 0;
}
```