Double-click (or enter) to edit

```python
# Basic Operations and Complexity
class ArrayOperations:
    def __init__(self, size):
        self.size = size
        self.array = [None] * size
        self.length = 0

    def access(self, index):
        if 0 <= index < self.length:
            return self.array[index]
        raise IndexError("Index out of bounds")

    def insert(self, index, element):
        if self.length >= self.size:
            raise OverflowError("Array is full")

        for i in range(self.length, index, -1):
            self.array[i] = self.array[i - 1]

        self.array[index] = element
        self.length += 1

    def delete(self, index):
        if index >= self.length:
            raise IndexError("Index out of bounds")

        for i in range(index, self.length - 1):
            self.array[i] = self.array[i + 1]

        self.array[self.length - 1] = None
        self.length -= 1

    def display(self):
        return [self.array[i] for i in range(self.length)]



#Shopping Cart System

# Example 1: Managing a Shopping Cart
def shopping_cart_example():
    print("\n=== Shopping Cart Example ===")
    # Initialize cart with size 10
    cart = ArrayOperations(10)

    # Add items to cart
    items = [
        {"id": 1, "name": "Laptop", "price": 999.99},
        {"id": 2, "name": "Mouse", "price": 29.99},
        {"id": 3, "name": "Keyboard", "price": 59.99}
    ]

    # Demonstrate insert operation
    print("Adding items to cart:")
    for i, item in enumerate(items):
        cart.insert(i, item)
        print(f"Added {item['name']} - ${item['price']}")

    print("\nCurrent cart contents:")
    print(cart.display())

    # Demonstrate delete operation
    print("\nRemoving second item (Mouse)...")
    cart.delete(1)
    print("Updated cart contents:")
    print(cart.display())

    # Demonstrate access operation
    first_item = cart.access(0)
    print(f"\nFirst item in cart: {first_item['name']} - ${first_item['price']}")

# Run example
shopping_cart_example()


```

```
 77
 78 # Example 2: Student Roster Management
 79 def student_roster_example():
 80     print("\n=== Student Roster Example ===")
 81     roster = ArrayOperations(20)
 82
 83     # Sample student data
 84     students = [
 85         {"id": "S001", "name": "John", "grade": "A"},
 86         {"id": "S002", "name": "Emma", "grade": "B"},
 87         {"id": "S003", "name": "Michael", "grade": "A-"}
 88     ]
 89
 90     # Add initial students
 91     print("Adding students to roster:")
 92     for i, student in enumerate(students):
 93         roster.insert(i, student)
 94         print(f"Added {student['name']} (ID: {student['id']}), Grade: {student['grade']}")
 95
 96     # Show current roster
 97     print("\nCurrent roster:")
 98     for student in roster.display():
 99         print(f"{student['name']}: {student['grade']}")
100
101     # Add new student in middle
102     new_student = {"id": "S004", "name": "Sarah", "grade": "B+"}
103     print(f"\nAdding new student {new_student['name']} at position 1")
104     roster.insert(1, new_student)
105
106     print("\nUpdated roster:")
107     for student in roster.display():
108         print(f"{student['name']}: {student['grade']}")
109
110 # Run example
111 student_roster_example()
112
113
114
115 #1.2 Search Operations in Arrays
116 def linear_search_detailed(array, target):
117     """Detailed linear search implementation"""
118     comparisons = 0
119
120     for index in range(len(array)):
121         comparisons += 1
122
123         if array[index] == target:
124             return {
125                 "index": index,
126                 "comparisons": comparisons,
127                 "found": True,
128                 "search_path": list(range(index + 1))
129             }
130
131     return {
132         "index": -1,
133         "comparisons": comparisons,
134         "found": False,
135         "search_path": list(range(len(array)))
136     }
137
138 def binary_search_detailed(array, target, key=lambda x: x):
139     """Detailed binary search implementation"""
140     left = 0
141     right = len(array) - 1
142     comparisons = 0
143     steps = []
144
145     while left <= right:
146         mid = (left + right) // 2
147         comparisons += 1
148         current_value = key(array[mid])
149
150         steps.append({
151             "left": left,
152             "right": right,
153             "mid": mid,
154             "current_value": current_value,
