NAMIBIA UNIVERSITY
OF SCIENCE AND TECHNOLOGY

FACULTY OF COMPUTING AND INFORMATICS
DEPARTMENT OF SOFTWARE ENGINEERING

DSA521S GROUP ASSIGNMENT

STUDENT NUMBER	FIRST NAME	LAST NAME
220117330	Shaningwa	willipard
224093630	Ndamononghenda	Kashava
223092479	Petrus	Shimbanda
221066535	Manuela K	Kambonge
200614282	Paulina	Shuuya
220002797	Luyanda	Ntini

Section A: Algorithms Representation (Pseudocode and Flowchart)

1. Problem Overview

You're tasked with implementing a phonebook application using basic linear data structures (e.g., arrays, linked lists) that can perform operations like insert, search, display, delete, update, and optional sorting of contacts. You will also analyze the efficiency of your search algorithm.

2. Modules

The phonebook application is divided into several distinct and well-structured modules, each responsible for handling a specific aspect of the operations. By using simple linear data structures such as arrays (or lists) and linear algorithms, the project ensures both simplicity and efficiency in solving the problem. Each module has been designed to focus on a single responsibility, ensuring that the problem becomes easy to solve, test, and maintain.

1. Main Phonebook Module

Module Functionality:

Acts as the central controller for the entire phonebook application. It handles the initialization of the phonebook (using a list structure) and coordinates the interaction between the user and other modules. This module also ensures the correct sequence of operations and flow of the program.

Data Structure Used: A list (array) is used to store all contact records in the phonebook.

Pseudocode:

```
function main():     phonebook
← []     displayMenu()
processUserInput(phonebook)
```

2. Insert Contact Module

Module Functionality:

This module is responsible for adding a new contact to the phonebook. It checks if the contact details are valid and inserts them into the list. The insertion follows a simple linear approach where the contact is appended to the end of the list.

Data Structure Used: A list is used to store contact records, where each record is a dictionary containing the name and phone number.

Pseudocode:

```
function insertContact(phonebook, name, phoneNumber):
   create newContact with name and phoneNumber
append newContact to phonebook
end function
```

3. Search Contact Module

Module Functionality:

This module searches for a contact within the phonebook. It uses a linear search algorithm to traverse the list and locate the contact by name. This approach ensures that even in unsorted lists, the search will be conducted efficiently, as required by the use of simple linear data structures.

Data Structure Used: Linear search on a list.

Pseudocode:

```
function searchContact(phonebook, searchName):
for each contact in phonebook:        if
contact.name equals searchName:
            return contact
   return "Contact not found"
```

4. Display All Contacts Module

Module Functionality:

This module outputs all contacts in the phonebook in a user-friendly format. The linear list is traversed, and each contact's details are displayed in order of insertion.

Data Structure Used: A list of contacts is traversed linearly for display.

```
function displayContacts(phonebook):
if phonebook is empty:
      print "No contacts available"
   else:        for each contact in
phonebook:
            print contact.name and contact.phoneNumber
```

5. Delete Contact Module

Module Functionality: This module deletes a specific contact from the phonebook by name. It uses a linear search to find the contact, and once found, the contact is removed from the list.

Data Structure Used: A list is linearly traversed to find the contact to delete.

Pseudocodes:

```
function deleteContact(phonebook, searchName):
for each contact in phonebook:
    if contact.name equals searchName:
        remove contact from phonebook
        return "Contact deleted"
return "Contact not found"
```

6. Update Contact Module

Module Functionality: This module allows the user to update the phone number of an existing contact. It uses linear search to locate the contact, and once found, updates the phone number.

Data Structure Used: A list is searched linearly to find the contact to update.

Pseudocode:

```
function updateContact(phonebook, searchName, newPhoneNumber):
for each contact in phonebook:
    if contact.name equals searchName:
        contact.phoneNumber = newPhoneNumber
        return "Contact updated"
return "Contact not found"
```

7. Sort Contacts Module

Module Functionality:

This module sorts the contacts in the phonebook alphabetically by name to improve search efficiency. It uses a basic sorting algorithm such as bubble sort or selection sort, keeping with the requirement of simple linear algorithms.

Data Structure Used: A list is sorted using a simple sorting algorithm.

Pseudocode:

```
function sortContacts(phonebook):
    sort phonebook by contact.name
    return phonebook
```

8. Efficiency Analysis Module

Module Functionality: This module analyzes the efficiency of the search algorithm used in the application. It calculates the time complexity of the search operation, which is O(n) due to the linear search being employed.

Data Structure Used: Linear search on a list.

Pseudocode:

```
function analyzeSearchEfficiency(phonebook, searchName):
    print "Search algorithm uses linear search with O(n) time complexity."
```

Detailed Analysis of the Search Contact Operation

The Search Contact operation in the phonebook application uses a linear search algorithm, which is the most straightforward approach for searching through an unsorted list. This algorithm traverses the list sequentially, checking each contact's name until a match is found or the entire list is searched. The following analysis provides a detailed breakdown of the search operation, focusing on the time and space complexity, as well as the reasoning behind the choice of linear search.

