Q1. Reverse a linked list.

i/p → 1 → 2 → 3 → 4 → X

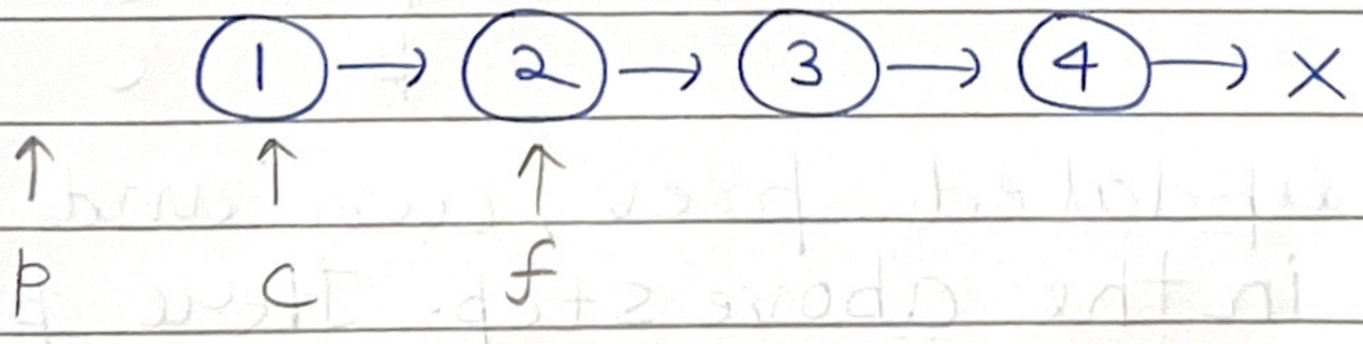
o/p → 4 → 3 → 2 → 1 → X

Here we will be maintaing 3 pointers namely previous, current & forward. For reversing a linked list, there are typically some steps

1) Connection of the current node to previous.
2) Updating previous node.
3) Updating current node.
4) The above 3 steps run until we /curr reaches NULL.
5) Return previous which will be the new node.

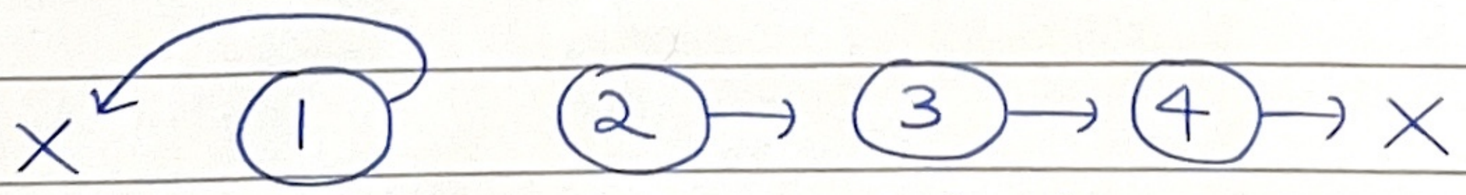
Dry run

1)



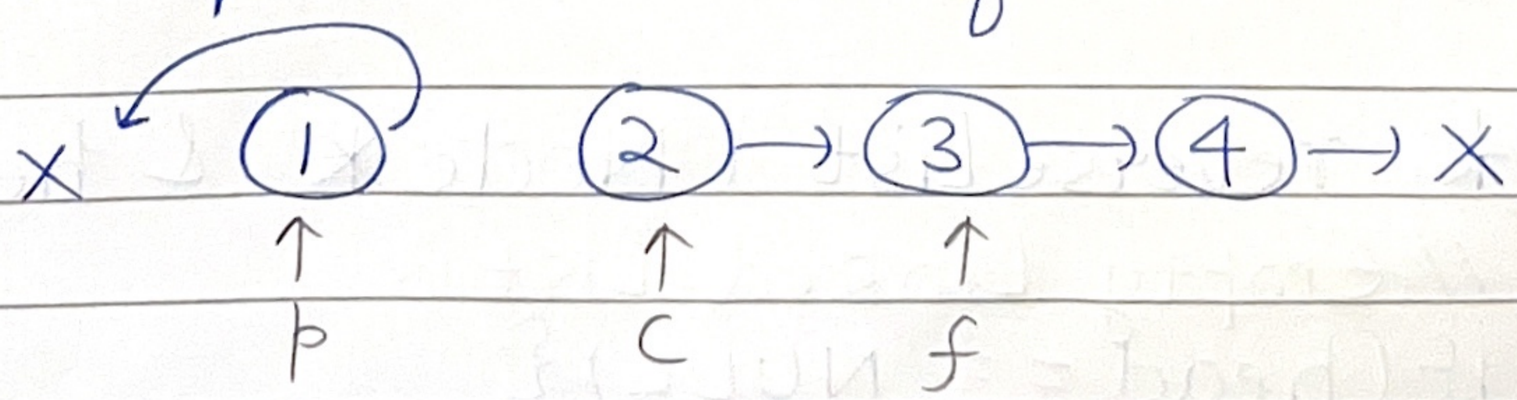
First off connect curr to the prev.

curr → next = prev.