155             "action": "comparison"
156         })
```

```
        })

        if current_value == target:
            steps[-1]["action"] = "found"
            return {
                "index": mid,
                "comparisons": comparisons,
                "steps": steps,
                "found": True
            }
        elif current_value < target:
            steps[-1]["action"] = "move_right"
            left = mid + 1
        else:
            steps[-1]["action"] = "move_left"
            right = mid - 1

    return {
        "index": -1,
        "comparisons": comparisons,
        "steps": steps,
        "found": False
    }


# Example 1: Library Book Search
def library_search_example():
    print("\n=== Library Book Search Example ===")

    # Book database
    books = [
        {"id": "B001", "title": "Python Programming", "author": "John Smith"},
        {"id": "B002", "title": "Data Structures", "author": "Jane Doe"},
        {"id": "B003", "title": "Algorithms", "author": "Alan Johnson"},
        {"id": "B004", "title": "Web Development", "author": "Sarah Wilson"},
        {"id": "B005", "title": "Machine Learning", "author": "Mike Brown"}
    ]

    # Linear search for a book by ID
    search_id = "B003"
    result = linear_search_detailed(books, search_id)

    print(f"\nSearching for book ID: {search_id}")
    if result["found"]:
        book = books[result["index"]]
        print(f"Found: {book['title']} by {book['author']}")
        print(f"Comparisons made: {result['comparisons']}")
    else:
        print("Book not found")

# Run all examples
library_search_example()


# Example 2: Student Grade Search
def grade_search_example():
    print("\n=== Student Grade Search Example ===")

    # Sorted student grades
    students = [
        {"id": "S001", "name": "Alice", "grade": 75},
        {"id": "S002", "name": "Bob", "grade": 82},
        {"id": "S003", "name": "Charlie", "grade": 88},
        {"id": "S004", "name": "David", "grade": 92},
        {"id": "S005", "name": "Eve", "grade": 95}
    ]

    # Binary search for grade
    target_grade = 88
    result = binary_search_detailed(
        students,
        target_grade,
        key=lambda x: x["grade"]
    )

    print(f"\nSearching for grade: {target_grade}")
    if result["found"]:
        student = students[result["index"]]
        print(f"Found student: {student['name']}")
```

```
        print("\nSearch steps:")
        for i, step in enumerate(result["steps"], 1):
            print(f"Step {i}:")
            print(f"  Checking range: {step['left']} to {step['right']}")
            print(f"  Middle element: {step['mid']}")
            print(f"  Action: {step['action']}")
    else:
        print("Grade not found")

# Run all examples
grade_search_example()



# Example 3: Phone Directory Search
def phone_directory_example():
    print("\n=== Phone Directory Search Example ===")

    # Sorted phone directory
    contacts = [
        {"name": "Adams, John", "phone": "123-456-7890"},
        {"name": "Brown, Mary", "phone": "234-567-8901"},
        {"name": "Davis, Steve", "phone": "345-678-9012"},
        {"name": "Johnson, Lisa", "phone": "456-789-0123"},
        {"name": "Wilson, Mike", "phone": "567-890-1234"}
    ]

    # Binary search by name
    search_name = "Davis, Steve"
    result = binary_search_detailed(
        contacts,
        search_name,
        key=lambda x: x["name"]
    )

    print(f"\nSearching for contact: {search_name}")
    if result["found"]:
        contact = contacts[result["index"]]
        print(f"Found: {contact['name']}")
        print(f"Phone: {contact['phone']}")
        print(f"Found in {result['comparisons']} comparisons")
    else:
        print("Contact not found")

# Run all examples
phone_directory_example()


#Student Management System
class StudentManagementSystem:
    def __init__(self, capacity):
        self.roster = ArrayOperations(capacity)
        self.id_index = {}  # For quick lookups by ID

    def add_student(self, student):
        # Add to array
        index = self.roster.length
        self.roster.insert(index, student)

        # Add to index
        self.id_index[student["id"]] = index

        print(f"Added student: {student['name']}")

    def find_student_by_id(self, student_id):
        # Direct lookup by ID
        if student_id in self.id_index:
            index = self.id_index[student_id]
            return self.roster.access(index)
        return None

