```
class Node:

    def __init__(self, element, prev, next):
        self._element = element
        self._prev = prev
        self._next = next


class SortedList:

    def __init__(self):
        self._head = Node(None, None, None)
        self._tail = Node(None, None, None)
        self.size = 0
        self._head._next = self._tail
        self._tail._prev = self._head

    def is_empty(self):

        return self.size == 0

    def size(self):

        return self.size

    def insert(self, element):

        if self.size == 0:
            newElement = Node(element, self._head, self._tail)
            self._head._next = newElement
            self._tail._prev = newElement

        else:
            newElement = Node(element, None, None)
            pointer = self._head._next

            while pointer is not self._tail and element > pointer._element:
                pointer = pointer._next

            newElement._next = pointer
            newElement._prev = pointer._prev
            pointer._prev._next = newElement
            pointer._prev = newElement

        self.size += 1

    def delete(self, element):
        pointer = self._head._next
        while pointer._next is not self._head and pointer._element is not element:
            pointer = pointer._next
```

```
        pointer._next._prev = pointer._prev._next
        pointer._prev._next = pointer._next

    def printList(self):
        pointer = self._head._next
        result = "["
        while pointer is not self._tail:
            result += str(pointer._element) + " "
            pointer = pointer._next
        result += ']'
        print(result)

    def printReverse(self):
        pointer = self._tail._prev
        result = "["
        while pointer is not self._head:
            result += str(pointer._element)
            pointer = pointer._prev
        result += "]"
        print(result)

    def find(self, element):
        #
        # Executes a sequential search of the Doubly Linked List starting at
        # self._head._next and continuing until element is found, element has a
        # greater value than the current Node's element, or self._tail is reached.
        #
        # self The implicit parameter for the list
        #
        # element A reference to the search value
        #
        # Returns a list with 2 values – the first value is True if element was
found,
        #                                 False otherwise - the second value is the
        #                                 number of locations visited during the
search.
        #
        temp = self._head._next
        visit = 0
        while temp._next is not None and element >= temp._element:
            if temp._element == element:
                return [True, visit + 1]
            temp = temp._next
            visit += 1

        return [False, visit+1]
```