Operations on Doubly Linked List

1. Creating
2. Traversal
3. Insertion
4. Deletion
5. Searching
6. Sorting

1. Creating a Doubly Linked List

```
// Function to create a new node
Node* createNode(int value) {
   Node* newNode = new Node();
   newNode->data = value;
   newNode->prev = NULL;
   newNode->next = NULL;
   return newNode;
}
```

2. Traversing a Doubly Linked List

Traversal in a **Doubly Linked List** means visiting each node **one by one**, either **from head to tail (forward)** or **from tail to head (backward)**.

```
//Forward Traversal (From Head to Tail)
void forwardTraversal(Node* head) {
   Node* temp = head;
   cout << "Forward Traversal: ";
   while (temp != NULL) {
      cout << temp->data << " ";
      temp = temp->next;
   }
   cout << endl;
}
```

3. Insertion at the end in a Doubly Linked List

```
void insertAtEnd(Node*& head, int data) {
   Node* newNode = new Node{data, NULL, NULL};
   if (head == NULL) {
       head = newNode;
       return;
   }
   Node* temp = head;
   while (temp->next != NULL)
       temp = temp->next;

   temp->next = newNode;
   newNode->prev = temp;
}
```

4. **Deletion in a Doubly Linked List**

```
void deleteFromEnd(Node*& head) {
    if (head == NULL)
        return;

    Node* temp = head;

    // Only one node
    if (temp->next == NULL) {
        delete head;
        head = NULL;
        return;
    }

    // Traverse to the last node
    while (temp->next != NULL)
        temp = temp->next;

    temp->prev->next = NULL;
    delete temp;
}
```

5. **Searching an Element (Key) in a Doubly Linked List**

```
Node* search(Node* head, int key) {
    Node* temp = head;
    while (temp != NULL) {
        if (temp->data == key)
            return temp; // Found
        temp = temp->next;
    }
    return NULL; // Not found
}
```

6. **Sorting a Doubly Linked List (Using Bubble Sort)**

```
void sortDoublyLinkedList(Node* head) {
    if (head == NULL)
        return;

    bool swapped;
    Node* ptr1;
    Node* lptr = NULL;

    do {
        swapped = false;
        ptr1 = head;

        while (ptr1->next != lptr) {
            if (ptr1->data > ptr1->next->data) {
                // Swap data
                int temp = ptr1->data;
                ptr1->data = ptr1->next->data;
                ptr1->next->data = temp;
                swapped = true;
            }
            ptr1 = ptr1->next;
        }
        lptr = ptr1;
    } while (swapped);
}
```