Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title: To implement a program in cpp for Array.

Steps to Implement the Program

1. **Start** the program.
2. Include the necessary header file <iostream> for input/output operations.
3. Use the using namespace std to avoid writing std:: repeatedly.
4. Declare the required variables:
   - An integer array A[100] to store the elements.
   - An integer variable n to store the size of the array.
   - An integer variable sum initialized to 0 to store the sum of array elements.
5. Prompt the user to **input the size of the array**.
6. Use a for loop to **input the elements** of the array and store them in the array A.
7. Use another for loop to **calculate the summation** of all the elements of the array by adding them to the sum variable.
8. Display the summation of the array elements.
9. **End** the program.

Algorithm:-

Step 1: Start
Step 2: Include <iostream> and use namespace std.
Step 3: Declare an array A[100], integer variables n (size of the array) and sum = 0.
Step 4: Display "Enter size of array: ".
Step 5: Input n (size of the array).
Step 6: For i = 0 to i < n:
- Display "Enter array element: ".
- Input A[i].

Step 7: For i = 0 to i < n:
- Add A[i] to sum.

Step 8: Display "Summation of Array is = " followed by the value of sum.
Step 9: Stop

Title: To implement a program in cpp for Array.

Code:

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

int main() {
    int A[100];
    int n, sum = 0;

    cout << "Enter size of array: ";
    cin >> n;

for (int i = 0; i < n; i++) {
cout << "Enter array element: ";
 cin >> A[i];
 }

  for (int i = 0; i < n; i++) {
 sum += A[i];
 }

cout << "Summation of Array is = " << sum << endl;

 return 0;
}
```

Output:

Enter size of array : 5
Enter Array elements:
10
20
30
40
50
Summation of Array is = 150

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title**: Implementing a Program in C++ for Multidimensional Arrays**

Objective**:**

To understand and demonstrate how to declare, initialize, and access multidimensional arrays in C++.

Steps**:**

1. **Start the Program**:
   - Include the required header file `#include<iostream>` to use input-output functions.
2. **Declare a Multidimensional Array**:
   - Choose a suitable size for the array (e.g., 2D or 3D array).
   - Syntax for declaring a 2D array: `data_type array_name[rows][columns];`
3. **Initialize the Array**:
   - Initialize the array elements either **statically** (fixed values) or **dynamically** (taking input from the user).
4. **Access and Display Elements**:
   - Use nested `for` loops to access and display the elements of the array.
5. **Perform Operations (Optional)**:
   - You can perform operations such as summation, matrix addition, or multiplication.
6. **End the Program**:
   - Print results and terminate the program.

Algorithm**:**

The following algorithm describes the steps for a 2D array.

1. **Step 1**: Start the program.
2. **Step 2**: Include the necessary header file: `#include<iostream>`.
3. **Step 3**: Declare the 2D array using syntax: `int arr[m][n];`, where `m` is the number of rows, and `n` is the number of columns.
4. **Step 4**: Use nested loops to take input from the user:
   - Outer loop: Iterates through rows.
   - Inner loop: Iterates through columns.
5. **Step 5**: Store the values entered by the user into the array.
6. **Step 6**: Use another set of nested loops to display the values.
7. **Step 7**: Perform any additional operations (like summing elements, etc., if needed).
8. **Step 8**: End the program.

Title: To implement a program in cpp for Multidimensional Arrays

Code:

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

 int main() {
   int rows, cols;

  // Input the dimensions of the matrix
  cout << "Enter the number of rows and columns: ";
  cin >> rows >> cols;

 int arr1[10][10], arr2[10][10], sum[10][10];

  // Input elements of the first matrix
    cout << "Enter elements of first matrix:\n";
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
      cin >> arr1[i][j];
 }
}

  // Input elements of the second matrix
  cout << "Enter elements of second matrix:\n";
  for (int i = 0; i < rows; i++) {
 for (int j = 0; j < cols; j++) {
  cin >> arr2[i][j];
  }
 }

  // Adding the two matrices
    for (int i = 0; i < rows; i++) {
for (int j = 0; j < cols; j++) { sum[i][j] = arr1[i][j]
+ arr2[i][j];
          }

      }

       // Display the sum matrix
       cout << "Sum of the matrices:\n";
      for (int i = 0; i < rows; i++) {
```

```
        for (int j = 0; j < cols; j++) {
          cout << sum[i][j] << " ";
      }
      cout << endl;
  }

  return 0;
   }
```

Output:

```
Enter the number of rows and columns: 2 2
Enter elements of first matrix:
1 2
2 1
Enter elements of second matrix:
2 2
1 1
Sum of the matrices:
3 4
3 2
```

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

: Title: To implement a program in cpp for Matrices.

Steps:

1. **Input Matrix Dimensions**: Accept the number of rows and columns of the matrices.
2. **Declare Matrices**: Define three 2D arrays - one for each input matrix and one for storing the result (sum).
3. **Input Elements of First Matrix**: Use nested loops to input elements of the first matrix.
4. **Input Elements of Second Matrix**: Use nested loops to input elements of the second matrix.
5. **Add Matrices**: Perform element-wise addition of the two matrices using nested loops.
6. **Store the Result**: Store the sum of each corresponding element in a third matrix.
7. **Display the Result**: Use nested loops to display the resultant matrix.

Algorithm:

1. **Start**
2. **Declare Variables**:
   - `rows` and `cols` for matrix dimensions
   - `arr1[10][10]`, `arr2[10][10]`, `sum[10][10]` for matrices.
3. **Input Dimensions**:
   - Prompt the user to enter the number of rows and columns.
   - Store the input in `rows` and `cols`.
4. **Input First Matrix**:
   - For `i = 0` to `rows-1`:
     - For `j = 0` to `cols-1`:
       - Input `arr1[i][j]`.
5. **Input Second Matrix**:
   - For `i = 0` to `rows-1`:
     - For `j = 0` to `cols-1`:
       - Input `arr2[i][j]`.
6. **Add Matrices**:
   - For `i = 0` to `rows-1`:
     - For `j = 0` to `cols-1`:
       - `sum[i][j] = arr1[i][j] + arr2[i][j]`.
7. **Display Sum Matrix**:
   - For `i = 0` to `rows-1`:
     - For `j = 0` to `cols-1`:
       - Print `sum[i][j]`.
     - Move to the next line after each row.
8. **Stop**

Title: To implement a program in cpp for Matrices.

Code:

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

int main() {
int rows, cols;

// Input the dimensions of the matrix
cout << "Enter the number of rows and columns: ";
cin >> rows >> cols;

int arr1[10][10], arr2[10][10], sum[10][10];

// Input elements of the first matrix
cout << "Enter elements of first matrix:\n";
for (int i = 0; i < rows; i++) {
for (int j = 0; j < cols; j++) {
cin >> arr1[i][j];
}
}

// Input elements of the second matrix
cout << "Enter elements of second matrix:\n";
for (int i = 0; i < rows; i++) {
for (int j = 0; j < cols; j++) {
cin >> arr2[i][j];
}
}

// Adding the two matrices
for (int i = 0; i < rows; i++) {
for (int j = 0; j < cols; j++) {
sum[i][j] = arr1[i][j] + arr2[i][j];
}
}

// Display the sum matrix
cout << "Sum of the matrices:\n";
for (int i = 0; i < rows; i++) {
for (int j = 0; j < cols; j++) {
cout << sum[i][j] << " ";
}
cout << endl;
}
```

```
 return 0;
}
```

Output:

```
Enter the number of rows and columns: 2 2
Enter elements of first matrix:
 2 1
 2 1
Enter elements of second matrix:
2 2
1 1
Sum of the matrices:
4 3
3 2
```

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

: Title: To implement a program in cpp for Stack .

Algorithm:

1. If `top` >= `MAX_SIZE - 1`:
   - Print "Stack Overflow".
   - Return.
2. Else:
   - Prompt the user to input an element.
   - Increment `top` by 1.
   - Store the element at `s[top]`.

2. Display Operation
Algorithm:

1. If `top == -1`:
   - Print "Stack is Empty".
   - Return.
2. Else:
   - Loop from `i = top` to `0`:
     - Print each element `s[i]`.

3. Pop Operation
Algorithm:

1. If `top == -1`:
   - Print "Underflow".
   - Return.
2. Else:
   - Print the element at `s[top]`.
   - Decrement `top` by 1.

4. Peep Operation
Algorithm:

1. Prompt the user to input a position `i`.
2. If `(top - i + 1) < 0`:
   - Print "Underflow".
   - Return.
3. Else:
   - Print the element at `s[top - i + 1]`.

5. Change Operation
Algorithm:

1. Prompt the user to input a position `i`.
2. If `(top - i + 1) < 0`:
   - Print "Underflow".
   - Return.
3. Else:
   - Prompt the user to input a new element `n`.
   - Replace `s[top - i + 1]` with `n`.

6. Exit Operation
Algorithm:

1. Call the `exit(0)` function to terminate the program.

: Title: To implement a program in cpp for Stack .

Code:

```
#include <iostream>
#include <cstdlib>
class stack{
private:
static const int MAX_SIZE = 10;
int s[MAX_SIZE];
int top;
int ele;
int i;

public:
stack() {
top = -1;
}
void push();
void display();
void pop();
void peep();
void change();
};
void stack::push() {
            if (top>=MAX_SIZE-1)
            std::cout << "\nstack is overflow:";
    else {
                std::cout << "\nEnter element:"; std::cin >> ele;
                top++; s[top] = ele;
    }
  }
```

```
void stack::display() {
        if (top == -1)
        std::cout << "\nstack is Empty";else {

  std::cout << "\nElements in stack are:\n";
      for (i = top; i >= 0; i--)
      std::cout << s[i] << "\t";
}

}
void stack::pop() {
       if (top == -1)
      std::cout << "\nUnderflow";
else {
      std::cout << "\npop ele is " << s[top]; top--;
}
}

void stack::peep() {
      std::cout << "\nEnter position:"; std::cin >> i;
       if ((top - i + 1) < 0)
      std::cout << "\nUnderflow"; else
       std::cout << "\nPeep ele is " << s[top - i + 1];
}

void stack::change() {
        std::cout << "\nEnter position: "; std::cin >> i;
        if ((top - i + 1) < 0)
        std::cout << "\nUnderflow";
else {
int n;
std::cout << "\nEnter element:"; std::cin >> n;
s[top - i + 1] = n;
}
}
int main() { stack s; int ch; int n;
std::cout << "Enter size of stack: "; std::cin >> n;
while (true) {
std::cout << "\n1. Push 2. Display 3. Pop 4. Peep 5. Change 6. Exit\n"; std::cout <<
"\nEnter choice: ";
std::cin >> ch;
switch (ch) {
```

```
case 1:

s.push();
break;

    case 2:
    s.display();
    break;
    case 3:
    s.pop();
    break;
    case 4:
    s.peep();
    break;
    case 5:
    s.change();
    break;
    case 6:
    exit(0);
    default:
    std::cout << "Invalid choice!";

    }

    }
    return 0;

}
```

Output:
Enter size of stack: 5

1. Push 2. Display 3. Pop 4. Peep 5. Change 6. Exit

Enter choice: 1

Enter element:1

1. Push 2. Display 3. Pop 4. Peep 5. Change 6. Exit

Enter choice: 1

Enter element:2

1. Push 2. Display 3. Pop 4. Peep 5. Change 6. Exit

Enter choice: 1

Enter element:3

1. Push 2. Display 3. Pop 4. Peep 5. Change 6. Exit

Enter choice: 1

Enter element:4

1. Push 2. Display 3. Pop 4. Peep 5. Change 6. Exit

Enter choice: 2

Elements in stack are:
4 3 2 1

1. Push 2. Display 3. Pop 4. Peep 5. Change 6. Exit

Enter choice: 3

pop ele is 4
1. Push 2. Display 3. Pop 4. Peep 5. Change 6. Exit

Enter choice: 4

Enter position:2

Peep ele is 2
1. Push 2. Display 3. Pop 4. Peep 5. Change 6. Exit

Enter choice: 5

Enter position: 1

Enter element:3

1. Push 2. Display 3. Pop 4. Peep 5. Change 6. Exit

Enter choice: 2

Elements in stack are:
3 2 1
1. Push 2. Display 3. Pop 4. Peep 5. Change 6. Exit

Enter choice:

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: __ _ _ _ _ _ _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

: Title: To implement a program in cpp for Infix to Postfix

Steps to Implement Infix to Postfix Conversion

1. **Understand Operator Precedence**: Operators have precedence levels (`^` > `*//` > `+/-`).
2. **Use a Stack**:
   - Store operators and parentheses temporarily.
   - Push `(` directly into the stack.
   - Pop operators when `)` is encountered or precedence rules dictate.
3. **Handle Operands**:
   - Directly append operands (numbers/letters) to the postfix expression.
4. **Scan the Infix Expression**:
   - From left to right, process each character:
     - Append operands to the result.
     - Push operators into the stack based on precedence.
     - Pop from the stack when encountering `)` or lower-precedence operators.
5. **Empty the Stack**: After traversing the expression, pop remaining operators from the stack and append to the result.

Algorithm

Input:

- A valid infix expression as a string.

Output:

- Corresponding postfix expression.

Algorithm Steps:

1. **Initialize**:
   - Create an empty stack `s` for operators.
   - Create an empty string `postfix` for the result.
2. **Traverse the Infix Expression** (for each character `ch`):
   - **If `ch` is an operand** (a letter or digit):
     - Append it to `postfix`.
   - **If `ch` is `(`**:
     - Push it to the stack.
   - **If `ch` is `)`**:
     - Pop operators from the stack and append to `postfix` until a `(` is encountered.
     - Remove `(` from the stack.

- **If `ch` is an operator**:
  - While the stack is not empty and the precedence of the top operator is **greater than or equal** to `ch`:
    - Pop the top operator from the stack and append it to `postfix`.
  - Push `ch` onto the stack.

3. **Pop Remaining Operators**:
   - After processing the entire expression, pop all remaining operators from the stack and append them to `postfix`.
4. **Return Result**:
   - The final `postfix` string contains the converted expression.

: Title: To implement a program in cpp for Infix to Postfix .

Code:

```
#include <iostream>
#include <stack>
#include <string>

using namespace std;

// Function to return precedence of operators
int precedence(char op) {
    if (op == '^') return 3;
    if (op == '*' || op == '/') return 2;
    if (op == '+' || op == '-') return 1;
    return 0;
}

// Function to check if the character is an operator
bool isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/' || c == '^');
}

// Function to convert infix expression to postfix
string infixToPostfix(string infix) {
    stack<char> s;  // Stack to hold operators and parentheses
    string postfix; // Resulting postfix expression

    for (int i = 0; i < infix.length(); i++) {
        char ch = infix[i];

        // If the character is an operand (number or letter), add it to postfix
        if (isalnum(ch)) {
            postfix += ch;
        }

    // If the character is '(', push it to the stack
```

```
  else if (ch == '(') {
            s.push(ch);
        }
        // If the character is ')', pop from stack until '(' is found
        else if (ch == ')') {
            while (!s.empty() && s.top() != '(') {
                postfix += s.top();
                s.pop();
            }
            s.pop(); // Remove '(' from the stack
        }
        // If the character is an operator
        else if (isOperator(ch)) {
            while (!s.empty() && precedence(s.top()) >= precedence(ch)) {
                postfix += s.top();
                s.pop();
            }
            s.push(ch);
        }
}

    // Pop remaining operators from the stack
    while (!s.empty()) {
        postfix += s.top();
        s.pop();
    }

    return postfix;
}

int main() {
    string infix;

    // Get user input for infix expression
    cout << "Enter an infix expression: ";
    cin >> infix;

    // Convert infix to postfix
    string postfix = infixToPostfix(infix); // Output the postfix expression
    cout << "Postfix Expression: " << postfix << endl;

    return 0;
}
```

Output:

Enter an infix expression: +ABC

Postfix Expression: ABC+

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _—_ _ _ _ _ _   Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _   Batch: _ _ _ _ _ _   Class: MCA. -I

Subject: Sign. of Teacher

Title: To implement a program in cpp for Infix to Prefix.

Algorithm

1. **Initialize a stack** to hold intermediate results.
2. **Input the postfix expression** as a string.
3. **Process each character** of the postfix expression:
   - If the character is an **operand**, push it into the stack.
   - If the character is an **operator**:
     - Pop the top two operands from the stack.
     - Form a subexpression as `"( operand1 operator operand2 )"`.
     - Push the resultant expression back into the stack.
4. **Repeat the process** until the entire postfix expression is processed.
5. **The final expression** on top of the stack will be the infix expression.

Steps

1. Start with a **postfix expression** as input.
2. Declare a **stack** to hold intermediate expressions.
3. Traverse the input string character by character.
4. For each character:
   - If it is an **operand**, push it into the stack.
   - If it is an **operator**, perform the following:
     - Pop the top two operands from the stack.
     - Combine them into an infix expression with the operator in the form `(operand1 operator operand2)`.
     - Push the resulting expression back onto the stack.
5. Once the traversal is complete, print the expression present on top of the stack.

Title: To implement a program in cpp for Infix to Prefix.
Code:

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

int main() {
    char p[30], temp[50], opr[10], op1[10], op2[10];
```

```
stack<string> stk;

// Take input postfix expression
cout << "\nEnter the postfix expression: ";
cin >> p;

int i = 0;

// Process each character of the postfix expression
while(p[i] != '\0') {
    // If the character is an operator
    if(p[i] == '+' || p[i] == '-' || p[i] == '*' || p[i] == '/' || p[i] == '^') {

        // Pop the two operands from the stack
        op2[0] = '\0'; op1[0] = '\0'; opr[0] = p[i];
        opr[1] = '\0';  // Set operator

        // Pop two operands from the stack
        op2 = stk.top(); stk.pop();
        op1 = stk.top(); stk.pop();
        // Create the infix expression in the form (op1 operator op2)
        strcpy(temp, "(");
        strcat(temp, op1.c_str());  // Append op1 to temp
        strcat(temp, opr);          // Append operator
        strcat(temp, op2.c_str());  // Append op2
        strcat(temp, ")");

        // Push the result back into the stack
        stk.push(temp);
    }
    else {
        // If it's an operand, push it to the stack
        temp[0] = p[i];
        temp[1] = '\0';
        stk.push(temp);
    }
    i++;
}
```

```
    // The top of the stack will contain the infix expression
    cout << "\nInfix expression is: " << stk.top() << endl;

    return 0;
}
```

Output:

Enter the postfix expression: ab+c*

Infix expression is: (a+b)*c

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: __ _ _ _ _ _ _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title:To implement a program in cpp for Queue using array.

Algorithm:

Step 1: Initialization

- Initialize `f = 0` and `r = 0`.

Step 2: Insert Operation
1. Check if the queue is full (r == m).
   If true, print "Overflow (Queue is full)" and exit.
2. If the queue is not full:
   a. Take input element 'n' to insert.
   b. If it's the first insertion (f == 0), set f = 1.
   c. Increment r by 1.
   d. Insert 'n' into q[r].

Step 3: Delete Operation
1. Check if the queue is empty (f == 0).
   If true, print "Underflow (Queue is empty)" and exit.
2. If the queue is not empty:
   a. Store the element at position 'f' (front) into 'n'.
   b. If 'f == r' (only one element in the queue), reset f = 0 and r = 0.
   c. Otherwise, increment f by 1.
   d. Print "Deleted element is n".

Step 4: Display Operation
1. Check if the queue is empty (f == 0).
   If true, print "Underflow (Queue is empty)".
2. If not empty:
   a. Print all elements from position 'f' to 'r' using a loop.

Step 5: Menu-Driven Execution
1. Print menu:
   1. Insert
   2. Display
   3. Delete
   4. Exit
2. Repeat until user selects "Exit":
   a. Take input 'ch' (user choice).
   b. Perform actions using switch-case:
      - Case 1: Call insert function.
      - Case 2: Call display function.
      - Case 3: Call delete function.

- Case 4: Exit program.
- Default: Print "Invalid choice!".

Title:To implement a program in cpp for Queue using array.

Code:

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

int m;  // Size of the queue

class queue {
   int f, r, q[10], n;  // f = front, r = rear, q = array for queue, n = element to insert

public:
   // Constructor to initialize front and rear to 0
   queue() {
      f = r = 0;
   }

   void insert();  // Insert function
   void del();   // Delete function
   void dis();    // Display function
};

// Function to insert an element in the queue
void queue::insert() {
   if (r == m) {
      cout << "\nOverflow (Queue is full)";
   } else {
      cout << "\nEnter Element in Queue = ";
      cin >> n;  // Read the element to be inserted
      if (f == 0) {
         f = 1;  // Set front to 1 if it's the first insertion
      }
```

```
    r++;  // Move the rear to the next position

 q[r] = n;  // Insert the element at the rear

}
}

// Function to delete an element from the queue
void queue::del() {
    if (f == 0) {
        cout << "\nUnderflow (Queue is empty)";
    } else {
        int n = q[f];  // Get the element at the front
        if (f == r) {
            // If front equals rear, the queue becomes empty after deletion
            f = r = 0;
        } else {
            f++;  // Move the front to the next position
        }
        cout << "\nDeleted element is " << n;
    }
}

// Function to display the elements in the queue
void queue::dis() {
    if (f == 0) {
        cout << "\nUnderflow (Queue is empty)";
    } else {
        cout << "\nElements in queue are: ";
        for (int i = f; i <= r; i++) {
            cout << q[i] << "\t";  // Display each element from front to rear
        }
    }
}

// Main function
int main() {
    queue q;  // Create an object of the queue class
    int ch;   // Variable to store user's choice
```

```
cout << "Enter size of queue: ";
    cin >> m;  // Read the maximum size of the queue

    cout << "\n1. Insert\n2. Display\n3. Delete\n4. Exit\n";


    while (ch != 4) {
        cout << "\nEnter choice: ";
        cin >> ch;  // Read user's choice

        switch (ch) {
            case 1:
                q.insert();  // Call insert function
                break;
            case 2:
                q.dis();     // Call display function
                break;
            case 3:
                q.del();     // Call delete function
                break;
            case 4:
                exit(0);     // Exit the program
            default:
                cout << "\nInvalid choice!";
        }
    }

    return 0;
}
```

Output:

```
Enter size of queue: 5

1. Insert
2. Display
3. Delete
4. Exit
```

Enter choice: 1

Enter Element in Queue = 1

Enter choice: 1

Enter Element in Queue = 2

Enter choice: 1

Enter Element in Queue = 3

Enter choice: 1

Enter Element in Queue = 4

Enter choice: 1

Enter Element in Queue = 5

Enter choice: 2

Elements in queue are: 1	2	3	4	5
Enter choice: 3

Deleted element is 1
Enter choice:

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title: To implement a program in CPP for Deques.

Algorithm for Deque Operations

1. **Initialize**:
   - Create a deque using std::deque from STL.
2. **Insert Elements**:
   - Use push_back() to insert at the end.
   - Use push_front() to insert at the front.
3. **Delete Elements**:
   - Use pop_back() to remove from the end.
   - Use pop_front() to remove from the front.
4. **Access Elements**:
   - Use front() to access the first element.
   - Use back() to access the last element.
   - Use indexing (deque[index]) to access elements at specific positions.
5. **Iterate and Display**:
   - Use an iterator or range-based loop to display elements.

Code:

```
#include <iostream>
#define SIZE 5

using namespace std;

int deque[SIZE];
int f = -1, r = -1;

// insert_front function will insert the value from the front
void insert_front(int x) {
   if ((f == 0 && r == SIZE - 1) || (f == r + 1)) {
      cout << "Overflow" << endl;
   } else if (f == -1 && r == -1) {
      f = r = 0;
      deque[f] = x;
   } else if (f == 0) {
```

```
        f = SIZE - 1;
        deque[f] = x;
    } else {
        f = f - 1;
        deque[f] = x;
    }
}

// insert_rear function will insert the value from the rear
void insert_rear(int x) {
    if ((f == 0 && r == SIZE - 1) || (f == r + 1)) {
        cout << "Overflow" << endl;
    } else if (f == -1 && r == -1) {
        r = 0;
        deque[r] = x;
    } else if (r == SIZE - 1) {
        r = 0;
        deque[r] = x;
  } else {
        r++;
        deque[r] = x;
    }
}

// display function prints all the values of deque
void display() {
    if (f == -1 && r == -1) {
        cout << "Deque is empty" << endl;
        return;
    }

    int i = f;
    cout << "Elements in deque are: ";
    while (i != r) {
        cout << deque[i] << " ";
        i = (i + 1) % SIZE;
    }
    cout << deque[r] << endl;
}

// getfront function retrieves the first value of the deque
void getfront() {
    if (f == -1 && r == -1) {
        cout << "Deque is empty" << endl;
    } else {
```

```
        cout << "The value at the front is: " << deque[f] << endl;
    }
}

// getrear function retrieves the last value of the deque
void getrear() {
    if (f == -1 && r == -1) {
        cout << "Deque is empty" << endl;
    } else {
        cout << "The value at the rear is: " << deque[r] << endl;
    }
}

