**[4] - Matrix in CPP –**

1) 2D Array Basic Introduction & Input, Output

2) Spiral matrix -

3) Diagonal Sum of Matrix -

4) Search for targeted element in Sorted Matrix -

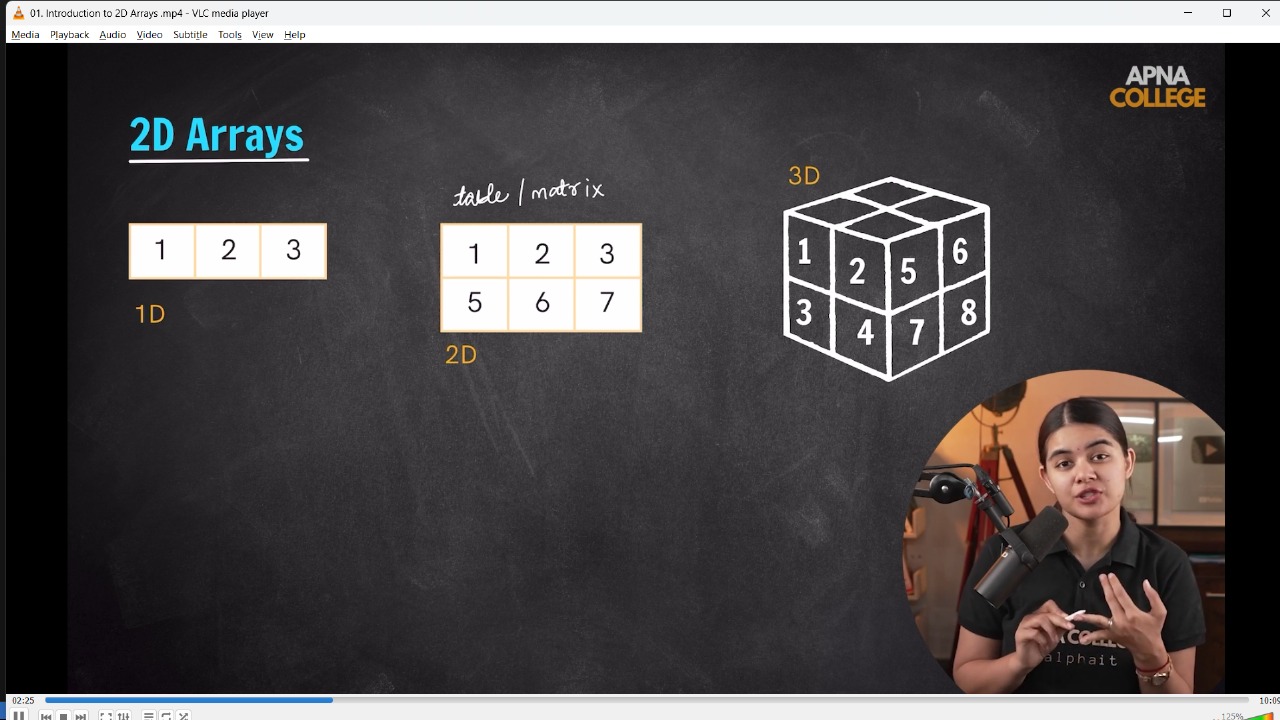
4.1) - Brute Force Approach - most simple method. T.C - O(n\*m)

