**Project Overview**

This project implements a **Magical Tree and Heap System** that integrates two common data structures: **Binary Search Tree (BST)** and **Max Heap**. The project provides an interactive system where users can insert, search, delete, and traverse elements in both a binary search tree and a heap. These operations simulate managing magical artifacts in an enchanted forest and a treasure heap, where the values represent the magical power of the artifacts.

* **Magical Tree (Binary Search Tree)**: The binary search tree is used to manage and organize magical artifacts in the enchanted forest. It follows the property that for any given node, the left child’s value is smaller, and the right child’s value is larger than the parent node.
* **Magical Heap (Max Heap)**: The heap represents a treasure heap, where the artifact with the highest magical power is always at the root. This heap maintains the max-heap property, where each parent’s value is greater than or equal to the values of its children.

The system includes basic operations for both data structures, and the user can interactively perform these operations to manage the magical artifacts.

**Description of Each Functionality**

The project implements the following functionalities, each corresponding to operations on either the **Magical Tree (BST)** or the **Magical Heap (Max Heap)**:

**1. Insert into Tree (BST)**

* **Function**: insertArtifact(int value) in the MagicalTree class
* **Description**: This function allows the user to insert a new artifact (integer value) into the binary search tree. The artifact is placed in the tree such that the left child’s value is smaller and the right child’s value is greater than the parent node.
* **Example Usage**: Insert an artifact with magical power 10 into the enchanted forest.
* **Behavior**: The tree grows with each new insertion, and it ensures that the BST property is maintained.

**2. Search in Tree (BST)**

* **Function**: searchArtifact(int value) in the MagicalTree class
* **Description**: This function searches for an artifact (node) with the given magical power value in the binary search tree. If found, it confirms the artifact is present; otherwise, it notifies that the artifact is not found.
* **Example Usage**: Search for an artifact with magical power 15.
* **Behavior**: The search traverses the tree from the root to find the artifact. The search follows the BST property to narrow down the location.

**3. Delete from Tree (BST)**

* **Function**: deleteArtifact(int value) in the MagicalTree class
* **Description**: This function deletes an artifact from the binary search tree. If the artifact exists in the tree, it is removed, and the tree is restructured to maintain the BST property.
* **Example Usage**: Delete the artifact with magical power 5.
* **Behavior**: The deletion process involves locating the node, adjusting pointers, and ensuring the BST property is maintained after the deletion.

**4. Traverse Tree (BST)**

* **Function**: traverseTree() in the MagicalTree class
* **Description**: This function performs an **in-order traversal** of the tree, which visits nodes in the left, root, right order. It displays the artifacts in ascending order of their magical power.
* **Example Usage**: Display all the artifacts currently in the enchanted forest.
* **Behavior**: The function recursively traverses the left subtree, prints the current node, and then recursively traverses the right subtree.

**5. Insert into Heap (Max Heap)**

* **Function**: insertArtifact(int value) in the MagicalHeap class
* **Description**: This function inserts an artifact with the given magical power into the max heap. The heap property is restored by "bubbling up" the inserted element if necessary.
* **Example Usage**: Insert an artifact with magical power 20 into the treasure heap.
* **Behavior**: The element is added at the end of the heap, and the heap property is restored by moving the element upward until the heap condition is met.

**6. Delete from Heap (Max Heap)**

* **Function**: deleteArtifact() in the MagicalHeap class
* **Description**: This function deletes the root artifact (the one with the highest magical power) from the heap. The last element in the heap replaces the root, and the heap property is restored by "bubbling down" the new root.
* **Example Usage**: Delete the artifact with the highest magical power from the treasure heap.
* **Behavior**: The root element is removed, and the heap is reorganized to maintain the max-heap property.

**7. Display Heap**

* **Function**: displayHeap() in the MagicalHeap class
* **Description**: This function displays the current state of the treasure heap by printing all its elements.
* **Example Usage**: Display the contents of the treasure heap to see the current magical powers.
* **Behavior**: It prints the contents of the heap in a level-order format, which corresponds to the heap's internal structure.

**Main Function:**

* **Description**: The main() function initializes instances of MagicalTree and MagicalHeap. It displays a menu of options for the user and processes user input to perform operations like inserting, searching, deleting, and traversing the tree, or inserting, deleting, and displaying the heap.
* **Menu Options**:
  1. Insert into Tree
  2. Search Tree
  3. Delete from Tree
  4. Traverse Tree
  5. Insert into Heap
  6. Delete from Heap
  7. Display Heap
  8. Exit

**How the Code Works:**

1. The program starts by displaying the welcome message and a menu of options.
2. Based on the user's input, the program calls the appropriate method for either the MagicalTree or MagicalHeap object.
3. The operations on both the tree and heap modify their respective structures, and the program gives feedback to the user (e.g., confirmation messages when inserting or deleting artifacts).
4. The user can repeatedly perform operations until they choose to exit the program.