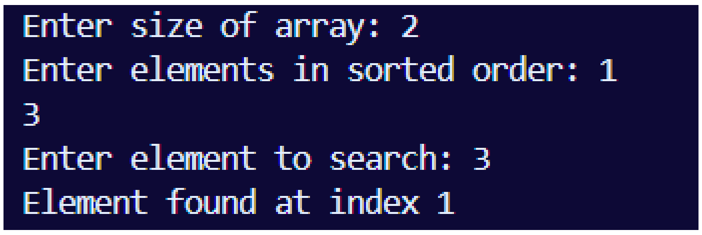
Assignment 2

AQ1

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

int main()
{
   int size, target;
   cout << "Enter size of array: ";
   cin >> size;

   int nums[size];
   cout << "Enter elements in sorted order: ";
   for(int i = 0; i < size; i++)
   {
      cin >> nums[i];
   }

   cout << "Enter element to search: ";
   cin >> target;
```

```
int left = 0, right = size - 1, midIndex, foundAt = -1;

while(left <= right)
{
   midIndex = (left + right) / 2;

   if(nums[midIndex] == target)
   {
      foundAt = midIndex;
      break;
   }
   else if(nums[midIndex] < target)
   {
      left = midIndex + 1;
   }
   else
   {
      right = midIndex - 1;
   }
}
```

```
    if(foundAt != -1)
    {
        cout << "Element found at index " << foundAt << endl;
    }
    else
    {
        cout << "Element not found" << endl;
    }
}
```

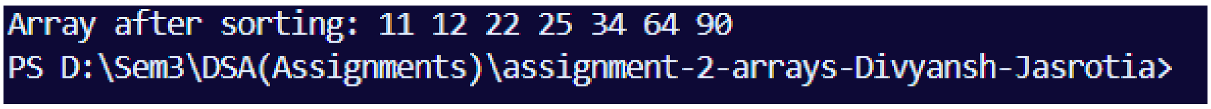


```
Enter size of array: 2
Enter elements in sorted order: 1
3
Enter element to search: 3
Element found at index 1
```

AQ2

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

int main()
{
    int data[] = {64, 34, 25, 12, 22, 11, 90};
int length = 7;

for(int pass = 0; pass < length - 1; pass++)
{
   for(int idx = 0; idx < length - pass - 1; idx++)
   {
      if(data[idx] > data[idx + 1])
      {
         int temp = data[idx];
         data[idx] = data[idx + 1];
         data[idx + 1] = temp;
      }
   }
}

cout << "Array after sorting: ";
for(int k = 0; k < length; k++)
{
   cout << data[k] << " ";
}
cout << endl;
```

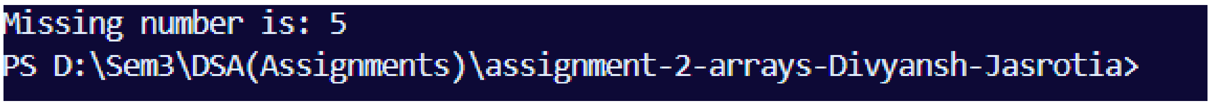
```
}
```



```
Array after sorting: 11 12 22 25 34 64 90
PS D:\Sem3\DSA(Assignments)\assignment-2-arrays-Divyansh-Jasrotia>
```

AQ3

```
#include <iostream>

using namespace std;


int main()

{

   int nums[] = {1, 2, 3, 4, 6, 7, 8};

   int size = 7;

   int missingNum = -1;


   for(int val = 1; val <= size + 1; val++)

   {

      bool found = false;

      for(int idx = 0; idx < size; idx++)

      {

         if(nums[idx] == val)

         {
                found = true;
                break;
            }
        }
        if(!found)
        {
            missingNum = val;
            break;
        }
    }

    cout << "Missing number is: " << missingNum << endl;
}
```



```
Missing number is: 5
PS D:\Sem3\DSA(Assignments)\assignment-2-arrays-Divyansh-Jasrotia>
```

AQ4

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


int main()
```

```
{
   cout << "Part (a)\n";

   char first[200], second[100];
   cout << "Enter the first string: ";
   cin.getline(first, 200);
   cout << "Enter the second string: ";
   cin.getline(second, 100);

   int idx1 = 0, idx2 = 0;
   while (first[idx1] != '\0')
   {
      idx1++;
   }
   while (second[idx2] != '\0')
   {
      first[idx1] = second[idx2];
      idx1++;
      idx2++;
   }
   first[idx1] = '\0';
```

```
cout << "Result after concatenation: " << first << endl;


cout << "\nPart (b)\n";


char revStr[200];
cout << "Enter a string to reverse: ";
cin.getline(revStr, 200);
int len = 0;
while (revStr[len] != '\0')
{
    len++;
}
for (int i = 0; i < len / 2; i++)
{
    char temp = revStr[i];
    revStr[i] = revStr[len - i - 1];
    revStr[len - i - 1] = temp;
}
cout << "Reversed string is: " << revStr << endl;


