Algorithm 1 Knapsack Algorithm - Dynamic Programming

```
1: procedure KNAPSACK(M)
2:     cost ← M + 1 lenght array of 0's
3:     best ← M + 1 lenght array of 0's
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
9:     print(cost[M])
10:    for k from M to 0 step size[best[k]] do
11:        print(best[k])
```

Algorithm 2 Held-Karp - Dynamic Programming

```
1: procedure TSP(G, n)
2:     for k := 2 to n do
3:         C({k}, k) := d_{1,k}
4:     for s := to n - 1 do
5:         for all S ⊆ {2, ..., n}, |S| = s do
6:             for all k ∈ S do
7:                 C(S, k) := min_{m≠k, m∈S} [C(S
8: {k}, m + d_{m,k}]
9:     return min_{k≠1}[C({2, ..., n}, k) + d_{k,1}]
```

Algorithm 3 Floyd-Warshall with path reconstruction

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

Algorithm 4 Nearest Neighbour

```
1: procedure NEAREST NEIGHBOUR(Vertex P)
2:     queue nodesSorted
3:     result = []
4:     V_t ← V
5:     while |V_t| > 0 do
6:         nodesSorted.resortRelativeTo(P)
7:         result.append(P)
8:         P ← result[0]
9:         nodesSort.removeTop()
10:    return result
```