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)`
        i. Worst-case time complexity: O(length of users)
        ii. Best case time complexity: O(1)
        iii. Best case scenario: User u is the first element in the users arraylist
    b. `computeMostUrgentQuestion(int k)/retrievePost(User u)`
        i. Compute most urgent Worst-case time complexity: O(length of unanswered questions arraylist)
        ii. Compute most urgent Best case time complexity: O(length of unanswered questions arraylist)
        iii. Compute most urgent Best case scenario: the algorithm will always have to iterate through the entire arraylist to check if there's a more urgent question.
        iv. RetrievePost worst-case: O(length of posts)
        v. RetrievePost best-case: O(length of posts)
        vi. RetrievePost Best case scenario: you have to loop through the entire arraylist to find the posts related to a certain user, which will be O(length of posts)
    c. `deletePostFromDatabase(User u, Post p)`
        i. Worst-case time complexity: O(length of posts arraylist)
        ii. Best case time complexity: O(length of posts arraylist)
        iii. Best case scenario: Same as the worst case, since deleting from an arraylist takes O(n) because it needs O(n) to delete an element.
2. What are the pros of using arraylists? Cons?
    a. Pro: arraylists are built into java, so users don't need to implement them. Also, arraylists keep the elements in insertion order. You can also have duplicates in an arraylist, while you can't have it in a set.
    b. Con: arraylists take O(n) for worst-case and average-case to search for a certain element, which means the same for deletion of a specific element. Since Piazza

Exchange uses a lot of searching, arraylists are not the optimal data structure to build the Piazza Exchange.

3. Explain any optimization you made on your code 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.
    a. HashMap<String,User> users;
        i. I used a hashmap to implement user list because each user has a unique PID. By switching from arraylist to a hashmap, we can query user with the worst case time complexity of O(1), which is a lot faster than O(length of users) when there are a lot of users.
    b. HashMap<String, ArrayList<Post>> posts;
        i. I used an arraylist inside a hashmap to implement posts list. Each user has a unique PID, and the arraylist of posts are the posts made by that user. By switching from arraylist to a hashmap, we can query all the posts by a certain user with the worst case time complexity of O(1), which is a lot faster than O(length of users) when there are a lot of users. If we need to query a certain post, the worst case would be O(nuber of posts) if all the posts in the piazza are made by that person. This could happen, but it's very unlikely. If we assume that posts are evenly distributed among all users, we will have the worst case = number of posts / number of users, which is a lot less than the number of posts when there are more than one user in the piazza.
    c. PriorityQueue<Post> unanswered;
        i. I used a priority queue to implement unanswered questions because we want to obtain the unanswered questions with the top priority over and over again. By implementing unanswered arraylist using priorityqueue, we will be able to obtain the maximum priority post with the worst case time complexity of O(logn) compared to O(n)