**Data Structures Used**

1. **LinkedList**
   * **Purpose**: The primary data structure used to store the contacts is a LinkedList<Contact>.
   * **Usage**:
     + **Dynamic Size**: LinkedList allows dynamic sizing, meaning you can easily add or remove contacts without worrying about predefining the size of the list. This is especially useful for a contact management application where the number of contacts can vary widely.
     + **Storage**: Each contact is represented by an instance of the Contact class, which encapsulates the properties (name, phone, and email) of a contact. The linked list will hold references to these Contact objects.
     + **Operations**:
       - **Add Contact**: When a new contact is added using the addContact method, the new Contact object is appended to the LinkedList using the add() method. This operation has an average time complexity of **O(1)** since it is adding to the end of the list.
       - **Delete Contact**: The deleteContact method removes a contact by index, which also has an average time complexity of **O(n)** since it may need to traverse the list to find the contact (if removing from the middle).
       - **Edit Contact**: When editing a contact, the application retrieves the Contact object from the list using its index and updates its attributes. This operation has a time complexity of **O(1)** as it's directly accessing an index.
       - **Search Contact**: In the searchContact function, the application loops through the LinkedList to find contacts that match the search query. This has a time complexity of **O(n)**, where n is the number of contacts.
2. **DefaultListModel**
   * **Purpose**: The DefaultListModel<String> is used for managing the list of contacts displayed in the GUI.
   * **Usage**:
     + It acts as a bridge between the data structure (the LinkedList of Contact objects) and the GUI component (the JList).
     + It allows the JList to update automatically when the model changes. When a new contact is added, removed, or modified, the DefaultListModel is updated to reflect those changes.
3. **Contact Class**
   * **Purpose**: This is a custom data structure (class) to represent each contact's details.
   * **Usage**:
     + Encapsulates the properties (name, phone, email) of a contact.
     + Provides methods to access and modify the contact details (getters and setters).
     + Includes a method to return a string representation of the contact for easy display in the GUI.

**Summary of Data Structure Usage**

* The **LinkedList** serves as the primary storage mechanism for contact data, providing efficient insertions and deletions.
* The **DefaultListModel** connects the underlying data structure with the GUI, allowing for dynamic updates of the contact list displayed to the user.
* The **Contact class** organizes the data associated with each contact, enabling easy management of contact information.

**Advantages of Using LinkedList**

* **Efficient Insertions/Deletions**: Because a LinkedList allows for constant-time insertions and deletions from the list (if adding/removing from the ends), it is suitable for applications where contacts are frequently added or removed.
* **Dynamic Growth**: Unlike arrays, a LinkedList can grow as needed without the need for resizing or reallocating memory.

**Disadvantages of Using LinkedList**

* **Memory Overhead**: Each element in a linked list requires additional memory for storing pointers to the next (and possibly previous) elements, making it less memory efficient than arrays when storing a large number of elements.
* **Sequential Access**: Accessing an element by index requires traversal from the beginning, which can make it slower than accessing elements in an array.