Implement Double-Ended Queue Using Array LeetCode

This program implements a Double-Ended Queue (Deque) data structure using an array. It defines a **Deque** class with methods for pushing elements to the front and rear of the deque, popping elements from the front and rear, checking if the deque is empty or full, getting the front and rear elements, getting the size of the deque, and displaying its elements. The deque supports circular updating of front and rear pointers to utilize the array efficiently. The program demonstrates the usage of the **Deque** class by performing various deque operations.

1. **Deque(int n)** - Constructor to initialize an empty deque with a given size.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(n), where n is the size of the deque.**
    - **Explanation:** This constructor allocates memory for the deque array and initializes other variables such as **front**, **rear**, **size**, and **length**. The time complexity is constant, and the space complexity depends on the size of the deque.
2. **void pushFront(int value)** - Pushes an element to the front of the deque.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method updates the **front** pointer circularly to add an element to the front position. It has constant time and space complexity.
3. **void pushRear(int value)** - Pushes an element to the rear of the deque.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method stores the value at the rear position and updates the **rear** pointer circularly to add an element to the rear position. It has constant time and space complexity.
4. **int popFront()** - Pops an element from the front of the deque.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method removes the front element, updates the **front** pointer circularly, and returns the deleted element. It has constant time and space complexity.

5. **int popRear()** - Pops an element from the rear of the deque.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method updates the **rear** pointer circularly, removes the rear element, and returns the deleted element. It has constant time and space complexity.
6. **int getFront()** - Returns the first element of the deque. If the deque is empty, it returns -1.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method returns the value of the **front** index, which represents the front element of the deque. It has constant time and space complexity.
7. **int getRear()** - Returns the last element of the deque. If the deque is empty, it returns -1.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method returns the value of the **rear** index, which represents the rear element of the deque. It has constant time and space complexity.
8. **bool isEmpty()** - Checks if the deque is empty.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method checks if the **length** variable is zero, indicating that the deque is empty. It has constant time and space complexity.
9. **bool isFull()** - Checks if the deque is full.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This method checks if the **length** variable is equal to the **size** variable, indicating that the deque is full. It has constant time and space complexity.
10. **int getSize()** - Returns the current number of elements in the deque.
    - **Time Complexity: O(1)**

- **Space Complexity: O(1)**
- **Explanation:** This method returns the **length** variable, representing the current number of elements in the deque. It has constant time and space complexity.

11. **void display()** - Displays the elements in the deque.
    - **Time Complexity: O(n), where n is the number of elements in the deque.**
    - **Space Complexity: O(1)**
    - **Explanation:** This method iterates through the deque elements, starting from **front** and stopping when it reaches **rear**, to display all elements. It has a time complexity proportional to the number of elements in the deque and constant space complexity.
12. **~Deque()** - Destructor to release memory allocated for the deque.
    - **Time Complexity: O(1)**
    - **Space Complexity: O(1)**
    - **Explanation:** This destructor deallocates the memory used by the deque array. It has constant time and space complexity.