Transaction t_i	Size s_i	Fee f_i
1	1	4
2	3	9
3	2	6
4	4	11
5	5	13

Table 1: A toy example of five transactions with their corresponding sizes and fees. The block size limit is $b = 8$.

Problem 2 (8 marks, 1 page)**. Profit maximisation in block mining - version 1**.

As the Bitcoin price has quadrupled in the past one year (9/2020-9/2021), you have made a decision of becoming a miner to earn some profit. You find out that in Bitcoin blockchain and the like, miners are responsible for constructing blocks and if successful (being the first to solve a puzzle), will receive not only a base reward but also transaction fees included in the transactions in the block. While the base reward is fixed and out of control of the miners, the transaction fees are not. You quickly figure out that one way for the miners to maximise their profit is to select the set of transactions that sum up to the highest fee. The problem formulation you come up with is: given n available transactions, in which transaction t_i has size s_i and pays fee f_i, $1 \le i \le n$, the miner should select a set I of transaction (indices) that has the maximum total fee $\sum_{i \in I} f_i$ while guaranteeing that the total size $\sum_{i \in I} s_i$ does not exceed the block size limit b.

a) [4 marks, 1/2 page] Design a greedy algorithm for this problem (note that it does NOT have to return the optimal solution): algorithm description (1 mark) + short pseudocode (1 mark) + complexity analysis (1 mark). Run it on the toy example in Table 1 with $b = 8$ and write down the list of transactions selected by the algorithm in their corresponding order (1 mark).

b) [4 marks, 1/2 page] Design a dynamic programming algorithm of complexity $O(nb)$ that can find a set of transactions that maximises the profit: write down the recursion formula (1 mark), build the dynamic programming table for the toy example in Table 1 with $b = 8$ (2 marks), and identify the set of transactions output by the algorithm together with the maximum profit (1 mark).

Solution. Solving this problem requires the knowledge of algorithmic approaches including greedy and dynamic programming, as well as asymptotic complexity.

a) Description. A greedy algorithm builds up the set S of transactions by picking the transactions one by one, always choosing t_i with the highest transaction fee satisfying the constraint that if t_i is added to S, the total sizes will not exceed the block size limit b. It terminates when no more transaction can be added to S, and returns S. To facilitate the algorithm, the set of transactions is sorted in a decreasing order of transaction fees; if two transactions have the same fee, then the one with smaller size will be prioritised (having smaller order); if two transactions have the same size and same fee, then break tie arbitrarily.

Algorithm greedyTransactionsSelect(n, b, t_i, s_i, f_i)

```
Sort the list T = [t_1, ..., t_n] according to their fees and then sizes;
curr_size = 0; curr_fee = 0; S = ∅;
for i = 1 → n do
  if curr_size + s_i ≤ b then
    Add i to S; curr_size = curr_size + s_i; curr_fee = curr_fee + f_i;
  end if
end for
return S, curr_size, curr_fee;
```

Pseudocode. See above.

Complexity analysis. The most costly operation is sorting, which takes time $O(n \log n)$. The for loop takes $O(n)$ time. Therefore, the overall complexity of the greedy algorithm is $O(n \log n)$.

Run the algorithm on the toy example. Running the greedy algorithm on the toy example in Table 1, the transactions are picking in the following order: t_5, t_2. The total size is $8 = 5 + 3$, while the total fee is $22 = 13 + 9$.

b) The transaction selection problem is precisely the Knapsack problem with b playing the role of the capacity, s_i the weights, and f_i the values of the n items. Therefore, we can reuse the dynamic programming algorithm taught in the lecture to solve this problem. Let $V[i, j]$ denotes the maximum value one can get by selecting a subset of transactions among the first i transactions while keeping the sum of sizes not exceeding j, for $0 \le i \le n$ and $0 \le j \le b$. Then the **recurrence formula** is

$$V[i,j] = \begin{cases} V[i-1,j], & \text{if } s_i > j, \\ \max\{V[i-1,j], f_i + V[i-1, j-s_i]\}, & \text{if } s_i \le j \end{cases}, \quad V[i,0] = V[0,j] = 0.$$

The dynamic programming table for the toy example is given below. The maximum fee possible is 24. A backtrace gives us the transactions t_1, t_2, t_4 and indeed, $f_1 + f_2 + f_4 = 4 + 9 + 11 = 24$ and $s_1 + s_2 + s_4 = 1 + 3 + 4 = 8 = b$.

s_i	f_i	$V[i,j]$	$j=0$	1	2	3	4	5	6	7	8
		$i=0$	**0**	0	0	0	0	0	0	0	0
1	**4**	1	0	**4**	4	4	4	4	4	4	4
3	**9**	2	0	4	4	9	**13**	13	13	13	13
2	6	3	0	4	6	10	**13**	15	19	19	19
4	**11**	4	0	4	6	10	13	15	19	21	**24**
5	13	5	0	4	6	10	13	15	19	21	**24**