cout << "\nPart (c)\n";
```

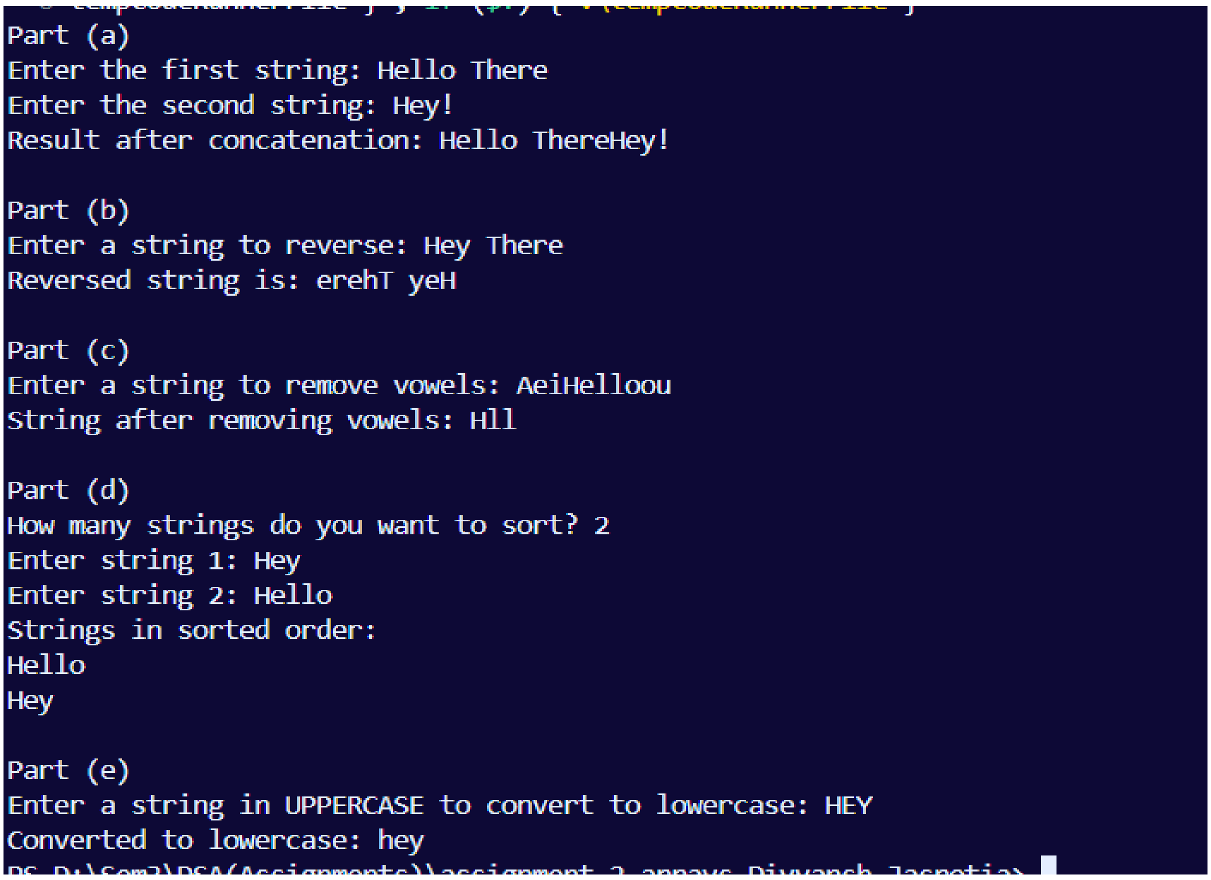
```
char original[200];
cout << "Enter a string to remove vowels: ";
cin.getline(original, 200);
int writeIdx = 0;
for (int i = 0; original[i] != '\0'; i++)
{
  char ch = original[i];
  if (!(ch == 'a' || ch == 'A' || ch == 'e' || ch == 'E' ||
      ch == 'i' || ch == 'I' || ch == 'o' || ch == 'O' ||
      ch == 'u' || ch == 'U'))
  {
    original[writeIdx++] = ch;
  }
}
original[writeIdx] = '\0';
cout << "String after removing vowels: " << original << endl;

cout << "\nPart (d)\n";

int count;
```

```
    cout << "How many strings do you want to sort? ";
    cin >> count;
    cin.ignore();

    char list[count][100];
    for (int i = 0; i < count; i++)
    {
        cout << "Enter string " << i + 1 << ": ";
        cin.getline(list[i], 100);
    }

    for (int i = 0; i < count - 1; i++)
    {
        for (int j = i + 1; j < count; j++)
        {
            int k = 0;
            while (list[i][k] != '\0' && list[j][k] != '\0' && list[i][k] ==
list[j][k])
            {
                k++;
            }
```

```
if (list[i][k] > list[j][k])
{
    char temp[100];
    int p = 0;
    while (list[i][p] != '\0')
    {
        temp[p] = list[i][p];
        p++;
    }
    temp[p] = '\0';

    p = 0;
    while (list[j][p] != '\0')
    {
        list[i][p] = list[j][p];
        p++;
    }
    list[i][p] = '\0';

    p = 0;
    while (temp[p] != '\0')
```

```
            {
                list[j][p] = temp[p];
                p++;
            }
            list[j][p] = '\0';
        }
    }
}

cout << "Strings in sorted order:\n";
for (int i = 0; i < count; i++)
{
    cout << list[i] << endl;
}

cout << "\nPart (e)\n";

char upperStr[200];
cout << "Enter a string in UPPERCASE to convert to
lowercase: ";
cin.getline(upperStr, 200);
```

```
    int i = 0;
    while (upperStr[i] != '\0')
    {
        if (upperStr[i] >= 'A' && upperStr[i] <= 'Z')
        {
            upperStr[i] = upperStr[i] + 32;
        }
        i++;
    }
    cout << "Converted to lowercase: " << upperStr << endl;
}
```



```
Part (a)
Enter the first string: Hello There
Enter the second string: Hey!
Result after concatenation: Hello ThereHey!

