Ques 1) Develop a Menu driven program to demonstrate the following operations of Arrays ——MENU——- 1.CREATE 2. DISPLAY 3. INSERT 4. DELETE 5. LINEAR SEARCH 6. EXIT

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


const int MAX = 100;


void createArray(int arr[], int &n) {
    cout << "Enter number of elements (1.." << MAX << "): ";
    cin >> n;
    if (n < 1) { cout << "Size must be >= 1. Setting to 1.\n"; n = 1; }
    if (n > MAX) { cout << "Exceeds MAX. Setting to " << MAX << ".\n";
n = MAX; }
    cout << "Enter " << n << " elements:\n";
    for (int i = 0; i < n; ++i) cin >> arr[i];
    cout << "Array created.\n";
}


void displayArray(const int arr[], int n) {
    if (n == 0) { cout << "Array is empty.\n"; return; }
    cout << "Array elements (" << n << "): ";
    for (int i = 0; i < n; ++i) cout << arr[i] << (i + 1 == n ? '\n' : ' ');
```

```
}

bool insertAt(int arr[], int &n, int pos, int value) {
   if (n == MAX) { cout << "Array is full. Cannot insert.\n"; return false;
}
   if (pos < 1 || pos > n + 1) { cout << "Invalid position. Use 1.." << (n +
1) << ".\n"; return false; }
   for (int i = n - 1; i >= pos - 1; --i) arr[i + 1] = arr[i]; // shift right
   arr[pos - 1] = value;
   ++n;
   return true;
}

bool deleteAt(int arr[], int &n, int pos) {
   if (n == 0) { cout << "Array is empty. Cannot delete.\n"; return false;
}
   if (pos < 1 || pos > n) { cout << "Invalid position. Use 1.." << n <<
".\n"; return false; }
   for (int i = pos - 1; i < n - 1; ++i) arr[i] = arr[i + 1]; // shift left
   --n;
   return true;
}

int linearSearch(const int arr[], int n, int key) {
   int count = 0;
   for (int i = 0; i < n; ++i) {
       if (arr[i] == key) {
           if (count == 0) cout << "Found at position(s): ";
           cout << (i + 1) << " ";
           ++count;
       }
   }
   if (count) cout << "\n";
   return count;
}

int main() {
   int arr[MAX];
   int n = 0;
   int choice;

   do {
       cout << "\n----- MENU -----\n"
            << "1. CREATE\n"
            << "2. DISPLAY\n"
            << "3. INSERT\n"
            << "4. DELETE\n"
            << "5. LINEAR SEARCH\n"
            << "6. EXIT\n"
            << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
          case 1: // CREATE
            createArray(arr, n);
            break;

          case 2: // DISPLAY
            displayArray(arr, n);
            break;

          case 3: { // INSERT
            int pos, value;
            cout << "Enter position to insert (1.." << (n + 1) << "): ";
            cin >> pos;
            cout << "Enter value to insert: ";
            cin >> value;
            if (insertAt(arr, n, pos, value))
              cout << "Inserted " << value << " at position " << pos <<
".\n";
```

```
    break;
}

case 4: { // DELETE
    int pos;
    cout << "Enter position to delete (1.." << n << "): ";
    cin >> pos;
    if (deleteAt(arr, n, pos))
        cout << "Deleted element at position " << pos << ".\n";
    break;
}

case 5: { // LINEAR SEARCH
    if (n == 0) { cout << "Array is empty.\n"; break; }
    int key;
    cout << "Enter value to search: ";
    cin >> key;
    int found = linearSearch(arr, n, key);
    if (!found) cout << key << " not found.\n";
    else cout << "Occurrences: " << found << ".\n";
    break;
}
```

```
            case 6:
                cout << "Exiting...\n";
                break;

            default:
                cout << "Invalid choice. Try again.\n";
        }
    } while (choice != 6);

    return 0;
}
```

Output->

```
----- MENU -----
1. CREATE
2. DISPLAY
3. INSERT
4. DELETE
5. LINEAR SEARCH
6. EXIT
Enter your choice: 1
Enter number of elements (1..100): 5
Enter 5 elements:
1
2
```

```
3
4
5
Array created.

