Project 5

100 points
Due March 15th by 8:00pm
This is not a team work, do not copy somebody else's work.
Reference: Author of this project is Cyndy Ishida

Assignment Overview
Quick Sort is often the preferred sorting algorithm for large scale applications. This assignment will be an implementation of this famous algorithm but, instead of a contiguous array, it will be a done with a Linked List Queue.

Assignment Deliverables
Be sure to use the specified file name(s) and to submit your files for grading **via D2L Dropbox** before the project deadline.

- Queue.py
- QuickSort.py

Assignment Specifications
Your task will be to complete the methods listed below.

in Queue.py

- **__len__(self)**
  - return (type int) the number of elements present in queue
  - O(1) space
  - O(1) time
- **is_empty(self)**
  - return (type Boolean) true if queue is empty
  - O(1) space
  - O(1) time
- **dequeue(self)**
  - return (type Node.val) element that was just removed from linked list queue * guaranteed built-in scalar type (i.e. ints, floats, chars)
  - O(1) space
  - O(1) time
- **enqueue(self, element)**
  - return (type None)
  - add element to end of linked list queue
  - O(1) space
  - O(1) time

- **get_middle(self)**
  - return (type Node.val) * guaranteed built-in scalar type (i.e. ints, floats, chars) return the element at the self.size//2 position
  - O(N) time
  - O(1) space

in QuickSort.py

- **insertion_sort(queue)**
  - parameter: queue (type LinkedQueue) the queue to sort
  - return (type None)
  - O(N^2) time, same as lecture, best case should be O(N)
  - O(1) space
  - Sorting should modify the given container
- **pick_pivot(queue)**
  - parameter: queue (type LinkedQueue) the queue
  - return (type Node.val) * guaranteed built-in scalar type (i.e. ints, floats, chars) median element between first element, the last element, and the middle index
  - O(N) time
  - O(1) space
  - this function should never be run with less than 3 elements in queue
- **quick_sort(queue)**
  - parameter: queue (type LinkedQueue) the queue to sort
  - return (type None)
  - O(n*log(n)) time
  - O(n*log(n)) space
  - Sorting should modify given container
  - Calls insertion_sort when queue is less than or equal to **10** elements

Assignment Notes

Points will be deducted if your solution has any warnings of type:

- Any use of a container type that isn't a LinkedQueue Type will result in a 50% decrease to final grade for assignment
- You are not allowed to use any outside module like 'import sys' or to modify the set recursion limit
- Mutators for Queue method will not be called on an empty list. i.e. error checking for empty queue is not needed.
- Elements for each node will only be built-in scalar types (int, float, chars)
- You are guaranteed that at least 5 elements will exist in queue to be sorted
- There is a sample testcase provided, that doesn't count for points 'testcase00.txt'
- You can add additional functions, however you aren't allowed to modify an function signatures
- Docstrings/Pre-Post Conditions are required on all function/method signatures.
  Example for Pre-Post Conditions Documentation:

```
def correctAnswers(questionNumber):
    """precondition: number of questions must be between 0 and maxNumQuestions
       postcondition: Updates game status to indicate that question number
questionNumber was answered correctly
    """
```

  If a method does not contain a precondition/postcondition simply state none.
  - For more examples on pre post conditions check this link out. (https://www.python.org/dev/peps/pep-0316/)

- It is the assumption you are using the python3.6 interpreter

Testing your work

Run your project on Pycharm see sample run below

```
Please Enter File Name: testcase00.txt
0.0, 1.0, 4.0, 5.0

Process finished with exit code 0
```

```
Please Enter File Name: testcase01.txt
0.0, 1.0, 1.0, 1.0, 2.0, 3.0, 4.0, 5.0, 7.0, 7.0, 7.0, 7.0, 8.0, 9.0, 10.0

Process finished with exit code 0
```