Part (b)
Enter a string to reverse: Hey There
Reversed string is: erehT yeH

Part (c)
Enter a string to remove vowels: AeiHelloou
String after removing vowels: Hll

Part (d)
How many strings do you want to sort? 2
Enter string 1: Hey
Enter string 2: Hello
Strings in sorted order:
Hello
Hey

Part (e)
Enter a string in UPPERCASE to convert to lowercase: HEY
Converted to lowercase: hey
```

AQ5

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

int main() {
   int n;

   cout << "Diagonal Matrix\n";
   cout << "Enter the size (n): ";
   cin >> n;

   int diagonalMatrix[n][n];
   cout << "Enter the " << n << "x" << n << " diagonal
matrix:\n";
   for (int row = 0; row < n; row++) {
      for (int col = 0; col < n; col++) {
         cin >> diagonalMatrix[row][col];
      }
   }

   cout << "You entered:\n";
```

```
for (int row = 0; row < n; row++) {
    for (int col = 0; col < n; col++) {
        cout << diagonalMatrix[row][col] << " ";
    }
    cout << "\n";
}

int diagStorage[n];
for (int i = 0; i < n; i++) {
    diagStorage[i] = diagonalMatrix[i][i];
}

cout << "Linear representation (Diagonal): ";
for (int i = 0; i < n; i++) {
    cout << diagStorage[i] << " ";
}
cout << "\n\n";

cout << "Tri-diagonal Matrix\n";
cout << "Enter the size (n): ";
cin >> n;
```

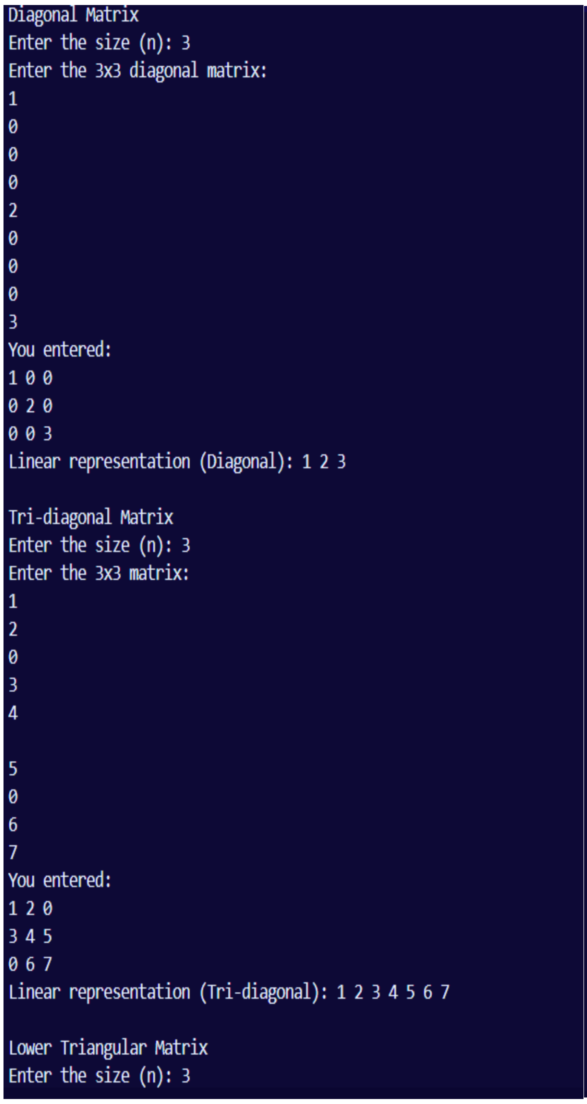
```
int triDiagMatrix[n][n];
cout << "Enter the " << n << "x" << n << " matrix:\n";
for (int row = 0; row < n; row++) {
    for (int col = 0; col < n; col++) {
        cin >> triDiagMatrix[row][col];
    }
}

cout << "You entered:\n";
for (int row = 0; row < n; row++) {
    for (int col = 0; col < n; col++) {
        cout << triDiagMatrix[row][col] << " ";
    }
    cout << "\n";
}

int triStorage[3 * n - 2], index = 0;
for (int row = 0; row < n; row++) {
    for (int col = 0; col < n; col++) {
        if (row == col || row == col + 1 || col == row + 1) {
            triStorage[index++] = triDiagMatrix[row][col];
        }
    }
}

cout << "Linear representation (Tri-diagonal): ";
for (int i = 0; i < 3 * n - 2; i++) {
    cout << triStorage[i] << " ";
}
cout << "\n\n";

cout << "Lower Triangular Matrix\n";
cout << "Enter the size (n): ";
cin >> n;

int lowerMatrix[n][n];
cout << "Enter the " << n << "x" << n << " matrix:\n";
for (int row = 0; row < n; row++) {
    for (int col = 0; col < n; col++) {
        cin >> lowerMatrix[row][col];
    }
```

```
}

cout << "You entered:\n";
for (int row = 0; row < n; row++) {
    for (int col = 0; col < n; col++) {
        cout << lowerMatrix[row][col] << " ";
    }
    cout << "\n";
}

int lowerStorage[n * (n + 1) / 2];
index = 0;
for (int row = 0; row < n; row++) {
    for (int col = 0; col <= row; col++) {
        lowerStorage[index++] = lowerMatrix[row][col];
    }
}

cout << "Linear representation (Lower Triangular): ";
for (int i = 0; i < n * (n + 1) / 2; i++) {
    cout << lowerStorage[i] << " ";
```

