You are not allowed to use any other data structures for the purposes of manipulating/sorting elements, nor may you use any modules from the Python to insert and remove elements from the list.
You are not allowed to swap data from the nodes when adding the node to be sorted. Traversing the linked list and updating 'next' references are the only required and acceptable operations
You are not allowed to use any kind of built-in sorting method or import any other libraries

add(self, item)
adds a new Node with value=item to the list making sure that the **ascending** order is preserved. It needs the item and returns nothing but modifies the linked list. The items in the list might not be unique, but you can assume every value will be numerical (int and float). You are not allowed to traverse the linked list more than once (only one loop required).

Input (excluding self)

int or float	*item*	A numerical value that represents the value of a Node object

Examples:

```
>>> x=SortedLinkedList()
>>> x.add(8.76)
>>> x.add(7)
>>> x.add(3)
>>> x.add(-6)
>>> x.add(58)
>>> x.add(33)
>>> x.add(1)
>>> x.add(-88)
>>> print(x)
Head:Node(-88)
Tail:Node(58)
List:-88 -> -6 -> 1 -> 3 -> 7 -> 8.76 -> 33 -> 58
```

replicate(self)
Returns a new SortedLinkedList object where each element of the linked list appears its node's value number of times (3 -> 1 results in 3 -> 3 -> 3 -> 1). Negative numbers and floats are repeated only once immediately after that node. For 0, the node is added, but not repeated. Method returns None if the list is empty. *Hint*: Using the add method from above could be very useful here!

Output

SortedLinkedList	A new linked list object that repeats values without disturbing the original list
None	None keyword is returned if the original list is empty

Examples:

```
>>> x=SortedLinkedList()
>>> x.add(4)
>>> x.replicate()
Head:Node(4)
Tail:Node(4)
List:4 -> 4 -> 4 -> 4
>>> x.add(-23)
>>> x.add(2)
>>> x.add(1)
>>> x.add(20.8)
>>> x
Head:Node(-23)
Tail:Node(20.8)
List:-23 -> 1 -> 2 -> 4 -> 20.8
>>> x.replicate()
Head:Node(-23)
Tail:Node(20.8)
List:-23 -> -23 -> 1 -> 2 -> 2 -> 4 -> 4 -> 4 -> 4 -> 20.8 -> 20.8
>>> x.add(-1)
>>> x.add(0)
>>> x.add(3)
>>> x.replicate()
Head:Node(-23)
Tail:Node(20.8)
List:-23 -> -23 -> -1 -> -1 -> 0 -> 1 -> 2 -> 2 -> 3 -> 3 -> 3 -> 4 -> 4 -> 4 -> 4 ->
20.8 -> 20.8
>>> x
Head:Node(-23)
Tail:Node(20.8)
List:-23 -> -1 -> 0 -> 1 -> 2 -> 3 -> 4 -> 20.8
>>> x.add(2)
>>> x.replicate()
Head:Node(-23)
Tail:Node(20.8)
List:-23 -> -23 -> -1 -> -1 -> 0 -> 1 -> 2 -> 2 -> 2 -> 2 -> 3 -> 3 -> 3 -> 4 -> 4 -> 4
-> 4 -> 20.8 -> 20.8
```

removeDuplicates(self)

Removes any duplicate nodes from the list, so it modifies the original list. This method must traverse the list only once. It modifies the original list.

Output

None	Nothing is returned when the method completes the work

Examples:

```
>>> x=SortedLinkedList()
>>> x.removeDuplicates()
>>> x
Head:None
Tail:None
List:
>>> x.add(1)
>>> x.add(1)
>>> x.add(1)
>>> x.add(1)
>>> x
Head:Node(1)
Tail:Node(1)
List:1 -> 1 -> 1 -> 1
>>> x.removeDuplicates()
>>> x
Head:Node(1)
Tail:Node(1)
List:1
>>> x.add(1)
>>> x.add(2)
>>> x.add(2)
>>> x.add(2)
>>> x.add(3)
>>> x.add(4)
>>> x.add(5)
>>> x.add(5)
>>> x.add(6.7)
>>> x.add(6.7)
>>> x
Head:Node(1)
Tail:Node(6.7)
List:1 -> 1 -> 2 -> 2 -> 2 -> 3 -> 4 -> 5 -> 5 -> 6.7 -> 6.7
>>> x.removeDuplicates()
>>> x
Head:Node(1)
Tail:Node(6.7)
List:1 -> 2 -> 3 -> 4 -> 5 -> 6.7
```