Algorithm 1 Knapsack Algorithm - Dynamic Programming

```
1: cost \leftarrow M + 1 length array of 0's

2: best \leftarrow M + 1 length 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 \infty
2: next \leftarrow |V| \times |V| length array of vertex indices initialized to null
3: procedure Floyd-Warshall(Path Reconstruction)
        for each edge(u, v) do
4:
            dist[u][v] \leftarrow w(u,v)
5:
            next[u][v] \leftarrow v
 6:
       for k = 1to|V| do
 7:
            for i = 1to|V| do
8:
                for j = 1to|V| do
9:
                   if dist[u][v] > dist[i][k] + dist[k][j] then
10:
                       dist[u][v] \leftarrow dist[i][k] + dist[k][j]
11:
12:
                       next[i][j] \leftarrow next[i][k]
    procedure GetPath(u, v)
13:
       if next[u][v] = null then
14:
            return []
15:
       path = [u]
16:
       while u \neq v do
17:
            u \leftarrow next[u][v]
18:
            path.append(u)
19:
20:
       return path
```

Algorithm 3 TSP by Nearest Neighbour

```
1: V \leftarrow vertices representing all destinations in the route
2: procedure Nearest Neighbour(Vertex P)
       sorted
Vertices \leftarrow V \setminus \{P\}
3:
       result = [P]
4:
       while |sortedVertices| > 0 do
5:
6:
           sorted Vertices. sortRelative To(P) \\
           P \leftarrow sortedVertices[0]
 7:
           sortedVertices.remove(P)
8:
           result.append(P) \\
9:
       result.append(result[0])
10:
       return result
11:
```