A linked list is given such that each node contains an additional random pointer which could point to any node in the list or null.

Return a deep copy of the list.

Solution: This problem can be solved in two ways. First is to use a map to map the old node to new node so that we know which node we’ve to point of a random pointer. But this takes O(N) space. If we’re to do it without extra memory, we’ll need to somehow access to access the corresponding random node which the original node’s random pointer points to. I mean, through original node’s random pointer, I should be able to access the copy of the original node pointed by random pointer. This can only be done with the copy nodes being the next node in the linked list. Thus, solution is

* Add the copy node as the node’s next node. I.,e keep to copy node next to the original node.
* Now, adjust the random pointer of the copy node. I.e., copy node’s random pointer will be original node’s random pointer’s next node.
* Now, adjust the pointer in the list to make them two different lists. I.e., Carefully remove the stitches to separate out the original node’s and copy nodes.

So, it requires three iterations of the list. So O(N) with O(1) memory.