Experiment **#1**		Student ID	
Date		Student Name	@KLWKS_BOT THANOS

Experiment Title: To implement basic Programs on Arrays and Linked Lists.

Aim/Objective: To understand the concept and implementation of programs on arrays and linked lists.

Description: The students will be able to implement programs on Arrays and Linked Lists.

Pre-Requisites:

Knowledge: Arrays and Linked Lists.

Tools: Code Blocks/Eclipse IDE

Pre-Lab:

1. An *array* is a type of data structure that stores elements of the same type in a contiguous block of memory. In an array A, of size N , each memory location has some unique index i, (where $0 \leq i < N$), that can be referenced as A[i] or A_i .

   Reverse an array of integers.

   Example

   A = [1,2,3]

   Return [3,2,1]

   Function Description

   Complete the function *reverseArray* in the editor below.

   reverseArray has the following parameter(s):

   - *int A[n]*: the array to reverse

   Returns

   - *int[n]*: the reversed array

   Input Format

   The first line contains an integer N , the number of integers in A .

   The second line contains N space-separated integers that make up A.

   Constraints

   $1 \leq N \leq 10^3$

   $1 \leq A[i] \leq 10^4$, where A[i] is the i^{th} integer in A.

Sample Input 1

```
4
1 4 3 2
```

Sample Output 1

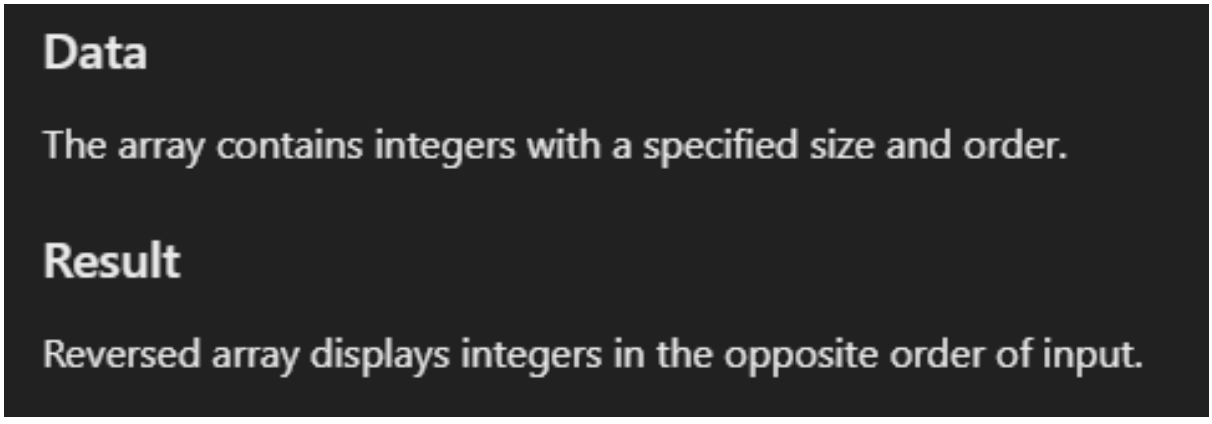
```
2 3 4 1
```

- **Procedure/Program**:

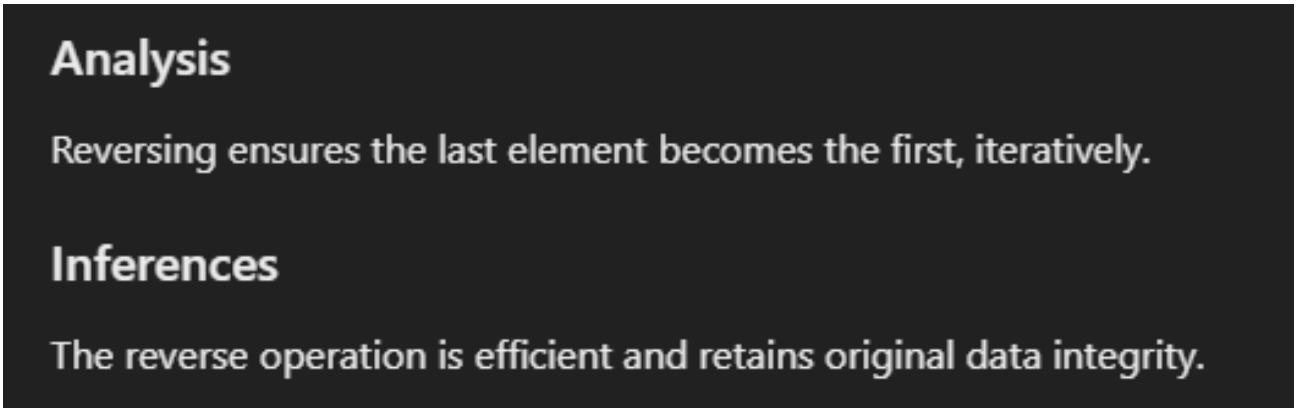
```
import java.util.Scanner;


public class ReverseArray {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);


      System.out.print("Enter the size of the array: ");
      int N = scanner.nextInt();


      int[] A = new int[N];


      System.out.println("Enter the elements of the array:");
      for (int i = 0; i < N; i++) {
         A[i] = scanner.nextInt();
      }
```

```
        System.out.print("Reversed Array: ");
        for (int i = N - 1; i >= 0; i--) {
            System.out.print(A[i] + " ");
        }

        scanner.close();
    }
}
```