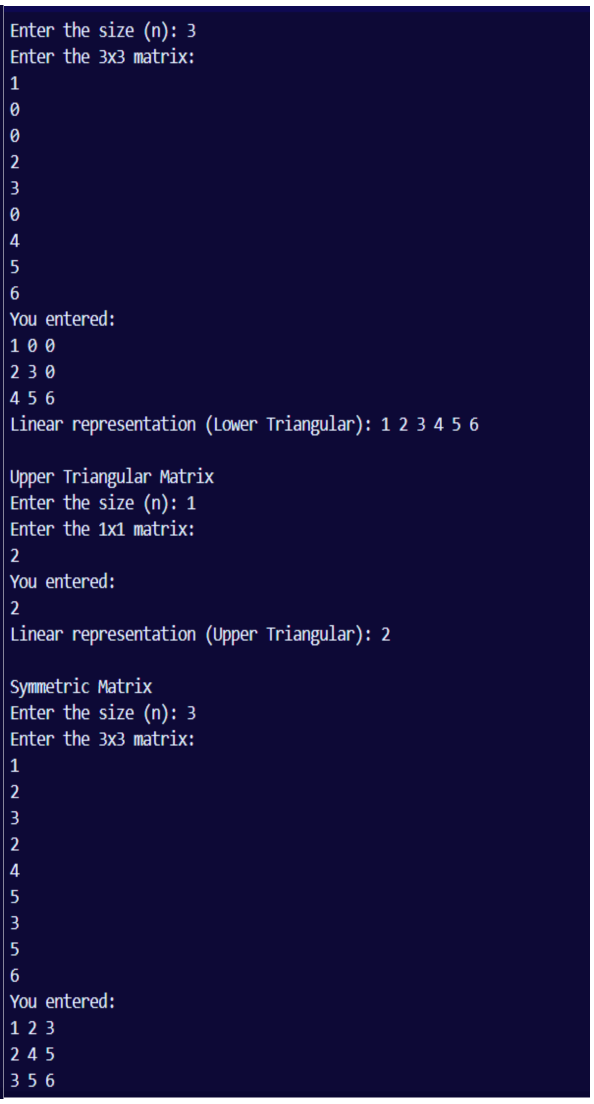
```
}
cout << "\n\n";

cout << "Upper Triangular Matrix\n";
cout << "Enter the size (n): ";
cin >> n;

int upperMatrix[n][n];
cout << "Enter the " << n << "x" << n << " matrix:\n";
for (int row = 0; row < n; row++) {
   for (int col = 0; col < n; col++) {
      cin >> upperMatrix[row][col];
   }
}

cout << "You entered:\n";
for (int row = 0; row < n; row++) {
   for (int col = 0; col < n; col++) {
      cout << upperMatrix[row][col] << " ";
   }
   cout << "\n";
```

```
}

int upperStorage[n * (n + 1) / 2];
index = 0;
for (int row = 0; row < n; row++) {
    for (int col = row; col < n; col++) {
        upperStorage[index++] = upperMatrix[row][col];
    }
}

cout << "Linear representation (Upper Triangular): ";
for (int i = 0; i < n * (n + 1) / 2; i++) {
    cout << upperStorage[i] << " ";
}
cout << "\n\n";

cout << "Symmetric Matrix\n";
cout << "Enter the size (n): ";
cin >> n;

int symmetricMatrix[n][n];
```

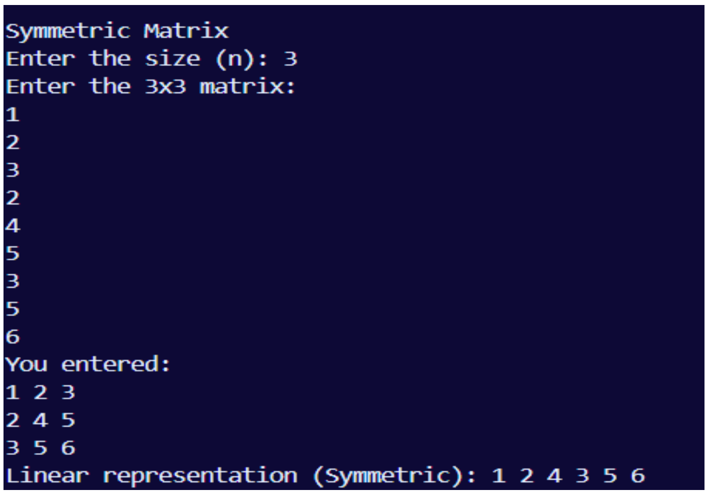
```
cout << "Enter the " << n << "x" << n << " matrix:\n";
for (int row = 0; row < n; row++) {
    for (int col = 0; col < n; col++) {
        cin >> symmetricMatrix[row][col];
    }
}

cout << "You entered:\n";
for (int row = 0; row < n; row++) {
    for (int col = 0; col < n; col++) {
        cout << symmetricMatrix[row][col] << " ";
    }
    cout << "\n";
}

int symStorage[n * (n + 1) / 2];
index = 0;
for (int row = 0; row < n; row++) {
    for (int col = 0; col <= row; col++) {
        symStorage[index++] = symmetricMatrix[row][col];
    }
    }

    cout << "Linear representation (Symmetric): ";
    for (int i = 0; i < n * (n + 1) / 2; i++) {
        cout << symStorage[i] << " ";
    }
    cout << "\n";

    return 0;
}
```



```
Diagonal Matrix
Enter the size (n): 3
Enter the 3x3 diagonal matrix:
1
0
0
0
2
0
0
0
3
You entered:
1 0 0
0 2 0
0 0 3
Linear representation (Diagonal): 1 2 3

Tri-diagonal Matrix
Enter the size (n): 3
Enter the 3x3 matrix:
1
2
0
3
4

5
0
6
7
You entered:
1 2 0
3 4 5
0 6 7
Linear representation (Tri-diagonal): 1 2 3 4 5 6 7

Lower Triangular Matrix
Enter the size (n): 3
```



```
Enter the size (n): 3
Enter the 3x3 matrix:
1
0
0
2
3
0
4
5
6
You entered:
1 0 0
2 3 0
4 5 6
Linear representation (Lower Triangular): 1 2 3 4 5 6

Upper Triangular Matrix
Enter the size (n): 1
Enter the 1x1 matrix:
2
You entered:
2
Linear representation (Upper Triangular): 2

Symmetric Matrix
Enter the size (n): 3
Enter the 3x3 matrix:
1
2
3
2
4
5
3
5
6
You entered:
1 2 3
2 4 5
3 5 6
```



```
Symmetric Matrix
Enter the size (n): 3
Enter the 3x3 matrix:
1
2
3
2
4
5
3
5
6
You entered:
1 2 3
2 4 5
3 5 6
Linear representation (Symmetric): 1 2 4 3 5 6
```