// delete_front() function deletes the element from the front
void delete_front() {
    if (f == -1 && r == -1) {
        cout << "Deque is empty" << endl;
} else if (f == r) {
        cout << "The deleted element is: " << deque[f] << endl;
        f = r = -1;
    } else if (f == SIZE - 1) {
        cout << "The deleted element is: " << deque[f] << endl;
        f = 0;
    } else {
        cout << "The deleted element is: " << deque[f] << endl;
        f = f + 1;
    }
}

// delete_rear() function deletes the element from the rear
void delete_rear() {
    if (f == -1 && r == -1) {
        cout << "Deque is empty" << endl;
    } else if (f == r) {
        cout << "The deleted element is: " << deque[r] << endl;
        f = r = -1;
    } else if (r == 0) {
        cout << "The deleted element is: " << deque[r] << endl;
        r = SIZE - 1;
    } else {
        cout << "The deleted element is: " << deque[r] << endl;
        r = r - 1;
    }
}
```

```
int main() {
    insert_front(20);
    insert_front(10);
    insert_rear(30);
    insert_rear(50);
    insert_rear(80);

    display(); // Display the values of deque
    getfront(); // Retrieve the value at front-end
    getrear();  // Retrieve the value at rear-end
    delete_front();
    delete_rear();
    display(); // Display the values after deletion

    return 0;
}
```

Output:

```
Elements in deque are: 10 20 30 50 80
The value at the front is: 10
The value at the rear is: 80
The deleted element is: 10
The deleted element is: 80
Elements in deque are: 20 30 50
```

Godavari Institute of Management & Research, Jalgaon **Masters Computer Application (MCA)** Name: _ Roll No: _ _ _ _ _ Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I Subject: Sign. of Teacher

Title : To implement a program in CPP for Linear(Single) Linked List

Algorithm:

1. **Create a Node Structure**
   Define a struct to represent a node with two members:
   - data to store the data value.
   - next to point to the next node.
2. **Initialize the Linked List**
   - Start with a head pointer set to nullptr.
3. **Insertion Operations**
   - Append a new node at the end of the list.
   - Handle edge cases like an empty list.
4. **Traversal Operation**
   - Start at the head and iterate through each node, printing its value until the end.
5. **Deletion Operation (Optional)**
   - Remove a specific node by value or position.
6. **Display the Linked List**
   - Print all elements sequentially to verify correctness.

Code:

```
#include<iostream>

#include<conio.h>

#include<process.h>

using namespace std;

class node {

int info, item, s; node *link;

public:


void insert();

void dis();

void del();
```

```
void search();
void sum();
};

node *move, *start = NULL, *temp;
void node::insert() {
cout << "\nEnter the item:";
cin >> item;
node *node1 = new node;
node1->link = NULL;
node1->info = item;
if (start == NULL) start = node1;
else {

move = start;

while (move->link != NULL) move = move->link;
move->link = node1;

}

}

void node::dis() { node *x;
x = start;

cout << "\n Elements in LL are:";
while (x != NULL) {
cout << "\t" << x->info; x = x->link;
}
```

```
}

void node::sum() {
node *x;
x = start;
s = 0;
while (x != NULL) {
s = s + x->info;
x = x->link;
}

cout << "\nSum of node is" << s;

}

void node::del() {
temp = start;
if (temp != NULL) {
 temp = temp->link;
cout << "\nDeleted node is" << start->info; delete start;
start = temp;

} else

cout << "\n List is empty:";

}

void node::search() {
```

```
int c = 0, f = 0, d;
cout << "\nEnter item"; cin >> item;
temp = start;

while (temp != NULL) {
c++;
if (temp->info == item) {
f = 1;
d = c;
break;
}

temp = temp->link;

}

if (f == 1)

cout << "\nElement is found at position " << d;
else
cout << "\nElement is not found";

}

int main() { node n; int ch;
cout << "\n1.Insert 2.Display 3. Delete 4.Search 5.Sum 6.Exit\n";
do {
cout << "\nEnter choice"; cin >> ch;
switch (ch) {
```

```
case 1:

n.insert();
break;
case 2:

n.dis();
 break;
case 3:

n.del();
break;
case 4:

n.search();
break;
case 5:

n.sum();
break;
case 6:

return 0;

}

}
while (ch != 6);
getch();
return 0;

}
```

Output:

1.Insert 2.Display 3. Delete 4.Search 5.Sum 6.Exit

Enter choice1

Enter the item:1

Enter choice1

Enter the item:2

Enter choice1

Enter the item:3

Enter choice1

Enter the item:4

Enter choice1

Enter the item:5

Enter choice2

Elements in LL are: 1 2 3 4 5
Enter choice3

Deleted node is1
Enter choice4

Enter item2

Element is found at position 1
Enter choice5

Sum of node is14
Enter choice

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ __ _ _ _ _ _ _   Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _   Batch: _ _ _ _ _ _   Class: MCA. -I

Subject: Sign. of Teacher

Title : To implement a program in CPP for Circular Linked List.

Algorithm to Implement Circular Linked List

1. **Start**
2. **Define a Node structure**:
   - Data field to store value.
   - Pointer to the next node.
3. **Initialize a head pointer to NULL**.
4. **Operations**:
   - **Create Node**:
     - Allocate memory for a new node.
     - Assign the data value and set the next pointer to head for circular linking.
   - **Insert at Beginning**:
     - Check if the list is empty.
     - If empty, make the new node point to itself and assign it as the head.
     - Otherwise, traverse to the last node and update pointers for circular linking.
   - **Insert at End**:
     - Traverse to the last node.
     - Update the new node's next to point to head and the last node's next to the new node.
   - **Delete Node**:
     - Handle special cases like empty list or single node.
     - Traverse and adjust pointers to skip the node to be deleted.
   - **Display**:
     - Traverse the list from head to tail, stopping when the head is reached again.
5. **End**

Code:

```
#include<iostream>
#include<conio.h>
#include<cstdlib>
using namespace
std;
 class node {
 int info,c,i;
 node *link;
 public:
```

```
node() {
c=0;
}

void insert();
void display();
void del();
};

node
*start=NULL,
*temp=NULL,
*move=NULL,
*temp1=NULL;
void node::insert()
{
int item;
node *p=new
node;
cout<<"\nEnter
Element:";
cin>>item;


p->info=item;
p->link=NULL;
if(start==NULL) {
start=p;
p->link=start;
c++;
}

else {

temp=start;

while(temp-
>link!=start)
temp=temp->link;
temp->link=p;
```

```
p->link=start;
c++;
}

}

void
node::display() {
if(start==NULL) {
cout<<"\n LL
empty";
return;
}

node *temp;
temp=start;
move=start->link;
cout<<temp->info;
while(move!=start)
{
cout<<"-
>"<<move->info;
move=move->link;
}

cout<<"\n Number
of


nodes in CLL are
:"<<c;

}

void node::del() {
int pos;
cout<<"\nEnter
Position:";
cin>>pos;
if(c==1) {
```

```
start=NULL;

}

if(start==NULL) {
cout<<"\n LL
Empty:";
return;
}

if(pos>c || pos<1) {
cout<<"\nInvalid
Position";
return;
}

if(pos==1) {
temp=start;
while(temp-
>link!=start)
temp=temp->link;
temp1=start;
start=start->link;
temp->link=start;
cout<<"\nDeleted
Element is
"<<temp1->info;
delete temp1;


c--;

}

else {

temp=start; i=1;
while(i<pos-1) {
temp=temp->link;
```

```
i++;
}

temp1=temp->link;

temp->link=temp1-
>link;

cout<<"\nDeleted
element
is"<<temp1->info;
delete temp1;
c--;
}

}



int main() {
node n;
int ch;
cout<<"\n 1.Insert
2.Display 3.Delete
4.Exit";
while(ch!=4) {
cout<<"\n Enter
Choice";
cin>>ch;
switch(ch) {


case 1: n.insert();
break;
case 2: n.display();
break;
case 3: n.del();
break;
case 4:
exit(0);
```

```
  }

  }

  getch();

  return 0;
  }
```

Output:

```
1. Insert
2. Display
3. Delete
4. Exit

Enter Choice: 1
Enter  Element:
10

Enter Choice: 1
Enter  Element:
20

Enter Choice: 1
Enter  Element:
30

Enter Choice: 2
10->20->30
Number of
nodes in CLL
are: 3

Enter Choice: 3
Enter Position:
2
Deleted
Element is 20

Enter Choice: 2
```

10->30
Number of
nodes in CLL
are: 2
Enter Choice: 4
Exit

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: __ _ _ _ _ _ _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title : To implement a program in CPP for Doubly Linked List.

Algorithm for Implementing a Doubly Linked List in C++

1. **Define the Node Structure**:
   - Create a structure or class for the node.
   - Include data, a pointer to the previous node (prev), and a pointer to the next node (next).
2. **Initialize the Head and Tail**:
   - Create two pointers, head and tail, initialized to nullptr.
3. **Insertion Operations**:
   - **Insert at the Beginning**:
     1. Create a new node.
     2. Set the new node's next to point to the current head.
     3. If the list is not empty, update the current head's prev to the new node.
     4. Update head to point to the new node.
   - **Insert at the End**:
     1. Create a new node.
     2. Set the new node's prev to point to the current tail.
     3. If the list is not empty, update the current tail's next to the new node.
     4. Update tail to point to the new node.
   - **Insert at a Specific Position**:
     1. Traverse the list to the desired position.
     2. Update the prev and next pointers of adjacent nodes to insert the new node.
4. **Deletion Operations**:
   - **Delete from the Beginning**:
     1. Check if the list is empty.
     2. Update head to the next node and set the new head's prev to nullptr.
   - **Delete from the End**:
     1. Check if the list is empty.
     2. Update tail to the previous node and set the new tail's next to nullptr.
   - **Delete a Specific Node**:
     1. Traverse the list to locate the node.
     2. Update the prev and next pointers of adjacent nodes to exclude the node.
5. **Traversal**:
   - **Forward Traversal**:
     1. Start from head.
     2. Traverse using the next pointer until reaching nullptr.
   - **Backward Traversal**:

    1. Start from tail.
    2. Traverse using the prev pointer until reaching nullptr.
6. **Search**:
    - Traverse the list to find a node with the desired value.
7. **Display**:
    - Print the list elements during forward or backward traversal.
8. **Exit**:
    - Ensure memory is deallocated by deleting all nodes before exiting the program.

Code:

```
#include<iostream>

#include<conio.h>

#include<cstdlib>

using namespace std;

class node {

int info,c,j;


node *left,*right;

public:

void insert();

void display();

void del();

};


node
*start=NULL,*temp=
NULL,*move=NULL,
*temp1=NULL;

void node::insert() {

int item;
```

```
node *p=new node;

cout<<"\nEnter
element:";

cin>>item;

p->info=item;

p->left=NULL;

p->right=NULL;

if(start==NULL) {

start=p;

return;

}


else {


temp=start;


while(temp-
>right!=NULL)


temp=temp->right;

temp->right=p;

p->left=start;


}
}
```

```
void node::display() {

move=start;

if(move==NULL) {

cout<<"\n LL Empty:";

return;

}


else {


cout<<"\n node in DLL
are :";

while(move!=NULL) {

cout<<move-
>info<<"\t";

move=move->right;

}


}


}


void node::del() {
```

```
if(start==NULL) {

cout<<"\n LL Empty:";

return;

}


temp=start; start=temp-
>right;

if(start != NULL)

start->left=NULL;

temp->right=NULL;


cout<<"\n deleted
element is"<<temp-
>info;

delete temp;

}


int main() {

node n;

int ch;

cout<<"\n1. Insert 2.
Display 3. Delete 4.
Exit";

while(ch!=4) {

cout<<"\nEnter
choice";
```

```
cin>>ch;

switch(ch) {


case 1: n.insert();

break;

case 2: n.display();

break;

case 3: n.del();

break;

case 4:

exit(0);

}


}


getch();

return 0;

}
```

Output :

