1.

a. O(1) the worst case and the best case is the same, since insertion into the arraylist is only appending the element at the end. The operation takes constant time.

b. O(n) for computeMostUrgentQuestion(), the best case and worst case is the same either way since query-for-max in arraylist requiring checking every element in the arraylist. For retrievePost(user u) the same argument applies: we always have to loop through the list, checking each element's association with the given user no matter what.

c. O(1) the best case when the first post is the one to be deleted. O(n) the worst case when the last one is the one to be deleted.

2.

Pros: Arraylist has no issue with size, so you can always initialize one empty and populate however many numbers of elements. Arraylist has very fast insertion, which is constant time. Arraylist's elements could be accessed by index, especially when we need to get a subset of the elements at the head/tail, it could be useful.

Cons: Arraylist does not support fast query: it is sorted in the order of the insertion. So whenever we need to search for some particular element, it always have to be on O(n) time complexity. There is a hidden capacity mechanism inside, so whenever the size goes up (i.e. when hitting some threshold), it cost O(n) to repopulate.

3.

a. by introducing priority queue with unanswered questions, we can always reorder the questions according to their priority. It supports very fast query for the extreme value to be of O(1) instead of O(n) given by the baseline implementation with Arraylist.

b. by introducing a hashmap whose keys are the keywords, values are the arraylist of posts. It supports on average O(1) query of posts with specified keywords. We can then reduce the size of posts we need to search since a subset with targeted keywords is taken out from the hashmap.