AQ6

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

int main()
{
   int rows, cols, count = 0;
   cout << "Part (a)\n";
   cout << "Enter number of rows and columns: ";
   cin >> rows >> cols;

   int mat[rows][cols];
   cout << "Enter matrix elements:\n";
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
         cin >> mat[i][j];
         if (mat[i][j] != 0) {
            count++;
         }
      }
   }
```

```
int sparse[3][count];
cout << "Matrix:\n";
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        cout << mat[i][j] << " ";
    }
    cout << "\n";
}

int idx = 0;
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        if (mat[i][j] != 0) {
            sparse[0][idx] = i;
            sparse[1][idx] = j;
            sparse[2][idx] = mat[i][j];
            idx++;
        }
    }
}

cout << "Sparse Matrix:\n";
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < count; j++) {
        cout << sparse[i][j] << " ";
    }
    cout << "\n";
}

for (int i = 0; i < count; i++) {
    int temp = sparse[0][i];
    sparse[0][i] = sparse[1][i];
    sparse[1][i] = temp;
}

cout << "Transpose of Sparse Matrix:\n";
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < count; j++) {
        cout << sparse[i][j] << " ";
    }
    cout << "\n";
}

cout << "Part (b)\n";
cout << "Enter number of rows and columns: ";
cin >> rows >> cols;
```

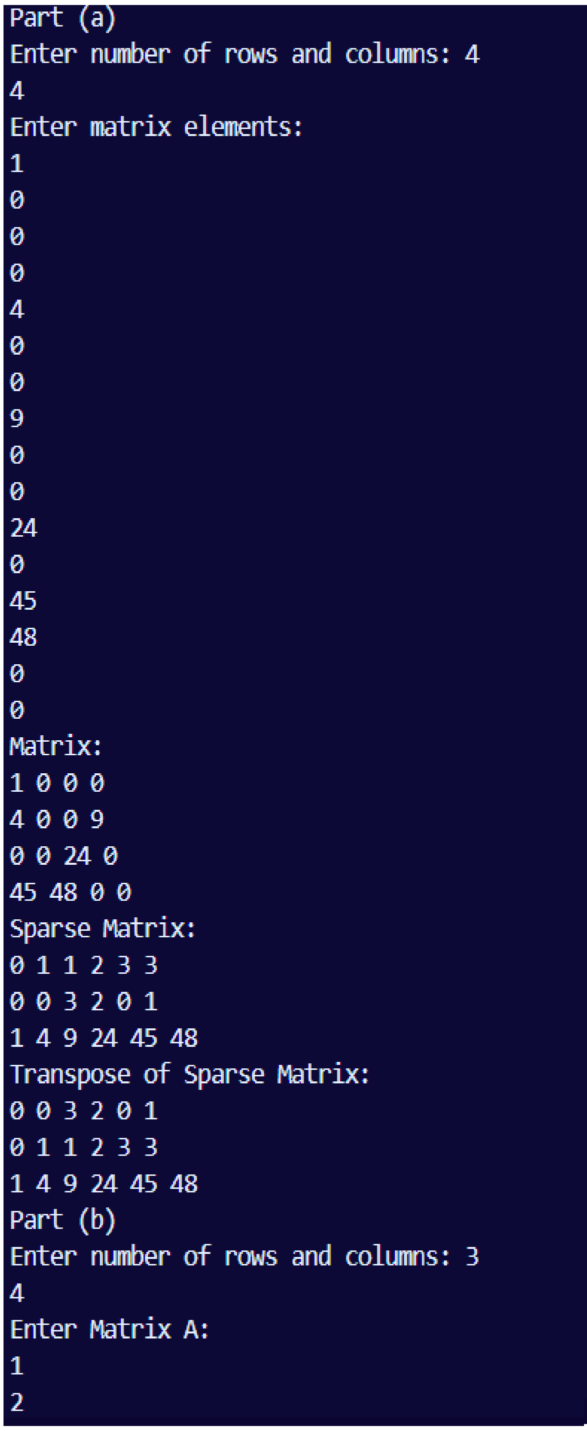
```
}

int nonZeroA = 0, nonZeroB = 0;
int A[rows][cols], B[rows][cols];

cout << "Enter Matrix A:\n";
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        cin >> A[i][j];
        if (A[i][j] != 0) nonZeroA++;
    }
}

int sparseA[3][nonZeroA];

