1a. The algorithm would not find an independent set of maximum total weight for the path (1-998-999-998-1).
1b. The algorithm would not find an independent set of maximum total weight for the path (1-999-1-1-999).
1c.
Algorithm description in Plain English:

First, convert G to a list and initialize a list S to cache the results of subproblems with S[0] being 0 and S[1] being the value of the first node in the path. Then, for every index i of S after 1, S[i] = max(S[i-1], S[i-2] + G[i-1]). Note that the index passed into G is i-1 since G is 0-indexed while S is 1-indexed. When every index of S is calculated, return the last index of S.

Proof of Correctness:

For every index i of S, note that the maximum path value or optimal solution with only the nodes G[0] to G[i-1] will be the maximum of S[i-1] or S[i-2] + G[i-1]. Essentially, the algorithm will either opt to ignore the current node and take the optimal solution of the previous path (without the current node), or it will opt to choose the current node and add it to the optimal solution of the previous path (without the previous neighboring node). Thus, S[n] must represent the optimal solution for all n nodes in G.

Complexity Analysis:

The above algorithm requires n iterations, and each iteration occurs in O(1) time where n is the number of nodes in path G. Therefore, the total time complexity is O(n).

2.
Algorithm description in Plain English:

First, initialize a list S of length v+1 to cache the results of subproblems with S[0] being 0 and S[1] being 1. Then, for every index i of S after 1, S[i] = min(1 + S[i-j] for each j in denominations). When every index of S is calculated, return the last index of S.

Proof of Correctness:

For every index i of S, note that the optimal solution for a value of i will be the minimum of 1 + S[i-j] for each j in the coin denominations. Essentially, for each iteration of S, the algorithm will calculate the current value's optimal solution by iterating through every coin denomination and adding 1 to the optimal solution of the current value minus the current denomination. Thus, S[v] must represent the optimal solution for the value v.

Complexity Analysis:

The above algorithm requires v iterations, and each iteration occurs in O(n) time where n is the number of different coin denominations. Therefore, the total time complexity is O(nv).

3.
Algorithm description in Plain English:

First, initialize a list S of length n+1 to cache the results of subproblems with S[0] being 0 and S[1] being p_1. Then, for every index i of S after 1, S[i] = max(S[i-1], S[i-k] + p_i). Note that if i-k is negative, S[i-k] should equal 0. When every index of S is calculated, return the last index of S.

Proof of Correctness:

For every index i of S, note that the optimal solution for a distance of m_i miles will be the maximum of S[i-1] or S[i-k] + p_i. Essentially, the algorithm will either opt to ignore the current location and take the optimal solution of the previous distance (without the current location), or it will opt to choose the current location and add it to the optimal solution of the previous distance (without all the locations within k distance of the current location). Thus, S[n] must represent the optimal solution for all locations along the highway.

Complexity Analysis:

The above algorithm requires n iterations, and each iteration occurs in O(1) time where n is the number of locations along the highway. Therefore, the total time complexity is O(n).

4.

Algorithm description in Plain English:

First, initialize a list S of length n+1 where n is the length of y to cache the results of subproblems with S[0] being 0 and S[1] being quality(y[0]). Then, for every index i of S after 1, S[i] = max(quality(y[j:i]) + S[j] for j in range(0, i)). When every index of S is calculated, return the last index of S.

Proof of Correctness:

For each index i of S, note that the optimal solution up to the corresponding index of the string y will be the maximum of quality(y[j:i]) + S[j] for each j in the range of 0 to i. Essentially, for each iteration of S, the algorithm will calculate the current index's optimal solution by iterating through every optimal quality solution made beforehand and adding the quality of a string that includes all characters from the end of that solution to the current index of y. Thus, S[n] must represent the optimal solution for all characters in the string y.

Complexity Analysis:

The above algorithm requires n iterations, and each iteration occurs in O(n) time where n is the length of y. Therefore, the total time complexity is $O(n^2)$.

5.

Algorithm description in Plain English:

First, initialize a list S of length n+1 to cache the results of subproblems with S[0] being 0, S[1] being r * s_1 + S[0], and S[2] being r * s_2 + S[1]. Then, for every index i of S after 2, S[i] = min((r * s_i) + S[i-1], 4c + S[i-4]). When every index of S is calculated, return the last index of S.

Proof of Correctness:

For each index i of S, note that the optimal cost solution of the schedule up to the week i will be the minimum of $(r * s_i) + S[i-1]$ and $4c + S[i-4]$. Essentially, for each iteration of S, the algorithm will either opt for Company A for the current week (and add $r * s_i$ to the last week's optimal solution), or it will opt for Company B for the 4 most recent weeks of the schedule (and add 4c to the optimal solution of 4 weeks ago). Thus, S[n] must represent the optimal solution for all weeks in the schedule.

Complexity Analysis:

The above algorithm requires n iterations, and each iteration occurs in O(1) time where n is the number of weeks in the schedule. Therefore, the total time complexity is O(n).