----- MENU -----
1. CREATE
2. DISPLAY
3. INSERT
4. DELETE
5. LINEAR SEARCH
6. EXIT
Enter your choice: 2
Array elements (5): 1 2 3 4 5

----- MENU -----
1. CREATE
2. DISPLAY
3. INSERT
4. DELETE
5. LINEAR SEARCH
6. EXIT
Enter your choice: 3
Enter position to insert (1..6): 2

Enter value to insert: 6

Inserted 6 at position 2.


----- MENU -----

1. CREATE

2. DISPLAY

3. INSERT

4. DELETE

5. LINEAR SEARCH

6. EXIT

Enter your choice: 2

Array elements (6): 1 6 2 3 4 5


----- MENU -----

1. CREATE

2. DISPLAY

3. INSERT

4. DELETE

5. LINEAR SEARCH

6. EXIT

Enter your choice: 4

Enter position to delete (1..6): 2
```

```
Deleted element at position 2.

----- MENU -----
1. CREATE
2. DISPLAY
3. INSERT
4. DELETE
5. LINEAR SEARCH
6. EXIT
Enter your choice: 2
Array elements (5): 1 2 3 4 5

----- MENU -----
1. CREATE
2. DISPLAY
3. INSERT
4. DELETE
5. LINEAR SEARCH
6. EXIT
Enter your choice: 5
Enter value to search: 3
Found at position(s): 3
Occurrences: 1.
```

----- MENU -----

1. CREATE

2. DISPLAY

3. INSERT

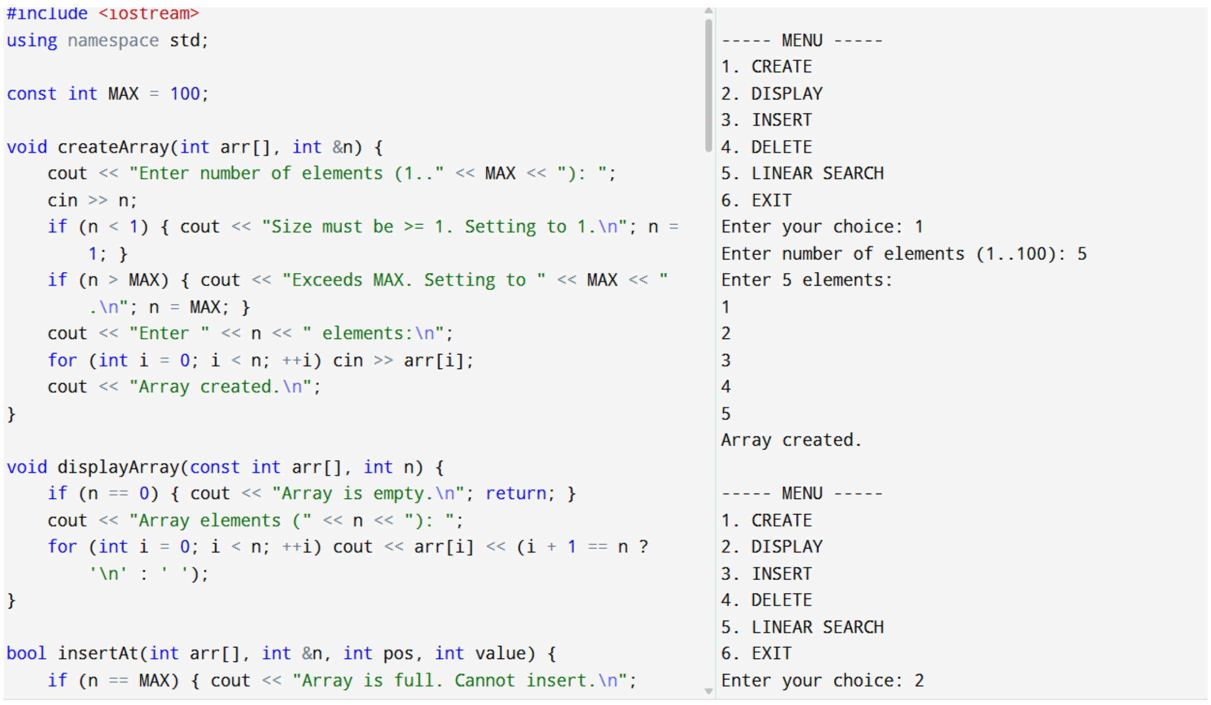
4. DELETE

5. LINEAR SEARCH

6. EXIT

Enter your choice: 6

Exiting...



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

const int MAX = 100;

void createArray(int arr[], int &n) {
    cout << "Enter number of elements (1.." << MAX << "): ";
    cin >> n;
    if (n < 1) { cout << "Size must be >= 1. Setting to 1.\n"; n =
        1; }
    if (n > MAX) { cout << "Exceeds MAX. Setting to " << MAX << "
        .\n"; n = MAX; }
    cout << "Enter " << n << " elements:\n";
    for (int i = 0; i < n; ++i) cin >> arr[i];
    cout << "Array created.\n";
}

void displayArray(const int arr[], int n) {
    if (n == 0) { cout << "Array is empty.\n"; return; }
    cout << "Array elements (" << n << "): ";
    for (int i = 0; i < n; ++i) cout << arr[i] << (i + 1 == n ?
        '\n' : ' ');
}

bool insertAt(int arr[], int &n, int pos, int value) {
    if (n == MAX) { cout << "Array is full. Cannot insert.\n";
```

