Abstract Data Structure to Store Flagged Posts

To handle the storage of flagged posts on social media, we carefully analyzed the problem to select an appropriate data structure. The main challenges identified were the **limited processing and storage capacity**, necessitating a solution that is both **memory-efficient and operationally efficient.** After breaking down the requirements, we identified the essential operations for this scenario:

- **Storing an element:** Adding flagged posts to the structure.
- **Retrieving an element:** Accessing the element with the highest priority (without deleting it).
- **Deleting an element:** Removing the processed post.
- **Checking capacity:** Determining the number of elements present in the structure.

After evaluating several options, including balanced binary trees, linked lists, and hashmaps, we chose the **Queue** , more specifically a **Priority Queue** as the most suitable abstract data structure. The primary reason for this choice is the priority queue's ability to process elements based on priority rather than the order in which they were added. This ensures that high-risk posts are always processed before medium- or low-risk ones. For our use case, the priority will be sorted in ascending order of risk level.

The chosen data structure efficiently supports all required operations:

1. **Insertion / Enqueue:** When a new post is flagged and assigned a priority, it is added to the queue. The priority queue ensures that insertion is efficient while maintaining the correct order of priorities.
2. **Retrieve / Peek the Highest-Risk Post:** The regulatory team processes the riskiest posts first. This is efficiently handled in constant time by accessing the top element of the queue.
3. **Remove the Processed Post / Dequeue :** Once a post is reviewed, the priority queue ensures that the next highest-priority post becomes available immediately.
4. **Check size:** Since the regulatory team has a storage limit, checking the current size of the queue is handled in constant time.

This setup solves the problems of limited storage and processing really well. By focusing on the highest-risk posts first, the team ensures that the most important issues are dealt with before anything else, which makes the process much more efficient. Plus, the priority queue is lightweight and and doesn't take up too much memory, so it can handle a lot of flagged posts without slowing down or using too much storage space.