cout << "Enter Matrix B:\n";
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        cin >> B[i][j];
        if (B[i][j] != 0) nonZeroB++;
    }
}

int sparseB[3][nonZeroB];
```

```
cout << "Matrix A:\n";
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        cout << A[i][j] << " ";
    }
    cout << "\n";
}

cout << "Matrix B:\n";
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        cout << B[i][j] << " ";
    }
    cout << "\n";
}

idx = 0;
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        if (A[i][j] != 0) {
            sparseA[0][idx] = i;
            sparseA[1][idx] = j;
            sparseA[2][idx] = A[i][j];
            idx++;
        }
    }
}

cout << "Sparse A:\n";
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < nonZeroA; j++) {
        cout << sparseA[i][j] << " ";
    }
    cout << "\n";
}

idx = 0;
for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
        if (B[i][j] != 0) {
            sparseB[0][idx] = i;
            sparseB[1][idx] = j;
            sparseB[2][idx] = B[i][j];
            idx++;
        }
    }
}

cout << "Sparse B:\n";
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < nonZeroB; j++) {
        cout << sparseB[i][j] << " ";
    }
    cout << "\n";
}

int sum[3][nonZeroA + nonZeroB], p = 0, q = 0;
int sumSize = 0;

while (p < nonZeroA && q < nonZeroB) {
    if (sparseA[0][p] < sparseB[0][q]) {
        for (int i = 0; i < 3; i++) sum[i][sumSize] = sparseA[i][p];
        p++;
    }
    else if (sparseB[0][q] < sparseA[0][p]) {
        for (int i = 0; i < 3; i++) sum[i][sumSize] = sparseB[i][q];
        q++;
    }
    else {
        if (sparseA[1][p] < sparseB[1][q]) {
            for (int i = 0; i < 3; i++) sum[i][sumSize] = sparseA[i][p];
            p++;
        }
        else if (sparseB[1][q] < sparseA[1][p]) {
            for (int i = 0; i < 3; i++) sum[i][sumSize] = sparseB[i][q];
            q++;
        }
        else {
            sum[0][sumSize] = sparseA[0][p];
            sum[1][sumSize] = sparseA[1][p];
            sum[2][sumSize] = sparseA[2][p] + sparseB[2][q];
            p++;
            q++;
        }
    }
    sumSize++;
}

while (p < nonZeroA) {
    for (int i = 0; i < 3; i++) sum[i][sumSize] = sparseA[i][p];
    p++;
    sumSize++;
}

while (q < nonZeroB) {
    for (int i = 0; i < 3; i++) sum[i][sumSize] = sparseB[i][q];
    q++;
    sumSize++;
}

cout << "Sum (Sparse A + B):\n";
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < sumSize; j++) {
        cout << sum[i][j] << " ";
    }
    cout << "\n";
}

cout << "Part (c)\n";
int r1, c1, r2, c2;
cout << "Enter size of Matrix X (rows cols): ";
cin >> r1 >> c1;
cout << "Enter size of Matrix Y (rows cols): ";
cin >> r2 >> c2;
```

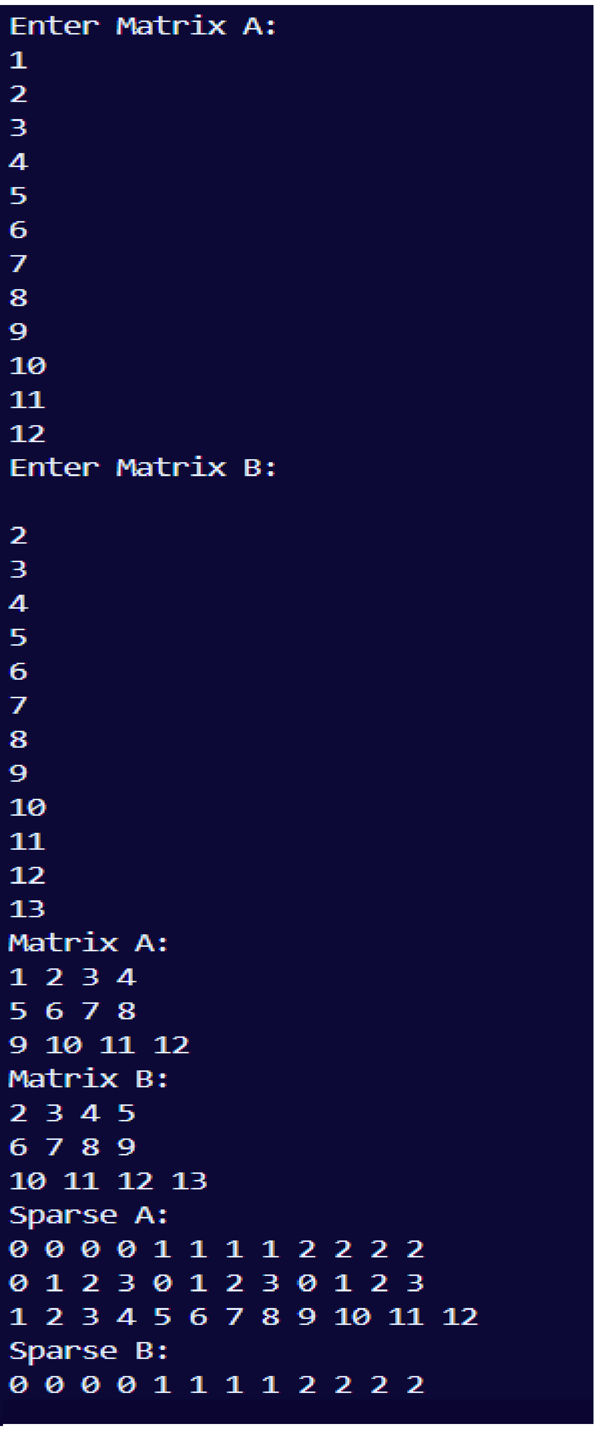
```
if (c1 == r2) {
    int X[r1][c1], Y[r2][c2], xCount = 0, yCount = 0;