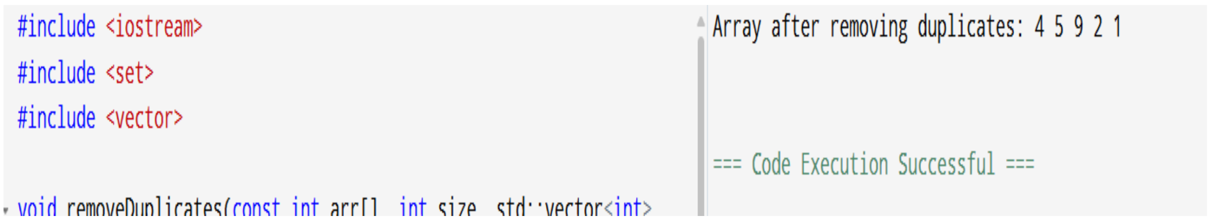
```
----- MENU -----
1. CREATE
2. DISPLAY
3. INSERT
4. DELETE
5. LINEAR SEARCH
6. EXIT
Enter your choice: 1
Enter number of elements (1..100): 5
Enter 5 elements:
1
2
3
4
5
Array created.

----- MENU -----
1. CREATE
2. DISPLAY
3. INSERT
4. DELETE
5. LINEAR SEARCH
6. EXIT
Enter your choice: 2
```

Ques 2 Design the logic to remove the duplicate elements from an Array and after the deletion the array should contain the unique elements.

Ans->#include <iostream>

```
// Remove duplicates without sorting or extra structures
int removeDuplicates(int arr[], int size) {
    if (size == 0 || size == 1)
        return size;

    int newSize = size;

    for (int i = 0; i < newSize; ++i) {
        for (int j = i + 1; j < newSize; ) {
            if (arr[j] == arr[i]) {
                // Shift elements left to remove duplicate at j
                for (int k = j; k < newSize - 1; ++k) {
                    arr[k] = arr[k + 1];
                }
                newSize--;  // Reduce size
                // Don't increment j here, because we want to check the
new element at position j
            } else {
                j++;  // Move to next element
            }
        }
    }
    return newSize;
}

int main() {
    int arr[] = {4, 5, 9, 4, 9, 2, 1, 5, 2};
    int size = sizeof(arr) / sizeof(arr[0]);

    size = removeDuplicates(arr, size);

    std::cout << "Array after removing duplicates: ";
    for (int i = 0; i < size; ++i) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```

Output->Array after removing duplicates: 4 5 9 2 1



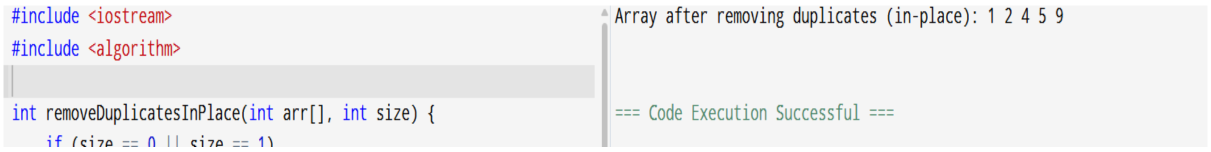
```
#include <iostream>
#include <set>
#include <vector>