1. Algorithm Description

The linear search algorithm works by iterating over each contact in the phonebook and comparing the contact's name with the search term provided by the user. If a match is found, the contact is returned. If no match is found after checking all contacts, the algorithm returns a message indicating that the contact was not found.

Pseudocode:

```
function searchContact(phonebook, searchName):
for each contact in phonebook:        if
contact.name equals searchName:
            return contact
    return "Contact not found"
```

2. Time Complexity

The efficiency of the search operation is measured by its time complexity, which refers to the number of comparisons the algorithm needs to perform as the size of the phonebook grows.

Best Case: The best-case scenario occurs when the contact being searched for is the first contact in the list. In this case, the search completes after just one comparison.

Time Complexity (Best Case): O(1)

Worst Case: The worst-case scenario occurs when the contact being searched for is either the last contact in the list or does not exist in the list at all. In this case, the algorithm must check every contact before returning a result.

Time Complexity (Worst Case): O(n), where n is the number of contacts in the phonebook.

Average Case: On average, the contact will be located somewhere in the middle of the list. The search will need to check about half of the contacts before finding the match.

Time Complexity (Average Case): O(n)

Since this algorithm uses a simple linear search, the time complexity is O(n) in both the average and worst cases. This means the number of comparisons increases linearly as the size of the phonebook increases.

3. Space Complexity

The space complexity of the linear search operation is O(1), meaning that the algorithm uses a constant amount of extra memory regardless of the size of the phonebook. This is because the search only requires a few variables to track the current contact being compared and the search term.

4. Justification for Using Linear Search

The linear search algorithm is well-suited for this project because it adheres to the requirement of using simple linear data structures and algorithms. The following points justify the use of linear search:

Simplicity: Linear search is easy to implement and does not require the phonebook to be sorted. It works directly on the list of contacts without the need for additional data structures or preprocessing.

Applicability: Since the phonebook is not necessarily sorted, linear search is the most practical option for locating a contact. More advanced search algorithms, such as binary search, would require the list to be sorted, which adds complexity to the solution.

Efficiency for Small Datasets: Although linear search has a time complexity of O(n), for small to moderately sized phonebooks, the performance is acceptable. As the number of contacts increases, the need for more efficient search algorithms may arise, but for the scale of this project, linear search provides a sufficient balance between simplicity and performance.

5. Alternative Approaches

While linear search is appropriate for this project, more efficient search algorithms can be considered in the future if the phonebook grows in size:

Binary Search: This algorithm would reduce the search time to O(log n), but it requires the phonebook to be sorted beforehand. The sorting process would incur additional time and space complexity, making it less suitable for this initial implementation.

Phonebook Application Pseudocode as a whole

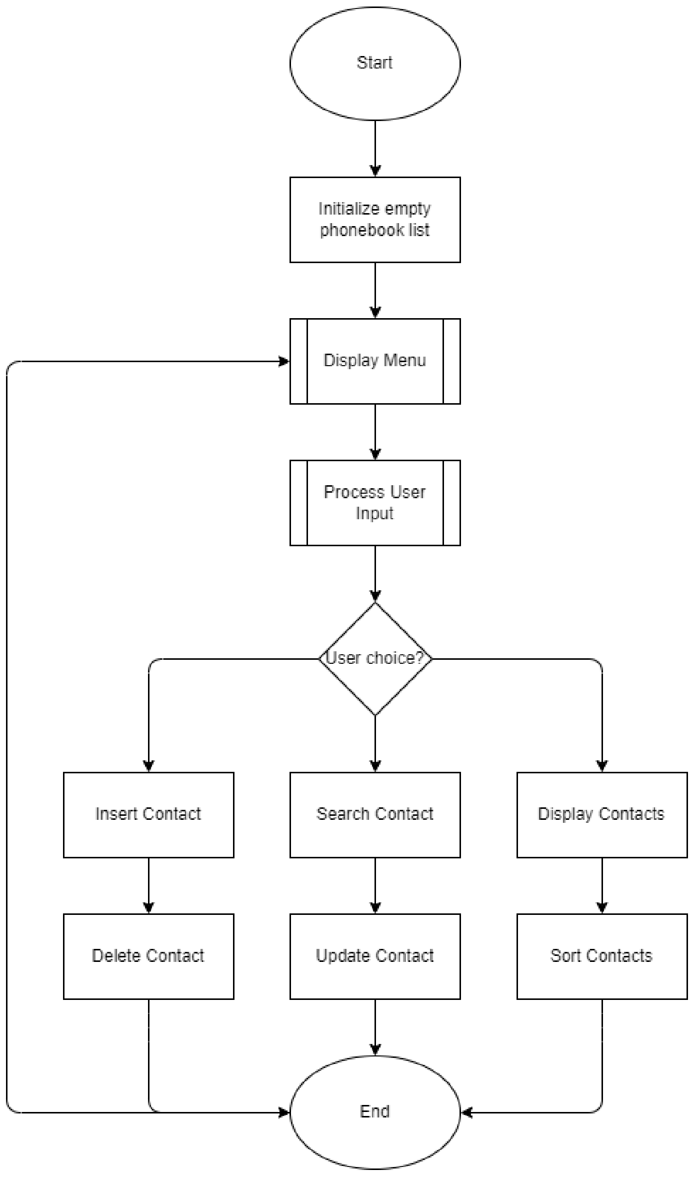
```
# Main function to control the flow of the application function
main():
    phonebook ← []  # Initialize an empty phonebook (list)

    while true:
        displayMenu()  # Show the available options
choice ← getUserChoice()  # Get user's menu choice

        if choice equals 1:
            name, phoneNumber ← getContactDetails()
insertContact(phonebook, name, phoneNumber)
        else if choice equals 2:
            searchName ← getSearchName()             contact
← searchContact(phonebook, searchName)
            displayContact(contact)
else if choice equals 3:
            displayContacts(phonebook)
else if choice equals 4:
```

```
            searchName ← getSearchName()
deleteContact(phonebook, searchName)
        else if choice equals 5:
            searchName, newPhoneNumber ← getUpdateDetails()
updateContact(phonebook, searchName, newPhoneNumber)
        else if choice equals 6:
            sortContacts(phonebook)
else if choice equals 7:
            searchName ← getSearchName()
analyzeSearchEfficiency(phonebook, searchName)         else if choice equals
0:              exit()  # Exit the program
        else:
            print "Invalid choice! Please try again."

# Function to display the menu options
function displayMenu():     print "1.
Insert Contact"     print "2. Search
Contact"     print "3. Display All
Contacts"     print "4. Delete Contact"
print "5. Update Contact"     print "6.
Sort Contacts"     print "7. Analyze
Search Efficiency"
    print "0. Exit"

# Function to get the user's menu choice
function getUserChoice():     print "Enter
your choice: "     return readInput()

# Function to get contact details for insertion
function getContactDetails():     print "Enter
contact name: "     name ← readInput()
```

