Algorithm 1 Knapsack Algorithm - Dynamic Programming

1: $cost \leftarrow M + 1$ lenght array of 0's
2: $best \leftarrow M + 1$ lenght array of 0's
3: **procedure** KNAPSACK(M)
4:      **for** i **from** 1 to N **do**
5:          **for** k **from** size[i] to M **do**
6:              **if** val[i] + cost[k-size[i]] > cost[k] **then**
7:                  cost[k] = val[i] + cost[k-size[i]]
8:                  best[k] = i

Algorithm 2 Floyd-Warshall with path reconstruction

1: $dist \leftarrow |V| \times |V|$ length array of minimum distances initialized to ∞
2: $next \leftarrow |V| \times |V|$ length array of vertex indices initialized to **null**
3: **procedure** FLOYD-WARSHALL(PATH RECONSTRUCTION)
4:      **for each** edge(u, v) **do**
5:          $dist[u][v] \leftarrow w(u, v)$
6:          $next[u][v] \leftarrow v$
7:      **for** $k = 1 to |V|$ **do**
8:          **for** $i = 1 to |V|$ **do**
9:              **for** $j = 1 to |V|$ **do**
10:                  **if** $dist[u][v] > dist[i][k] + dist[k][j]$ **then**
11:                      $dist[u][v] \leftarrow dist[i][k] + dist[k][j]$
12:                      $next[i][j] \leftarrow next[i][k]$
13: **procedure** GETPATH(u, v)
14:      **if** $next[u][v] = $ **null then**
15:          **return** []
16:      $path = [u]$
17:      **while** $u \neq v$ **do**
18:          $u \leftarrow next[u][v]$
19:          path.append(u)
20:      **return** path

Algorithm 3 TSP by Nearest Neighbour

1: V ← vertices representing all destinations in the route
2: **procedure** NEAREST NEIGHBOUR(**Vertex** P)
3:     **queue** sortedVertices ← $V \setminus \{P\}$
4:     result = $[P]$
5:     **while** $|sortedVertices| > 0$ **do**
6:         sortedVertices.sortRelativeTo(P)
7:         $P \leftarrow sortedVertices.front()$
8:         sortedVertices.pop()
9:         result.append(P)
10:     result.append(result[0])
11:     **return** result