Test Instructions for Data Structure Implementation

You will implement a data structure that supports the following operations:

1. add() - Inserts numbers into a doubly linked list.
2. sort() - Transfers elements from the linked list into an array and sorts them using Heap Sort.
3. search(value) - Performs Depth-First Search (DFS) on the data structure and returns the traversal path.
4. display() - Displays the numbers in the doubly linked list in both forward and backward order.

1. Implement the add() Method

- Create a doubly linked list (DLL) where each node contains:
  - value: The integer added.
  - prev: Pointer to the previous node.
  - next: Pointer to the next node.
- Implement the add(value) function:
  - If the list is empty, initialize it with the first node.
  - Otherwise, append new nodes to the end of the list while maintaining prev and next pointers.

Expected Behavior:

- Adding multiple numbers should create a doubly linked list where each node connects forward and backward.

2. Implement the sort() Method

- Extract all elements from the doubly linked list and store them in an array.
- Implement the Heap Sort algorithm:
  - Convert the array into a max-heap.
  - Swap the first and last elements, reduce the heap size, and heapify.
  - Repeat until the array is sorted.
- After sorting, the array should be sorted in ascending order.

Expected Behavior:

- Sorting should not modify the linked list.

- It should only convert DLL to array, then sort.

3. Implement the search(value) Method

- Perform Depth-First Search (DFS) to look for the value in the doubly linked list.
- Return the traversal order of nodes until the target is found.
- If the value is found, return 'Found at position X'.
- If not, return 'Not found'.

Expected Behavior:

- The function should print all nodes visited in DFS order before finding the value.

4. Implement the display() Method

- Display all elements in the doubly linked list in:
  - Forward order (head to tail)
  - Backward order (tail to head)

Expected Behavior:

- The list should be correctly traversed in both directions without errors.

Example Usage

```
dll = DoublyLinkedList()

dll.add(5)

dll.add(2)

dll.add(8)

dll.add(1)


dll.display()

# Output: Forward: [5, 2, 8, 1]

#         Backward: [1, 8, 2, 5]


dll.sort()

# Output: Sorted array: [1, 2, 5, 8]


dll.search(8)
```

```
# Output: DFS Traversal: [5, 2, 8]
#          Found at position 3
```

Testing & Validation

1. Adding Elements:
   - Add multiple elements and check if DLL maintains correct prev and next links.
2. Sorting:
   - Verify that sorting does not alter the linked list but correctly sorts the array.
3. Searching:
   - Test searching for:
     - A number in the list.
     - A number not in the list.
     - Edge cases like first or last element.
4. Displaying:
   - Check if the list displays correctly in both directions.

Deliverables

- A Python class DoublyLinkedList with correctly implemented methods.
- Example test cases that validate functionality.
- Proper documentation/comments explaining your approach.