```
1. Insert 2. Display 3. Delete 4. Exit
Enter choice: 1
Enter element: 10

Enter choice: 1
Enter element: 20

Enter choice: 1
Enter element: 30
```

Enter choice: 2
Nodes in DLL are: 10 20 30

Enter choice: 3
Deleted element is: 10

Enter choice: 2
Nodes in DLL are: 20 30

Enter choice: 4

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: __ _ _ _ _ _ _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title: To implement the program in CPP for Polynomial Addition.

Algorithm for Polynomial Addition in C++

1. **Start**
2. **Initialize Polynomials**
   - Represent the polynomials as arrays or linked lists, where each node contains:
     - Coefficient
     - Exponent
3. **Input Polynomials**
   - Input the terms of the first polynomial.
   - Input the terms of the second polynomial.
4. **Set Pointers**
   - Set two pointers ptr1 and ptr2 to traverse the two polynomials.
5. **Traverse and Compare Terms**
   - While both pointers are not null:
     - Compare the exponents of the terms pointed to by ptr1 and ptr2:
       - **Case 1:** If exponents are equal:
         - Add the coefficients.
         - Insert the result into the result polynomial.
         - Move both pointers to the next term.
       - **Case 2:** If exponent of ptr1 is greater:
         - Copy the term pointed to by ptr1 to the result.
         - Move ptr1 to the next term.
       - **Case 3:** If exponent of ptr2 is greater:
         - Copy the term pointed to by ptr2 to the result.
         - Move ptr2 to the next term.
6. **Add Remaining Terms**
   - If ptr1 is not null, copy all remaining terms of the first polynomial to the result.
   - If ptr2 is not null, copy all remaining terms of the second polynomial to the result.
7. **Display Result Polynomial**
   - Traverse the result polynomial and display each term.
8. **End**

Code:

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

    int main() {
    int a[10], b[10], c[10];
    int m, n, k = 0, k1, i, j;

    cout << "\n\tPolynomial Addition\n";
    cout << "\t===================\n";

    // Input for the first polynomial
    cout << "\n\tEnter the number of terms of the polynomial: ";
    cin >> m;
    cout << "\n\tEnter the degrees and coefficients: ";
    for (i = 0; i < 2 * m; i++) {
        cin >> a[i];
    }

    // Display the first polynomial
    cout << "\n\tFirst polynomial is: ";
    k1 = 0;
    if (a[k1 + 1] == 1) {
        cout << "x^" << a[k1];
    } else {
        cout << a[k1 + 1] << "x^" << a[k1];
    }
   k1 += 2;
   while (k1 < 2 * m) {
      cout << "+" << a[k1 + 1] << "x^" << a[k1];
      k1 += 2;
}
// Input for the second polynomial
cout << "\n\n\tEnter the number of terms of the second polynomial: ";
cin >> n;
cout << "\n\tEnter the degrees and coefficients: ";
 for (j = 0; j < 2 * n; j++) {
 cin >> b[j];
 }

  // Display the second polynomial
cout << "\n\tSecond polynomial is: ";
k1 = 0;
```

```
if (b[k1 + 1] == 1) {
   cout << "x^" << b[k1];
} else {
  cout << b[k1 + 1] << "x^" << b[k1];
           }
 k1 += 2;
while (k1 < 2 * n) {
 cout << "+" << b[k1 + 1] << "x^" << b[k1];
 k1 += 2;
 }

  // Add the two polynomials
 i = 0;
  j = 0;
while (m > 0 && n > 0) {
 if (a[i] == b[j]) {
 c[k + 1] = a[i + 1] + b[j + 1];
 c[k] = a[i];
  m--;
 n--;
 i += 2;
j += 2;
} else if (a[i] > b[j]) {
  c[k + 1] = a[i + 1];
c[k] = a[i];
           m--;
 i += 2;
} else {
 c[k + 1] = b[j + 1];
c[k] = b[j];
  n--;
 j += 2;
 }
 k += 2;
}

  // Add remaining terms of the first polynomial
while (m > 0) {
  c[k + 1] = a[i + 1];
 c[k] = a[i];
k += 2;
 i += 2;
 m--;
}
```

```
 // Add remaining terms of the second polynomial
 while (n > 0) {
 c[k + 1] = b[j + 1];
 c[k] = b[j];
  k += 2;
 j += 2;
  n--;
  }

 // Display the resulting polynomial
cout << "\n\n\tSum of the two polynomials is: ";
k1 = 0;
 if (c[k1 + 1] == 1) {
 cout << "x^" << c[k1];
 } else {
     cout << c[k1 + 1] << "x^" << c[k1];
}
  k1 += 2;
  while (k1 < k) {
if (c[k1 + 1] == 1) {
   cout << "+x^" << c[k1];
  } else {
   cout << "+" << c[k1 + 1] << "x^" << c[k1];
}
  k1 += 2;
 }

cout << endl;
return 0;
}
```

Output:

Enter the number of terms of the polynomial: 2
Enter the degrees and coefficients: 2 3 0 4

First polynomial is: 3x^2+4x^0

Enter the number of terms of the second polynomial: 2
Enter the degrees and coefficients: 3 1 2 2

Second polynomial is: 1x^3+2x^2

Sum of the two polynomials is: 1x^3+5x^2+4x^0

Godavari Institute of Management & Research, Jalgaon **Masters Computer Application (MCA)** Name: __ _ _ _ _ _ _ Roll No: _ _ _ _ _ Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I Subject: Sign. of Teacher

Title: To implement the program in CPP for Polynomial Addition sing Linked List.

Algorithm for Polynomial Addition using Linked List in C++

1. **Define the Node Structure**
   - Create a structure for the nodes of the linked list.
   - Each node should store the coefficient, exponent, and a pointer to the next node.
2. **Create Functions to Handle Operations**
   - Write functions to:
     - **Insert terms** into a polynomial linked list in descending order of exponents.
     - **Display the polynomial** for debugging or visualization.
     - **Add two polynomials** represented as linked lists.
3. **Input Polynomials**
   - Take input for two polynomials, inserting terms in descending order of exponents.
4. **Traverse and Add Polynomials**
   - Initialize pointers for both polynomial linked lists.
   - Compare the exponents of terms pointed to by the two pointers:
     - If the exponents are equal:
       - Add the coefficients, create a new term with the sum, and move both pointers forward.
     - If one exponent is larger:
       - Copy the term with the larger exponent to the result list and move the pointer of the corresponding list forward.
   - If terms remain in either polynomial after traversing, append them to the result.
5. **Output the Resultant Polynomial**
   - Traverse the resultant linked list and display the terms.
6. **End**
   - Ensure proper memory management and exit the program

Code:

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

// Node structure containing power and coefficient of variable
```

```
struct Node {
    int coeff;
    int pow;
    Node* next;
};

// Function to create a new node
void create_node(int coeff, int pow, Node** temp) {
    Node *r, *z;
    z = *temp;
    if (z == nullptr) {
        r = new Node();
        r->coeff = coeff;
        r->pow = pow;
        *temp = r;
        r->next = new Node();
        r = r->next;
        r->next = nullptr;
    } else {
        r->coeff = coeff;
        r->pow = pow;
        r->next = new Node();
        r = r->next;
        r->next = nullptr;
    }
}
// Function to add two polynomial numbers
void polyadd(Node* poly1, Node* poly2, Node* poly) {
    while (poly1->next != nullptr && poly2->next != nullptr) {
        // If power of 1st polynomial is greater than 2nd
        if (poly1->pow > poly2->pow) {
            poly->pow = poly1->pow;
            poly->coeff = poly1->coeff;
            poly1 = poly1->next;
        }
 // If power of 2nd polynomial is greater than 1st
          else if (poly1->pow < poly2->pow) {
poly->pow = poly2->pow;
poly->coeff = poly2->coeff;
poly2 = poly2->next;
          }
// If power of both polynomials is the same
else {
```

```
poly->pow = poly1->pow;
poly->coeff = poly1->coeff + poly2->coeff;
poly1 = poly1->next;
poly2 = poly2->next;
}
// Dynamically create a new node
poly->next = new Node();
poly = poly->next;
poly->next = nullptr;
 }

 // Add remaining terms of the first polynomial
while (poly1->next != nullptr) {
  poly->pow = poly1->pow;
poly->coeff = poly1->coeff;
poly1 = poly1->next;
poly->next = new Node();
 poly = poly->next;
poly->next = nullptr;
 }

           // Add remaining terms of the second polynomial
while (poly2->next != nullptr) {
poly->pow = poly2->pow;
poly->coeff = poly2->coeff;
```

```
poly2 = poly2->next;

            poly->next = new Node();
                poly = poly->next;
                poly->next = nullptr;
              }
            }

            // Function to display the polynomial
            void show(Node* node) {
              while (node->next != nullptr) {
                cout << node->coeff << "x^" << node->pow;
                node = node->next;
                if (node->coeff >= 0 && node->next != nullptr) {
                  cout << "+";
                }
              }
            }

            int main() {
              Node *poly1 = nullptr, *poly2 = nullptr, *poly = nullptr;

              // Create first polynomial: 5x^2 + 4x^1 + 2x^0
              create_node(5, 2, &poly1);
              create_node(4, 1, &poly1);
              create_node(2, 0, &poly1);

              // Create second polynomial: -5x^1 - 5x^0
              create_node(-5, 1, &poly2);
              create_node(-5, 0, &poly2);

              cout << "1st Polynomial: ";
              show(poly1);

              cout << "\n2nd Polynomial: ";
              show(poly2);

              // Create result polynomial
              poly = new Node();

              // Add the two polynomials
              polyadd(poly1, poly2, poly);

              // Display the result
```

```
 cout << "\nAdded Polynomial: ";

 show(poly);

return 0;

}
```

Output:

1st Polynomial: 5x^2+4x^1+2x^0
2nd Polynomial: -5x^1-5x^0
Added Polynomial: 5x^2-3x^0

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title:To implement a program in cpp for Binary Search Tree.

Algorithm for Implementing a Binary Search Tree (BST) in C++

1. **Define the Node Structure**
   - Create a class or structure to represent a node in the BST.
   - Each node should have the following properties:
     - data to store the value.
     - left pointer for the left child.
     - right pointer for the right child.
2. **Create Insert Function**
   - If the tree is empty, create a new node and make it the root.
   - If the value to be inserted is smaller than the current node's value:
     - Recursively call the insert function on the left subtree.
   - If the value is greater than the current node's value:
     - Recursively call the insert function on the right subtree.
3. **Create Search Function**
   - Start at the root.
   - If the tree is empty or the root's value matches the search value, return the node or NULL.
   - If the search value is smaller than the current node's value:
     - Recursively search in the left subtree.
   - If the search value is greater than the current node's value:
     - Recursively search in the right subtree.
4. **Create Traversal Functions**
   - Implement the following traversal methods:
     - **Inorder Traversal**: Left -> Node -> Right
     - **Preorder Traversal**: Node -> Left -> Right
     - **Postorder Traversal**: Left -> Right -> Node
5. **Delete a Node**
   - If the node to be deleted is found:
     - If it has no children, delete it directly.
     - If it has one child, replace it with the child.
     - If it has two children, find the in-order successor or predecessor and replace the node with it, then delete the successor/predecessor.
6. **Create Main Function**
   - Instantiate the BST.
   - Allow the user to perform operations like insert, search, delete, and traverse.
   - Use appropriate menu-driven or input-driven logic.

Code:

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

// Definition of the BST node
struct Node {
    int data;
    Node *left_child;
    Node *right_child;

    Node(int value) {
        data = value;
        left_child = nullptr;
        right_child = nullptr;
    }
};

// Function to create a new node
Node* new_node(int x) {
    return new Node(x);
}

// Function to search for a value in the BST
Node* search(Node* root, int x) {
    if (root == nullptr || root->data == x)
        return root;

    if (x > root->data)
        return search(root->right_child, x);
    else
        return search(root->left_child, x);
}
```

```
// Function to insert a value into the BST
Node* insert(Node* root, int x) {
    if (root == nullptr)
        return new_node(x);

    if (x > root->data)
        root->right_child = insert(root->right_child, x);
    else
        root->left_child = insert(root->left_child, x);

    return root;
}

// Function to find the minimum value in the BST
Node* find_minimum(Node* root) {
    if (root == nullptr)
        return nullptr;

    if (root->left_child != nullptr)
        return find_minimum(root->left_child);

    return root;
}

// Function to delete a value from the BST
Node* delete_node(Node* root, int x) {
    if (root == nullptr)
        return nullptr;

    if (x > root->data)
        root->right_child = delete_node(root->right_child, x);
    else if (x < root->data)
        root->left_child = delete_node(root->left_child, x);
    else {
        // Node with no children
        if (root->left_child == nullptr && root->right_child == nullptr) {
            delete root;
            return nullptr;
        }
        // Node with one child
        else if (root->left_child == nullptr || root->right_child == nullptr) {
            Node* temp = (root->left_child == nullptr) ? root->right_child : root-
>left_child;
```

```
  delete root;
return temp;
}
           // Node with two children
  else {
 Node* temp = find_minimum(root->right_child);
root->data = temp->data;
 root->right_child = delete_node(root->right_child, temp->data);
}
}
    return root;
}

// Function for in-order traversal
void inorder(Node* root) {
    if (root != nullptr) {
        inorder(root->left_child);
        cout << root->data << " ";
        inorder(root->right_child);
    }
}

int main() {
    Node* root = new_node(20);
    insert(root, 5);
    insert(root, 1);
    insert(root, 15);
    insert(root, 9);
    insert(root, 7);
    insert(root, 12);
    insert(root, 30);
    insert(root, 25);
    insert(root, 40);
    insert(root, 45);
    insert(root, 42);

    cout << "In-order traversal before deletion: ";
    inorder(root);
    cout << endl;

    // Deleting nodes
    root = delete_node(root, 1);
    root = delete_node(root, 40);
```

```
 root = delete_node(root, 45);
    root = delete_node(root, 9);

    cout << "In-order traversal after deletion: ";
    inorder(root);
    cout << endl;

    return 0;
}
```

Output:

```
1  5  7  9  12  15  20  25  30  40  42  45
 5  7  12  15  20  25  30  42
```

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ __ _ _ _ _ _ _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title: To implement a program in cpp for In-order, Pre-order, Post-order Traversals.