- **Data and Results**:



- **Analysis and Inferences**:



2. There is a collection of input strings and a collection of query strings. For each query string, determine how many times it occurs in the list of input strings. Return an array of the results.

Example

stringList = [‘ab’, ‘ab’, ‘abc’]

queries = [‘ab’, ‘abc’, ‘bc’]

There are 2 instances of ‘ab’, 1 of ‘abc’ and 0 of ‘bc’. For each query, add an element to the return array, *results = [2,1,0].*

Function Description

Complete the function matchingStrings in the editor below. The function must return an array of integers representing the frequency of occurrence of each query string in strings.

matchingStrings has the following parameters:

string stringList[n] – an array of strings to search

string queries[q] – an array of query strings

Returns

int[q]: an array of results for each query

Input Format

The first line contains and integer n, the size of stringList[].

Each of the next n lines contains a string stringList[i].

The next line contains q, the size of queries[].

Each of the next q lines contains a string queries[i] .

Constraints

$1 \leq n \leq 10^3$

$1 \leq q \leq 10^3$,

$1 \leq |stringList[i], queries[i]| \leq 20$

- **Procedure/Program**:

```
import java.util.Scanner;

public class StringQueryCount {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Enter the size of stringList: ");
      int n = scanner.nextInt();
      scanner.nextLine();

      String[] stringList = new String[n];
      System.out.println("Enter the strings for stringList:");
      for (int i = 0; i < n; i++) {
         stringList[i] = scanner.nextLine();
      }

      System.out.print("Enter the size of queries: ");
      int q = scanner.nextInt();
      scanner.nextLine();

      String[] queries = new String[q];
      System.out.println("Enter the strings for queries:");
      for (int i = 0; i < q; i++) {
         queries[i] = scanner.nextLine();
      }
```

```
        int[] results = new int[q];
        for (int i = 0; i < q; i++) {
            results[i] = 0;
        }

        for (int i = 0; i < q; i++) {
            for (int j = 0; j < n; j++) {
                if (queries[i].equals(stringList[j])) {
                    results[i]++;
                }
            }
        }

        System.out.print("Results: ");
        for (int i = 0; i < q; i++) {
            System.out.print(results[i] + " ");
        }

        scanner.close();
    }
}
```

- **Data and Results**:

Data

The input consists of strings and queries for frequency comparison.

Result

Counts of each query's occurrences in the string list returned.

- **Analysis and Inferences**:

Analysis

Query strings are checked against input list using nested iteration.

Inferences

Higher frequency queries indicate repeated patterns in the string list.

In-Lab:

1. Given pointers to the heads of two sorted linked lists, merge them into a single, sorted linked list. Either head pointer may be null meaning that the corresponding list is empty.

Example

headA refers to 1 -> 3 -> 7 -> NULL

headB refers to 1 -> 2 -> NULL

The new list is 1 -> 2 -> 3 -> 7 -> NULL

Function Description:

Complete the mergeLists function in the editor below.

mergeLists has the following parameters:

- SinglyLinkedListNode pointer headA: a reference to the head of a list
- SinglyLinkedListNode pointer headB: a reference to the head of a list

Returns

SinglyLinkedListNode pointer: a reference to the head of the merged list

Input Format

The first line contains an integer t, the number of test cases.