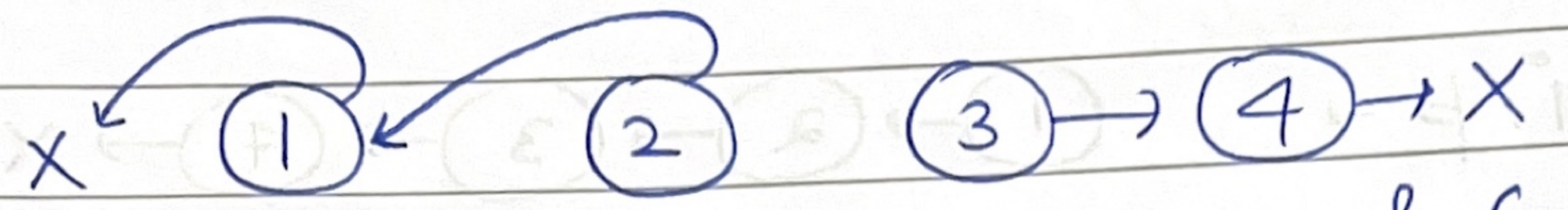


Update prev, curr & forward.

2)

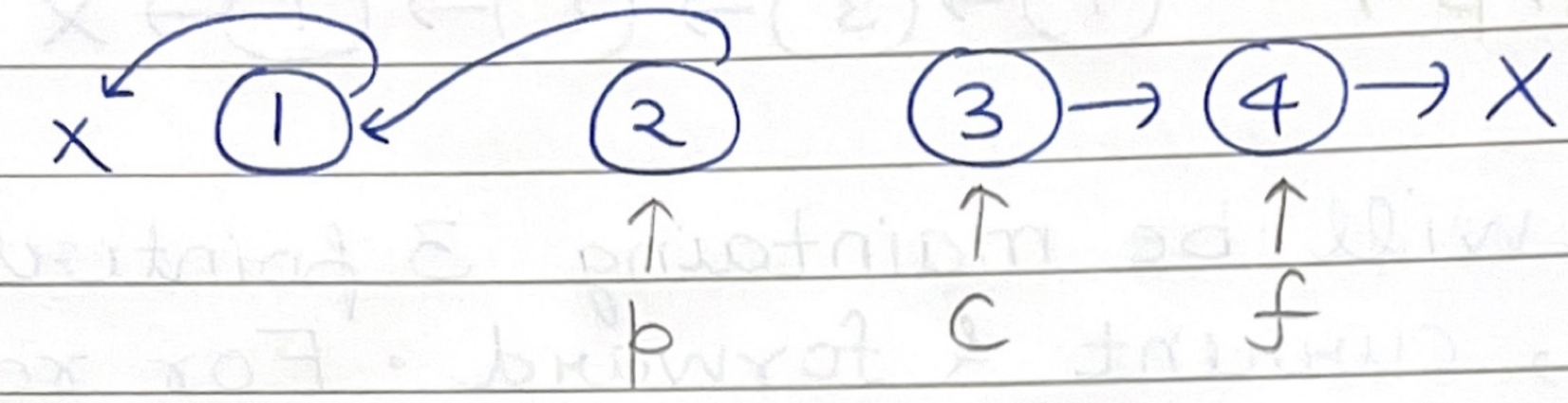


Connect curr to prev

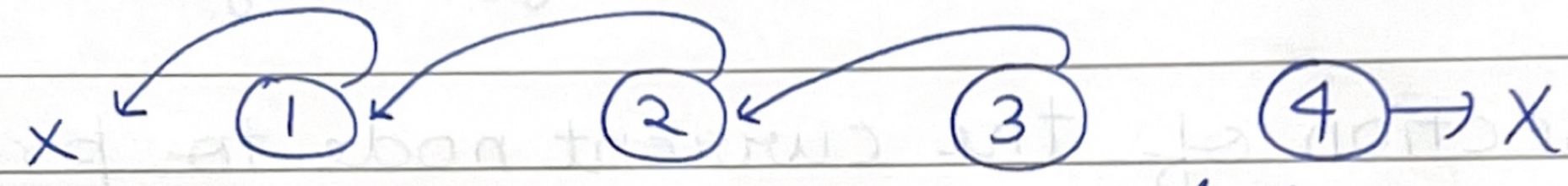


Now update prev, curr & forward.