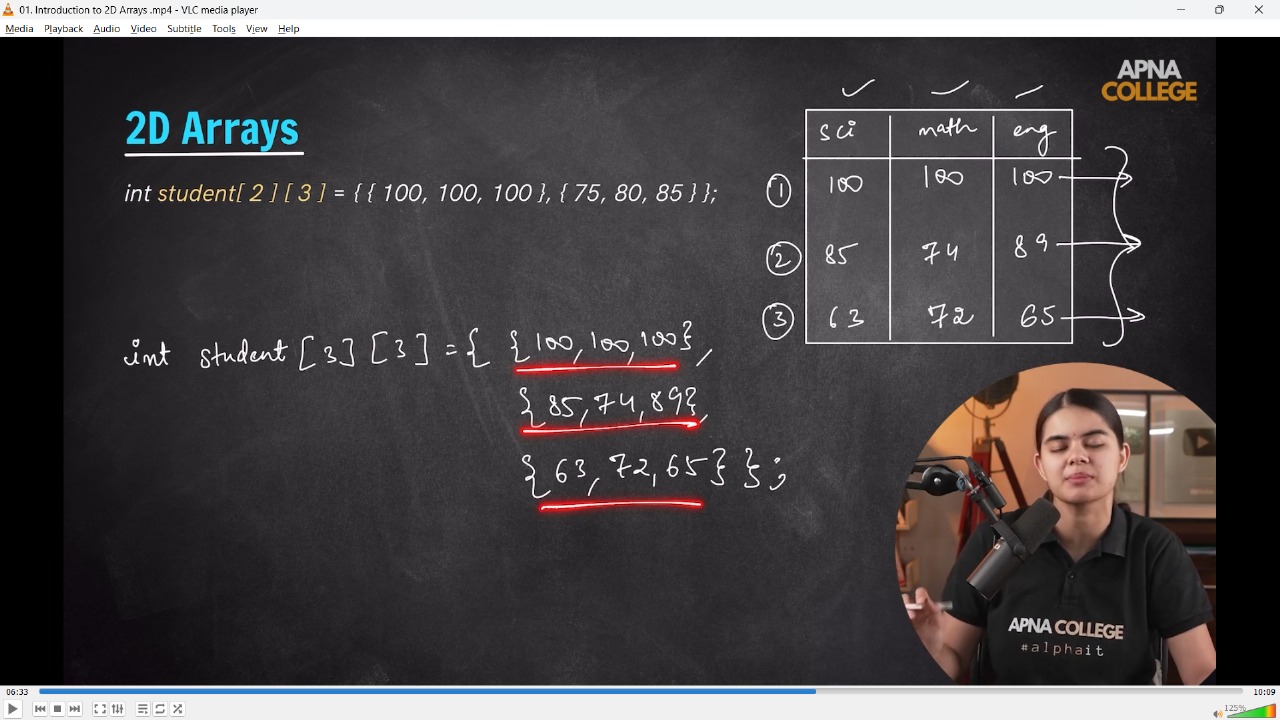
4.2) - Using Binary Search - 2nd Optimum - Row Wise - O(n\*logn), Colm