What is a Doubly Linked List?

- A linked list is a node where each node contains three parts:
    1. Data
    2. Pointer to the next node
    3. Pointer to the previous node

- **Advantages:**
    - Can traverse forwards and backwards.
    - More flexibility in operations.

Insertion at Head:

```
void insertAtHead(Node head, int newData) {
   Node newNode = new Node(newData);
   newNode.next = head;
   if (head != null) {
      head.prev = newNode;
   }
   head = newNode;
}
```

Insertion at End:

```
void insertAtEnd(Node head, int newData) {
   Node newNode = new Node(newData);
   if (head == null) {
      head = newNode;
      return;
   }
   Node temp = head;
   while (temp.next != null) {
      temp = temp.next;
   }
```

```
    temp.next = newNode;
    newNode.prev = temp;
}
```

Deletion from a Specific Position:

```
void deleteFromPosition(Node head, int position) {
    if (head == null || position <= 0) return;
    Node temp = head;
    for (int i = 1; temp != null && i < position; i++) {
        temp = temp.next;
    }
    if (temp == null) return;
    if (temp.prev != null) {
        temp.prev.next = temp.next;
    } else {
        head = temp.next;
    }
    if (temp.next != null) {
        temp.next.prev = temp.prev;
    }
}
```

Search for a Value:

```
boolean search(Node head, int key) {
    Node temp = head;
    while (temp != null) {
        if (temp.data == key) return true;
        temp = temp.next;
    }
    return false;
}
```

Display Doubly Linked List:

```
void display(Node head) {
   Node temp = head;
   while (temp != null) {
      System.out.print(temp.data + " ");
      temp = temp.next;
   }
   System.out.println();
}
```

What is a Circular Linked List?

- linked list where the last node points back to the first node, forming a circle.

2. Singly Circular Linked List
3. Doubly Circular Linked List

- **Advantages:**
  - Can traverse the entire list from any node.
  - Useful for applications requiring a cyclic traversal.

Insertion at Head:

```
void insertAtHead(Node head, int newData) {
   Node newNode = new Node(newData);
   if (head == null) {
      newNode.next = newNode;
      head = newNode;
      return;
   }
   Node temp = head;
   while (temp.next != head) {
      temp = temp.next;
```

```
    }
    newNode.next = head;
    temp.next = newNode;
    head = newNode;
}
```

Insertion at End:

```
void insertAtEnd(Node head, int newData) {
    Node newNode = new Node(newData);
    if (head == null) {
        newNode.next = newNode;
        head = newNode;
        return;
    }
    Node temp = head;
    while (temp.next != head) {
        temp = temp.next;
    }
    temp.next = newNode;
    newNode.next = head;
}
```

Deletion from a Specific Position:

```
void deleteFromPosition(Node head, int position) {
    if (head == null) return;
    Node temp = head;
    if (position == 1) {
        while (temp.next != head) {
            temp = temp.next;
        }
        temp.next = head.next;
```

```
        head = head.next;
        return;
    }
    for (int i = 1; i < position - 1 && temp.next != head; i++) {
        temp = temp.next;
    }
    if (temp.next == head) return;
    temp.next = temp.next.next;
}
```

Search for a Value:

```
boolean search(Node head, int key) {
    if (head == null) return false;
    Node temp = head;
    do {
        if (temp.data == key) return true;
        temp = temp.next;
    } while (temp != head);
    return false;
}
```

Display Circular List:

```
void displayCircular(Node head) {
    if (head == null) return;
    Node temp = head;
    do {
        System.out.print(temp.data + " ");
        temp = temp.next;
    } while (temp != head);
    System.out.println();
}
```