Algorithm for In-order, Pre-order, and Post-order Tree Traversals:

1. In-order Traversal (Left, Root, Right):

- Input: Binary Tree
- Output: Elements in ascending order
- Algorithm:
  1. Start at the root node.
  2. Traverse the left subtree by calling In-order recursively.
  3. Visit the root node.
  4. Traverse the right subtree by calling In-order recursively.

2. Pre-order Traversal (Root, Left, Right):

- Input: Binary Tree
- Output: Root followed by left and right subtrees
- Algorithm:
  1. Start at the root node.
  2. Visit the root node.
  3. Traverse the left subtree by calling Pre-order recursively.
  4. Traverse the right subtree by calling Pre-order recursively.

3. Post-order Traversal (Left, Right, Root):

- Input: Binary Tree
- Output: Left and right subtrees followed by the root
- Algorithm:
  1. Start at the root node.
  2. Traverse the left subtree by calling Post-order recursively.
  3. Traverse the right subtree by calling Post-order recursively.
  4. Visit the root node.

Code:

```
include <iostream>
using namespace std;
struct ver {
int data;

ver *left, *right;
};
class tree {
public:
ver* create(int, ver*);
void in(ver*);
void post(ver*);
void pre(ver*);
};


ver* tree::create(int c, ver* node) {
if (node == NULL) {
node = new ver;
node->data = c;
node->left = NULL;
node->right = NULL;
return node;
}

else {
if (c < node->data)
node->left = create(c, node->left);
else
node->right = create(c, node->right);
return node;

}
```

```
}


void tree::in(ver* node) {
if (node) {
in(node->left);
cout << node->data << "\t";
in(node->right);
}
}


void tree::pre(ver* node) {
if (node) {
cout << node->data << "\t";
pre(node->left);
pre(node->right);
}

}


void tree::post(ver* node) {
if (node) {
post(node->left);
post(node->right);
cout << node->data << "\t";
}
}


int main() {
tree t;
ver* r = NULL;
int n, ch;


cout << "\n 1: Insert 2: Inorder 3: Preorder 4: Postorder 5: Exit :\n";
while (ch != 5) {
cout << "\nEnter Choice:";
cin >> ch;
switch (ch) {
```

```
case 1:
cout << "\nEnter Node:";
cin >> n;

r = t.create(n, r);
break;
case 2:
cout << "\nInorder Traversal:";
t.in(r);
break;
case 3:
cout << "\nPreorder Traversal:";
t.pre(r);
break;
case 4:
cout << "\nPostorder Traversal:";
t.post(r);
break;
case 5:
return 0;

}

}

}
```

Output:

```
1: Insert  2: Inorder  3: Preorder  4: Postorder  5: Exit :

Enter Choice: 1
Enter Node: 50

Enter Choice: 1
Enter Node: 30

Enter Choice: 1
Enter Node: 70

Enter Choice: 1
Enter Node: 20
```

Enter Choice: 1
Enter Node: 40

Enter Choice: 1
Enter Node: 60

Enter Choice: 1
Enter Node: 80

Enter Choice: 2
Inorder Traversal:
20 30 40 50 60 70 80

Enter Choice: 3
Preorder Traversal:
50 30 20 40 70 60 80

Enter Choice: 4
Postorder Traversal:
20 40 30 60 80 70 50

Enter Choice: 5

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title: To implement a program in cpp for Max-Heap Tree.

Algorithm for Max-Heap Tree in C++

1. **Initialization**:
   - Define an array to store the elements of the max-heap.
   - Initialize the size of the heap as 0.
2. **Insertion**:
   - Add a new element at the end of the heap array.
   - Increment the size of the heap.
   - **Heapify-Up**:
     - Compare the newly added element with its parent.
     - If the new element is greater than its parent, swap them.
     - Repeat the process until the max-heap property is satisfied.
3. **Deletion (Extract-Max)**:
   - Swap the root (maximum element) with the last element in the heap.
   - Remove the last element from the heap.
   - Decrement the size of the heap.
   - **Heapify-Down**:
     - Compare the new root with its children.
     - If the root is smaller than either child, swap it with the larger child.
     - Repeat the process until the max-heap property is satisfied.
4. **Heapify-Up and Heapify-Down**:
   - For any node, if it violates the max-heap property (i.e., parent is smaller than a child), perform appropriate swaps.
   - Ensure the max-heap property is restored throughout the tree.
5. **Max-Heap Property**:
   - The parent node must always be greater than or equal to its children.
6. **Implementation**:
   - Use an array to represent the tree, with the index structure allowing easy access to parents and children.

Code:

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

class MaxHeap {
private:
    vector<int> heap;

    // Function to heapify up after insertion
    void heapifyUp(int index) {
        if (index == 0) return; // Base case
        int parent = (index - 1) / 2;
        if (heap[parent] < heap[index]) {
            swap(heap[parent], heap[index]);
            heapifyUp(parent); // Recursive call to ensure max-heap property
        }
    }

    // Function to heapify down after deletion
    void heapifyDown(int index) {
        int left = 2 * index + 1;
        int right = 2 * index + 2;
        int largest = index;

        if (left < heap.size() && heap[left] > heap[largest]) {
            largest = left;
        }
        if (right < heap.size() && heap[right] > heap[largest]) {
            largest = right;
        }
        if (largest != index) {
            swap(heap[index], heap[largest]);
            heapifyDown(largest); // Recursive call to maintain max-heap property
        }
    }
```

```
public:
    // Function to insert a new element into the heap
    void insert(int value) {
        heap.push_back(value);
        heapifyUp(heap.size() - 1); // Adjust position to maintain max-heap
    }

    // Function to remove and return the maximum element (root) from the heap
    int extractMax() {
        if (heap.empty()) {
            cout << "Heap is empty!" << endl;
            return -1;
        }
        int maxElement = heap[0];
        heap[0] = heap.back();
        heap.pop_back();
        heapifyDown(0); // Adjust position to maintain max-heap
        return maxElement;
    }

    // Function to display the elements of the heap
    void printHeap() {
        for (int i = 0; i < heap.size(); ++i) {
            cout << heap[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    MaxHeap maxHeap;

    // Insert elements into the max heap
    maxHeap.insert(10);
    maxHeap.insert(20);
    maxHeap.insert(15);
    maxHeap.insert(30);
    maxHeap.insert(40);

    cout << "Max Heap after insertions: ";
    maxHeap.printHeap();

    // Extract maximum elements
```

```
 cout << "Extracted max: " << maxHeap.extractMax() << endl;
    cout << "Heap after extraction: ";
    maxHeap.printHeap();

    return 0;
}
```

Output:

Max Heap after insertions: 40 30 15 10 20

Extracted max: 40

Heap after extraction: 30 20 15 10

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title: To implement a program in c for Min-Heap Tree.

Algorithm to Implement a Min-Heap in C:

1. **Structure Definition**:
   - Define a structure for the Min-Heap, typically including a dynamic array to store elements and a variable to track the current size.
2. **Insertion Operation**:
   - Add a new element to the heap.
   - Place the new element at the end of the heap array.
   - Perform the **heapify-up** operation to maintain the min-heap property:
     - Compare the newly added element with its parent.
     - If it is smaller, swap it with the parent.
     - Continue this process until the heap property is satisfied.
3. **Heapify Down (Extract-Min)**:
   - To extract the minimum element, swap the root (minimum element) with the last element.
   - Remove the last element from the heap.
   - Perform the **heapify-down** operation to maintain the min-heap property:
     - Compare the root with its children.
     - If the smallest child is smaller than the root, swap them.
     - Continue this process to ensure the heap property is maintained.
4. **Heapify Function**:
   - Write a function heapify to restore the heap property for any node.
   - It handles both **heapify-up** and **heapify-down**.
5. **Heap Structure**:
   - Maintain the array to hold the heap elements.
   - Keep track of the current size of the heap.
6. **Min-Heap Operations**:
   - Insertion: Add elements and perform heapify-up.
   - Extraction: Remove the minimum element, perform heapify-down.
   - Peeking: Return the root element (minimum) without removing it.

Code:

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

// Function to heapify a subtree rooted with node i which is an index in arr[]
void minHeapify(vector<int>& arr, int n, int i) {
    int smallest = i; // Initialize smallest as root
    int left = 2 * i + 1; // left child
    int right = 2 * i + 2; // right child

    // If left child is smaller than root
    if (left < n && arr[left] < arr[smallest])
        smallest = left;

    // If right child is smaller than smallest so far
    if (right < n && arr[right] < arr[smallest])
        smallest = right;

    // If smallest is not root
    if (smallest != i) {
        swap(arr[i], arr[smallest]);

        // Recursively heapify the affected sub-tree
        minHeapify(arr, n, smallest);
    }
}

// Function to build a min heap from an array
void buildMinHeap(vector<int>& arr, int n) {
    // Start from the last non-leaf node and heapify all nodes in reverse order
    for (int i = n / 2 - 1; i >= 0; i--) {
        minHeapify(arr, n, i);
    }
}

// Function to print an array
void printArray(const vector<int>& arr) {
    for (int i = 0; i < arr.size(); ++i)
        cout << arr[i] << " ";
    cout << endl;
```

```
}

int main() {
    // Test with a random array
    vector<int> randomArray = {4, 10, 3, 5, 15};
    int n1 = randomArray.size();

    cout << "Original Random Array: ";
    printArray(randomArray);

    // Build heap
    buildMinHeap(randomArray, n1);

    cout << "Min Heap from Random Array: ";
    printArray(randomArray);
    cout << endl;

    // Test with a sorted array
    vector<int> sortedArray = {8, 6, 5, 4, 2};
    int n2 = sortedArray.size();

    cout << "Original Sorted Array: ";
    printArray(sortedArray);

    // Build heap
    buildMinHeap(sortedArray, n2);

    cout << "Min Heap from Sorted Array: ";
    printArray(sortedArray);

    return 0;
}
```

Output:

```
Original Random Array: 4 10 3 5 15
Min Heap from Random Array: 3 5 4 10 15

Original Sorted Array: 8 6 5 4 2
Min Heap from Sorted Array: 2 4 5 8 6
```

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: __ _ _ _ _ _ _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title: To implement a program in cpp for Depth First Traversal.

Algorithm for Depth First Traversal (DFS) in C++:

1. **Input Representation**:
   - Represent the graph using adjacency list or matrix.
2. **DFS Traversal Function**:
   - Create a recursive function DFS(node) that takes a starting node as input.
   - Maintain a visited array to keep track of visited nodes.
3. **Initialize**:
   - Create a stack to manage the traversal.
   - Push the starting node onto the stack and mark it as visited.
4. **DFS Loop**:
   - While the stack is not empty:
     a. Pop a node from the stack.
     b. Process (or output) the node.
     c. For each adjacent unvisited node, push it onto the stack and mark it as visited.
5. **Repeat**:
   - Continue the process until all nodes have been visited.
6. **Output**:
   - The nodes are visited in the order they are processed, reflecting the DFS traversal.