The format for each test case is as follows:

The first line contains an integer n, the length of the first linked list.

The next n lines contain an integer each, the elements of the linked list.

The next line contains an integer m, the length of the second linked list.

The next n lines contain an integer each, the elements of the second linked list.

Sample Input:

1

3

1

2

3

2

3

4

Sample Output:

1 2 3 3 4

Explanation:

The first linked list is: 1 -> 2 -> 3 -> NULL

The second linked list is: 3 -> 4 -> NULL

Hence, the merged linked list is: 1 -> 2 -> 3 -> 3 -> 4 -> NULL

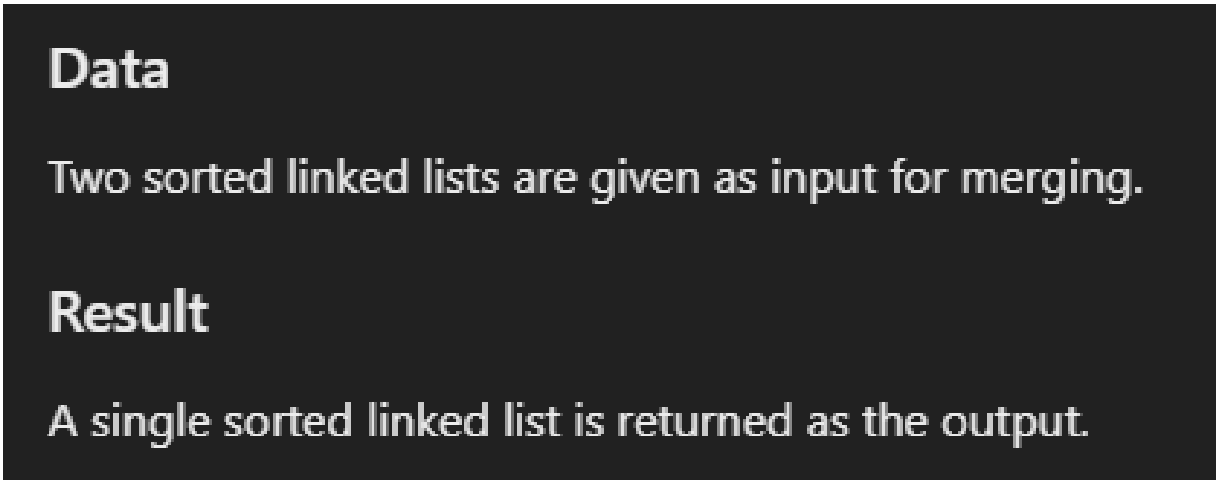
- **Procedure/Program:**

```
import java.util.Scanner;

public class MergeLinkedLists {

  static class SinglyLinkedListNode {
    int data;
    SinglyLinkedListNode next;

    SinglyLinkedListNode(int data) {
      this.data = data;
      this.next = null;
    }
  }

  public static void printList(SinglyLinkedListNode head) {
    while (head != null) {
      System.out.print(head.data + " ");
      head = head.next;
    }
    System.out.println();
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int t = scanner.nextInt();

    while (t > 0) {
```