void removeDuplicates(const int arr[], int size, std::vector<int>
```

```
Array after removing duplicates: 4 5 9 2 1


=== Code Execution Successful ===
```

In-place removal (no extra space, but sorted array required)

```
#include <iostream>
#include <algorithm>  // for std::sort

// Removes duplicates from sorted array, returns new size
int removeDuplicatesInPlace(int arr[], int size) {
   if (size == 0 || size == 1)
      return size;

   std::sort(arr, arr + size);  // Sort the array

   int j = 0;  // Index of last unique element

   for (int i = 1; i < size; i++) {
      if (arr[i] != arr[j]) {
         j++;
         arr[j] = arr[i];
      }
   }
   return j + 1;
}
```

```
int main() {
    int arr[] = {4, 5, 9, 4, 9, 2, 1, 5, 2};
    int size = sizeof(arr) / sizeof(arr[0]);

    size = removeDuplicatesInPlace(arr, size);

    std::cout << "Array after removing duplicates: ";
    for (int i = 0; i < size; ++i) {
        std::cout << arr[i] << " ";
    }
    std::cout << std::endl;

    return 0;
}
```



```
#include <iostream>
#include <algorithm>

int removeDuplicatesInPlace(int arr[], int size) {
    if (size == 0 || size == 1)
```

```
Array after removing duplicates (in-place): 1 2 4 5 9


=== Code Execution Successful ===
```

Ques 3) Predict the Output of the following program int main() { int i; int arr[5] = {1}; for (i = 0; i < 5; i++) printf("%d",arr[i]); return 0; }

Ans->#include <iostream>

```
using namespace std;

int main()

{
    int arr[5] = {1};

    for (int i = 0; i < 5; i++)

        cout << arr[i]<<endl;

    return 0;

}
```

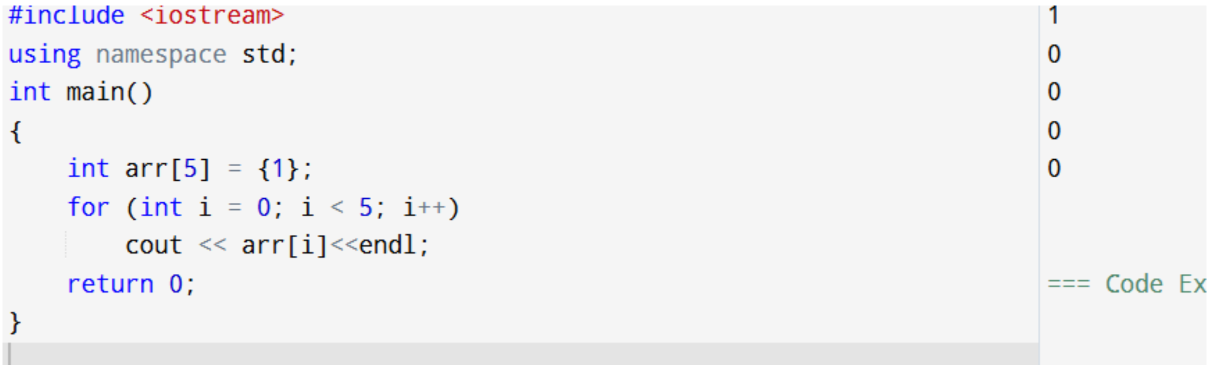
Output->1

0

0

0

0



```
#include <iostream>
using namespace std;
int main()
{
    int arr[5] = {1};
    for (int i = 0; i < 5; i++)
        cout << arr[i]<<endl;
    return 0;
}
```