Code:

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

int main() {
   cout << "===== Program to demonstrate the DFS Traversal on a Graph, in CPP
=====\n\n";
```

```
    // Variable declarations
    int cost[10][10] = {0}; // Adjacency matrix, initialized to 0
    int i, j, k, n, e, top = -1, v;
    int stk[10], visit[10] = {0}, visited[10] = {0};

    cout << "Enter the number of vertices in the Graph: ";
    cin >> n;

    cout << "\nEnter the number of edges in the Graph: ";
    cin >> e;

    cout << "\nEnter the start and end vertex of the edges:\n";
    for (k = 1; k <= e; k++) {
        cin >> i >> j;
        cost[i][j] = 1;
        cost[j][i] = 1; // For undirected graph
    }

    cout << "\nEnter the initial vertex to start the DFS traversal with: ";
    cin >> v;

    cout << "\nThe DFS traversal on the given graph is:\n";
    cout << v << " ";
    visited[v] = 1; // Mark the starting vertex as visited
k = 1;

 while (k < n) {
        for (j = n; j >= 1; j--) {
            if (cost[v][j] != 0 && visited[j] != 1 && visit[j] != 1) {
                visit[j] = 1;
                stk[++top] = j; // Push vertex onto the stack
            }
        }

        if (top == -1) {
            break; // If stack is empty, traversal is complete
        }

        v = stk[top--]; // Pop vertex from the stack
        cout << v << " ";
        visit[v] = 0;  // Mark it as removed from the visit stack
        visited[v] = 1; // Mark it as visited
        k++;
```

```
    }

cout << "\n";
    return 0;
}
```

Output:

```
===== Program to demonstrate the DFS Traversal on a Graph, in CPP =====

Enter the number of vertices in the Graph: 5

Enter the number of edges in the Graph: 4

Enter the start and end vertex of the edges:
1 2
1 3
2 4
3 5

Enter the initial vertex to start the DFS traversal with: 1

The DFS traversal on the given graph is:
1 3 5 2 4
```

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: __ _ _ _ _ _ _   Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _   Batch: _ _ _ _ _ _   Class: MCA. -I

Subject: Sign. of Teacher

Title: To implement a program in cpp for Breadth First Traversal.

Algorithm for Breadth First Traversal (BFS) in C++:

1. **Input Graph Representation**:
   - Represent the graph using an adjacency list or adjacency matrix.
2. **Initialization**:
   - Create a queue to store vertices.
   - Create an array to keep track of visited nodes.
   - Enqueue the starting vertex.
   - Mark the starting vertex as visited.
3. **BFS Loop**:
   - While the queue is not empty:
     - Dequeue the front vertex.
     - Print or process the vertex.
     - For each adjacent vertex of the dequeued vertex:
       - If the vertex has not been visited, mark it as visited.
       - Enqueue the adjacent vertex.
4. **Output**:
   - The vertices are processed in the order they are dequeued, ensuring BFS traversal.

Code:

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

int n, i, j, visited[10], queue[10], front = -1, rear = -1;
int adj[10][10];

void bfs(int v) {
   // Mark the starting node as visited
   visited[v] = 1;
   queue[++rear] = v; // Enqueue the starting vertex

   while (front < rear) {
      // Dequeue a vertex and explore its adjacent nodes
      int current = queue[++front];

      // Explore all adjacent vertices of the current vertex
```

```
        for (i = 1; i <= n; i++) {
            if (adj[current][i] && !visited[i]) {
                queue[++rear] = i;  // Enqueue the unvisited adjacent vertex
                visited[i] = 1;      // Mark it as visited
            }
        }
    }
}

int main() {
    int v;
    cout << "Enter the number of vertices: ";
    cin >> n;

    // Initialize visited array and queue
    for (i = 1; i <= n; i++) {
        visited[i] = 0;
 queue[i] = 0;
    }

    cout << "Enter graph data in matrix form:\n";
    for (i = 1; i <= n; i++) {
        for (j = 1; j <= n; j++) {
            cin >> adj[i][j];
        }
    }

    cout << "Enter the starting vertex: ";
    cin >> v;

bfs(v);

    cout << "The nodes which are reachable are:\n";
    bool allVisited = true;
    for (i = 1; i <= n; i++) {
        if (visited[i]) {
            cout << i << "\t";
        } else {
            allVisited = false;
        }
    }

    if (!allVisited) {
        cout << "\nBFS is not possible. Not all nodes are reachable.\n";
    }
```

```
    return 0;
}
```

Output:

```
Enter the number of vertices: 5
Enter graph data in matrix form:
0 1 0 0 1
1 0 1 0 0
1 1 0 1 0
0 0 1 0 0
1 0 0 0 0
Enter the starting vertex: 1
The node which are reachable are:
1       2       3       4       5
```

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _   Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _   Batch: _ _ _ _ _ _   Class: MCA. -I

Subject: Sign. of Teacher

Title: To implement a program in cpp for obtaining shortest path using Dijkstra Algorithm.

Dijkstra's Algorithm:

1. **Initialization**:
   - Create a distance array (dist[]) initialized with infinity for all nodes except the starting node, which is set to 0.
   - Create a set or priority queue to keep track of the minimum distance.
   - Mark all nodes as unvisited.
2. **Main Loop**:
   - Extract the node with the smallest distance from the priority queue.
   - For each neighbor of the extracted node:
     - Calculate the tentative distance by adding the distance of the current node and the weight of the edge to the neighbor.
     - If this calculated distance is smaller than the currently known distance to the neighbor, update the neighbor's distance.
   - Mark the current node as visited.
3. **Termination**:
   - Repeat the process until all nodes have been visited or the priority queue is empty.

Code:

```
#include <iostream>
#include <vector>
#include <set>
#include <climits>

using namespace std;

// Define infinity as a large value for initialization
#define INF INT_MAX

// Function to implement Dijkstra's algorithm
void dijkstra(int V, vector<pair<int, int>> adj[], int src) {
    // Create a distance array and initialize all distances to infinity
```

```
    vector<int> dist(V, INF);
    dist[src] = 0;

    // Use a set to keep track of vertices that are being processed
    set<pair<int, int>> s;
    s.insert({0, src}); // Insert source with distance 0

    while (!s.empty()) {
        // Get the vertex with the minimum distance
        int u = s.begin()->second;
        s.erase(s.begin());

        // Explore the neighbors of vertex u
        for (auto neighbor : adj[u]) {
            int v = neighbor.first;
            int weight = neighbor.second;

          // If a shorter path to v is found, update the distance
          if (dist[u] + weight < dist[v]) {
            // Remove the old pair and insert the new one with updated distance
          if (dist[v] != INF) {
                    s.erase(s.find({dist[v], v}));
                }
                dist[v] = dist[u] + weight;
                s.insert({dist[v], v});
            }
        }
    }

    // Print the shortest distances from the source
    cout << "Vertex\tDistance from Source\n";
    for (int i = 0; i < V; i++) {
        if (dist[i] == INF)
            cout << i << "\tINF\n"; // If no path exists
        else
            cout << i << "\t" << dist[i] << endl;
    }
}

int main() {
    // Number of vertices
    int V = 9;
```

```
    // Adjacency list representation of the graph
    vector<pair<int, int>> adj[V];

    // Add edges (undirected graph)
    adj[0].push_back({1, 6});
    adj[0].push_back({7, 8});
    adj[1].push_back({0, 6});
    adj[1].push_back({2, 8});
    adj[1].push_back({7, 13});
    adj[2].push_back({1, 8});
    adj[2].push_back({3, 7});
    adj[2].push_back({5, 6});
    adj[2].push_back({8, 2});
    adj[3].push_back({2, 7});
    adj[3].push_back({4, 9});
    adj[3].push_back({5, 14});
    adj[4].push_back({3, 9});
    adj[4].push_back({5, 10});
    adj[5].push_back({2, 6});
    adj[5].push_back({3, 14});
    adj[5].push_back({4, 10});
    adj[5].push_back({6, 2});
        adj[6].push_back({5, 2});
    adj[6].push_back({7, 1});
    adj[6].push_back({8, 6});
    adj[7].push_back({0, 8});
    adj[7].push_back({1, 13});
    adj[7].push_back({6, 1});
    adj[7].push_back({8, 7});
    adj[8].push_back({2, 2});
    adj[8].push_back({6, 6});
    adj[8].push_back({7, 7});

    // Run Dijkstra's algorithm from vertex 0
    dijkstra(V, adj, 0);

    return 0;
}
```

Output:

```
Vertex  Distance from Source
 0       0
 1       6
```

2	14
3	13
4	21
5	20
6	22
7	8
8	12

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ __ _ _ _ _ _ _   Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _   Batch: _ _ _ _ _ _   Class: MCA. -I

Subject: Sign. of Teacher

Title: To implement a program in c for obtaining shortest Path using Floyd- Warshall Algorithm.

Floyd-Warshall Algorithm for Shortest Path:

1. Initialization:

- Create a 2D array dist[][] representing the graph's adjacency matrix, where dist[i][j] holds the weight of the edge from vertex i to vertex j.
- Set dist[i][i] = 0 for all vertices since the distance from a vertex to itself is zero.

2. Iterative Updates:

- For each intermediate vertex k from 1 to n (where n is the number of vertices):
  - For each pair of vertices i and j:
    - If a shorter path from i to j through k exists (dist[i][j] > dist[i][k] + dist[k][j]), update dist[i][j] = dist[i][k] + dist[k][j].

3. Output the Shortest Paths:

- After completing the algorithm, dist[i][j] contains the shortest distance from vertex i to vertex j.
- If a path does not exist between i and j, set dist[i][j] to a large value (infinity).

Complexity:

- Time Complexity: O(V^3) where V is the number of vertices.
- Space Complexity: O(V^2) for the distance matrix.

Code:

```
#include <iostream>
#include <iomanip> // for std::setw
#define V 4
#define INF 99999 // Define Infinite as a large enough value

using namespace std;
```

```
// A function to print the solution matrix
void printSolution(int dist[][V]);

// Solves the all-pairs shortest path problem using Floyd Warshall algorithm
void floydWarshall(int dist[][V]) {
    int i, j, k;

// Add all vertices one by one to the set of intermediate vertices
// Before start of an iteration, we have shortest distances between all pairs of
vertices
 // such that the shortest distances consider only the vertices in set {0, 1, 2, .. k-1} as
intermediate vertices
  // After the end of an iteration, vertex no. k is added to the set of intermediate
vertices
 for (k = 0; k < V; k++) {
 // Pick all vertices as source one by one
for (i = 0; i < V; i++) {
   // Pick all vertices as destination for the above picked source
  for (j = 0; j < V; j++) {
   // If vertex k is on the shortest path from i to j, then update the value of dist[i][j]
  if (dist[i][k] + dist[k][j] < dist[i][j])
    dist[i][j] = dist[i][k] + dist[k][j];
           }
          }
           }
// Print the shortest distance matrix
printSolution(dist);
}

/* A utility function to print solution */
void printSolution(int dist[][V]) {
   cout << "The following matrix shows the shortest distances between every pair of
vertices\n";
    for (int i = 0; i < V; i++) {
      for (int j = 0; j < V; j++) {
 if (dist[i][j] == INF)
  cout << setw(7) << "INF"; // Formatting for better output
          else
 cout << setw(7) << dist[i][j];
 }
 cout << endl;
}
}

int main() {
```

```
 // Let us create the following weighted graph
 //     10
 // (0)------ >(3)
//  |         /|\
//  5 |       |  |
//  |  |1      |  |
           //  \|/       |  |
/        / (1) ------>(2)
         //      3

    int graph[V][V] = { { 0, 5, INF, 10 },
      { INF, 0, 3, INF },
      { INF, INF, 0, 1 },
      { INF, INF, INF, 0 } };

   // Function call
    floydWarshall(graph);

   return 0;
  }
```

Output:

The following matrix shows the shortest distances between every pair of vertices

```
 0      5    INF     10
INF      0      3     INF
INF    INF      0      1
INF    INF    INF      0
```

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ __ _ _ _ _ _ _   Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _   Batch: _ _ _ _ _ _   Class: MCA. -I

Subject: Sign. of Teacher

Title: To implement a program in cpp for Minimum spanning tree using Kruskal Algorithm.

