In the <u>Knapsack problem</u>, we are given a set of n items $I = \{1, \cdots, n\}$, where each item i has a value v_i and a size s_i. All sizes and values are positive integers. The knapsack has capacity B, where B is also a positive integer. The goal is to find a subset of items $S \subseteq I$ that maximizes the value $\sum_{i \in S} v_i$ of items in the knapsack subject to the constraint that the total size of these items is no more than the capacity; that is, $\sum_{i \in S} s_i \le B$. We assume that we consider only items that could actually fit in the knapsack by themselves, so that $s_i \le B$ for each $i \in I$.

<u>A dynamic programming algorithm for the knapsack problem:</u>

```
A(1) ← {(0,0), (s1,v1)}
for j ← 2 to n do
    A(j) ← A(j-1)
    for each (t,w) ∈ A(j-1) do
        if t + sj ≤ B then
            Add (t+sj, w+vj) to A(j)
    Remove dominated pairs from A(j)

return    max      w
       (t,w) ∈ A(n)
```

We maintain an array entry $A(j)$ for $j = 1, \cdots, n$. Each entry $A(j)$ is a list of pairs (t, w). A pair (t, w) in the list of entry $A(j)$ indicates that there is a set S from the first j items that uses space exactly $t \le B$ and has value exactly w; that is, there exists a set $S \subseteq \{1, \cdots, j\}$, such that $\sum_{i \in S} s_i = t \le B$ and $\sum_{i \in S} v_i = w$. Each list keeps track of only the most efficient pairs (dominating pairs) from which optimal solution is possible. Non-dominating pairs cannot give optimal solutions, so they are eliminated.

A pair (t, w) dominates another pair (t', w') if $t \leq t'$ and $w \geq w'$. In any list, no pair dominates another one; this means that we can assume each list $A(j)$ is of the form $(t_1, w_1), \ldots, (t_k, w_k)$ with $t_1 < t_2 < \cdots < t_k$ and $w_1 < w_2 < \cdots < w_k$. Since the sizes of the items are integers, this implies that there are at most $B+1$ pairs in each list. Furthermore, if we let $V = \sum_{i=1}^{n} v_i$ be the maximum possible value for the knapsack, then there can be at most $V+1$ pairs in the list. Therefore the above algorithm as time complexity $O(n \min(B, V))$. This is not a polynomial-time algorithm. It is a pseudopolynomial-time algorithm:

An algorithm for a problem Π is said to be pseudopolynomial if its running time is polynomial in the size of the input when the numeric part of the input is encoded in unary.

PTAS: A polynomial time approximation scheme (PTAS) is a family algorithms $\{A_\epsilon\}$, where there is an algorithm for each $\epsilon > 0$, such that A_ϵ is a $(1+\epsilon)$-approximation algorithm (for minimization problems) or a $(1-\epsilon)$-approximation algorithm (for maximization problems).

FPTAS: A fully polynomial-time approximation scheme (FPTAS) is an approximation scheme such that the running time of A_ϵ is bounded by a polynomial in $1/\epsilon$.

If the maximum possible value V were some polynomial in n, then the running time would indeed be a polynomial in the input size. Suppose that we measure value in (integer) multiples of μ (to be defined later), and convert each value v_i by rounding down to the nearest integer multiple of μ; more precisely, we set v_i' to be $\lfloor v_i/\mu \rfloor$ for each item i.

We can then run the dynamic programming algorithm on the items with sizes s_i and values v_i', and output the optimal solution for the rounded data as a near-optimal solution for the true data. If we use values $\overline{v_i = v_i' \mu}$ instead of v_i, then each value is inaccurate by at most μ, and so each feasible solution has its value changed by at most $n\mu$. We want the error introduced to be at most ϵ times a lower bound on the optimal value (and so be sure that the true relative error is at most ϵ). Let M be the maximum value of an item; that is $M = \max_{i \in I} v_i$. Then M is a lower bound on OPT since one possible solution is to pack the most valuable item in the knapsack by itself. We set μ such that $n\mu = \epsilon M \Rightarrow \underline{\mu = \epsilon M/n}$.

With the modified values, $V' = \sum_{i=1}^{n} v_i' = \sum_{i=1}^{n} \left\lfloor \frac{v_i}{\epsilon M/n} \right\rfloor = O(n^2/\epsilon)$.
Thus, the running time of the algorithm is $O(n.\min(B, V'))$ $= O(n^3/\epsilon)$ and is bounded by a polynomial in $1/\epsilon$.

<u>FPTAS for knapsack:</u>

$M \leftarrow \max_{i \in I} v_i$

$\mu \leftarrow \epsilon M/n$

$v_i' \leftarrow \lfloor v_i/\mu \rfloor$ forall $i \in I$

Run the dynamic programming algorithm for knapsack with values v_i'

For proving the above algorithm to be FPTAS, we have to show that it returns a solution whose value is at least $(1-\epsilon)$ times the value of an optimal solution. Let S be the set of items returned by the algorithm. Let O be an optimal set of items. We have $M \leq OPT$. By definition of v_i': $\mu v_i' \leq v_i \leq \mu(v_i'+1)$, so that $\mu v_i' \geq v_i - \mu$.

Applying the definitions of the rounded data, along with the fact that S is an optimal solution for the values v_i':

$$\begin{aligned}
\sum_{i\in S} v_i &\geq \mu \sum_{i \in S} v_i' \\
&\geq \mu \sum_{i\in O} v_i' \\
&\geq \sum_{i\in O} v_i - |O|\mu \\
&\geq \sum_{i\in O} v_i - n\mu \\
&= \sum_{i\in O} v_i - \epsilon M \\
&\geq OPT - \epsilon OPT = \underline{(1-\epsilon)\, OPT}
\end{aligned}$$