Wise - O(m\*logn)

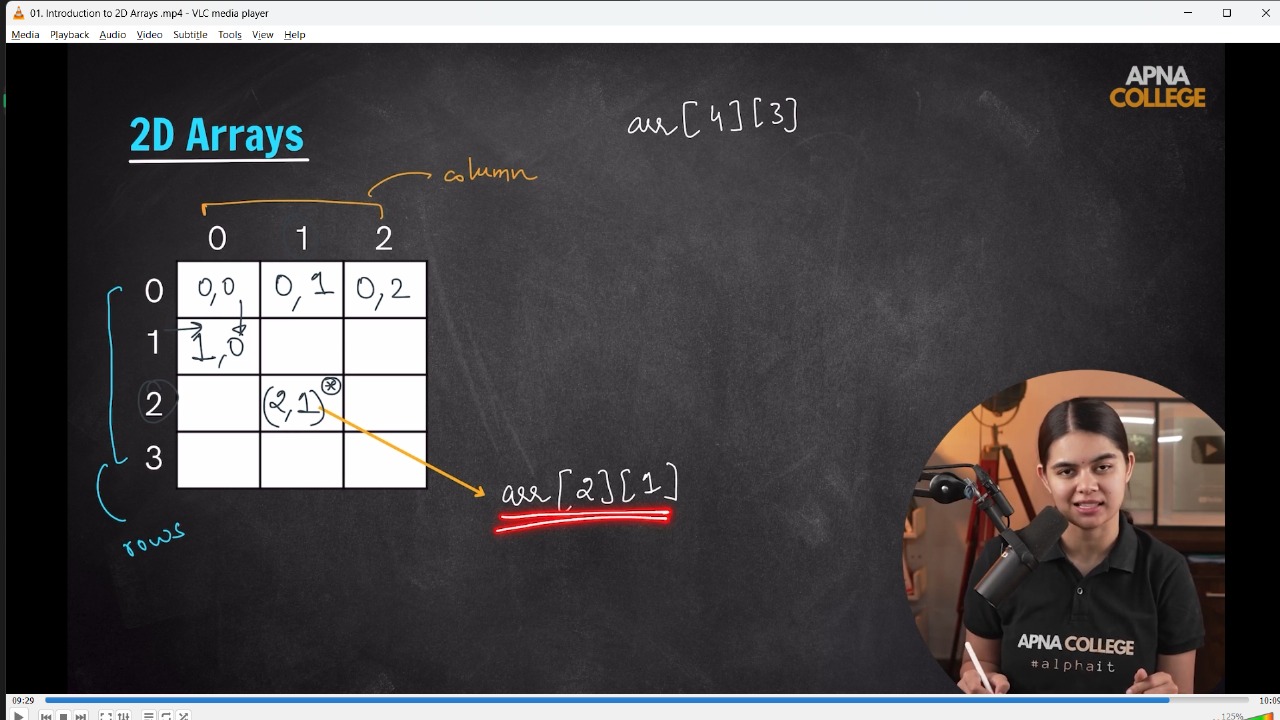
4.3) - Staircase Search technique - O(n+m)

