1. Consider the 0-1 Knapsack problem that we discussed in the lecture. Given n positive integers $w_1, ..., w_n$ and another positive integer W, the goal is to find a subset S of indices such that $\sum_{i \in S} w_i$ is maximised subject to $\sum_{i \in S} w_i \leq W$. We studied a dynamic program for this problem that has a running time $O(n \cdot W)$. We know that this is bad when the integers are very large. The goal of this task is to improve the running time at the cost of the optimality of the solution. Let us make this more precise. Let OPT denote the value of the optimal solution. For any given $\varepsilon \in (0, 1]$, design an algorithm that outputs a solution that has a value at least $\frac{OPT}{(1+\varepsilon)}$. The running time of your algorithm should be of the form $O\left(\frac{f(n)}{\varepsilon}\right)$, where $f(n)$ is some polynomial in n. You need not give the best algorithm that exists for this question. As long as $f(n)$ in the running time is polynomial in n; you will get the full credit.

   (Hint: Try modifying the dynamic programming solution for the knapsack problem.)

Solution: Let us assume that for all i, $w_i \leq W$. Since otherwise, we can remove the item i and then solve the problem with the remaining items. We will first check if $\sum_i w_i \leq W$. If so, the optimal solution will pick all items. So, now we only need to focus on the case when $\sum_i w_i > W$. Let w be the weight of the item with maximum weight. Since $\sum_i w_i > W$, we have:

$$n \cdot w > W \tag{1}$$

A DP that does not work: Let us try to use the same Dynamic Programming formulation as in the class and try to "scale". We will see that this formulation has an issue. However, this will nicely set us up for the DP that works.

Given the problem instance $I = (w_1, ..., w_n, W)$ we create the following new instance of the 0-1 knapsack problem $I' = (w'_1, ..., w'_n, W')$, where for all $i, w'_i = \lfloor w_i/m \rfloor$, $W' = \lfloor W/m \rfloor$, and $m = \lfloor w/(2n/\varepsilon) \rfloor$. We solve the second instance using the dynamic programming that we discussed in class. Let O' denote the optimal solution for $(w'_1, ..., w'_n, W')$. That is, O' is the set of items picked by the dynamic programming algorithm when executed on instance I'. Note that the running time of the algorithm in instance I' will be $O(n \cdot W') = O(n \cdot W/m) = O\left(\frac{n^2}{\varepsilon} \cdot \frac{W}{w}\right) = O(n^3/\varepsilon)$ (using inequality (1)). Let us analyze if we can return O' as an approximate solution to instance I. The fact that $\sum_{i \in O'} w'_i \leq W'$ does not ensure that $\sum_{i \in O'} w_i$ will be at most W. This is where the problem lies with this formulation.

We will use an alternative DP formulation. Consider $w'_1, ..., w'_n$ as in the above DP formulation. Let

$$L(i, w) = \min_{S \subseteq \{1,...,i\} \text{ s.t. } \sum_{i \in S} w'_i = w} \left\{ \sum_{i \in S} w_i \right\}.$$

For base cases, we have $L(i, 0) = 0$ for all $i \geq 0$ and $L(0, w) = \infty$ for all $w > 0$. Furthermore, we can write:

$$L(i, w) = \begin{cases} \min\{L(i-1, w - w'_i) + w_i, L(i-1, w)\} & \text{If } w'_i \leq w \\ L(i-1, w) & \text{Otherwise} \end{cases}$$

We can compute the values $L(i, w)$ similar to how we compute the values $M(i, w)$ as in the previous formulation. We can also output the set $S \subseteq \{1, ..., i\}$ that minimizes $\sum_{i \in S} w_i$ such that $\sum_{i \in S} w'_i = w$ for a given (i, w). The following program computes these values:

```
0-1-Knap(w', w, W)
  - For i = 0 to n
      - L[i, 0] ← 0
      - P[i, 0] ← "↑"
  - For t = 0 to W
      - L[0, t] ← ∞
  - For i = 1 to n
      - For t = 1 to W
          - If (w'[i] > t)
              - L[i, t] ← L[i − 1, t]
              - P[i, t] = "↑"
          - elseif (L[i − 1, t] ≤ L[i − 1, t − w'[i]] + w[i])
              - L[i, t] ← L[i − 1, t]
              - P[i, t] = "↑"
          - else
              - L[i, t] ← L[i − 1, t − w'[i]] + w[i]
              - P[i, t] = "↖"
```

```
FindItems(w', P, t)
  - i ← n; S ← {}
  - While (i ≠ 0)
      - If (P[i, t] = "↑")
          - i ← i − 1
      - else
          - S ← S ∪ {i}
          - i ← i − 1
          - t ← t − w'[i]
  - return(S)
```

Let t be the largest index such that $L(n,t) \leq W$. Then we run the program `FindItems(`w', P, t`)` to find a set S. We return S as the approximate solution to the original problem.

First note that $\sum_{i \in S} w_i \leq W$. This follows from the definition of L. Let O denote the optimal solution to the given problem. We prove the following claim.

Claim 1: $\sum_{i \in O} w'_i \leq t$.

Proof. Since otherwise, t will not be the largest index such that $L(n,t) \leq W$. □

Finally, we have

$$\sum_{i \in S} w'_i = t \geq \sum_{i \in O} w'_i$$

Multiplying both sides by m, we get:

$$
\begin{aligned}
& m \cdot (\sum_{i \in S} w_i') \geq m \cdot (\sum_{i \in O} w_i') \\
\Rightarrow \quad & \sum_{i \in S} w_i \geq m \cdot \left(\sum_{i \in O} \lfloor w_i / m \rfloor \right) \\
\Rightarrow \quad & \sum_{i \in S} w_i \geq \sum_{i \in O} w_i - mn \\
\Rightarrow \quad & \sum_{i \in S} w_i \geq OPT - (\varepsilon/2)w \\
\Rightarrow \quad & \sum_{i \in S} w_i \geq OPT - (\varepsilon/2) \cdot OPT \quad (\text{since } w \leq OPT) \\
\Rightarrow \quad & \sum_{i \in S} w_i \geq (1 - \varepsilon/2) \cdot OPT \geq \frac{OPT}{1+\varepsilon}
\end{aligned}
$$