    cout << "Enter Matrix X:\n";
    for (int i = 0; i < r1; i++) {
        for (int j = 0; j < c1; j++) {
            cin >> X[i][j];
            if (X[i][j] != 0) xCount++;
        }
    }

    int sparseX[3][xCount];

    cout << "Enter Matrix Y:\n";
    for (int i = 0; i < r2; i++) {
        for (int j = 0; j < c2; j++) {
            cin >> Y[i][j];
            if (Y[i][j] != 0) yCount++;
        }
    }

    int sparseY[3][yCount];
```

```
cout << "Matrix X:\n";
for (int i = 0; i < r1; i++) {
    for (int j = 0; j < c1; j++) {
        cout << X[i][j] << " ";
    }
    cout << "\n";
}

cout << "Matrix Y:\n";
for (int i = 0; i < r2; i++) {
    for (int j = 0; j < c2; j++) {
        cout << Y[i][j] << " ";
    }
    cout << "\n";
}

idx = 0;
for (int i = 0; i < r1; i++) {
    for (int j = 0; j < c1; j++) {
        if (X[i][j] != 0) {
            sparseX[0][idx] = i;
            sparseX[1][idx] = j;
            sparseX[2][idx] = X[i][j];
            idx++;
        }
    }
}

cout << "Sparse X:\n";
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < xCount; j++) {
        cout << sparseX[i][j] << " ";
    }
    cout << "\n";
}

idx = 0;
for (int i = 0; i < r2; i++) {
    for (int j = 0; j < c2; j++) {
        if (Y[i][j] != 0) {
            sparseY[0][idx] = i;
            sparseY[1][idx] = j;
            sparseY[2][idx] = Y[i][j];
            idx++;
        }
    }
}

cout << "Sparse Y:\n";
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < yCount; j++) {
        cout << sparseY[i][j] << " ";
    }
    cout << "\n";
}

int sparseProd[3][xCount * yCount], prodCount = 0;

for (int i = 0; i < xCount; i++) {
    for (int j = 0; j < yCount; j++) {
        if (sparseX[1][i] == sparseY[0][j]) {
            int row = sparseX[0][i];
            int col = sparseY[1][j];
            int val = sparseX[2][i] * sparseY[2][j];
            bool exists = false;

            for (int z = 0; z < prodCount; z++) {
                if (sparseProd[0][z] == row && sparseProd[1][z] == col) {
                    sparseProd[2][z] += val;
                    exists = true;
                    break;
                }
            }

            if (!exists) {
                sparseProd[0][prodCount] = row;
                sparseProd[1][prodCount] = col;
                sparseProd[2][prodCount] = val;
                prodCount++;
            }
        }
    }
}

cout << "Sparse Product:\n";
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < prodCount; j++) {
        cout << sparseProd[i][j] << " ";
    }
    cout << "\n";
}
```

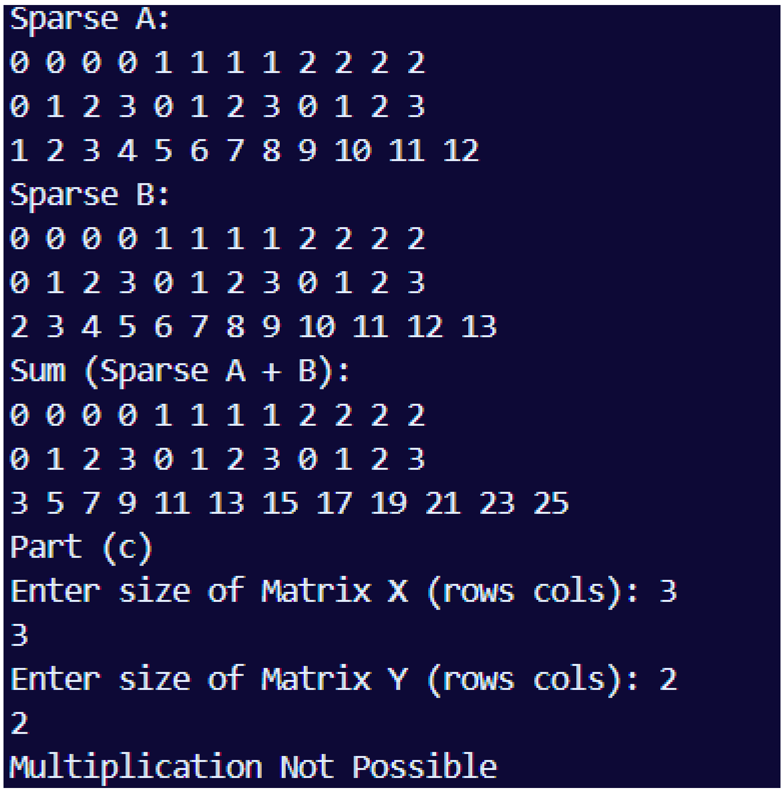
```
    } else {
        cout << "Multiplication Not Possible\n";
    }