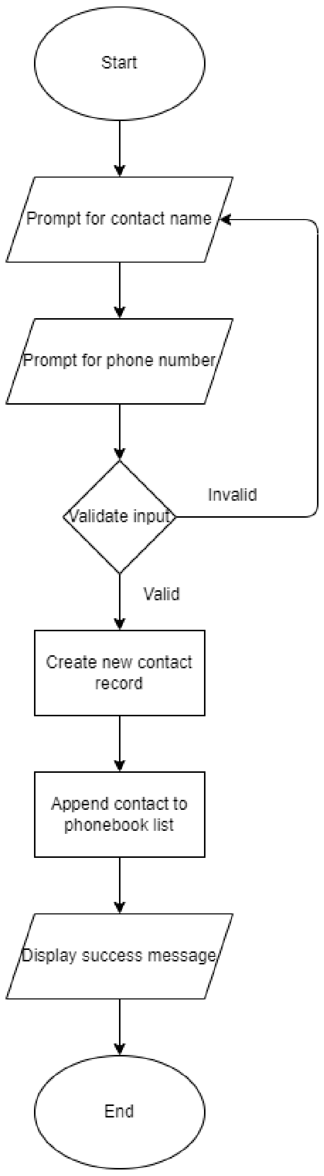
```
print "Enter contact phone number: "
phoneNumber ← readInput()     return name,
phoneNumber

# Function to insert a contact
function insertContact(phonebook, name, phoneNumber):
   contact ← createContact(name, phoneNumber)
append contact to phonebook     print "Contact
added successfully."

# Function to create a contact object function
createContact(name, phoneNumber):
   contact ← {'name': name, 'phoneNumber': phoneNumber}
   return contact

# Function to get the contact name for searching or deletion function
getSearchName():
   print "Enter contact name to search/delete: "
return readInput()

# Function to search for a contact by name
function searchContact(phonebook, searchName):
for each contact in phonebook:         if
contact['name'] equals searchName:
         return contact
   return "Contact not found"

# Function to display a single contact
function displayContact(contact):     if
contact equals "Contact not found":
        print contact
else:
        print "Name: " + contact['name'] + ", Phone: " + contact['phoneNumber'] # Function to
display all contacts function displayContacts(phonebook):     if phonebook is empty:
        print "No contacts available."
    else:         for each contact in
phonebook:
            print "Name: " + contact['name'] + ", Phone: " + contact['phoneNumber']

# Function to delete a contact by name function
deleteContact(phonebook, searchName):     for
each contact in phonebook:         if contact['name']
equals searchName:             remove contact from
phonebook             print "Contact deleted
successfully."
            return
    print "Contact not found"

# Function to get the details for updating a contact function
getUpdateDetails():
    print "Enter contact name to update: "
name ← readInput()     print "Enter new
phone number: "     newPhoneNumber
← readInput()     return name,
newPhoneNumber

# Function to update a contact's phone number function
updateContact(phonebook, searchName, newPhoneNumber):     for
each contact in phonebook:
        if contact['name'] equals searchName:
contact['phoneNumber'] = newPhoneNumber
            print "Contact updated successfully."
            return
    print "Contact not found"

# Function to sort contacts by name
function sortContacts(phonebook):
sort phonebook by contact['name']
print "Contacts sorted alphabetically."

# Function to analyze the efficiency of the search algorithm function
analyzeSearchEfficiency(phonebook, searchName):
    print "The search algorithm uses linear search with O(n) time complexity."

# Function to exit the program function
exit():     print "Exiting the phonebook
application."
    stop execution
```