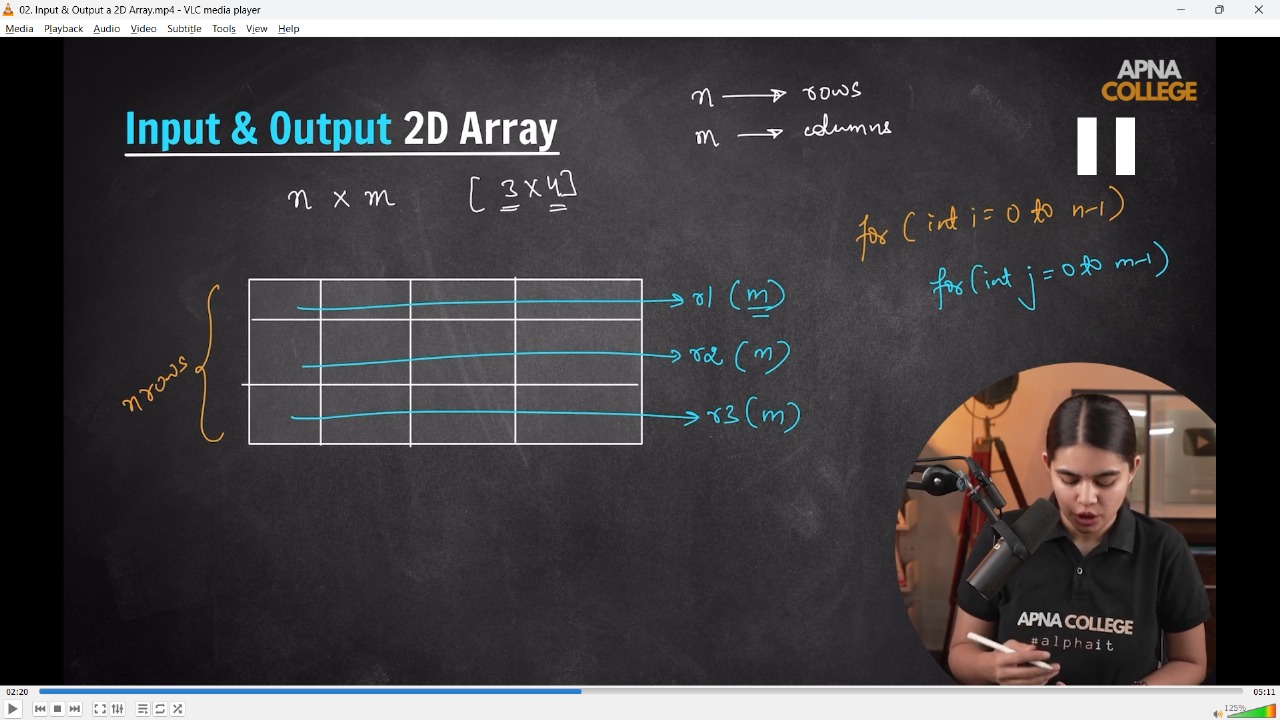
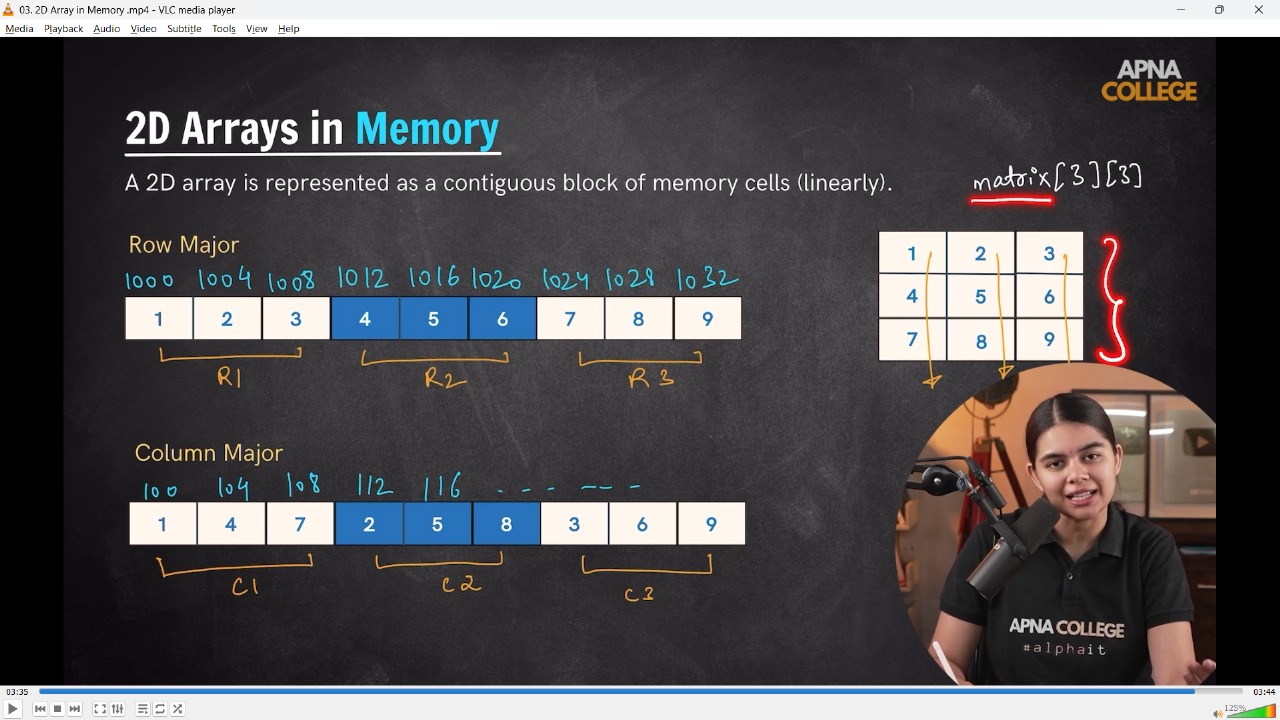
**Matrix in CPP –**