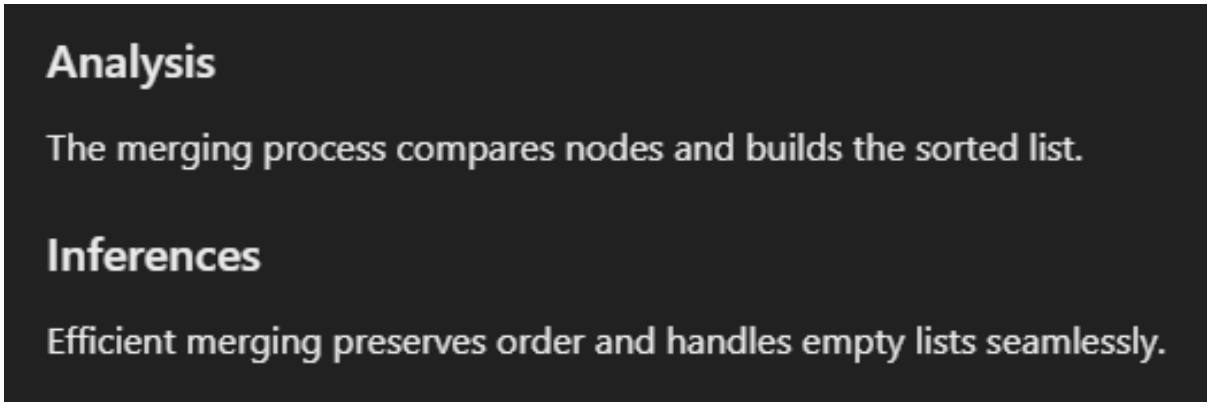
```
t--;

int n = scanner.nextInt();
SinglyLinkedListNode headA = null, currentA = null;
if (n > 0) {
   int data = scanner.nextInt();
   headA = new SinglyLinkedListNode(data);
   currentA = headA;
   for (int i = 1; i < n; i++) {
      data = scanner.nextInt();
      currentA.next = new SinglyLinkedListNode(data);
      currentA = currentA.next;
   }
}

int m = scanner.nextInt();
SinglyLinkedListNode headB = null, currentB = null;
if (m > 0) {
   int data = scanner.nextInt();
   headB = new SinglyLinkedListNode(data);
   currentB = headB;
   for (int i = 1; i < m; i++) {
      data = scanner.nextInt();
      currentB.next = new SinglyLinkedListNode(data);
      currentB = currentB.next;
   }
}

SinglyLinkedListNode dummy = new SinglyLinkedListNode(0);
SinglyLinkedListNode tail = dummy;

while (headA != null && headB != null) {
   if (headA.data <= headB.data) {
      tail.next = headA;
      headA = headA.next;
   } else {
      tail.next = headB;
```

```
            headB = headB.next;
          }
          tail = tail.next;
        }

        if (headA != null) {
          tail.next = headA;
        } else {
          tail.next = headB;
        }

        SinglyLinkedListNode mergedHead = dummy.next;
        printList(mergedHead);
      }

      scanner.close();
    }
}
```

- **Data and Results:**



- **Analysis and Inferences:**



2. You are given the pointer to the head node of a sorted linked list, where the data in the nodes is in ascending order. Delete nodes and return a sorted list with each distinct value in the original list. The given head pointer may be null indicating that the list is empty.

Example

head refers to the first node in the list.

1->2->2->3->3->3->3->NULL

Remove 1 of the 2 data values and return head pointing to the revised list, 1->2->3->NULL

Function Description

Complete the removeDuplicates function in the editor below.

removeDuplicates has the following parameter:

- SinglyLinkedListNode pointer head: a reference to the head of the list

Returns

- SinglyLinkedListNode pointer: a reference to the head of the revised list

Input Format

The first line contains an integer t , the number of test cases.
The format for each test case is as follows:
The first line contains an integer n, the number of elements in the linked list.
Each of the next n lines contains an integer, the data value for each of the elements of the linked list.

Constraints

1<=t<=10
1<=n<=1000
1<=list[i]<=1000

Sample Input

```
STDIN   Function
-----   --------
1       t = 1
5       n = 5
1       data values = 1, 2, 2, 3, 4
2
2
3
4
```

