Problem 1: We can use array to do this:

① initialize (m):

create an array values of size m

set all elements in values to 0

· The runing time is O(m) because it creates an array of size "m", which takes linear times related to the size of array.

② update (i, x):

values[i] ← x

updateValues[i] ← x

updateCount[i] += 1

latestUpdateIndex ← i

· The runing time is O(1) because the update operation involves direct access to the array index.

③ get (i):

return values[i]

· The runing time is O(1) because accessing the value at a specific index in an array is a constant-time operation.

④ undo():

i ← latestUpdateIndex

values[i] ← updateValues[i]

updateCount[i] -= 1

The runing time is O(1) because it directly accesses the information about the latest update.

⑤ reset():

values ← [0] · len(values)

updateValues ← [0] · len(updateValues)

updateCount ← [0] · len(updateCount)

· The runing time is O(1) because it involves creating new arrays and the size of the array is constant.

⑥ count_undo():

return sum(updateCount)

· The runing time is O(1) because it involves summing the elements in the "updateCount" array, which is constant time.