**1) 2D Array Basic Introduction & Input, Output**







// int main()

// {

//     int students[3][3] = {{100, 100, 100},

//                           {85, 74, 89},

//                           {63, 72, 65}};

//     cout << students[1][2] << endl; // 89

//     cout << students[2][0] << endl; // 63

//                                     // \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

//     int n, m;

//     cout << "no. of rows & colms are - " << endl;

//     cin >> n >> m;

//     int arr[n][m];

//     // Taking input & printing the values

//     cout << "Now, enter the array elements - " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             cin >> arr[i][j];

//         }

//     }

//     cout << "So, the entered 2D Array si - " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             cout << arr[i][j] << " ";

//         }

//         cout << endl;

//     }

// /\*

// no. of rows & colms are -

// 3 3

// Now, enter the array elements -

// 1 2 3 4 5 6 7 8 9

// So, the entered 2D Array si -

// 1 2 3

// 4 5 6

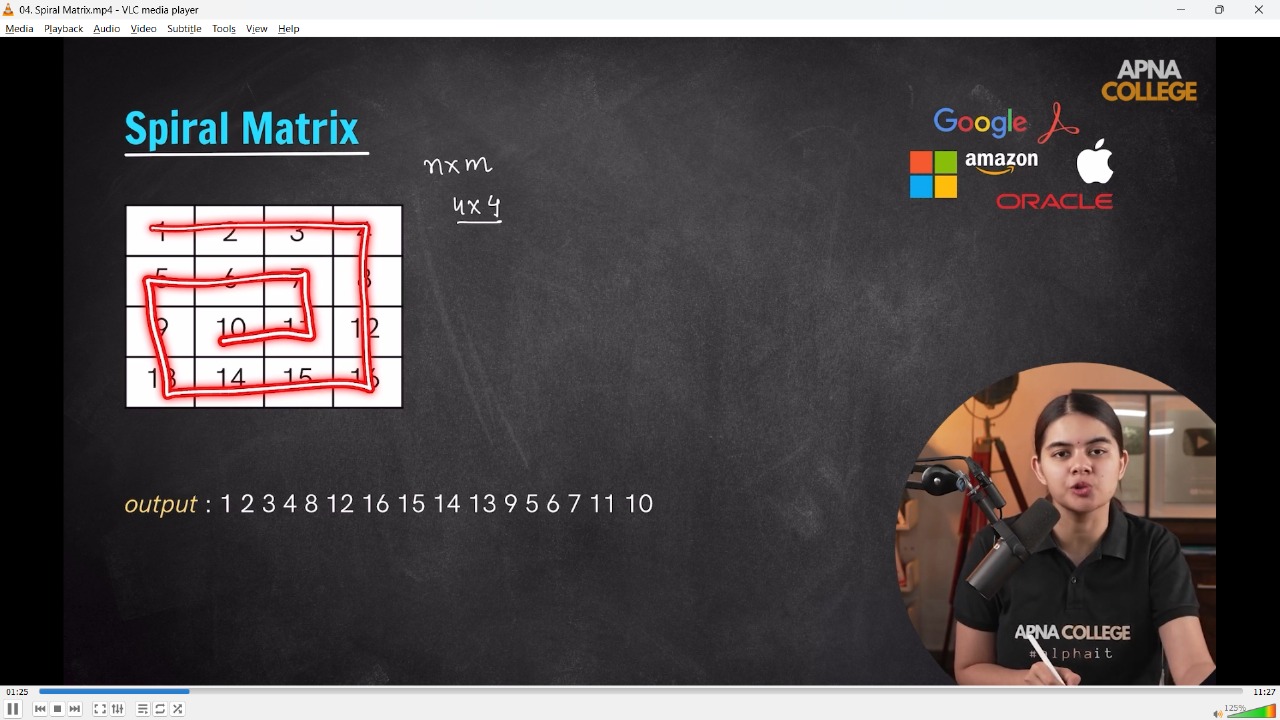
// 7 8 9

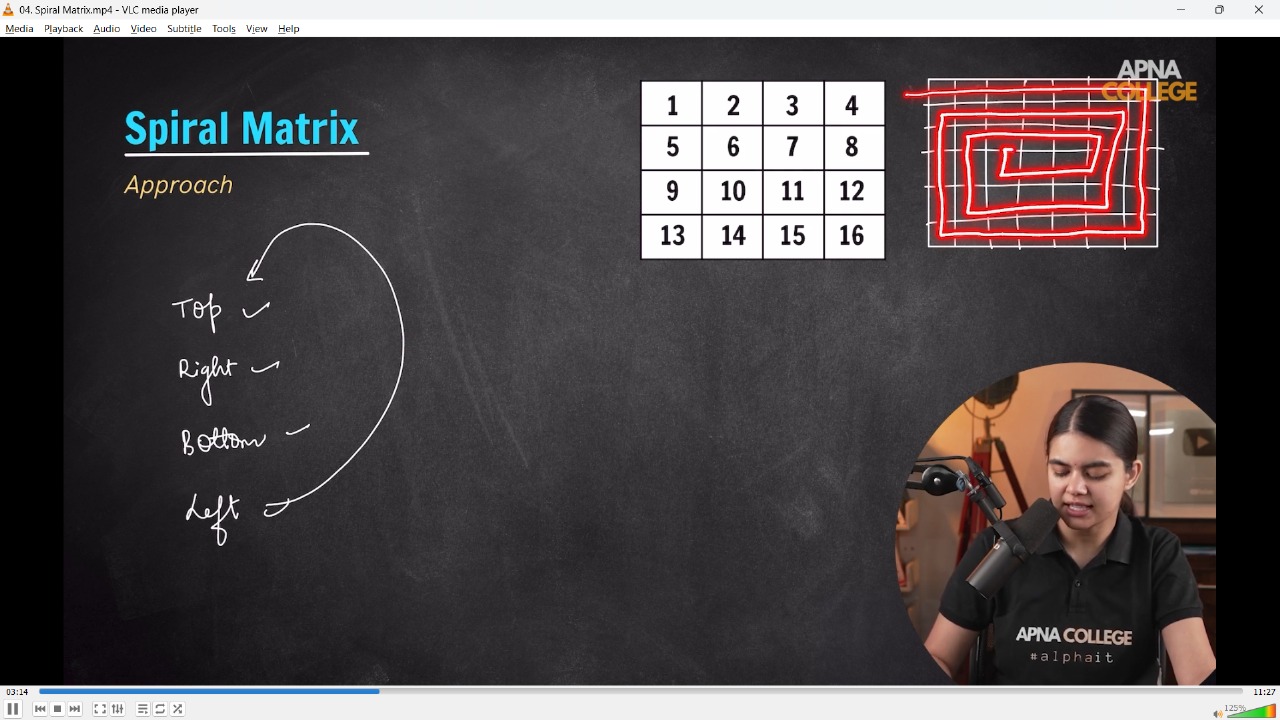
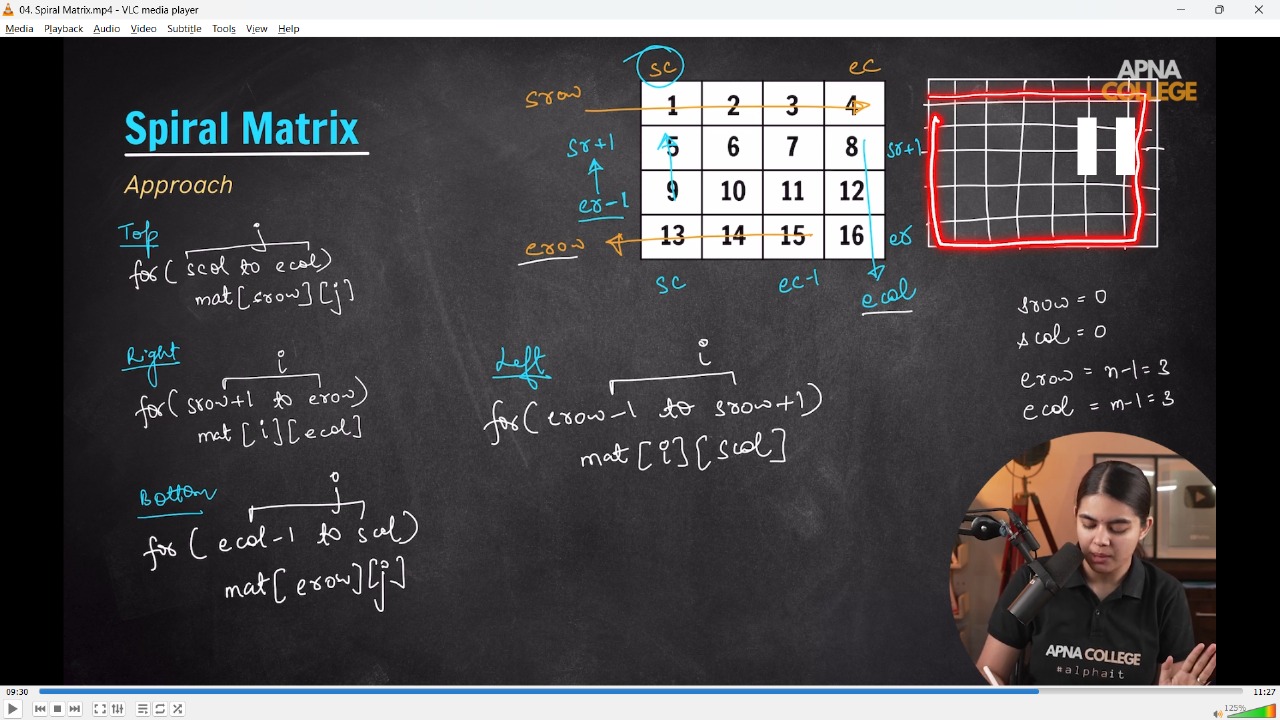
//  \*/