3)

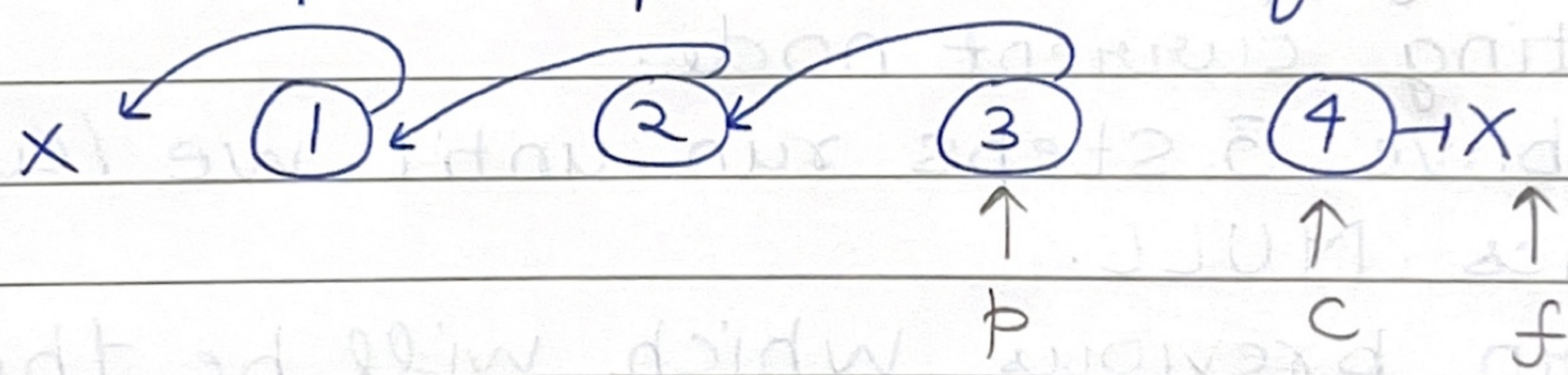


Connect curr to prev

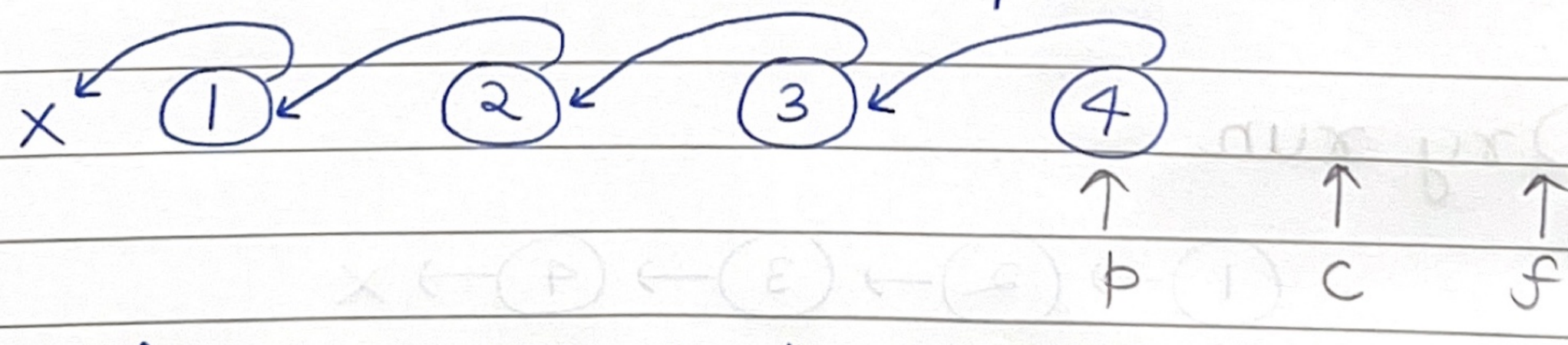


Now update prev, curr & forward.

4)



Now connect curr to prev.



We have updated prev, curr and forward in the above step. Here prev is returned as the new Head.

Code

(i) Iterative

```
Node* reverseList (Node* &head){
    // Empty Linked List
    if(head == NULL){
```

```
        return head; // Nothing to reverse
    }

    // Maintaing prev & curr pointers
    Node* prev = NULL;
    Node* curr = head;
    // Run loop until curr != NULL
    While (curr != NULL) {
        // 3rd pointer
        Node * forward = curr -> next;
        curr -> next = prev; // Connection
        prev = curr; // Updating prev
        curr = forward; // Updating curr
    }
    return prev; // new Head is prev
}
```

(ii) Iterative → ✓, Recursive implementation

```
Node* reverseList (Node* & prev, Node* &
                   curr) {
    // Base case
    if (curr == NULL) {
        return prev; // new head
    }
    // Maintaing forward (1 case solve)
    Node * forward = curr -> next;
    curr -> next = prev;
    // Recursive relation
    return reverseList (curr, forward);
}
```

Time complexity = O(n)
Space complexity = O(n) → Recursive else O(1)