```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None

class BinarySearchTree:
    def __init__(self):
        self.root = None

    def insert(self, key):
        if self.root is None:
            self.root = Node(key)
        else:
            self.insert_recursive(self.root, key)

    def insert_recursive(self, node, key):
        if key < node.key:
            if node.left is None:
                node.left = Node(key)
            else:
                self.insert_recursive(node.left, key)
        elif key > node.key:
            if node.right is None:
                node.right = Node(key)
            else:
                self.insert_recursive(node.right, key)
        else:
            print(f"Value {key} already exists in the tree.")

    def search(self, key):
```

```
    return self.search_recursive(self.root, key)

def search_recursive(self, node, key):
    if node is None:
        return False
    if node.key == key:
        return True
    elif key < node.key:
        return self.search_recursive(node.left, key)
    else:
        return self.search_recursive(node.right, key)

def delete(self, key):
    self.root = self.delete_recursive(self.root, key)

def delete_recursive(self, node, key):
    if node is None:
        return node

    if key < node.key:
        node.left = self.delete_recursive(node.left, key)
    elif key > node.key:
        node.right = self.delete_recursive(node.right, key)
    else:
        # Node with only one child or no child
        if node.left is None:
            return node.right
        elif node.right is None:
            return node.left

        # Node with two children: Get the inorder successor (smallest in the right subtree)
```

```
            temp = self.min_value_node(node.right)
            node.key = temp.key
            node.right = self.delete_recursive(node.right, temp.key)
        return node

    def min_value_node(self, node):
        current = node
        while current.left is not None:
            current = current.left
        return current

    def display_inorder(self):
        if self.root is None:
            print("Tree is empty.")
        else:
            print("Inorder Traversal:", end=" ")
            self.display_inorder_recursive(self.root)
            print()

    def display_inorder_recursive(self, node):
        if node:
            self.display_inorder_recursive(node.left)
            print(node.key, end=" ")
            self.display_inorder_recursive(node.right)

def main():
    bst = BinarySearchTree()

    while True:
        print("\nBinary Search Tree Operations:")
        print("1. Insert")
```

```
print("2. Delete")
print("3. Search")
print("4. Display (Inorder Traversal)")
print("5. Exit")

choice = input("Enter your choice: ")

if choice == '1':
    key = int(input("Enter value to insert: "))
    bst.insert(key)

elif choice == '2':

    key = int(input("Enter value to delete: "))
    if bst.search(key):
        bst.delete(key)
        print(f"Value {key} deleted.")
    else:
        print(f"Value {key} not found in the tree.")

elif choice == '3':

    key = int(input("Enter value to search: "))
    if bst.search(key):
        print(f"Value {key} found in the tree.")
    else:
        print(f"Value {key} not found in the tree.")

elif choice == '4':
  bst.display_inorder()
elif choice == '5':
```

```
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please try again.")

if __name__ == "__main__":
    main()
```