(a)

In this part, we consider the min-cost path using exactly i edges. So the only difference from what we have discussed in the lecture is the recurrence step. Since during iteration i we have only one choice which is adding an edge, there will be no "$OPT(i-1,v)$" term in our recursion process. Hence, the recursive formula will be:

$$OPT(i,v) = \min_{w \in V}(OPT(i-1,w) + c_{vm}) \tag{1}$$

Denote the number of nodes in G as n, and the algorithm is stated as follow:

Algorithm 1 Algorithm for problem 2a

Array $OPT[0, \cdots, V]$
Define $OPT[0,t] = 0$
Define $OPT[0,v] = \infty$ for $v \neq t$
For $i = 1, 2, \cdots n-1$
    For $v \in V$ in any order
        Compute $OPT[i,v]$ using $OPT[i,v] = \min_{w \in V}(OPT(i-1,w) + C_{vw})$
    Endfor
Endfor
Return $OPT[n-1,s]$

Proof of Correctness:
The correctness of this algorithm follows directly by induction from Equation (1), and is also proved during the lecture.

Time complexity analysis:
The array OPT has n^2 entries. For each entry it takes $O(n)$ time to compute. Hence, the total running time will be $O(n^3)$.

(b)

Now we would also like to compute the number of different shortest path. The only thing we need to do is to add another array C to record the count of the min-path in each recursion step. And finally we multiply them together to get the final result. The algorithm is stated as follow:

Algorithm 2 Algorithm for problem 2b

Array $OPT[0, \cdots, V]$
Array $C[0, \cdots, V]$
Define $OPT[0, t] = 0$
Define $OPT[0, v] = \infty$ for $v \neq t$
For $i = 1, 2, \cdots n-1$
    For $v \in V$ in any order
        Compute $OPT[i, v]$ using $OPT[i, v] = \min_{w \in V}(OPT(i-1, w) + C_{vw})$
        Compute the number of w (i.e., minimum points) which leads to this optimum value and assign it to $C[i, v]$
    Endfor
Endfor
Return $OPT[n-1, s]$
Return $\prod_{i=1,j=1}^{n} C[i, j]$

Proof of Correctness:
The correctness of this algorithm follows directly by induction from Equation (1), and is also proved during the lecture.

Time complexity analysis:
Both array OPT and C have n^2 entries. For each entry of both arrays it takes $O(n)$ time to compute. Hence, the total running time will be $O(n^3)$.

(c)

This time we use any number of shortest paths, just as we discussed during the lecture. We only need to add the counting process shown in part (b) to the original algorithm, the recursion of which is:

$$OPT(i, v) = \min(OPT(i-1, v), \min_{w \in V}(OPT(i-1, w) + c_{vm})) \tag{2}$$

The algorithm is stated as follow:

Algorithm 3 Algorithm for problem 2c

Array $OPT[0, \cdots, V]$
Array $C[0, \cdots, V]$
Define $OPT[0, t] = 0$
Define $OPT[0, v] = \infty$ for $v \neq t$
For $i = 1, 2, \cdots n-1$
    For $v \in V$ in any order
        Compute $OPT[i, v]$ using $OPT(i, v) = \min(OPT(i-1, v), \min_{w \in V}(OPT(i-1, w) + c_{vm}))$
        Compute the number of w (i.e., minimum points) which leads to this optimum value and assign it to $C[i, v]$; **If opt$(\mathbf{i-1, v})$ is chosen then we assign C$[\mathbf{i, v}]$ by 1**
    Endfor
Endfor
Return $OPT[n-1, s]$
Return $\prod_{i=1,j=1}^{n} C[i, j]$

Proof of Correctness:
The correctness of this algorithm follows directly by induction from Equation (2), and is also proved during the lecture.

Time complexity analysis:
Both array OPT and C have n^2 entries. For each entry of both arrays it takes $O(n)$ time to compute. Hence, the total running time will be $O(n^3)$.