Flowcharts For The Modules

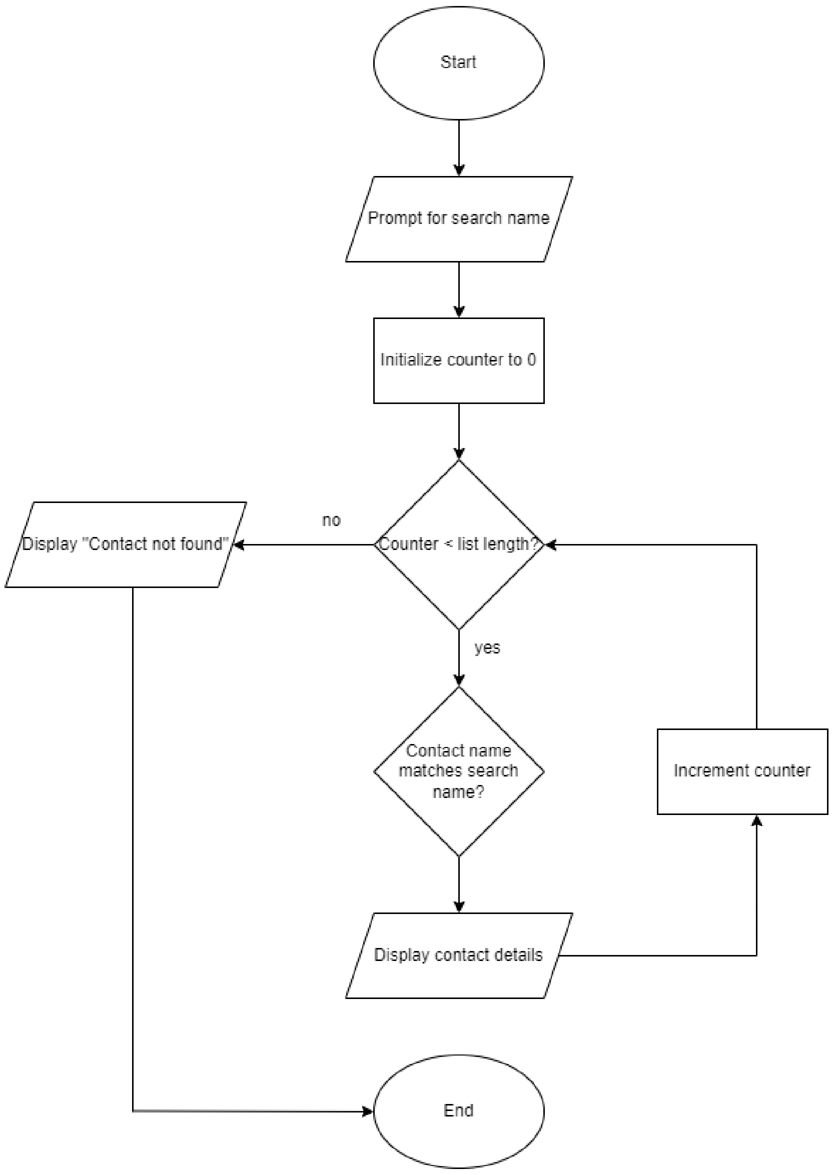
1. Main Phonebook module



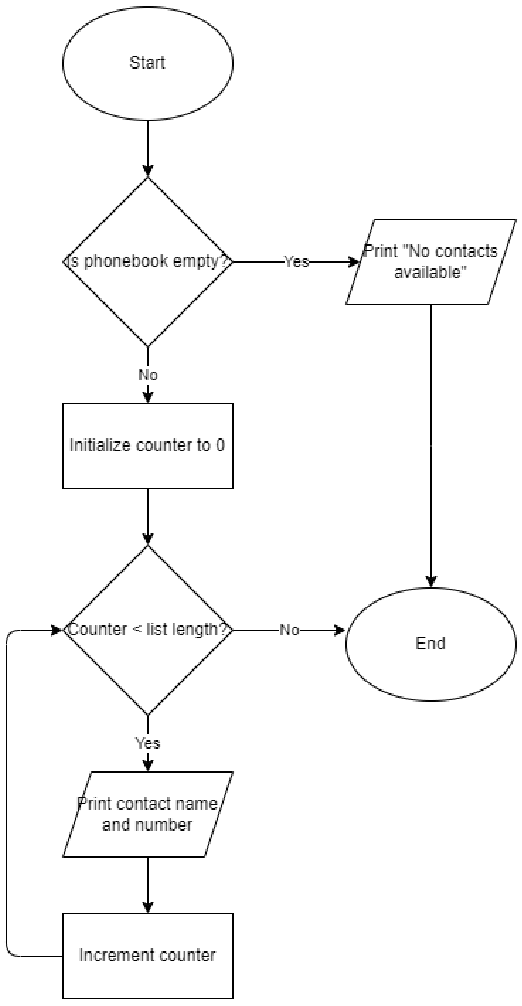
2. Insert Contact Module:



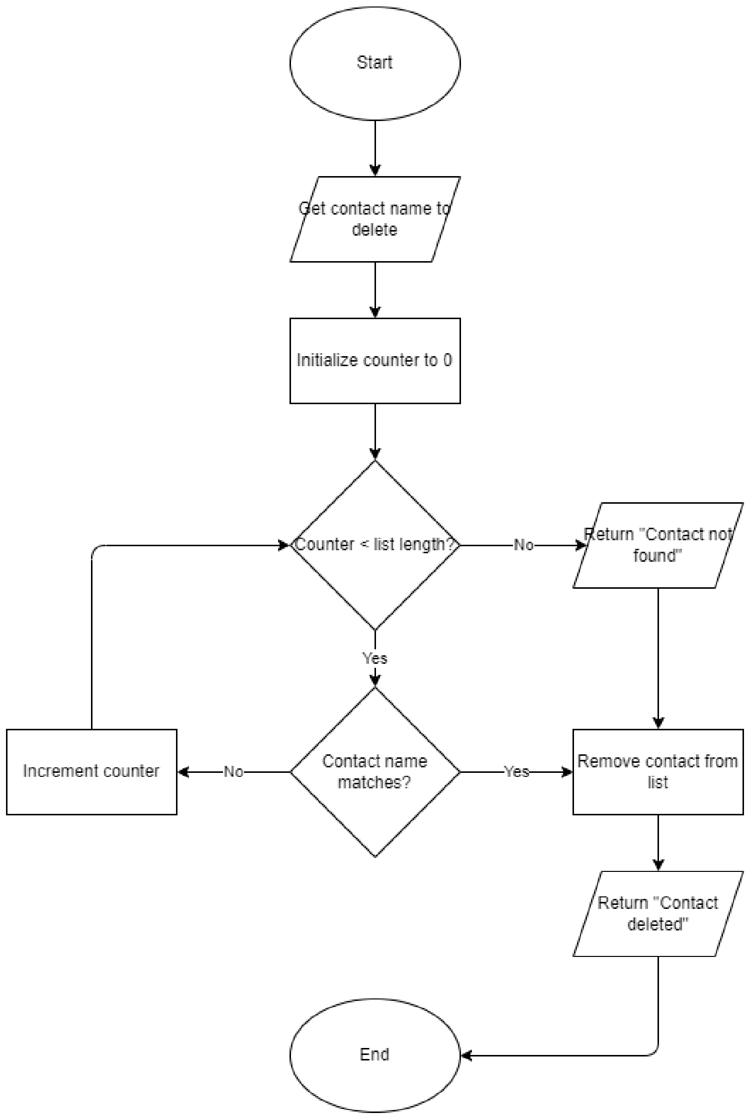
3. Search Contact Module:



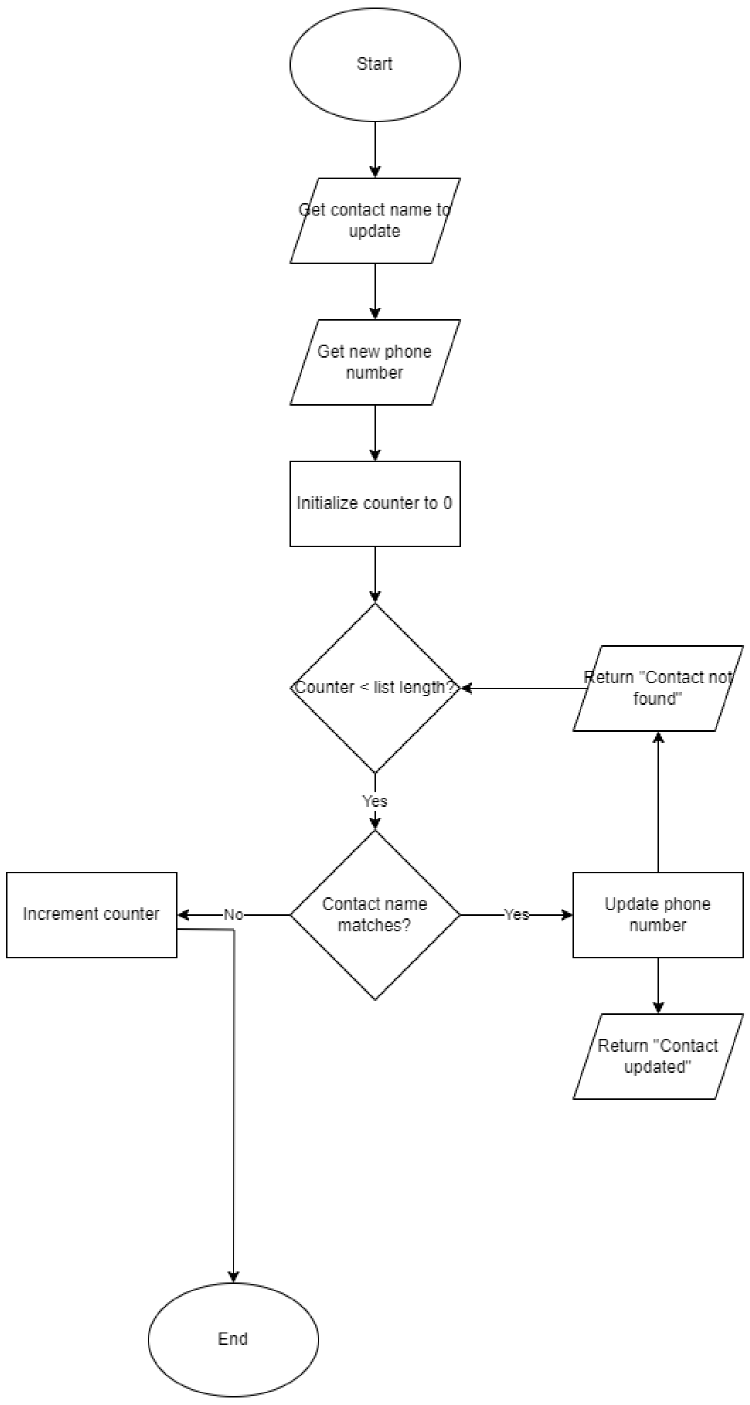
4. Display All Contacts Module:



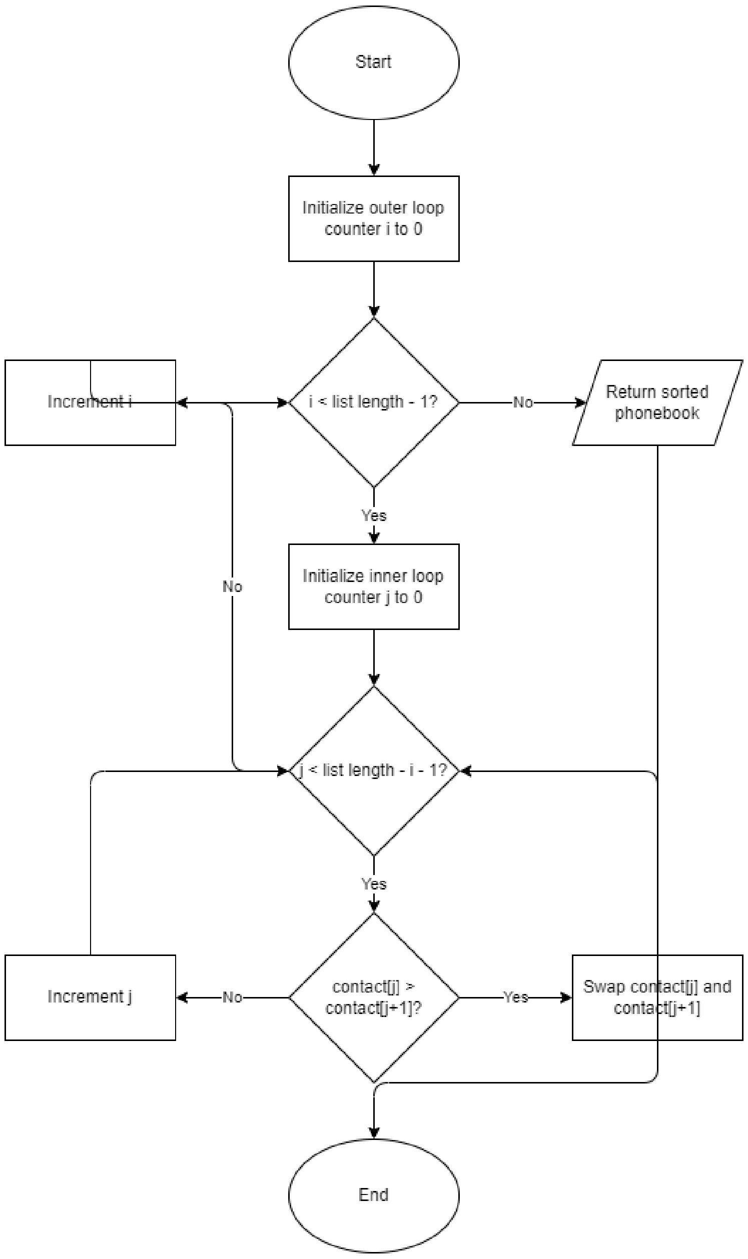
5. Delete Contact Module:



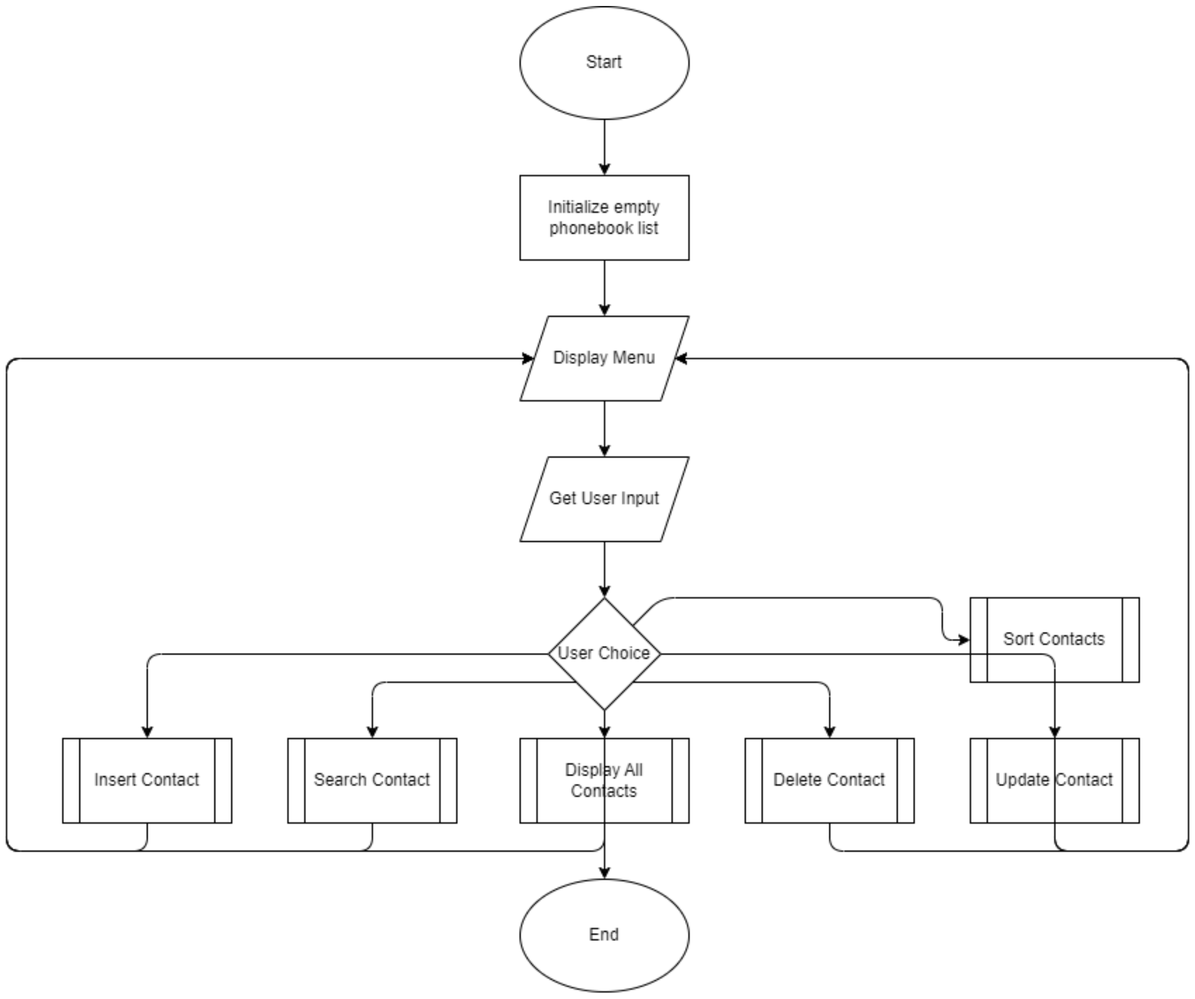
6. Update Contact Module:



7. Sort Contacts Module:


Start
Initialize outer loop counter i to 0
i < list length - 1?
No
Return sorted phonebook
Increment i
Yes
Initialize inner loop counter j to 0
No
j < list length - i - 1?
Yes
contact[j] > contact[j+1]?
No
Increment j
Yes
Swap contact[j] and contact[j+1]
End

Overview Flowchart



SECTION B: Practical implementation of the program designed in section A. [25 Marks]

Use ideas from programming to implement your solution algorithm designed in section A. Students are free

to use any programming language of their choice to implement the data structures and the operations

