Conceptual Questions:

1. The starter code uses arraylists for every collection. Given that, what are the worst/best time complexities of the methods below? Also briefly describe the best case scenario for each method.
    a. `addPostToDatabase(User u, Post p)`
    b. `computeMostUrgentQuestion()/retrievePost(User u)`
    c. `deletePostFromDatabase(User u, Post p)`
2. What are the pros of using arraylists? Cons?

3. Explain any optimization you made on your code (or at least any optimization that the course provided) and explain how each optimization contributes in reducing the runtime of relevant methods. Evaluating the baseline runtime complexities for all the benchmarking methods, especially the ones listed on question1, will help you get some ideas.

1. a. `addPostToDatabase(User u, Post p)`
    - Best time complexity:
        - this.users.contains(u) takes O(1), this.status.equals("inactive") takes O(1), this.posts.add(p) takes O(1), u.posts.add(p) takes O(1), u.numOfPostSubmitted++ takes O(1), this.keywordForest.insert(p) takes O(1), this.unansweredQueue.add(p) also takes O(1)
        - The best case time complexity would be O(1). The best case scenario is when the .contains finds the user right away (for instance, the user is the first element in the users arraylist), .add doesn't need to resize, .insert on the forest in uniform among all keywords.
    - Worst time complexity:
        - this.users.contains(u) takes O(n), this.status.equals("inactive") takes O(1), this.posts.add(p) takes O(n), u.posts.add(p) takes O(n), u.numOfPostSubmitted++ takes O(1), this.keywordForest.insert(p) takes O(n), this.unansweredQueue.add(p) also takes O(n)
        - The worst case time complexity would be O(n)

    b. `computeMostUrgentQuestion()/retrievePost(User u)`
    - Both best time complexity and worst time complexity for both of these methods is O(n) because in both best and worst cases:
        - In computeMostUrgentQuestion, we need to loop through the entire unanswered arraylist and find the most urgent question which results in O(n).

- In retrievePost, we need to loop through the entire posts of the users, which takes O(n) time.
- The best case scenario for these two methods is the same as worst case scenario, where we would need to run the for loop for both of the methods.

c. `deletePostFromDatabase(User u, Post p)`

- Best time complexity:
  - u instanceof Instructor takes O(1), if (this.posts.contains(p)) takes O(1), this.posts.remove(p) and this.unanswered.remove(p) takes O(1).
  - The best case time complexity would be O(1). The best case scenario is when the .contains finds the post right away (for instance, that post is the first element in the arraylist of posts), .remove is the last element in the posts and unanswered arraylists.
- Worst time complexity:
  - u instanceof Instructor takes O(1), if (this.posts.contains(p)) takes O(n), this.posts.remove(p) and this.unanswered.remove(p) takes O(n)
  - The worst case time complexity would be O(n).

2. Pros and cons of using arraylists
   - Pros:
     - It's a resizable array. For a traditional array, once we specify its size, we can't change it. We would have to create a whole new array with more size and run a for loop and add each element plus the new element to the new array. Arraylists are resizable so we don't have to specify its size from the beginning.
     - We can make index-based access. For a linkedlist, for instance, we don't have index-based access. We need to start from the head and iterate through. For arraylists, however, we can do arr.get(2) and get to the 3rd element.
     - We can remove elements at any index. For a traditional array, we can't delete an element of a certain index but in arraylist, we have the flexibility of removing an element at any index.
   - Cons:
     - It cannot hold primitive data types. Unlike traditional arrays which can hold primitive data types like int, arraylist requires we turn primitive data types like int into objects like Integer. This is because Arraylist is implemented using Objects.

- As mentioned in pros, we can remove elements from any index but removing it takes O(n) because after removal of any element other than the last, the data points need to be shifted down.

3. Explain any optimization you made on your code (or at least any optimization that the course provided) and explain how each optimization contributes in reducing the runtime of relevant methods. Evaluating the baseline runtime complexities for all the benchmarking methods, especially the ones listed on question1, will help you get some ideas.
    - On PiazzaExchange class, I added
      HashMap<String, ArrayList<Post>> keywordHash;
      In this keywordHash, string is the keyword associated with a post and ArrayList<Post> is the arraylist of posts that we add when we run addPostToDatabase. I used this hashmap for retrievePost(String keyword) method because I saw that we output all the posts that contain the keyword and hashmap is very efficient at querying values using key. I added all the posts containing the same keyboard in the same arraylist which counted as a value for the keyword. On the retrievePost(String keyword), I called this.keywordHash.get(keyword). This get operation's time complexity is O(1) with the assumption that the posts are well distributed across the buckets.

      Before, I had to loop through each post in the posts arraylist to see if a particular post had the same keyword and add it to an arraylist. I would then have to loop through that arraylist to turn it into an array. This was 2 * O(n). Using hashmap, the get operation was only O(1). Unfortunately, the return type was array so after getting the arraylist of posts with the keyword, I needed to do .toArray(new Post[0]) to turn it into an array which took O(n). However, hashmap is still faster than my previous implementation.