Advance Data Structure & Algorithm

Course Code: **R1UC503B**

Lab File

For

BACHELOR OF

ENGINEERING & TECHNOLOGY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

GALGOTIAS UNIVERSITY, GREATER NOIDA

UTTAR PRADESH

Student Name: Srasti Bhardwaj

Admission No: 22SCSE1180243

Semester : V

INDEX

S. No.	Date	Tital	Page No.	Teacher's Sign

ADSA Practical File

Name: Srasti Bhardwaj, 22SCSE1180243 , Sec – 08

1. Write a function to find the maximum and minimum elements in an array.

   Source code:

```
#include <bits/stdc++.h>
using namespace std;
void findMaxMin(int arr[], int size, int &maxVal, int &minVal) {
    maxVal = INT_MIN;
    minVal = INT_MAX;
    for (int i = 0; i < size; i++) {
        if (arr[i] > maxVal) {
            maxVal = arr[i];
        }
        if (arr[i] < minVal) {
            minVal = arr[i];
        }
    }
}
int main() {
    int arr[] = {7, 2, 8, 1, 4, 10, 6};
    int size = sizeof(arr) / sizeof(arr[0]);
    int maxVal, minVal;
    findMaxMin(arr, size, maxVal, minVal);
    cout << "Maximum element: " << maxVal << endl;
    cout << "Minimum element: " << minVal << endl;
    return 0;
}
```

   Output:

```
Array elements:  12 3 45 7 89 1 -4 100
Maximum element:  100
Minimum element:  -4
```

2. Write a function to reverse an array in place.

Source code:

```
#include <iostream>
using namespace std;
void reverseArray(int arr[], int n) {
   int start = 0, end = n - 1;
   while (start < end) {
      int temp = arr[start];
      arr[start] = arr[end];
      arr[end] = temp;
      start++;
      end--;
   }
}
int main() {
   int arr[] = {1, 2, 3, 4, 5, 6, 7};
   int n = sizeof(arr) / sizeof(arr[0]);
   cout << "Original array: ";
   for (int i = 0; i < n; i++) {
      cout << arr[i] << " ";
   }
   cout << endl;
   reverseArray(arr, n);
   cout << "Reversed array: ";
   for (int i = 0; i < n; i++) {
      cout << arr[i] << " ";
   }
   cout << endl;
return 0;
}
```

Output:

```
Original array: 1 2 3 4 5 6 7
Reversed array: 7 6 5 4 3 2 1
```

3. Write a function to find the Kth smallest or largest element in an array.

Source code:

```
#include <iostream>
#include <algorithm>
using namespace std;
int findKthSmallest(int arr[], int n, int k) {
    sort(arr, arr + n);
    return arr[k - 1];
}
int findKthLargest(int arr[], int n, int k) {
    sort(arr, arr + n, greater<int>());
    return arr[k - 1];
}
int main() {
    int arr[] = {7, 10, 4, 3, 20, 15};
    int n = sizeof(arr) / sizeof(arr[0]);
    int k = 3;
    cout << "Array elements: ";
    for (int i = 0; i < n; i++) {
      cout << arr[i] << " ";
   }
   cout << endl;
   cout << k << "rd smallest element: " << findKthSmallest(arr, n,k)<<endl;
   cout << k << "rd largest element: " << findKthLargest(arr, n, k) << endl;
return 0;
}
```

Output:

```
Array elements: 7 10 4 3 20 15
3rd smallest element: 7
```

3rd largest element: 10

4. Given an array containing only 0s, 1s, and 2s, sort the array in linear time.

   Source code:

```
#include <iostream>
using namespace std;
void sortArray(int arr[], int n) {
  int low = 0, mid = 0, high = n - 1;
  while (mid <= high) {
    switch (arr[mid]) {
      case 0:
        swap(arr[low], arr[mid]);
        low++;
        mid++;
        break;
      case 1:
        mid++;
        break;
      case 2:
        swap(arr[mid], arr[high]);
        high--;
        break;
    }
  }
}
int main() {
  int arr[] = {2, 0, 1, 2, 0, 1, 1, 0, 2};
  int n = sizeof(arr) / sizeof(arr[0]);
  cout << "Original array: ";
  for (int i = 0; i < n; i++) {
    cout << arr[i] << " ";
  }
  cout << endl;
  sortArray(arr, n);
```

```
    cout << "Sorted array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
 return 0;
}
```

Output:

```
Original array: 2 0 1 2 0 1 1 0 2
Sorted array: 0 0 0 1 1 1 2 2 2
```

5. Write a function to move all zeroes in an array to the end while maintaining the relative order of other elements.

Source code:

```
include<iostream>

using namespace std;

void moveZeroesToEnd(int arr[], int n) {

    int nonZeroIndex = 0;

    for (int i = 0; i < n; i++) {

        if (arr[i] != 0) {

            arr[nonZeroIndex++] = arr[i];

        }

    }

    while (nonZeroIndex < n) {

        arr[nonZeroIndex++] = 0;

    }

}
```

```
int main() {
    int arr[] = {0, 1, 0, 3, 12};
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << "Original array: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
    moveZeroesToEnd(arr, n);
    cout << "Array after moving zeroes to the end: ";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Output:

```
Original array: 0 1 0 3 12
Array after moving zeroes to the end: 1 3 12 0 0
```

6. Write a function to reverse a singly linked list.

   Source code:

```
#include <iostream>
using namespace std;
```

```
struct Node {
       int data;
       Node* next;
       Node(int val) : data(val), next(nullptr) {}
};
Node* reverseList(Node* head) {
    Node* prev = nullptr;
    Node* current = head;
    Node* next = nullptr;
    while (current != nullptr) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}
void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = new Node(4);
    head->next->next->next->next = new Node(5);
    cout << "Original Linked List: ";
    printList(head);
    head = reverseList(head);
    cout << "Reversed Linked List: ";
    printList(head);
    return 0;
}
```

Output:

```
Original Linked List: 1 2 3 4 5
Reversed Linked List: 5 4 3 2 1
```

7. Write a function to detect if a cycle exists in a linked list.

Source code:

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* next;
   Node(int val) : data(val), next(nullptr) {}
};
bool hasCycle(Node* head) {
   if (head == nullptr) return false;
   Node* slow = head;
   Node* fast = head;
   while (fast != nullptr && fast->next != nullptr) {
      slow = slow->next;
      fast = fast->next->next;
       if (slow == fast) {
         return true;
      }
   }
   return false;
}
void printList(Node* head) {
   Node* temp = head;
   while (temp != nullptr) {
      cout << temp->data << " ";
      temp = temp->next;
   }
   cout << endl;
}
int main() {
     Node* head = new Node(1);
   head->next = new Node(2);
   head->next->next = new Node(3);
   head->next->next->next = new Node(4);
   head->next->next->next->next = new Node(5);
   head->next->next->next->next->next = head->next->next;
     if (hasCycle(head)) {
       cout << "Cycle detected in the linked list!" << endl;
    } else {
        cout << "No cycle detected in the linked list." << endl;
     }
     return 0;
}
```

Output:

```
Cycle detected in the linked list!
```

8. Write a function to find the middle element of a linked list.

Source code:

```
#include <iostream>
using namespace std;
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};
int findMiddle(Node* head) {
    if (head == nullptr) return -1;
    Node* slow = head;
    Node* fast = head;
    while (fast != nullptr && fast->next != nullptr) {
        slow = slow->next;
        fast = fast->next->next;
    }
return slow->data;
}
void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}

int main() {
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = new Node(4);
    head->next->next->next->next = new Node(5);
    cout << "Original Linked List: ";
    printList(head);
    int middle = findMiddle(head);
    cout << "Middle element: " << middle << endl;
    return 0;
}
```

Output:

```
Original Linked List: 1 2 3 4 5
Middle element: 3
```

9. Write a function to merge two sorted linked lists into one sorted linked list.

   Source code:

```
#include <iostream>
```

```
using namespace std;
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};
int findMiddle(Node* head) {
    if (head == nullptr) return -1;
    Node* slow = head;
    Node* fast = head;
    while (fast != nullptr && fast->next != nullptr) {
        slow = slow->next;
        fast = fast->next->next;
    }
    return slow->data;
}
void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = new Node(4);
    head->next->next->next->next = new Node(5);
    cout << "Original Linked List: ";
    printList(head);
    int middle = findMiddle(head);
    cout << "Middle element: " << middle << endl;
    return 0;
}
```

Output:

```
Original Linked List: 1 2 3 4 5
Middle element: 3
```

10. Write a function to remove the Nth node from the start/end of a linked list.

Source code:

```
#include <iostream>
using namespace std;
struct Node {
   int data;
   Node* next;
   Node(int val) : data(val), next(nullptr) {}
};
void removeNthNode(Node*& head, int n, bool fromStart) {
   if (head == nullptr) return;
   if (fromStart) {
      if (n == 1) {
         Node* temp = head;
         head = head->next;
         delete temp;
         return;
      }
      Node* temp = head;
      for (int i = 1; i < n - 1 && temp != nullptr; i++) {
         temp = temp->next;
      }
      if (temp != nullptr && temp->next != nullptr) {
         Node* nodeToDelete = temp->next;
         temp->next = temp->next->next;
         delete nodeToDelete;
      }
   } else {
      Node* fast = head;
```

```
        Node* slow = head;
        for (int i = 0; i < n; i++) {
            if (fast == nullptr) return;
            fast = fast->next;
        }
        if (fast == nullptr) {
            Node* temp = head;
            head = head->next;
            delete temp;
            return;
        }
        while (fast != nullptr) {
            fast = fast->next;
            slow = slow->next;
        }
        Node* nodeToDelete = slow->next;
        slow->next = slow->next->next;
        delete nodeToDelete;
    }
}
void printList(Node* head) {
    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
}
int main() {
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = new Node(4);
    head->next->next->next->next = new Node(5);
    cout << "Original Linked List: ";
    printList(head);
    removeNthNode(head, 3, true);
    cout << "After removing 3rd node from start: ";
    printList(head);
    removeNthNode(head, 2, false);
    cout << "After removing 2nd node from end: ";
    printList(head);
    return 0;
}
```

Output:

```
Original Linked List: 1 2 3 4 5
After removing 3rd node from start: 1 2 4 5
After removing 2nd node from end: 1 2 4
```