PRIORITY QUEUE ANALYSIS OF COMPLEXITY.

Temporal complexity:

add(): O(log n) - In the worst case, the added element may have to climb through all levels of the complete binary tree, meaning that log(n) steps are required to reach the appropriate position.

remove(): O(log n) - In the worst case, the removed element may have to descend through all levels of the complete binary tree, meaning that log(n) steps are required to reach the appropriate position.

isEmpty(): O(1) - Simply returns a boolean value, which does not depend on the number of elements in the PriorityQueue.

size(): O(1) - Simply returns the value of the size of the PriorityQueue, which is kept up-to-date after each add() and remove() operation.

peek(): O(1) - Simply returns the element at the top of the PriorityQueue, which is always kept in position 1 of the heap array.

Spatial complexity:

PriorityQueue: O(n) - since the size of the heap array depends on the number of elements stored in the PriorityQueue. In the worst case, the heap array will have a length of 2n if all elements are distinct, and as a result, the spatial complexity will be O(n).

BST ANALYSIS OF COMPLEXITY.

Temporal Complexity:

insert() O(n): In the worst case, the insertion of a node can traverse the entire tree from the root to a leaf. Since the tree has a maximum height of O(n), where n is the number of nodes. Therefore, the temporal complexity of the insert method is O(n).

search() O(n): In the worst case, the search for a node can traverse the entire tree from the root to a leaf. Since the tree has a maximum height of O(n), where n is the number of nodes. Therefore, the temporal complexity of the search method is O(n).

delete() O(n): In the worst case, the deletion of a node can traverse the entire tree from the root to a leaf. Since the tree has a maximum height of O(n), where n is the number of nodes. Thus, the temporal complexity of the delete method is O(n).

inOrder() O(n): The inOrder method traverses all the nodes of the tree and prints them in increasing order, which takes time O(n), where n is the number of nodes.

Spatial Complexity:

BinarySearchTree: O(n) -The spatial complexity of a BST is O(n), where n is the number of nodes in the tree. This is because it has to store each of the nodes in the tree, including its value, key, and pointers to its left and right child nodes. In the worst case, if the tree is completely unbalanced (like a linked list), the number of nodes would be equal to the height of the tree, resulting in a spatial complexity of O(h), where h is the height of the tree, and h is equal to n.