Sample Output

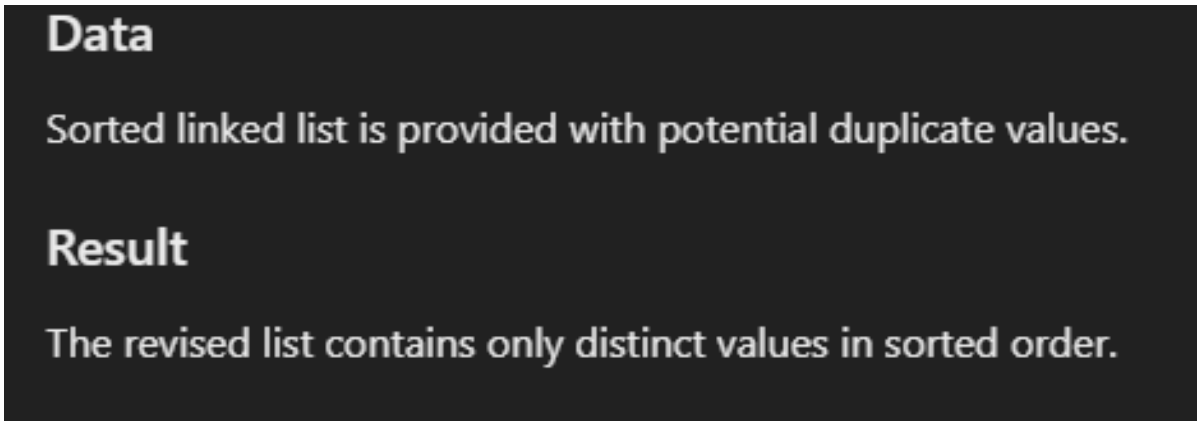
1 2 3 4

- **Procedure/Program:**

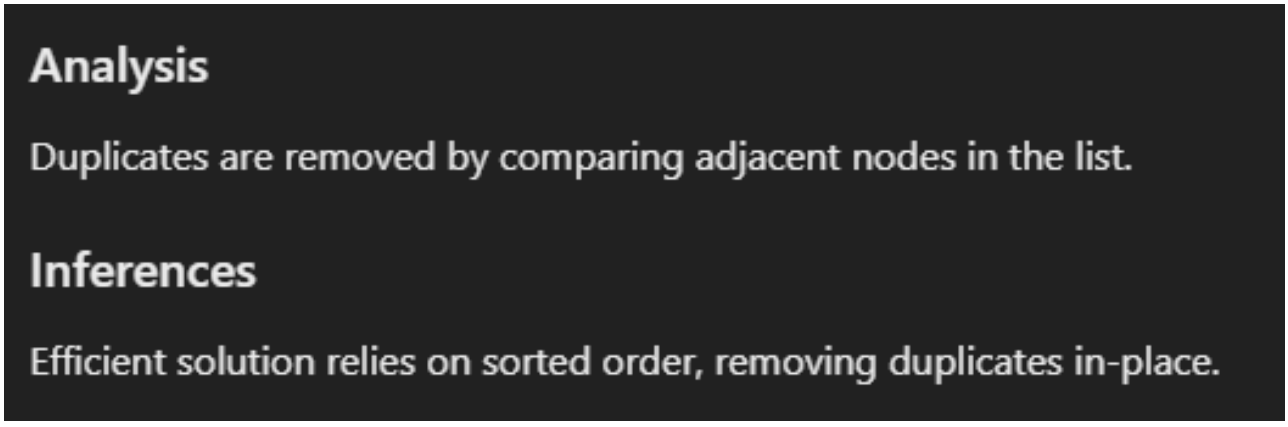
```
import java.util.Scanner;

public class RemoveDuplicatesFromList {

  static class SinglyLinkedListNode {
    int data;
    SinglyLinkedListNode next;

    SinglyLinkedListNode(int data) {
      this.data = data;
      this.next = null;
    }
  }

  public static void printList(SinglyLinkedListNode head) {
    while (head != null) {
      System.out.print(head.data + " ");
      head = head.next;
    }
    System.out.println();
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int t = scanner.nextInt();

    while (t > 0) {
      t--;

      int n = scanner.nextInt();
      SinglyLinkedListNode head = null, current = null;

      if (n > 0) {
```

```
            int data = scanner.nextInt();
            head = new SinglyLinkedListNode(data);
            current = head;
            for (int i = 1; i < n; i++) {
                data = scanner.nextInt();
                current.next = new SinglyLinkedListNode(data);
                current = current.next;
            }
        }

        current = head;
        while (current != null && current.next != null) {
            if (current.data == current.next.data) {
                current.next = current.next.next;
            } else {
                current = current.next;
            }
        }

        printList(head);
    }

    scanner.close();
  }
}
```

- **Data and Results:**



- **Analysis and Inferences:**



Post-Lab:

1. How will you modify a linked list of integers so that all even numbers appear before all odd numbers in the modified linked list? Also, keep the even and odd numbers in the same order.
   Example:
   Input: 17->15->8->12->10->5->4->1->7->6->NULL
   Output: 8->12->10->4->6->17->15->5->1->7->NULL.

