$0-1$ **Knapsack summary**

- The problem: We are given a knapsack of capacity W and a set of n items, where item i is worth $v[i]$ and has weight $w[i]$ pounds. The weights $w[i]$ and the total weight W are **integers**. Fill the knapsack so that the overall value of all items in the knapsack is maximized.

- Notation and choice of subproblem: Let $optknapsack(k, w)$ return the maximal value for a knapsack of size w using items among items 1 through k. To solve our problem we call $optknapsack(n, W)$.

  for simplicity, we'll assume that the weight and value arrays $w[], v[]$ are global

- Recursive definition of $optknapsack(k, w)$:

  @returns the max value to pack a knapsack of capacity w using the first k items
  $\textsc{optknapsack}(k, w)$
  if $(w == 0)$: return 0
  if $(k \leq 0)$: return 0
  if $(w[k] \leq w)$: $with = v[k] +$ $\textsc{optknapsack}$ $(k-1, w-w[k])$
  else: $with = 0$
  $without = \textsc{optknapsack}(k-1, w)$
  RETURN max $\{$ $with, without$ $\}$

- Why correct? It tries both possibilities for each item (with or without) and recurses on the rest, which is correct bec. it has optimal substructure (why?).

- Dynamic programming, recursive (top-down) with memoization:

  Create a table a $table[1..n][1..W]$, where $table[i][w]$ will store the result of optknapsack(i, w).
  Initialize all entries in the table as 0.
  Call $optknapsackDP(n, W)$ and return the result.

  @returns the max value to pack a knapsack of capacity w using the first k items
  $\textsc{optknapsack}\text{DP}(k, w, table)$
  if $(w == 0)$: return 0
  if $(k == 0)$: return 0
  IF $(table[k][w] \neq 0)$: RETURN $table[k][w]$
  IF $(w[k] \leq w)$: $with = v[k] +$ $\textsc{optknapsack}\text{DP}(k-1, w-w[k], table)$
  ELSE: with $= 0$
  $without = \textsc{optknapsack}\text{DP}(k-1, w, table)$
  $table[k][w]$ $=$ max $\{$ $with, without$ $\}$
  RETURN $table[k][w]$

  Running time for $optknapsackDP(n, W)$: $O(n \cdot W)$

- Dynamic programming, iterative (bottom-up):

@returns the max value to pack a knapsack of capacity W using the n items

OPTKNAPSACKDP_ITERATIVE()

create table[0..n][0..W] and initialize all entries to 0
for $k = 1; k \leq n; k++$
    for $w = 1; w \leq W; w++$
        if $(w[k] \leq w)$ with $= v[k] + table[k-1][w - w[k]]$ else with= 0
        without $= table[k-1][w]$
        $table[k][w] = \max$ { with, without }
RETURN $table[n][W]$

Running time: $O(n \cdot W)$

- Computing full solution (without storing additional information while filling the table):

@param: The table $table[1..n][1..W]$ as computed above, where $table[i][x]$ stores the max value to pack a knapsack of weight x using items $1..i$.
@return: prints the set of items corresponding to $table[n][W]$

FINDITEMS(table)

$i = n, w = W$
while $(i > 0$ and $w > 0)$
    // is the value $table[i][w]$ achieved with item i or without?
    if $table[i][w] == v[i] + table[i-1][w - w[i]]$ //WITH item i
        output item i
        $w = w - w[i]$, $i = i - 1$
    else // WITHOUT item i
        $i = i - 1$

Running time: $O(n)$, with no extra space