Kruskal's Algorithm for Minimum Spanning Tree (MST)

1. **Sort all the edges in ascending order by their weights.**
2. **Initialize a disjoint-set (or union-find) data structure.**
3. **Initialize an empty set to store the MST edges.**
4. **For each edge in the sorted edge list:**
   - If the edge doesn’t form a cycle (using union-find), add it to the MST.
   - Otherwise, discard the edge.
5. **Repeat step 4 until the MST contains (V - 1) edges, where V is the number of vertices.**
6. **Output the edges of the MST along with their weights.**

Code:

```
//Kruskal

#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

// Edge structure to represent an edge
struct Edge {
    int u, v, weight;
};

// Disjoint Set Union (DSU) or Union-Find structure
class DSU {
public:
    vector<int> parent, rank;
```

```
        DSU(int n) {
            parent.resize(n);
            rank.resize(n, 0);
            for (int i = 0; i < n; i++) {
                parent[i] = i;
            }
        }

        // Find the parent of a node with path compression
        int find(int u) {
            if (u != parent[u]) {
                parent[u] = find(parent[u]);  // Path compression
            }
            return parent[u];
        }
    // Union by rank
    void unionSet(int u, int v) {
    int rootU = find(u);
     int rootV = find(v);

     if (rootU != rootV) {
      // Union by rank (attach the smaller tree under the larger one)
    if (rank[rootU] > rank[rootV]) {
      parent[rootV] = rootU;
     } else if (rank[rootU] < rank[rootV]) {
     parent[rootU] = rootV;
     } else {
parent[rootV] = rootU;
rank[rootU]++;
    }
    }
    }
    };

    // Function to implement Kruskal's algorithm
    int kruskal(vector<Edge>& edges, int n) {
        // Sort edges by their weights in non-decreasing order
        sort(edges.begin(), edges.end(), [](Edge a, Edge b) {
            return a.weight < b.weight;
        });

        DSU dsu(n);  // Initialize DSU for n vertices
        int mstWeight = 0;  // Total weight of the MST
        vector<Edge> mstEdges;  // To store the edges of the MST
```

```
// Go through the sorted edges and add them to the MST if no cycle is formed
for (Edge& edge : edges) {
    int u = edge.u;
    int v = edge.v;
    int weight = edge.weight;

    // If u and v are in different sets, add this edge to the MST
    if (dsu.find(u) != dsu.find(v)) {
        dsu.unionSet(u, v);
        mstEdges.push_back(edge);
        mstWeight += weight;
    }
      }
   // Print the MST edges and total weight
      cout << "\nEdges in the Minimum Spanning Tree (MST):\n";
      for (Edge& edge : mstEdges) {
          cout << edge.u << " - " << edge.v << " : " << edge.weight << endl;
      }

      return mstWeight;
   }

   int main() {
      int n, e;

      // Input the number of vertices and edges
      cout << "Enter the number of vertices: ";
      cin >> n;
      cout << "Enter the number of edges: ";
      cin >> e;

      vector<Edge> edges(e);

      // Input the edges (u, v, weight)
      cout << "Enter the edges (u, v, weight):\n";
      for (int i = 0; i < e; i++) {
          cin >> edges[i].u >> edges[i].v >> edges[i].weight;
      }

      // Call Kruskal's algorithm to find the MST
      int mstWeight = kruskal(edges, n);

      // Print the total weight of the MST
```

```
    cout << "\nTotal weight of the Minimum Spanning Tree: " << mstWeight
<< endl;

    return 0;
}
```

Output:

```
Enter the number of vertices: 4
Enter the number of edges: 5
Enter the edges (u, v, weight):
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
Edges in the Minimum Spanning Tree (MST):
3 - 2 : 4
0 - 3 : 5
0 - 1 : 10

Total weight of the Minimum Spanning Tree: 19
```

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title:To implement a program in cpp for Minimum spanning tree using Prims Algorithm.

Prims Algorithm for Minimum Spanning Tree (MST)

1. **Initialization**:
   - Create a graph with V vertices and E edges.
   - Initialize a set MST to store the MST vertices.
   - Select a starting vertex src and include it in MST.
   - Set key[] values of all vertices as infinite, except for src, which is set to 0.
   - Create a priority queue to store vertices and their keys.
2. **Main Loop**:
   - Extract the vertex u with the smallest key from the priority queue.
   - Add u to MST.
   - Update the key values of the adjacent vertices of u that are not yet in MST.
   - For each adjacent vertex v of u, if the edge weight from u to v is smaller than the current key of v, update the key and parent of v.
3. **Termination**:
   - Repeat step 2 until all vertices are included in the MST.
   - The parent array contains the edges of the MST.
4. **Output**:
   - The MST is constructed with minimum weight edges connecting all vertices.

Code:

```
#include <iostream>
#include <climits> // for INT_MAX

using namespace std;

int main() {
int a, b, n, ne = 1, i, j, min, cost[10][10], mincost = 0;

// Taking the number of vertices as input
cout << "\nEnter the number of vertices: ";
cin >> n;

cout << "\nEnter the adjacency matrix (use 0 for no edge): \n";
```

```
 // Input the adjacency matrix
 for (i = 1; i <= n; i++) {
for (j = 1; j <= n; j++) {
 cin >> cost[i][j];
if (cost[i][j] == 0) {
  cost[i][j] = INT_MAX; // Replace 0 with a large number (infinity)
 }
 }
 }

          // Prim's Algorithm to find the MST
while (ne < n) {
min = INT_MAX;

// Search for the minimum edge
for (i = 1; i <= n; i++) {
          for (j = 1; j <= n; j++) {
          if (cost[i][j] < min) {
          min = cost[i][j];
 a = i;
 b = j;
}
}
}

// Print the edge that is part of MST
cout << "Edge (" << a << ", " << b << ") = " << min << endl;

 // Add the minimum weight edge to the total cost
 mincost += min;

// Mark the edge as processed by setting the cost to a large number (effectively
removing it)
cost[a][b] = cost[b][a] = INT_MAX;

// Increment the number of edges in the MST
  ne++;
  }

  // Output the total minimum cost of the MST
 cout << "\nMinimum Spanning Tree total weight = " << mincost << endl;

  return 0;
```

}

Output:

Enter the number of vertices: 4

Enter the adjacency matrix (use 0 for no edge):
0 10 999 30
10 0 50 999
999 50 0 20
30 999 20 0
Edge (1, 2) = 10
Edge (1, 4) = 30
Edge (3, 4) = 20

Minimum Spanning Tree total weight = 60

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title:To implement a program in cpp for Hash Table.

Algorithm for Hash Table Implementation in C++

1. **Define a Hash Table Structure:**
   - Create a struct for the hash table that includes:
     - An array to store key-value pairs.
     - Size of the hash table.
     - A function to compute the hash.
2. **Hash Function:**
   - Implement a hash function that calculates the index by taking the modulus of the key with the table size.
3. **Insertion:**
   - Compute the index using the hash function.
   - If the index is empty or already contains the key, insert the key-value pair.
   - Handle collisions using methods like:
     - **Chaining**: Store linked lists at each index to handle collisions.
     - **Open Addressing**: Probe for the next available slot (e.g., linear probing, quadratic probing).
4. **Search:**
   - Compute the index using the hash function.
   - Search for the key at the computed index.
   - If the key is found, return the corresponding value.
   - If not, handle the collision using the same strategy as insertion.
5. **Deletion:**
   - Compute the index using the hash function.
   - Remove the key-value pair if found.
   - Rehash the table if necessary.
6. **Resize:**
   - If the load factor exceeds a threshold, rehash the table by resizing and rehashing existing elements.
7. **Handle Edge Cases:**
   - Resize when table reaches a certain load factor.
   - Ensure the hash function distributes keys uniformly.

Code:

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

// HashTable class
class HashTable {
private:
    static const int hashGroups = 10; // Number of groups (buckets)
    list<int> table[hashGroups];      // Array of lists to handle collisions

    // Hash function to calculate hash value
    int hashFunction(int key) {
        return key % hashGroups;
    }

public:
    // Insert a key into the hash table
    void insert(int key) {
        int index = hashFunction(key);
        table[index].push_back(key);
    }

    // Delete a key from the hash table
    void remove(int key) {
        int index = hashFunction(key);
        table[index].remove(key);
    }

    // Search for a key in the hash table
    bool search(int key) {
        int index = hashFunction(key);
        for (auto it : table[index]) {
            if (it == key)
                return true;
        }
```

```
 return false;
 }

 // Display the hash table
 void display() {
     for (int i = 0; i < hashGroups; i++) {
cout << "Bucket " << i << ": ";
for (auto it : table[i]) {
cout << it << " -> ";
}
Cout << "null" << endl;
}
 }
};

int main() {
 HashTable ht;

// Insert elements
ht.insert(10);
ht.insert(20);
ht.insert(15);
ht.insert(7);
ht.insert(25);

 // Display hash table
cout << "Hash Table: " << endl;
ht.display();

    // Search for a key
    int key = 15;
    cout << "\nSearching for key " << key << ": ";
    if (ht.search(key)) {
        cout << "Found" << endl;
    } else {
        cout << "Not Found" << endl;
    }

    // Remove a key
    key = 20;
    cout << "\nRemoving key " << key << endl;
    ht.remove(key);

          // Display updated hash table
```

```
cout << "\nUpdated Hash Table: " << endl;

ht.display();

 return 0;
}
```

Output:

```
Hash Table:
Bucket 0: null
Bucket 1: null
Bucket 2: null
Bucket 3: null
Bucket 4: null
Bucket 5: null
Bucket 6: null
Bucket 7: 7 -> null
Bucket 8: null
Bucket 9: 10 -> 20 -> 15 -> 25 -> null

Searching for key 15: Found

Removing key 20

Updated Hash Table:
Bucket 0: null
Bucket 1: null
Bucket 2: null
Bucket 3: null
Bucket 4: null
Bucket 5: null
Bucket 6: null
Bucket 7: 7 -> null
Bucket 8: null
Bucket 9: 10 -> 15 -> 25 -> null
```

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title:To implement a program in cpp for Linear Search using array.

Algorithm for Linear Search in C++:

1. **Start**
2. Read the size of the array (n).
3. Read the array elements into memory.
4. Prompt the user to input the target value (x) to search.
5. Iterate over each element of the array from the beginning to the end:
   - Compare each element with the target value (x).
   - If a match is found, return the index of the element and terminate.
6. If the loop completes without finding the target, return -1 to indicate the value is not present in the array.
7. **End**.

Code:

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

// Function to perform linear search
int linearSearch(int arr[], int n, int target) {
    for (int i = 0; i < n; ++i) {
        if (arr[i] == target)
            return i; // Return the index if target is found
    }
    return -1; // Return -1 if target is not found
}

int main() {
    int arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
    int n =

sizeof(arr) / sizeof(arr[0]);
    int target = 23;
```

```
    int result = linearSearch(arr, n, target);
    if (result != -1)
        cout << "Element found at index " << result << endl;
    else
        cout << "Element not found in the array" << endl;

    return 0;
}
```

Output:

How many elements u want to enter:

5

Enter array elements:

10

20

30

40

50

Enter element to search: 30

Element found at index 2

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title:To implement a program in cpp for Binary Search using array.

Algorithm for Binary Search in C++:

1. **Initialize low and high pointers**:
   - Set low to the starting index of the array (0).
   - Set high to the last index of the array (n - 1), where n is the size of the array.
2. **Repeat until the element is found or the pointers cross**:
   - Calculate the middle index mid as (low + high) / 2.
3. **Compare the middle element with the target value**:
   - If the middle element is the target, return the index mid.
   - If the target is smaller than the middle element, set high = mid - 1.
   - If the target is greater than the middle element, set low = mid + 1.
4. **Target not found**:
   - If the loop ends without finding the target, return -1 to indicate that the element is not present in the array.

Code:

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

// Function to perform binary search
int binarySearch(int arr[], int left, int right, int target) {
    while (left <= right) {
        int mid = left + (right - left) / 2;

        // Check if target is present at mid
        if (arr[mid] == target)
            return mid;

        // If target is greater, ignore left half
        if (arr[mid] < target)
            left = mid + 1;
        else
            right = mid - 1;
```

```
    }

    return -1; // Target is not present
}

int main() {
    int arr[] = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
    int n = sizeof(arr) / sizeof(arr[0]);
    int target = 23;

    int result = binarySearch(arr, 0, n - 1, target);
    if (result != -1)
        cout << "Element found at index " << result << endl;
    else
        cout << "Element not found in the array" << endl;

    return 0;
}
```

Output:

Enter number of elements:
5
Enter 5 integers:
10
20
30
40
50
Enter value to find: 50
50 found at location 5

Godavari Institute of Management & Research, Jalgaon **Masters Computer Application (MCA)**		
Name: _		Roll No: _ _ _ _ _
Date of Performance: _ _ /_ _/20_ _	Batch: _ _ _ _ _ _	Class: MCA. -I
Subject:		Sign. of Teacher

Title:To implement a program in cpp for Bubble Sort.

Bubble Sort Algorithm:

1. **Initialize**: Start with an array of n elements.
2. **Outer Loop**: Iterate through the array from the beginning to the end (n-1 passes).
   - The last element is already in its correct position after each pass.
3. **Inner Loop**: For each pass, compare adjacent elements.
   - If the current element is greater than the next, swap them.
4. **Repeat**: Continue the process until no more swaps are needed.
5. **Output**: The array is sorted in ascending order.

Code:

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

int main() {
    int array[100], n, i, j, swap;

    cout << "Enter number of elements: ";
    cin >> n;

    cout << "\nEnter " << n << " Numbers:\n";
    for (i = 0; i < n; i++)
        cin >> array[i];

    // Bubble Sort
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (array[j] > array[j + 1]) {
                swap = array[j];
                array[j] = array[j + 1];
```

```
                array[j + 1] = swap;
            }
        }
    }

    cout << "\nSorted Array:\n";
    for (i = 0; i < n; i++)
        cout << array[i] << endl;

    return 0;
}
```

Output:

```
Enter number of elements: 5

Enter 5 Numbers:
42
25
33
17
8

Sorted Array:
8
17
25
33
42
```

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: __ _ _ _ _ _ _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title:To implement a program in cpp for Selection Sort.