- **Procedure**:

```
import java.util.Scanner;
import java.util.List;
import java.util.ArrayList;

public class SeparateEvenOdd {

  static class SinglyLinkedListNode {
    int data;
    SinglyLinkedListNode next;

    SinglyLinkedListNode(int data) {
      this.data = data;
      this.next = null;
    }
  }

  public static void printList(SinglyLinkedListNode head) {
    while (head != null) {
```

```
        System.out.print(head.data + " ");
        head = head.next;
    }
    System.out.println();
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    System.out.print("Enter the number of elements: ");
    int n = scanner.nextInt();

    System.out.print("Enter the elements: ");
    List<Integer> dataList = new ArrayList<>();
    for (int i = 0; i < n; i++) {
        dataList.add(scanner.nextInt());
    }

    SinglyLinkedListNode head = new SinglyLinkedListNode(dataList.get(0));
    SinglyLinkedListNode current = head;
    for (int i = 1; i < n; i++) {
        current.next = new SinglyLinkedListNode(dataList.get(i));
        current = current.next;
    }

    SinglyLinkedListNode evenHead = null, evenTail = null;
```

```
SinglyLinkedListNode oddHead = null, oddTail = null;


current = head;
while (current != null) {
   if (current.data % 2 == 0) {
      if (evenHead == null) {
         evenHead = evenTail = current;
      } else {
         evenTail.next = current;
         evenTail = evenTail.next;
      }
   } else {
      if (oddHead == null) {
         oddHead = oddTail = current;
      } else {
         oddTail.next = current;
         oddTail = oddTail.next;
      }
   }
   current = current.next;
}


if (evenTail != null) {
   evenTail.next = oddHead;
}
if (oddTail != null) {
```

```
            oddTail.next = null;
        }

        head = (evenHead != null) ? evenHead : oddHead;

        printList(head);

        scanner.close();
    }
}
```

- **Data and Results:**

Data:

The input is a linked list with unsorted integers.

Result:

The output is a reordered linked list of integers.

- **Analysis and Inferences:**

Analysis:

Even numbers are placed before odd numbers, preserving relative order.

Inferences:

Reordering preserves structure; even elements come before odd elements.

2. Given a linked list that is sorted based on absolute values. Sort the list based on actual values.

   Examples:

   Input-1 : 1 -> -10

   Output-1: -10 -> 1

   Input-2 : 1 -> -2 -> -3 -> 4 -> -5

   Output-2: -5 -> -3 -> -2 -> 1 -> 4

- **Procedure**:

```
import java.util.Scanner;

public class BubbleSortLinkedList {

  static class SinglyLinkedListNode {
    int data;
    SinglyLinkedListNode next;

    SinglyLinkedListNode(int data) {
      this.data = data;
      this.next = null;
    }
  }

  public static void printList(SinglyLinkedListNode head) {
    while (head != null) {
      System.out.print(head.data + " ");
      head = head.next;
```

```
    }
    System.out.println();
}

public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);

    int n = scanner.nextInt();

    SinglyLinkedListNode head = null;
    SinglyLinkedListNode current = null;

    for (int i = 0; i < n; i++) {
        int data = scanner.nextInt();
        SinglyLinkedListNode newNode = new SinglyLinkedListNode(data);
        if (head == null) {
            head = newNode;
            current = head;
        } else {
            current.next = newNode;
            current = current.next;
        }
    }

    SinglyLinkedListNode dummy = new SinglyLinkedListNode(0);
    dummy.next = head;
```

```
        SinglyLinkedListNode prev = dummy;
        current = head;

        while (current != null && current.next != null) {
            if (current.data > current.next.data) {
                SinglyLinkedListNode temp = current.next;
                current.next = temp.next;
                prev.next = temp;
                temp.next = current;
                prev = dummy;
            } else {
                prev = current;
                current = current.next;
            }
        }

        current = dummy.next;
        printList(current);

        scanner.close();
    }
}
```

- **Sample VIVA-VOCE Questions (In-Lab):**

1. What is the difference between an array and a linked list?

- **Array**: Fixed size, contiguous memory, direct access via index.
- **Linked List**: Dynamic size, nodes with pointers, sequential access.

2. What are the advantages and disadvantages of using linked lists?

- **Advantages**: Dynamic size, efficient insertions/deletions.
- **Disadvantages**: Extra memory for pointers, slower access time, complex implementation.

3. How will you find the middle element of a singly linked list without iterating the list more than once?

- Use two pointers: slow (moves 1 step) and fast (moves 2 steps). When fast reaches end, slow is at the middle.

4. What is the time complexity for accessing an element in an array?

- **O(1)** (constant time).

5. How would you reverse an array in-place in linear time and constant space?

- Swap elements from both ends of the array towards the center in **O(n)** time and **O(1)** space.

Evaluator Remark (if Any):	**Marks Secured ___out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.