Circular Linked List

1. **Node Class**:
    - This class represents the individual nodes in the circular linked list.
    - Attributes: **value** (stores the value of the node), **next** (points to the next node in the list).
    - Constructor: Initializes the value and sets the next pointer to **nullptr**.
2. **CircularLinkedList Class**:
    - This class implements the circular linked list and provides various methods to manipulate it.
3. **isEmpty()**:
    - Checks whether the linked list is empty or not.
    - Time Complexity: O(1)
    - Space Complexity: O(1)
4. **insertWhileEmpty(Node* newNode)**:
    - Inserts a node when the linked list is empty.
    - Time Complexity: O(1)
    - Space Complexity: O(1)
5. **deleteOnlyElement()**:
    - Deletes the only element in the linked list.
    - Time Complexity: O(1)
    - Space Complexity: O(1)
6. **findIndex(int index)**:
    - Finds the node at a given index.
    - Time Complexity: O(n), where n is the length of the linked list
    - Space Complexity: O(1)
7. **findValue(int value)**:
    - Finds the index of a given value in the linked list.
    - Time Complexity: O(n), where n is the length of the linked list
    - Space Complexity: O(1)

8. **swap(Node* first, Node* second)**:
    - Swaps the values of two nodes.
    - Time Complexity: O(1)
    - Space Complexity: O(1)
9. **Constructor**:
    - Initializes an empty circular linked list.
    - Time Complexity: O(1)
    - Space Complexity: O(1)
10. **pushBack(int value)**:
    - Adds a new node with the given value to the end of the linked list.
    - Time Complexity: O(1)
    - Space Complexity: O(1)
11. **pushFront(int value)**:
    - Adds a new node with the given value to the front of the linked list.
    - Time Complexity: O(1)
    - Space Complexity: O(1)
12. **insertAfterIndex(int index, int value)**:
    - Inserts a new node with the given value after a specified index.
    - Time Complexity: O(n), where n is the length of the linked list
    - Space Complexity: O(1)
13. **insertBeforeIndex(int index, int value)**:
    - Inserts a new node with the given value before a specified index.
    - Time Complexity: O(n), where n is the length of the linked list
    - Space Complexity: O(1)
14. **popFront()**:
    - Removes the first element from the linked list.
    - Time Complexity: O(1)
    - Space Complexity: O(1)
15. **popBack()**:

- Removes the last element from the linked list.
- Time Complexity: O(n), where n is the length of the linked list
- Space Complexity: O(1)

16. **deleteNode(int index)**:
    - Deletes a node at a specified index.
    - Time Complexity: O(n), where n is the length of the linked list
    - Space Complexity: O(1)
17. **deleteLinkedList()**:
    - Deletes the entire linked list.
    - Time Complexity: O(1)
    - Space Complexity: O(1)
18. **display()**:
    - Displays the elements of the linked list.
    - Time Complexity: O(n), where n is the length of the linked list
    - Space Complexity: O(1)
19. **reverse()**:
    - Reverses the linked list.
    - Time Complexity: O(n), where n is the length of the linked list
    - Space Complexity: O(1)
20. **search(int value)**:
    - Searches for a value in the linked list and returns its index.
    - Time Complexity: O(n), where n is the length of the linked list
    - Space Complexity: O(1)
21. **update(int index, int value)**:
    - Updates the value of a node at a specified index.
    - Time Complexity: O(n), where n is the length of the linked list
    - Space Complexity: O(1)
22. **sort()**:
    - Sorts the linked list in ascending order using bubble sort.

- Time Complexity: O(n^2), where n is the length of the linked list
- Space Complexity: O(1)

23. **headNode()**:
    - Displays the value of the head node.
    - Time Complexity: O(1)
    - Space Complexity: O(1)
24. **tailNode()**:
    - Displays the value of the tail node.
    - Time Complexity: O(1)
    - Space Complexity: O(1)
25. **linkedListLength()**:
    - Displays the length of the linked list.
    - Time Complexity: O(1)
    - Space Complexity: O(1)
26. **Destructor**:
    - Frees the memory allocated for the linked list nodes.
    - Time Complexity: O(n), where n is the length of the linked list
    - Space Complexity: O(1)