TAD of data structures - Integrative task one

- Priority queue:

TAD <Priority queue>
Abstract object: A Priority Queue is a collection of items in which each item is assigned a numeric or other value that determines its priority. Elements are stored in such a way that the element with the highest priority is always at the head of the queue and is accessed first. These items can be inserted and removed from the priority queue, and can also search for items with higher priority without having to delete them.
{ **inv:** ● The Priority Queue must always be ordered, meaning that the element with the highest priority should be at the head of the queue. ● The Priority Queue cannot contain duplicate elements. }
Primitive operations: ● add: (T element) → void ○ Modifier operation: Because it modifies the structure of the priority queue to Add a new item. ● remove: (No inputs) → T () ○ Modifier operation: Because it modifies the structure of the priority queue to Delete an item. ● isEmpty: (No inputs) → boolean ○ Parser operation: Because it only checks if the priority queue is empty and returns a boolean. ● size: (No inputs) → int ○ Parser operation: Because it only checks the size of the priority queue and returns it. ● peek: (No inputs) → T ○ Parser operation: Because it only checks what the queue peek is and returns that element.

- Binary search tree:

TAD <Generic Binary Search Tree>
Abstract Object: A binary search tree is a tree consisting of a set of nodes, each with a value and a reference to a right and left child node. The tree is organized in such a way that the value of each node is greater than all values in its left subtree and less than all values in its right subtree.
{**inv**: ● For all nodes in the tree, all values in its left subtree are less than the node's value, and all values in its right subtree are greater than the node's value. }
Primitive operations: ● getRoot: (No inputs) → T (Root node) ○ Analyzing operation, as its function is to return the root node of the tree, without modifying it. ● search: key (Node identifier) → T (Object or node being searched for) ○ Analyzing operation, as it searches for an object in the tree by its identifier and returns it if found, without modifying the tree. ● insert: key (Node identifier), value (Complete object) → void ○ Modifying operation, as it modifies the structure of the tree by adding a new node. ● delete: key (Node identifier) → T (Complete object) ○ Modifying operation, as it modifies the structure of the tree by removing a node. ● inOrder: (No inputs) → String (String of the tree in order) ○ Analyzing operation, as it returns a String of the binary tree in order, without modifying it.