```
1
0
0
0
0


=== Code Ex
```

Ques 4Implement the logic to a. Reverse the elements of an array b. Find the matrix multiplication c. Find the Transpose of a Matrix

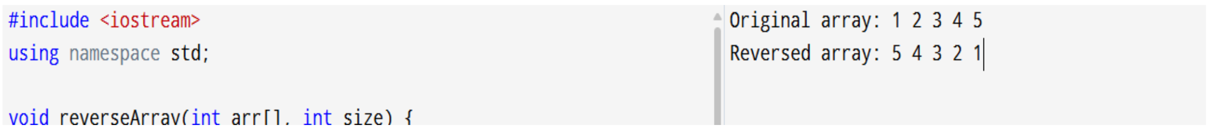
Ans-> Reverse the elements of an array

```
#include <iostream>

using namespace std;


void reverseArray(int arr[], int size) {

    int start = 0, end = size - 1;
    while (start < end) {
        // Swap arr[start] and arr[end]
        int temp = arr[start];
        arr[start] = arr[end];
        arr[end] = temp;

        start++;
        end--;
    }
}

int main() {
    int arr[] = {1, 2, 3, 4, 5};
    int size = sizeof(arr) / sizeof(arr[0]);

    cout << "Original array: ";
    for(int i = 0; i < size; i++)
        cout << arr[i] << " ";

    reverseArray(arr, size);

    cout << "\nReversed array: ";
    for(int i = 0; i < size; i++)
```

```
        cout << arr[i] << " ";

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

Output-> Original array: 1 2 3 4 5

Reversed array: 5 4 3 2 1



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

void reverseArray(int arr[], int size) {
```

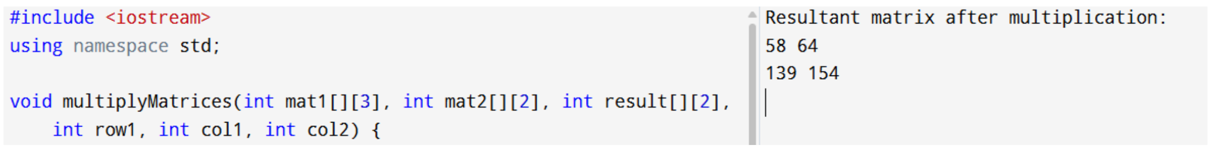
```
Original array: 1 2 3 4 5
Reversed array: 5 4 3 2 1
```

Find the matrix multiplication

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

void multiplyMatrices(int mat1[][3], int mat2[][2], int result[][2], int
row1, int col1, int col2) {
    // Initialize result matrix to 0
    for (int i = 0; i < row1; i++) {
        for (int j = 0; j < col2; j++) {
            result[i][j] = 0;
        }
    }

    // Matrix multiplication logic
    for (int i = 0; i < row1; i++) {
        for (int j = 0; j < col2; j++) {
            for (int k = 0; k < col1; k++) {
                result[i][j] += mat1[i][k] * mat2[k][j];
            }
        }
    }
}

int main() {
    int mat1[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    int mat2[3][2] = {
        {7, 8},
        {9, 10},
        {11, 12}
    };

    int result[2][2];  // Resultant matrix will be 2x2
    multiplyMatrices(mat1, mat2, result, 2, 3, 2);

    cout << "Resultant matrix after multiplication:\n";
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            cout << result[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Output->Resultant matrix after multiplication:

```
58 64
139 154
```



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

void multiplyMatrices(int mat1[][3], int mat2[][2], int result[][2],
    int row1, int col1, int col2) {
```

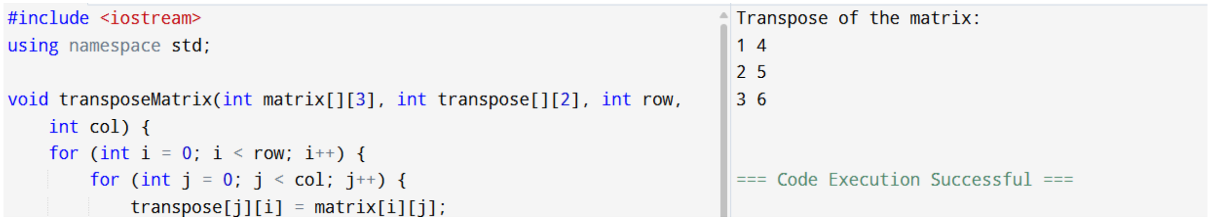
```
Resultant matrix after multiplication:
58 64
139 154
```

