Project 2(GrellSim) Milestone 1 analysis.

Preface: this analysis attempts to prove why each public method in my GrellFindableQueue class runs in the allotted time.

Method 1. (Contains):
In this method, the allotted time for this is amortized O(1) time. My implementation of this method does meet the time requirement because I am using a HashMap to check if the todo queue has the item in it. This method runs in amortized O(1) time.

Method 2. (Length):
In this method, the allotted time for this is O(1) time. My implementation of this method does meet the time requirements, because on insert, remove, and pop, the global variable(size) is updated accordingly. Therefore when length() is called. all it is doing is retiring a variable

Method 3. (get):
In this method, the allotted time for this is O(1) time. My implementation of this method does meet the time requirements, because I am using a hashMap to look up whether or not the element is contained in the todo queue. HashMap.get() runs in amortized O(1) time. Therefore, this method runs in O(1) time.

Method 4. (isEmpty):
In this method, the allotted time for this is O(1) time. My implementation of this method does meet the time requirements, because I am checking isEmpty on a hashMap. This is known to run in O(1) time. Therefore my method runs in O(1) time.

Method 5. (insert):
In this method, the allotted time for this is O(log(n)) time. My implementation of this method does meet the time requirements, because, 1. I am not using any methods that run in over O(log(n)) time.(ex. IndexOf is not used) 2. when doing the swaps. the node will have to be at most, moved up in the arrayList at a speed of O(log(n)) time, because it just has to go up one branch of the tree. Therefore, my method runs in O(log(n)) time.

method 6. (first):
In this method, the allotted time for this is O(1) time. My implementation of this method does meet the time requirements, because all it has to do is grab the top element of the array list at index 0. Insert ensures that there is no shifting that needs to be done in first() so this method runs in O(1) time.

method 7. (pop):
In this method, the allotted time for this is O(log(n)) time. My implementation of this method does meet the time requirements, because when removing the 0th element from the arraylist. my implementation grabs the last element, switches it with the element to remove then percolates that element down the arraylist. this is known to run in O(log(n)) time because it has to go through at most one branch of the arraylist.

method 8. (remove):
In this method, the allotted time for this is O(log(n)) time. My implementation of this method does meet the time requirements, because when removing the nth element from the arraylist. my implementation grabs the last element, switches it with the element to remove then percolates that element down the arraylist. this is known to run in O(log(n)) time because it has to go through at most one branch of the arraylist. As well, the index is already known because it is stored in a hashmap that has amortized O(1) look up time. So indexOf is not needed here.