    def find_students_by_grade(self, grade):
        # Linear search for all students with matching grade
        matches = []
        for i in range(self.roster.length):
            student = self.roster.access(i)
            if student["grade"] == grade:
                matches.append(student)
        return matches

```

```python
# Usage example
def run_student_management_example():
    print("\n=== Student Management System Example ===")

    # Initialize system
    sms = StudentManagementSystem(100)

    # Add some students
    students = [
        {"id": "S001", "name": "John Doe", "grade": "A", "major": "CS"},
        {"id": "S002", "name": "Jane Smith", "grade": "B", "major": "Math"},
        {"id": "S003", "name": "Bob Wilson", "grade": "A", "major": "Physics"},
        {"id": "S004", "name": "Alice Brown", "grade": "A", "major": "CS"}
    ]

    for student in students:
        sms.add_student(student)

    # Find student by ID
    print("\nSearching for student S003:")
    student = sms.find_student_by_id("S003")
    if student:
        print(f"Found: {student['name']} - {student['major']}")

    # Find students with grade A
    print("\nSearching for students with grade A:")
    students = sms.find_students_by_grade("A")
    for student in students:
        print(f"{student['name']} - {student['major']}")

# Run the example
run_student_management_example()



#Hash Tables

class SimpleHashTable:
    """A basic hash table implementation using chaining for collision resolution."""

    def __init__(self, size=100):
        self.size = size
        self.table = [[] for _ in range(size)]

    def _hash(self, key):
        """Simple hash function."""
        return hash(key) % self.size

    def insert(self, key, value):
        """Insert a key-value pair"""
        index = self._hash(key)

        # Check for existing key
        for item in self.table[index]:
            if item[0] == key:
                item[1] = value  # Update value
                return

        # Add new key-value pair
        self.table[index].append([key, value])

    def get(self, key):
        """Retrieve value for given key"""
        index = self._hash(key)

        for item in self.table[index]:
            if item[0] == key:
                return item[1]
        return None

    def delete(self, key):
        """Delete a key-value pair"""
        index = self._hash(key)

        for i, item in enumerate(self.table[index]):
            if item[0] == key:
                return self.table[index].pop(i)[1]
        return None


```

```python
#Multi-Index Hash Tables

class EmployeeDirectory:
    """
    Employee directory with multiple search indices
    """
    def __init__(self):
        self.by_id = {}           # Primary index
        self.by_department = {}   # Department index
        self.by_salary_range = {}  # Salary range index

    def add_employee(self, emp_id, name, department, salary):
        """Add employee with multiple indices"""
        employee = {
            "id": emp_id,
            "name": name,
            "department": department,
            "salary": salary
        }

        # Add to primary index
        self.by_id[emp_id] = employee

        # Add to department index
        if department not in self.by_department:
            self.by_department[department] = []
        self.by_department[department].append(employee)

        # Add to salary range index
        salary_range = (salary // 10000) * 10000
        if salary_range not in self.by_salary_range:
            self.by_salary_range[salary_range] = []
        self.by_salary_range[salary_range].append(employee)

    def find_by_id(self, emp_id):
        """O(1) lookup by ID"""
        return self.by_id.get(emp_id)

    def find_by_department(self, department):
        """O(1) lookup by department"""
        return self.by_department.get(department, [])

    def find_by_salary_range(self, min_salary, max_salary):
        """Find employees within salary range"""
        results = []
        for range_start in self.by_salary_range:
            if min_salary <= range_start <= max_salary:
                results.extend(self.by_salary_range[range_start])
        return results

# Example usage
def demonstrate_employee_directory():
    print("\n=== Employee Directory Example ===\n")
    directory = EmployeeDirectory()

    # Add sample employees
    employees = [
        ("E001", "John Doe", "IT", 75000),
        ("E002", "Jane Smith", "HR", 50000),