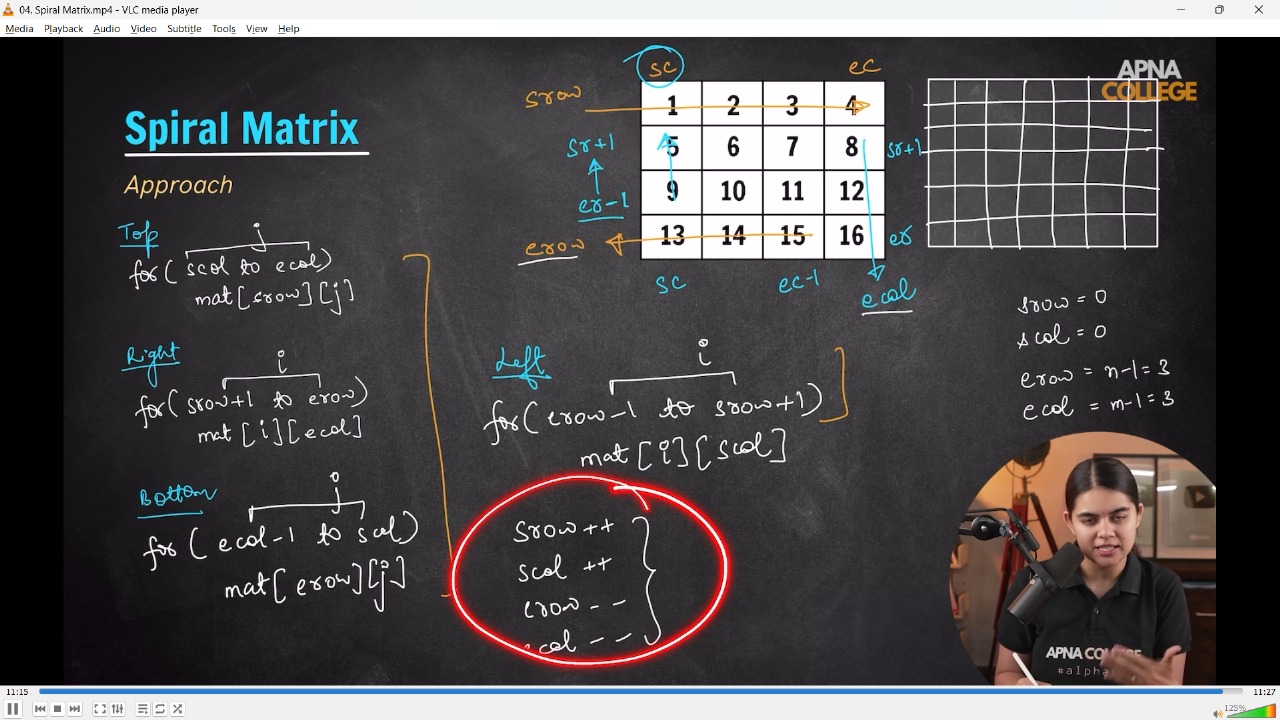
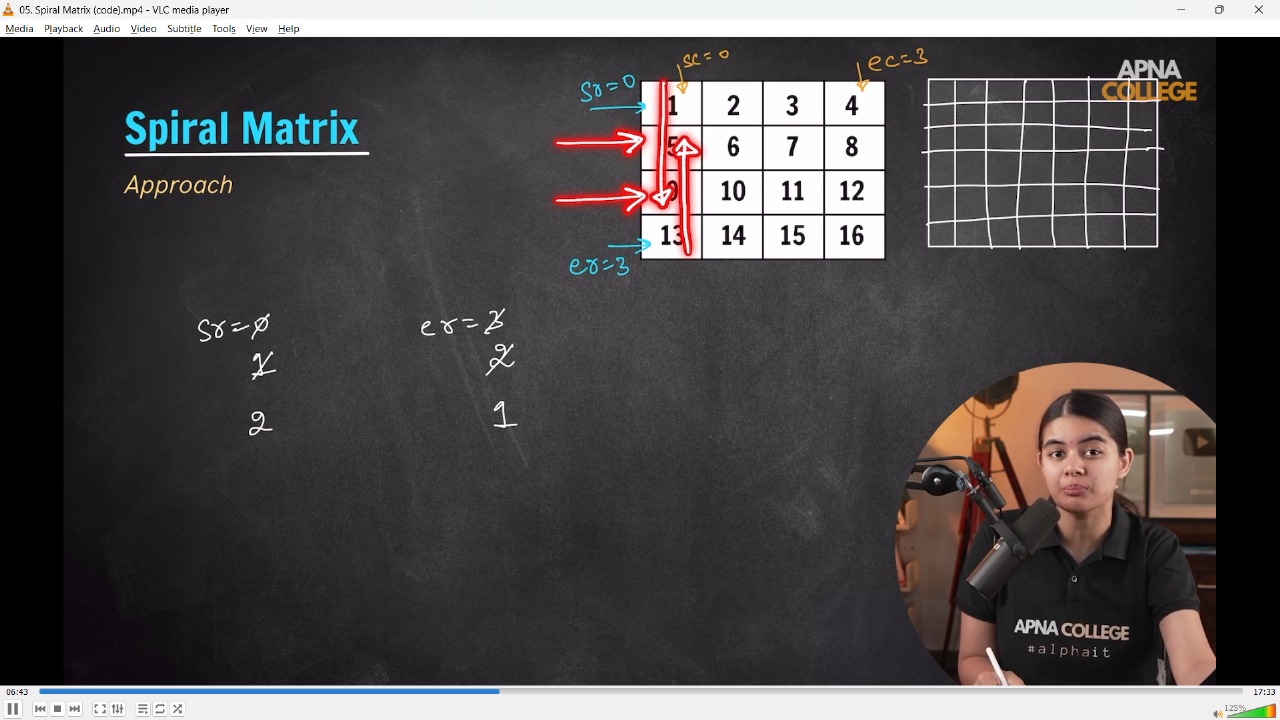
// }