Algorithm for Selection Sort in C++:

1. **Initialize**:
   Start with an unsorted array of n elements.
2. **Outer Loop**:
   Iterate over each element i from 0 to n-1.
3. **Find Minimum Element**:
   Set min_idx = i.
   Iterate through the unsorted portion of the array from i+1 to n-1.
   If any element is smaller than the current minimum, update min_idx.
4. **Swap**:
   If min_idx is not equal to i, swap the element at min_idx with the element at i.
5. **Repeat**:
   Continue this process for the remaining elements until the entire array is sorted.
6. **Output**:
   The array is now sorted.

Code:

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

/*
 * Function to swap two variables
 */
void swap(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}

/*
 * Selection sort function
 */
```

```
void selectionSort(int arr[], int size) {
    int i, j;
    for (i = 0; i < size; i++) {
        for (j = i; j < size; j++) {
            if (arr[i] > arr[j])
                swap(&arr[i], &arr[j]);
        }
    }
}

/*
 * Main Function
 */
int main() {
int array[10], size;

    cout << "How many numbers you want to sort: ";
    cin >> size;
 cout << "\nEnter " << size << " numbers:\n";

for (int i = 0; i < size; i++)
 cin >> array[i];

 selectionSort(array, size);

cout << "\nSorted array is:\n";
for (int i = 0; i < size; i++)
cout << array[i] << endl;

 return 0;
}
```

Output:

```
How many numbers you want to sort: 5


Enter 5 numbers:
42
17
33
8
25
Sorted array is:
8
```

17

25

33

42

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title: To implement a program in cpp for Insertion Sort.

Insertion Sort Algorithm:

1. **Start with the second element (index 1)** as the key.
2. **Compare the key with elements before it** (in the sorted portion of the array).
3. **Shift elements greater than the key one position to the right**.
4. **Insert the key at its correct position** in the sorted portion.
5. **Repeat steps 2-4** for each remaining element in the array until the entire array is sorted.

Code:

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

int main() {
    int n, i, j, temp;
    int arr[64];

    cout << "Enter number of elements: ";
    cin >> n;

    cout << "\nEnter " << n << " integers:\n";
    for (i = 0; i < n; i++) {
        cin >> arr[i];
    }

    // Insertion Sort Algorithm
    for (i = 1; i < n; i++) {
        j = i;
        while (j > 0 && arr[j - 1] > arr[j]) {
            temp = arr[j];
            arr[j] = arr[j - 1];
            arr[j - 1] = temp;
            j--;
```

```
        }
    }

    cout << "\nSorted list in ascending order:\n";
    for (i = 0; i < n; i++) {
        cout << arr[i] << endl;
    }

    return 0;
}
```

Output:

```
Enter number of elements: 5

Enter 5 integers:
42
17
33
8
25
Sorted list in ascending order:
8
17
25
33
42
```

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title: To implement a program in cpp for Radix Sort.

Radix Sort Algorithm:

1. **Find the maximum number in the array** to determine the number of digits.
2. **Iterate through each digit (from least significant to most significant)**.
3. **Apply counting sort** on each digit.
4. **Sort elements based on each digit position**.
5. **Repeat until all digits are processed**.
6. **Combine the sorted results** to get the fully sorted array.

Code:

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

vector<int> radixSort(vector<int> arr, int size);
vector<int> countSort(vector<int> arr, int size, int exponent);
int maximum(const vector<int>& arr);
int minimum(const vector<int>& arr);

int main() {
    int size;
    cout << "Enter the array size: ";
    cin >> size;

    vector<int> arr(size);
    cout << "Enter the array elements:\n";
    for (int i = 0; i < size; i++) {
        cout << "arr[" << i << "] = ";
        cin >> arr[i];
    }
```

```
    cout << "\nArray before sorting:\n";
    for (int i = 0; i < size; i++) {
 cout << "arr[" << i << "] = " << arr[i] << endl;
}

arr = radixSort(arr, size);

cout << "\nArray after sorting:\n";
for (int i = 0; i < size; i++) {
 cout << "arr[" << i << "] = " << arr[i] << endl;
}

 return 0;
}

vector<int> radixSort(vector<int> arr, int size) {
    int maxOfArr = maximum(arr);
    int exponent = 1;

while (exponent <= maxOfArr) {
arr = countSort(arr, size, exponent);
exponent *= 10;
}

 return arr;
}

vector<int> countSort(vector<int> arr, int size, int exponent) {
int range = 10; // For decimal numbers (0-9)
    vector<int> frequencyArray(range, 0);
    vector<int> newArr(size);

    // Count occurrences of digits at the current place
    for (int i = 0; i < size; i++) {
        frequencyArray[(arr[i] / exponent) % 10]++;
    }

    // Update frequency array to store cumulative counts
    for (int i = 1; i < range; i++) {
        frequencyArray[i] += frequencyArray[i - 1];
    }

// Build the sorted array
for (int i = size - 1; i >= 0; i--) {
  int pos = frequencyArray[(arr[i] / exponent) % 10] - 1;
```

```
newArr[pos] = arr[i];
frequencyArray[(arr[i] / exponent) % 10]--;
}

return newArr;
        }

int maximum(const vector<int>& arr) {
 int max = INT_MIN;
for (int num : arr) {
if (num > max) {
max = num;
}
}
return max;
}

int minimum(const vector<int>& arr) {
int min = INT_MAX;
for (int num : arr) {
if (num < min) {
min = num;
}
}
return min;
}
```

Output:

```
Enter the array size: 6
Enter the array elements:
arr[0] = 170
arr[1] = 45
arr[2] = 75
arr[3] = 90
arr[4] = 802
arr[5] = 24

Array before sorting:
arr[0] = 170
arr[1] = 45
arr[2] = 75
arr[3] = 90
arr[4] = 802
arr[5] = 24
```

Array after sorting:
arr[0] = 24
arr[1] = 45
arr[2] = 75
arr[3] = 90
arr[4] = 170
arr[5] = 802

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title:To implement a program in cpp for Quick Sort.

Quick Sort Algorithm:

1. **Select a pivot element** from the array.
2. **Partition** the array into two sub-arrays:
   - Elements smaller than the pivot go to the left.
   - Elements larger than the pivot go to the right.
3. **Recursively apply** the Quick Sort to the sub-arrays.
4. **Combine** the sorted sub-arrays along with the pivot to form the sorted array.
5. Repeat the process until the entire array is sorted.

Code:

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

// QuickSort Function
void quickSort(int *arr, int low, int high) {
   int i = low, j = high;
   int pivot = arr[(low + high) / 2]; // Choose middle element as pivot

   while (i <= j) {
      while (arr[i] < pivot) i++;
      while (arr[j] > pivot) j--;

      if (i <= j) {
         swap(arr[i], arr[j]); // Swap elements
         i++;
         j--;
      }
   }

   // Recursively sort the two halves
   if (low < j)
```

```
        quickSort(arr, low, j);
    if (i < high)
        quickSort(arr, i, high);
}

int main() {
    int n;

    cout << "Enter the number of elements in the array: ";
    cin >> n;

    int arr[n];
    cout << "Enter the elements of the array:\n";
    for (int i = 0; i < n; i++) {
        cout << "arr[" << i << "]: ";
        cin >> arr[i];
    }

    // Perform QuickSort
    quickSort(arr, 0, n - 1);

    // Output the sorted array
    cout << "The sorted array is:\n";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << endl;
    }

    return 0;
}
```

Output:

```
Enter the number of elements in the array: 6

Enter the elements of the array:
arr[0]: 42
arr[1]: 17
arr[2]: 33
arr[3]: 8
arr[4]: 25
arr[5]: 90

The sorted array is:
8
17
25
```

33
42
90

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: __ _ _ _ _ _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title:To implement a program in cpp for Merge Sort.

Merge Sort Algorithm:

1. **Divide** the input array into two halves recursively until each subarray has one or no elements.
2. **Merge** two halves by comparing the elements and sorting them.
3. In each merge step:
   - **Copy** elements from two halves.
   - **Merge** them into a sorted order by comparing the elements.
4. **Repeat** the merging process until the entire array is sorted

Code:

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

// Merge function to merge two halves
void merge(vector<int>& arr, int left, int mid, int right) {
    int size1 = mid - left + 1;
    int size2 = right - mid;

    // Create temporary arrays
    vector<int> Left(size1), Right(size2);

    // Copy data to temp arrays
    for (int i = 0; i < size1; i++) {
        Left[i] = arr[left + i];
    }
    for (int j = 0; j < size2; j++) {
        Right[j] = arr[mid + 1 + j];
    }

    // Merge the temp arrays back into arr
    int i = 0, j = 0, k = left;
```

```
    while (i < size1 && j < size2) {
        if (Left[i] <= Right[j]) {
            arr[k] = Left[i];
            i++;
        } else {
            arr[k] = Right[j];
            j++;
        }
        k++;
    }

    // Copy remaining elements of Left[] if any
while (i < size1) {
        arr[k] = Left[i];
        i++;
        k++;

  }

    // Copy remaining elements of Right[] if any
    while (j < size2) {
        arr[k] = Right[j];
        j++;
        k++;
    }
}


// Merge Sort function to divide and conquer

void mergeSort(vector<int>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;

        // Recursively divide the array into two halves
        mergeSort(arr, left, mid);
        mergeSort(arr, mid + 1, right);

        // Merge the two halves
        merge(arr, left, mid, right);
    }
}

int main() {
int size;
```

```
 cout << "Enter the size: ";
    cin >> size;

    vector<int> arr(size);
    cout << "Enter the elements of the array:\n";
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }

    // Call Merge Sort
mergeSort(arr, 0, size - 1);

    // Print the sorted array
    cout << "The sorted array is:\n";



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

cout << arr[i] << endl;
    }

    return 0;
}
```

Output:

Enter the size: 6
Enter the elements of the array:
34
7
23
90
12
5

The sorted array is:
5
7
12
23
34
90

Godavari Institute of Management & Research, Jalgaon
Masters Computer Application (MCA)

Name: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ __ _ _ _ _ _ _ Roll No: _ _ _ _ _

Date of Performance: _ _ /_ _/20_ _ Batch: _ _ _ _ _ _ Class: MCA. -I

Subject: Sign. of Teacher

Title:To implement a program in cpp for Heap Sort.

Heap Sort Algorithm:

1. **Build a max heap** from the input array.
   - For a given array of size n, create a max heap by arranging elements such that the parent is greater than its children.
2. **Extract the maximum element** from the heap (root of the max heap) and place it at the end of the array.
3. **Heapify the reduced heap** (excluding the last element) to maintain the max heap property.
4. **Repeat steps 2-3** until the heap size reduces to 1.
5. The array is now sorted in ascending order.

Code:

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

// Function prototypes
void heapify(int*, int, int);
void heapsort(int*, int);
void print_array(int*, int);

int main()
{
    int arr[] = {10, 3, 5, 16, 92, 12, 56, 43};
    int n = sizeof(arr) / sizeof(arr[0]);

cout << "\nArray before sorting:\n";
print_array(arr, n);

heapsort(arr, n);

cout << "\n\nArray after sorting:\n";
 print_array(arr, n);
```

```
return 0;
}

/* Function to perform heapify on the subtree with root at index i */
void heapify(int* arr, int n, int i)
{
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    // Check if the left child is larger than the root
    if (left < n && arr[left] > arr[largest]) {
        largest = left;
    }
    // Check if the right child is larger than the largest so far
    if (right < n && arr[right] > arr[largest]) {
        largest = right;
    }

    // If the largest is not the root, swap them and continue heapifying
    if (largest != i) {
        swap(arr[i], arr[largest]);
        heapify(arr, n, largest);
    }
}

/* Function to perform heapsort */
void heapsort(int* arr, int n)
{
    // Build the max heap
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }

    // One by one extract elements from the heap
    for (int i = n - 1; i >= 0; i--) {
        // Move the current root (largest) to the end
        swap(arr[0], arr[i]);

        // Call heapify on the reduced heap
        heapify(arr, i, 0);
    }
}

/* Function to print the array */
```

```
void print_array(int* arr, int n)
{
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
```

Output:

```
int arr[] = {10, 3, 5, 16, 92, 12, 56, 43};

Array before sorting:
10 3 5 16 92 12 56 43

Array after sorting:
3 5 10 12 16 43 56 92
```