        ("E003", "Bob Wilson", "IT", 88000),
        ("E004", "Alice Brown", "Finance", 70000)
    ]

    for emp_id, name, dept, salary in employees:
        directory.add_employee(emp_id, name, dept, salary)
        print(f"Added {name} (ID: {emp_id})")

    # Demonstrate different search methods
    print("\nSearching by ID (E003):")
    emp = directory.find_by_id("E003")
    print(f"Found: {emp['name']} - {emp['department']}")

    print("\nIT Department employees:")
    it_emps = directory.find_by_department("IT")
    for emp in it_emps:
        print(f"{emp['name']} - {emp['salary']}")

    print("\nEmployees with salary 70000-80000:")
    salary_range = directory.find_by_salary_range(70000, 80000)
    for emp in salary_range:
```

```
        print(f"{emp['name']} - ${emp['salary']}")

# Run example
demonstrate_employee_directory()


#Linked List
class Node:
    """A single node in the linked list

    Attributes:
        data: The node's data
        next: Reference to the next node
    """
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    """Linked List implementation with search capabilities

    Attributes:
        head: First node in the list
    """
    def __init__(self):
        self.head = None

    def append(self, data):
        """Add a new node at the end O(n)"""
        if not self.head:
            self.head = Node(data)
            return

        current = self.head
        while current.next:
            current = current.next
        current.next = Node(data)

    def insert_front(self, data):
        """Add a new node at the beginning O(1)"""
        new_node = Node(data)
        new_node.next = self.head
        self.head = new_node

    def delete(self, data):
        """Delete first occurrence of data O(n)"""
        if not self.head:
            return False

        if self.head.data == data:
            self.head = self.head.next
            return True

        current = self.head
        while current.next:
            if current.next.data == data:
                current.next = current.next.next
                return True
            current = current.next
        return False

    def linear_search(self, target):
        """Search for an element and return its position"""
        current = self.head
        position = 0

        while current:
            if current.data == target:
                return {
                    "Position": position,
                    "found": True,
                    "data": current.data
                }
            current = current.next
            position += 1

        return {
            "Position": -1,
            "found": False,
            "data": None
```

```
            data = None
        }

    def display(self):
        """Returns list of all elements"""
        elements = []
        current = self.head
        while current:
            elements.append(current.data)
            current = current.next
        return elements



# Example: Library Book Management
class Book:
    """Book record for library management"""
    def __init__(self, isbn, title, author):
        self.isbn = isbn
        self.title = title
        self.author = author

    def __eq__(self, other):
        """Enable comparison with ISBN string"""
        if isinstance(other, str):
            return self.isbn == other
        return False

    def __str__(self):
        return f"{self.title} by {self.author} (ISBN: {self.isbn})"

def demonstrate_library_linked_list():
    print("\n=== Library Book Management Example ===")

    # Create a LinkedList catalog
    catalog = LinkedList()

    # Add sample books
    books = [
        Book("978-0001", "Python Basics", "John Smith"),
        Book("978-0002", "Data Structures", "Jane Doe"),
        Book("978-0003", "Algorithms", "Bob Wilson")
    ]

    print("Adding books to catalog:")
    for book in books:
        catalog.append(book)
        print(f"Added: {book}")

    # Search for a book
    print("\nSearching for ISBN: 978-0002")
    result = catalog.linear_search("978-0002")
    if result["found"]:
        book = result["data"]
        print(f"Found at position {result['Position']}: {book}")
    else:
        print("Book not found")

    # Delete a book
    print("\nDeleting book with ISBN: 978-0001")
    catalog.delete("978-0001")
    print("Updated catalog:")
    for book in catalog.display():
        print(book)

# Run the example
demonstrate_library_linked_list()



#Binary Search Tree
class TreeNode:
    """Binary Search Tree Node

    Attributes:
        data: Node's data
        left: Left child
        right: Right child
    """
    def __init__(self, data):
```

```
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

class BinarySearchTree:
    """Binary Search Tree implementation with search operations"""

    def __init__(self):
        self.root = None

    def insert(self, data):
        """Insert new node with data"""
        if not self.root:
            self.root = TreeNode(data)
        else:
            self._insert_recursive(self.root, data)

    def _insert_recursive(self, node, data):
        """Recursively find correct position and insert"""
        if data["id"] < node.data["id"]:
            if node.left is None:
                node.left = TreeNode(data)
            else:
                self._insert_recursive(node.left, data)
        else:
            if node.right is None:
                node.right = TreeNode(data)
            else:
                self._insert_recursive(node.right, data)

    def search(self, target_id):
        """Search for node with target_id"""
        result = self._search_recursive(self.root, target_id)
        return {
            "found": result is not None,
            "data": result,
            "comparisons": self._count_comparisons(self.root, target_id)
        }

    def _search_recursive(self, node, target_id):
        """Recursive search operation"""
        if node is None or node.data["id"] == target_id:
            return node.data if node else None

        if target_id < node.data["id"]:
            return self._search_recursive(node.left, target_id)
        return self._search_recursive(node.right, target_id)

    def _count_comparisons(self, node, target_id, count=0):
        """Count number of comparisons in search"""
        if node is None:
            return count

        count += 1
        if node.data["id"] == target_id:
            return count
        if target_id < node.data["id"]:
            return self._count_comparisons(node.left, target_id, count)
        return self._count_comparisons(node.right, target_id, count)



def demonstrate_student_bst():
    print("\n=== Student Record System Using BST ===")

    # Create BST
    student_tree = BinarySearchTree()

    # Add sample students
    students = [
        {"id": 101, "name": "Alice", "grade": 95},
        {"id": 103, "name": "Bob", "grade": 88},
        {"id": 102, "name": "Charlie", "grade": 92},
        {"id": 105, "name": "David", "grade": 85},
        {"id": 104, "name": "Eve", "grade": 90}
    ]

    print("Adding students to BST:")
    for student in students:
```

```
        student_tree.insert(student)
        print(f"Added: {student['name']} (ID: {student['id']})")

    # Search demonstrations
    test_ids = [102, 106]  # One existing, one non-existing
    for student_id in test_ids:
        print(f"\nSearching for student ID: {student_id}")
        result = student_tree.search(student_id)
        if result["found"]:
            student = result["data"]
            print(f"Found: {student['name']} - Grade: {student['grade']}")
        else:
            print("Student not found")

        print(f"Number of comparisons: {result['comparisons']}")

# Run the demonstration
demonstrate_student_bst()



#Enhanced Binary Search Tree
class TreeNode:
    """
    Enhanced Binary Search Tree Node
    Attributes:
        data: Node's data
        left: Left child
        right: Right child
        height: Height of node for AVL balancing
    """
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
        self.height = 1  # Used for AVL balancing

class BalancedBST:
    """
    Self-balancing Binary Search Tree (AVL Tree) implementation
    with traversal methods
    """
    def __init__(self):
        self.root = None

    def _get_height(self, node):
        """Get height of node"""
        if not node:
            return 0
        return node.height

    def _get_balance(self, node):
        """Get balance factor of node"""
        if not node:
            return 0
        return self._get_height(node.left) - self._get_height(node.right)

    def _update_height(self, node):
        """Update height of node"""
        if not node:
            return
        node.height = max(self._get_height(node.left),
                          self._get_height(node.right)) + 1

    def _right_rotate(self, y):
        """Right rotation for balancing"""
        x = y.left
        T2 = x.right