    return 0;
}
```



```
Part (a)
Enter number of rows and columns: 4
4
Enter matrix elements:
1
0
0
0
4
0
0
9
0
0
24
0
45
48
0
0
Matrix:
1 0 0 0
4 0 0 9
0 0 24 0
45 48 0 0
Sparse Matrix:
0 1 1 2 3 3
0 0 3 2 0 1
1 4 9 24 45 48
Transpose of Sparse Matrix:
0 0 3 2 0 1
0 1 1 2 3 3
1 4 9 24 45 48
Part (b)
Enter number of rows and columns: 3
4
Enter Matrix A:
1
2
```



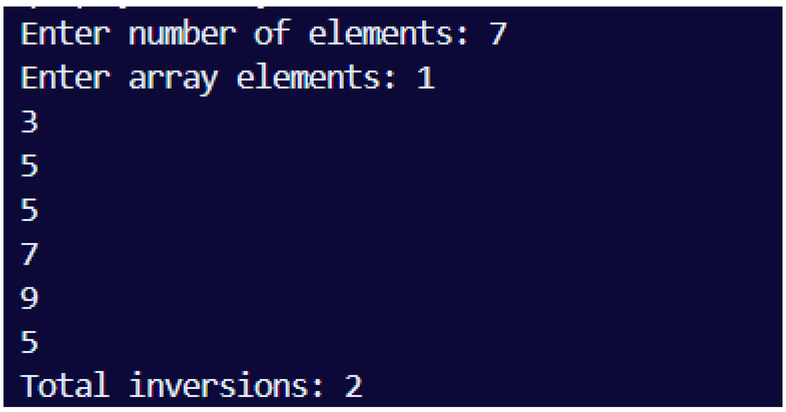
```
Enter Matrix A:
1
2
3
4
5
6
7
8
9
10
11
12
Enter Matrix B:

2
3
4
5
6
7
8
9
10
11
12
13
Matrix A:
1 2 3 4
5 6 7 8
9 10 11 12
Matrix B:
2 3 4 5
6 7 8 9
10 11 12 13
Sparse A:
0 0 0 0 1 1 1 1 2 2 2 2
0 1 2 3 0 1 2 3 0 1 2 3
1 2 3 4 5 6 7 8 9 10 11 12
Sparse B:
0 0 0 0 1 1 1 1 2 2 2 2
```



```
Sparse A:
0 0 0 0 1 1 1 1 2 2 2 2
0 1 2 3 0 1 2 3 0 1 2 3
1 2 3 4 5 6 7 8 9 10 11 12
Sparse B:
0 0 0 0 1 1 1 1 2 2 2 2
0 1 2 3 0 1 2 3 0 1 2 3
2 3 4 5 6 7 8 9 10 11 12 13
Sum (Sparse A + B):
0 0 0 0 1 1 1 1 2 2 2 2
0 1 2 3 0 1 2 3 0 1 2 3
3 5 7 9 11 13 15 17 19 21 23 25
Part (c)
Enter size of Matrix X (rows cols): 3
3
Enter size of Matrix Y (rows cols): 2
2
Multiplication Not Possible
```

AQ7

```
#include <iostream>

using namespace std;


int countInversions(int nums[], int len) {
    int inv = 0;
    for (int a = 0; a < len - 1; a++) {
        for (int b = a + 1; b < len; b++) {
            if (nums[a] > nums[b]) {
                inv++;
            }
        }
    }
    return inv;
}

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

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

    int total = sizeof(data) / sizeof(data[0]);
    int result = countInversions(data, total);

    cout << "Total inversions: " << result << endl;
    return 0;
}
```

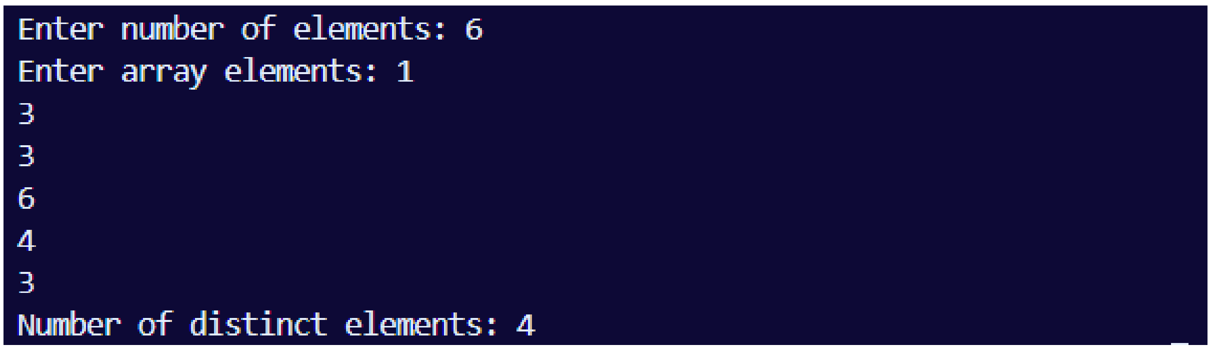


```
Enter number of elements: 7
Enter array elements: 1
3
5
5
7
9
5
Total inversions: 2
```

AQ8

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


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


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

    int distinctCount = 0;

    for (int i = 0; i < n; i++) {
        int isDuplicate = 0;
        for (int j = 0; j < i; j++) {
            if (arr[i] == arr[j]) {
                isDuplicate = 1;
                break;
            }
        }
        if (isDuplicate == 0) {
            distinctCount++;
        }
    }

    cout << "Number of distinct elements: " << distinctCount
<< endl;
    return 0;
```

```
}
```



```
Enter number of elements: 6
Enter array elements: 1
3
3
6
4
3
Number of distinct elements: 4
```