Operations on singly-linked lists without an encapsulating class

Here we will work with lists simply as a reference to the first node.

- To begin you must have an implementation of the class **Node** and some functions to add to it and print it, so that you can test the following functions.
    - *This is the assignments from the previous class. Start with some of those.*
- Make a function that takes a node (***head***) and returns the length of the list (# of nodes)
    - Can you do it both iteratively and recursively?
        - Which is better?
    - Does your implementation work if the node that is sent in is ***None*** (empty list)?
    - *How much is changed so that the function returns the sum of the list's values?*
- Implement an operation that takes a node (***head*** *of an ordered list*) and a value (***data***) as parameters and adds the item in the correct location in the list, so that it is still ordered.
    - This should be able to take an empty list, so that you can use it to fully populate an ordered list.
    - *Is this one more elegant when done recursively?*
- Implement a function that takes a node (***head***) as a parameter and returns a node, the ***head*** of a list that has the same items as the previous list, but in reverse order.
    - Can you do this by using all the same nodes, not by making new ones?
        - Only change their ***next*** links.
        - Could you do that with previous operations as well (i.e. ***merge_lists***)?
- Implement an operation (**merge_lists**) that takes two nodes (the heads of two ***ordered*** lists) as parameters and returns one node, the head of a list which has all the elements from the other two in one ordered list.
    - *Make this one recursive!*
    - *This is fairly advanced. Skip ahead, if having trouble.*
- Implement ***merge_sort*** on a singly-linked list, splitting the list recursively into halves until each part has only 1 or 0 items, then using ***merge_lists*** to reconnect them ordered.
    - *This is advanced. It's OK to finish the linked_list class assignment below first.*
    - It can be tricky when a list is down to 2 items. Make sure the split happens between them, not behind the second one, as that will give another list of length 2, which results in endless recursion. *Video lecture is not definitive enough :)*

Nodes and lists with an encapsulating class

Here we work with a class that has nodes as instance variables and optional helper variables.

- Make a class that implements a ***linked_list*** using singly-linked list nodes
    - The following operations should be implemented:
        - ***push_front(data)***
        - ***pop_front()*** (returns data)
        - ***push_back(data)***
        - ***pop_back()*** (returns data)
        - ***get_size()*** (returns the size) *(what is the best implementation?)*
        - ***__str__(self)*** (returns a string with all the items)
    - What is the time complexity of each of these operations?