Lab 4

This lab provides an introduction to doubly-linked lists. In particular, you are asked to implement a circular doubly-linked list where the elements are in order.

1 Doubly Linked Ordered List

The structure of an ordered list is a collection of items where each item holds a relative position that is based upon some underlying characteristic of the item. The ordering is typically either ascending or descending, and we assume that list items have a meaningful comparison operation that is already defined. Many of the ordered list operations are the same as those of the unordered list.

Implement the following operations for an ordered list of items **ordered in ascending order** using a **circular doubly linked list**. That is to say the "first" item in the list will be the smallest and the "last" item in the list will be the largest.

Your implementation should use a **single dummy node** as the "head" as seen in class.

- `insert` This function takes an ordered list and a value as arguments and inserts the value into the proper location in the list.

  This function *must* have $O(n)$ performance in the worst case.

- `remove` This function takes an ordered list and a value as arguments and removes the value from the list. If the value is not in the list, then this function should raise a `ValueError`.

  This function *must* have $O(n)$ performance in the worst case.

- `contains` This function takes an ordered list and a value as arguments and returns whether or not the value is in the list (`True` if it is, `False` otherwise).

  This function *must* have $O(n)$ performance in the worst case.

- `index` This function takes an ordered list and a value as arguments and returns the index of the first occurrence of the value in the list. If the value is not in the list, then this function should raise a `ValueError`.

  This function *must* have $O(n)$ performance in the worst case.

- `get` This function takes an ordered list and an index as arguments and returns the value at the given index. If the index is outside the bounds of the list, then this function should raise an `IndexError`.

  This function *must* have $O(n)$ performance in the worst case.

- `pop` This function takes an ordered list and an index as arguments and removes (and returns) the value at the given index. If the index is outside the bounds of the list, then this function should raise an `IndexError`.

  This function *must* have $O(n)$ performance in the worst case.

- `is_empty` This function takes an ordered list as an argument and returns whether or not the list is empty (`True` if it is, `False` otherwise).

  This function *must* have $O(1)$ performance in the worst case.

- `size` This function takes an ordered list as an argument and returns the number of items in the list.

  This function *must* have $O(1)$ performance in the worst case.

Note that you can (and should) assume that all items added to your list are comparable with the `<` operator and can be compared for equality with `==`. You cannot make *any* other assumptions about the items in your list.

2 Testing

As mentioned in the syllabus, your code is periodically graded prior to the deadline and you will receive automated feedback based on the results of my tests. *But*, in order to receive *any* feedback, your tests much provide 100% test coverage of the code you are submitting. If you do not have 100% coverage, this is the only feedback you will receive.

100% test coverage means that every line of your code is run at some point in some test. Imagine if that weren't the case. That means that you have a line of code (or multiple lines of code) that could do anything, and you'd never know. You never tested them!

Your tests will go in files named `ordered_list_tests.py`.

3 GitHub Submission

Push your finished code back to GitHub. Refer to Lab 0, as needed, to remember how to push your local code.