1.a

We are given a graph $G = (V, E)$, and a vertex $v \in V$, and we want to construct a graph G' such that G' has the same vertices as G except with v removed, and the shortest path distances between all remaining vertex pairs are the same.

If we consider removing v, for each individual path of length 2 going through v, we replace it with a single edge of the same total length, thus bypassing v with the same cost. Then we can remove v from the graph and return it.

```
p1(G = (V,E), v):
    G' := copy(G) without v
    for i in inc(v):
        for j in out(v):
            G'_w(i,j) := min(G'_w(i,j), G_w(i,v) + G_w(v,j))
    return G'
```

1.b

We are given a graph G and a vertex v, along with a graph G' generated from running part a on G and v, where we already know all the shortest path distances in G'. We want to compute the remaining shortest path distances in G, which are those from any other vertex to v and v to any other vertex.

Since we already know the distance from j to x for any j in the outgoing vertices of v, we can compute the minimum distance from v to x by minimizing the sum of that distance with the edge weight $w(v, j)$ over all choices of j.

The same argument applies to computing the minimum distance from x to v. We repeat these steps for all vertices in G'.

```
p2(G = (V,E), G' = (V-v,E')):
    for x in V-v:
        minval := inf
        for j in G_out(v):
            minval := min(minval, G_w(v,j) + spd(j,x))
        spd(v,x) := minval
    for x in V-v:
        minval := inf
        for j in G_in(v):
            minval := min(minval, G_w(j,v) + spd(x,j))
        spd(x,v) := minval
    return
```

1.c

To obtain an $O(n^3)$ all-pairs-shortest-path algorithm, we can combine the algorithms from part a and b. The idea is to recursively strip off one vertex at a time with part a, and on the way back up the recursion tree, to compute the distances.

First, we choose some vertex v in G. we remove it with the part a algorithm. Since all the remaining shortest path distances remain the same from part a, when we call our function on G', it will compute the correct shortest path distances for the subset of vertex pairs excluding v.

Then, we use the part b algorithm to determine the distances to and from v. In addition, we calculate the distance from v to itself, using a similar argument as from part b.

In the base case, there will be only one vertex in the graph. Since the part a algorithm replaces edges in a simple graph instead of adding multiple edges between the same vertex pairs, always choosing the shorter edge, there will be only one edge from our one vertex to itself, and it will be of length corresponding to the shortest path distance to itself.

```
APSP(G = (V,E)):
    v := arbitrary vertex in V
    if |V| = 1:
        spd(v,v) := w(v,v)
        return
    G' := P1(G,v)
    APSP(G')
    P2(G,G',v)
    spd(v,v) := inf
    for x in out(v):
        spd(v,v) := min(spd(v,v), spd(x,v) + w(v,x))
    return
```

Runtime:

The first two algorithms are $O(n^2)$. It takes $O(n)$ time to compute the distance from v to itself. The part c algorithm therefore does $O(n^2)$ work per recursive call, and makes one smaller recursive call. There will be n recursive calls since each one removes one vertex and the base case is reached when only one vertex is left. The runtime is $O(n^3)$.