HW5: Stacks and Queues

Problem 1: Implement Queue using a dictionary (20 points)

In our Queue implementation using lists either the enqueue or the dequeue had time complexity $O(n)$. Your job is to build a more efficient Queue by using a dictionary rather than a list as the internal data structure. Create a Queue class in a file called **EfficientQ.py**. Your Queue must implement ALL the methods described for a Queue. Think about the following when designing your solution:

a. A Queue has to keep track of its head and tail. You may find it helpful to store this information as attributes of self (i.e. `self.head, self.tail`)
b. Items are removed from the Queue in order of their arrival.
c. It may be helpful to think about the deli counter at a grocery store where customers are essentially put into a Queue. When a customer arrives, they grab a ticket with a number. The deli serves customers calling out ticket numbers starting with the lowest numbered ticket.
d. A dictionary can store a key value pair.
e. In the grocery deli example, customers are essentially being placed into a dictionary with their ticket numbers as keys.

(5 points) Write a main function with tests for each method of the Queue class. Make sure you list the expected results next to each test. Make sure you test your Queue thoroughly.

(4 points) Write header comments (using Purpose, input, output and assumptions) for `enqueue`, `dequeue`, `isEmpty` and `size` methods.

Problem 2 (15 pts)

Complete the program Baggage.py, which has already been started for you. It simulates luggage handling for a large aircraft. As bags arrive they are loaded into containers. Each bag has a different weight and containers can hold a maximum amount of weight. When a container has reached its maximum weight capacity, or if the next bag in the line will cause it to exceed its weight capacity, or there are no more bags, the container is sent to be loaded onto the aircraft.
At the aircraft loading station, containers are unloaded such that the last container which arrives will be the first to be unloaded. The bags in each container are unloaded in the same order as they were loaded, i.e., first in, first out.

Your program Bagage.py should prompt the user for a filename and the maximum container weight. The file will contain a list of bag weights in the order that the bags arrive.
Your program should output the weights of the bags in the order they are unloaded. Functions for obtaining user input and reading file data are already completed for you in Baggage.py. Your job is to complete the two functions (denoted with TODOs) to compute the aircraft loading order.

Two sample data files, bags25.dat and bags100.dat, are provided for you to test your program.
For example, here are the contents of the file bags25.dat:

```
16 24 25 3 20 18 7 17 4 15 13 22 2 12 10 5 8 1 11 21 19 6 23 9 14
```

Here is sample output from your program:

```
Enter filename:
```

```
>>bags25.dat
Enter container weight:
>>100
The unloading order is:
23 9 14 12 10 5 8 1 11 21 19 6 18 7 17 4 15 13 22 2 16 24 25 3 20

Enter filename:
>>bags25.dat
Enter container weight:
>>50
The unloading order is:
23 9 14 21 19 6 2 12 10 5 8 1 11 15 13 22 18 7 17 4 25 3 20 16 24
```

The files necessary for this problem are:

- bags25.dat
- bags100.dat
- QueueDS.py
- StackDS.py
- Baggage.py