Task 1

A: No nodes are marked.

B: An all-marked distance is the length of a path from node `u` to node `v`, on which only `v` is unmarked and all other nodes (if any) are marked.
(For clarity, in this document, we define such a path as an "all-marked path"; in contrast, define a "not-all-marked path" as a path having at least two unmarked nodes, including the destination node.)
Since all nodes are unmarked initially, there is no all-marked path from node 0 to any node other than node 0 itself, and the path from 0 to 0 has length 0, so `m[0]` should be ***min*** {0} == 0, and all other `m[i]`s should be ***min*** ∅ = ∞, which is exactly how we have assigned `m`.

C: Because from initialization node `tn` (node 0) is unmarked and has the smallest value (0) in `m`.

D: The precondition of this statement is that I1, I2, I3 all holds. By I2, `m[tn]` is the length of the shortest all-marked path from 0 to `tn`. If there exists a path which is not-all-marked, meaning that at least one node `u (u ≠ tn)` on this path is unmarked, then by I3, we know that `m[u] >= m[tn]`, showing that any not-all-marked path is no shorter than `m[tn]`. So `m[tn]` is indeed the least distance from 0 to `tn`. Thus, we can extend I1 by marking node `tn` and it still holds.

E: Now that node `tn` is marked, there may appear an all-marked path from node 0 via node `tn` to some unmarked node `i`, whose length is smaller than `m[i]`, thus breaking I2.

F: Node `tn` is marked now.

G: This code updates the least all-marked distance from 0 to every unmarked node `i` which is directly reachable from node `tn`, using the length of the shortest path from 0 to `tn` plus the length of the edge `tn -> i`. Only updating directly reachable nodes is sufficient, because if there appears a new all-marked path `P: 0 -> ... -> tn -> ... -> j -> i` where the last marked node is not `tn` but some already marked `j`, then by I1, we know the path `0 -> ... -> tn -> ... -> j` cannot be shorter than `m[j]`, who has already contributed to the minimisation of `m[i]`, so path `P` can never contribute a smaller value.
Since the old value of `m[i]` is the least all-marked distance when node `tn` was unmarked, now we have taken the only newly-marked node `tn` into consideration, `m[i]` becomes the current least all-marked distance from 0, re-establishing I2.

H: This code checks all unmarked nodes to find out the m-least one and assign it to `tn`, thus re-establishing I3. (It will always update `tn` with an unmarked node from `un`, because the `m` values are always less than or equal to INFTY, as they can only be decreased in the first for loop but never increased.)

I: It is not necessary for termination, because in every iteration another unmarked node `tn` will be removed from `un` (we can guarantee this from question H), when `un` becomes empty, the loop will be terminated by the first `break`.
For any unmarked node `v`, if `m[v] == ∞`, there does not exist any edge from some marked node `u` to `v` (otherwise `m[v]` must have been decreased in the first `for` loop after `u` has been marked).
If `minD==INFTY`, all unmarked nodes have infinity `m`, so there is no edge from any marked node to any unmarked node, meaning that all remaining (unmarked) nodes are unreachable, therefore the statement allows the algorithm to terminate as early as possible.

J: I1 is never broken anywhere in the loop.

K: If the loop terminates at the first `break`, no nodes are left unmarked, so I2 holds. If it terminates at the second `break`, I2 has been re-established in question G and not broken afterwards, so I2 still holds.

L: From the termination condition, if `un` is not empty, then the minimum `m` of all nodes in `un` must have been INFTY, but INFTY is the largest possible value, so they must all be INFTY.

M: I1 holds, so `m` is correct for all marked nodes.
For any unmarked node `i`, I2 holds, so `m[i]` is the least all-marked distance, but from the previous question we know it is INFTY, so, no all-marked path exists from 0 to any unmarked node. On the other hand, for any unmarked `i`, if there exists a not-all-marked path `P: 0 -> ... -> x -> y -> ... -> i`, where `x` is the last marked node (can be 0) and `y` is the first unmarked node (can be `i`), then by definition the path `P': 0 -> ... -> x -> y` is an all-marked path, which contradicts with the last conclusion (no all-marked path exists from 0 to any unmarked node). So, neither all-marked nor not-all-marked path exists from 0 to `i`, meaning node `i` is unreachable. Therefore `m[i] == INFTY` is correct.