Find the transpose of a matrix

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

void transposeMatrix(int matrix[][3], int transpose[][2], int row, int
col) {
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            transpose[j][i] = matrix[i][j];
        }
    }
}

int main() {
    int matrix[2][3] = {
        {1, 2, 3},
        {4, 5, 6}
    };

    int transpose[3][2];  // Transpose will be 3x2

    transposeMatrix(matrix, transpose, 2, 3);

    cout << "Transpose of the matrix:\n";
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 2; j++) {
            cout << transpose[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Output-> Transpose of the matrix:

```
1 4
2 5
3 6
```



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

void transposeMatrix(int matrix[][3], int transpose[][2], int row,
    int col) {
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            transpose[j][i] = matrix[i][j];
```

```
Transpose of the matrix:
1 4
2 5
3 6


=== Code Execution Successful ===
```

Ques 5 Write a program to find sum of every row and every column in a two-dimensional array.

Ans->

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


int main() {
    const int ROWS = 3;
    const int COLS = 4;

    int arr[ROWS][COLS] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    };

    // Calculate and print sum of each row
    for (int i = 0; i < ROWS; ++i) {
        int rowSum = 0;
        for (int j = 0; j < COLS; ++j) {
            rowSum += arr[i][j];
        }
        cout << "Sum of row " << i << " = " << rowSum << endl;
    }

    // Calculate and print sum of each column
    for (int j = 0; j < COLS; ++j) {
        int colSum = 0;
        for (int i = 0; i < ROWS; ++i) {
            colSum += arr[i][j];
        }
        cout << "Sum of column " << j << " = " << colSum << endl;
    }

    return 0;
}
```

Output->Sum of row 0 = 10

Sum of row 1 = 26

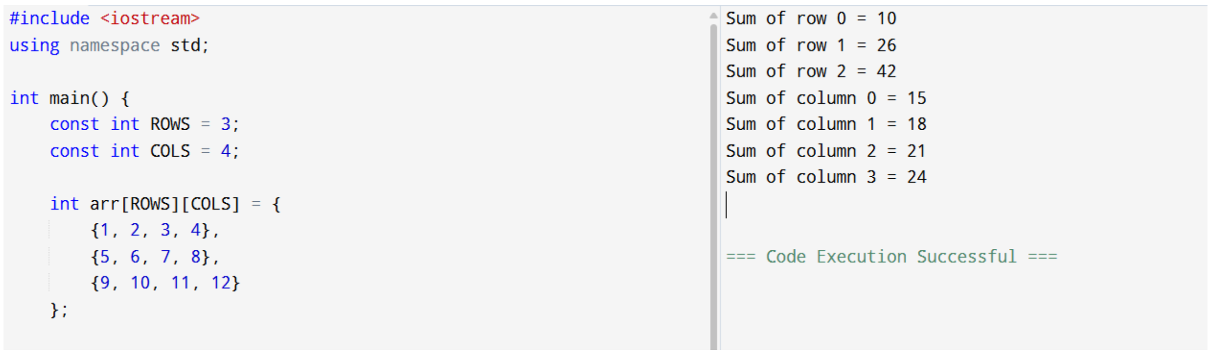
Sum of row 2 = 42

Sum of column 0 = 15

Sum of column 1 = 18

Sum of column 2 = 21

Sum of column 3 = 24



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

int main() {
    const int ROWS = 3;
    const int COLS = 4;

    int arr[ROWS][COLS] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    };
```

```
Sum of row 0 = 10
Sum of row 1 = 26
Sum of row 2 = 42
Sum of column 0 = 15
Sum of column 1 = 18
Sum of column 2 = 21
Sum of column 3 = 24


=== Code Execution Successful ===
```