        x.right = y
        y.left = T2

        self._update_height(y)
        self._update_height(x)

        return x

    def _left_rotate(self, x):
        """Left rotation for balancing"""
```

```
        y = x.right
        T2 = y.left

        y.left = x
        x.right = T2

        self._update_height(x)
        self._update_height(y)

        return y

    def insert(self, data):
        """Insert new node and maintain balance"""
        self.root = self._insert_recursive(self.root, data)

    def _insert_recursive(self, node, data):
        """Recursive insert with balancing"""
        if not node:
            return TreeNode(data)

        if data["id"] < node.data["id"]:
            node.left = self._insert_recursive(node.left, data)
        else:
            node.right = self._insert_recursive(node.right, data)

        # Update height
        self._update_height(node)

        # Get balance factor
        balance = self._get_balance(node)

        # Balance the tree if needed
        # Left Left Case
        if balance > 1 and data["id"] < node.left.data["id"]:
            return self._right_rotate(node)

        # Right Right Case
        if balance < -1 and data["id"] > node.right.data["id"]:
            return self._left_rotate(node)

        # Left Right Case
        if balance > 1 and data["id"] > node.left.data["id"]:
            node.left = self._left_rotate(node.left)
            return self._right_rotate(node)

        # Right Left Case
        if balance < -1 and data["id"] < node.right.data["id"]:
            node.right = self._right_rotate(node.right)
            return self._left_rotate(node)

        return node

    # Traversal Methods
    def inorder(self):
        """Inorder traversal: left -> root -> right"""
        result = []
        self._inorder_recursive(self.root, result)
        return result

    def _inorder_recursive(self, node, result):
        if node:
            self._inorder_recursive(node.left, result)
            result.append(node.data)
            self._inorder_recursive(node.right, result)

    def preorder(self):
        """Preorder traversal: root -> left -> right"""
        result = []
        self._preorder_recursive(self.root, result)
        return result

    def _preorder_recursive(self, node, result):
        if node:
            result.append(node.data)
            self._preorder_recursive(node.left, result)
            self._preorder_recursive(node.right, result)

    def postorder(self):
        """Postorder traversal: left -> right -> root"""
        result = []
```

```
        self._postorder_recursive(self.root, result)
        return result

    def _postorder_recursive(self, node, result):
        if node:
            self._postorder_recursive(node.left, result)
            self._postorder_recursive(node.right, result)
            result.append(node.data)

    def level_order(self):
        """Level order (breadth-first) traversal"""
        if not self.root:
            return []

        result = []
        queue = [self.root]

        while queue:
            node = queue.pop(0)
            result.append(node.data)

            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)

        return result

    def search(self, target_id):
        """Search with path tracking"""
        path = []
        result = self._search_recursive(self.root, target_id, path)
        return {
            "found": result is not None,
            "data": result,
            "path": path
        }

    def _search_recursive(self, node, target_id, path):
        if not node:
            return None

        path.append(node.data["id"])

        if node.data["id"] == target_id:
            return node.data

        if target_id < node.data["id"]:
            return self._search_recursive(node.left, target_id, path)
        return self._search_recursive(node.right, target_id, path)


#Example Usage and Demonstration
def demonstrate_balanced_bst():
    print("\n=== Balanced BST Demonstration ===")

    tree = BalancedBST()

    # Sample student data
    students = [
        {"id": 5, "name": "Alice", "grade": 95},
        {"id": 3, "name": "Bob", "grade": 88},
        {"id": 7, "name": "Charlie", "grade": 92},
        {"id": 1, "name": "David", "grade": 85},
        {"id": 9, "name": "Eve", "grade": 90},
        {"id": 4, "name": "Frank", "grade": 87},
        {"id": 6, "name": "Grace", "grade": 93}
    ]

    # Insert students
    print("Inserting students...")