specified.

```
import java.util.ArrayList;

import java.util.Scanner;


public class PhonebookApp {
   // ArrayList to store names and phone numbers of contacts
static ArrayList<String> names = new ArrayList<>();     static
ArrayList<String> phoneNumbers = new ArrayList<>();

   // Scanner object to capture user input     static
Scanner scanner = new Scanner(System.in);

   public static void main(String[] args) {
// Run the phonebook application
runPhonebook();
   }

   // Main method to handle the menu and user choices
public static void runPhonebook() {        boolean
running = true;

       while (running) {
           // Display menu options to the user
displayMenu();

           // Get the user's choice of action
int choice = getUserChoice();

           // Handle the user's choice
switch (choice) {
               case 1:
insertContact();
break;                case 2:
searchContact();
```

```
                break;
case 3:
displayAllContacts();
                break;
case 4:
deleteContact();
                break;
case 5:
updateContact();
                break;
case 6:
sortContacts();
break;            case 7:
                analyzeSearchEfficiency();                break;            case 0:
                running = false;  // Exit the loop to end the program
System.out.println("Exiting the phonebook application.");
                break;
default:
                System.out.println("Invalid choice. Please try again.");
            }
        }
    }

    // Method to display the phonebook menu
    public static void displayMenu() {
        System.out.println("\nPhonebook Menu:");
        System.out.println("1. Insert Contact");
        System.out.println("2. Search Contact");
        System.out.println("3. Display All Contacts");
        System.out.println("4. Delete Contact");
        System.out.println("5. Update Contact");
        System.out.println("6. Sort Contacts");
        System.out.println("7. Analyze Search Efficiency");
        System.out.println("0. Exit");
        System.out.print("Enter your choice: ");
    }

    // Method to get the user's menu choice     public static int
getUserChoice() {         return scanner.nextInt();  // Capture user
input as an integer
    }

    // Method to insert a new contact into the phonebook
    public static void insertContact() {
        System.out.print("Enter contact name: ");
        String name = scanner.next();  // Capture the contact name
        System.out.print("Enter phone number: ");
        String phoneNumber = scanner.next();  // Capture the phone number

        // Add the contact details to the lists
names.add(name);         phoneNumbers.add(phoneNumber);

        System.out.println("Contact added successfully.");
    }

    // Method to search for a contact by name
public static void searchContact() {
        System.out.print("Enter name to search: ");
        String searchName = scanner.next();  // Get the name to search for
boolean found = false;
```

```
        // Search through the list of names to find the contact
        for (int i = 0; i < names.size(); i++) {
            if (names.get(i).equalsIgnoreCase(searchName)) {
                System.out.println("Contact found:");
                System.out.println("Name: " + names.get(i) + ", Phone: " + phoneNumbers.get(i));
                found = true;
                break;
            }
        }

        if (!found) {
            System.out.println("Contact not found.");
        }
    }

    // Method to display all contacts in the phonebook
public static void displayAllContacts() {          if
(names.isEmpty()) {
            System.out.println("No contacts available.");
        } else {
            System.out.println("All Contacts:");
            // Loop through the list and display each contact
            for (int i = 0; i < names.size(); i++) {
                System.out.println("Name: " + names.get(i) + ", Phone: " + phoneNumbers.get(i));
            }
        }
    }

    // Method to delete a contact by name
public static void deleteContact() {
        System.out.print("Enter name of the contact to delete: ");
String searchName = scanner.next();  // Get the name to delete
boolean deleted = false;

        // Search through the list to find the contact to delete
        for (int i = 0; i < names.size(); i++) {
           if (names.get(i).equalsIgnoreCase(searchName)) {
                names.remove(i);  // Remove the name
phoneNumbers.remove(i);  // Remove the corresponding phone number
System.out.println("Contact deleted successfully.");
                deleted = true;
break;
            }
        }

        if (!deleted) {
            System.out.println("Contact not found.");
        }
    }

    // Method to update a contact's phone number
public static void updateContact() {
        System.out.print("Enter name of the contact to update: ");
String searchName = scanner.next();  // Get the name to update
boolean updated = false;

        // Search through the list to find the contact to update
        for (int i = 0; i < names.size(); i++) {
            if (names.get(i).equalsIgnoreCase(searchName)) {
                System.out.print("Enter new phone number: ");
                String newPhoneNumber = scanner.next();  // Get the new phone number
phoneNumbers.set(i, newPhoneNumber);  // Update the phone number
System.out.println("Contact updated successfully.");
                updated = true;
                break;
            }
        }

        if (!updated) {
            System.out.println("Contact not found.");
        }
    }

    // Method to sort contacts alphabetically by name
public static void sortContacts() {
        // Bubble sort algorithm to sort contacts by name
        for (int i = 0; i < names.size() - 1; i++) {
for (int j = 0; j < names.size() - i - 1; j++) {
                if (names.get(j).compareToIgnoreCase(names.get(j + 1)) > 0) {
                    // Swap the names and phone numbers
                    String tempName = names.get(j);
String tempPhone = phoneNumbers.get(j);
names.set(j, names.get(j + 1));
phoneNumbers.set(j, phoneNumbers.get(j + 1));
names.set(j + 1, tempName);
phoneNumbers.set(j + 1, tempPhone);
                }
            }
        }
        System.out.println("Contacts sorted alphabetically by name.");
    }
```

```
  // Method to analyze the search efficiency of the phonebook
public static void analyzeSearchEfficiency() {
      // Explain the efficiency of the linear search used in this application         System.out.println("The
search operation uses linear search with O(n) time complexity.");
      System.out.println("The more contacts in the phonebook, the longer it takes to search.");
  }
}
```