    for student in students:
        tree.insert(student)
        print(f"Added: {student['name']} (ID: {student['id']})")

    # Demonstrate different traversals
    print("\nTraversal Demonstrations:")

    print("\nInorder Traversal (sorted by ID):")
    for student in tree.inorder():
```

```
        print(f"{student['name']}: {student['id']}")

    print("\nLevel Order Traversal (by tree level):")
    for student in tree.level_order():
        print(f"{student['name']}: {student['id']}")

    # Search demonstration
    print("\nSearch Demonstrations:")
    search_ids = [4, 8]  # One existing, one non-existing

    for search_id in search_ids:
        result = tree.search(search_id)
        print(f"\nSearching for ID: {search_id}")
        if result["found"]:
            print(f"Found: {result['data']['name']}")
        else:
            print("Student not found")
        print(f"Search path: {' -> '.join(map(str, result['path']))}")

# Run demonstration
def main():
    demonstrate_balanced_bst()

if __name__ == "__main__":
    main()



#Array Operations
class DynamicArray:
    def __init__(self, initial_size=4):
        self.array = [None] * initial_size
        self.size = initial_size
        self.length = 0

    # TODO: Implement these methods
    def append(self, element):
        """Add element to end of array, resize if needed"""
        pass

    def insert(self, index, element):
        """Insert element at index"""
        pass

    def remove(self, index):
        """Remove element at index"""
        pass

    def get(self, index):
        """Get element at index"""
        pass


# Test cases
def test_dynamic_array():
    arr = DynamicArray()

    # Test 1: Append beyond initial size
    for i in range(6):
        arr.append(i)
    assert arr.size >= 6, "Array should resize"

    # Test 2: Insert in middle
    arr.insert(2, 10)
    assert arr.get(2) == 10, "Insert failed"



#Array Search
def find_duplicates(arr):
    """
    Find all elements that appear more than once in the array

    Example:
    Input: [1, 3, 4, 2, 2, 1, 5, 3]
    Output: [1, 2, 3]
    """
    pass

def find_missing_number(arr):
```

```
    """
    Find the missing number in an array containing n-1 numbers from 1 to n

    Example:
    Input: [1, 2, 4, 6, 3, 7, 8], n=8
    Output: 5
    """
    pass

# Test cases
def test_search_problems():
    # Test duplicates
    result = find_duplicates([1, 3, 4, 2, 2, 1, 5, 3])
    assert set(result) == {1, 2, 3}, "Duplicate finding failed"

    # Test missing number
    result = find_missing_number([1, 2, 4, 6, 3, 7, 8])
    assert result == 5, "Missing number finding failed"


```

```
=== Shopping Cart Example ===
Adding items to cart:
Added Laptop - $999.99
Added Mouse - $29.99
Added Keyboard - $59.99

Current cart contents:
[{'id': 1, 'name': 'Laptop', 'price': 999.99}, {'id': 2, 'name': 'Mouse', 'price': 29.99}, {'id': 3, 'name': 'Keyboard', 'price': 59.99}]

Removing second item (Mouse)...
Updated cart contents:
[{'id': 1, 'name': 'Laptop', 'price': 999.99}, {'id': 3, 'name': 'Keyboard', 'price': 59.99}]

First item in cart: Laptop - $999.99

=== Student Roster Example ===
Adding students to roster:
Added John (ID: S001), Grade: A
Added Emma (ID: S002), Grade: B
Added Michael (ID: S003), Grade: A-

Current roster:
John: A
Emma: B
Michael: A-

Adding new student Sarah at position 1

Updated roster:
John: A
Sarah: B+
Emma: B
Michael: A-

=== Library Book Search Example ===

Searching for book ID: B003
Book not found

=== Student Grade Search Example ===

Searching for grade: 88
Found student: Charlie

Search steps:
Step 1:
 Checking range: 0 to 4
 Middle element: 2
 Action: found

=== Phone Directory Search Example ===

Searching for contact: Davis, Steve
Found: Davis, Steve
Phone: 345-678-9012
Found in 1 comparisons
```

Double-click (or enter) to edit