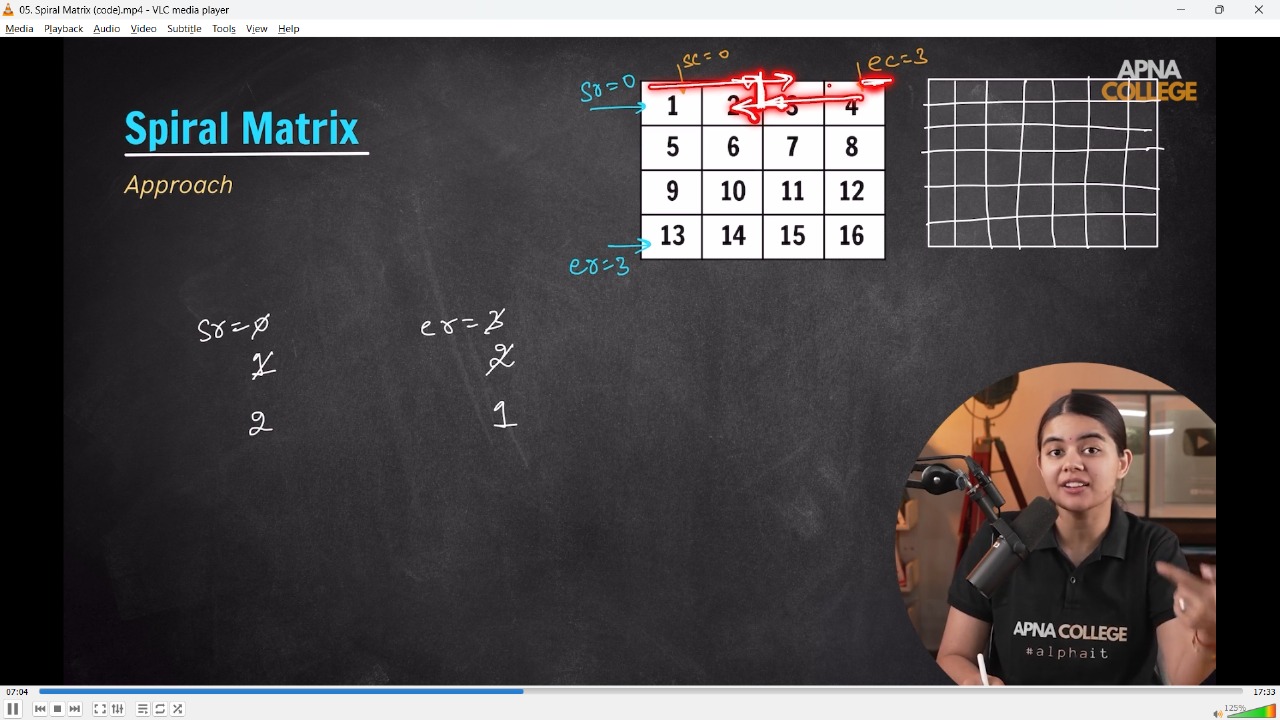
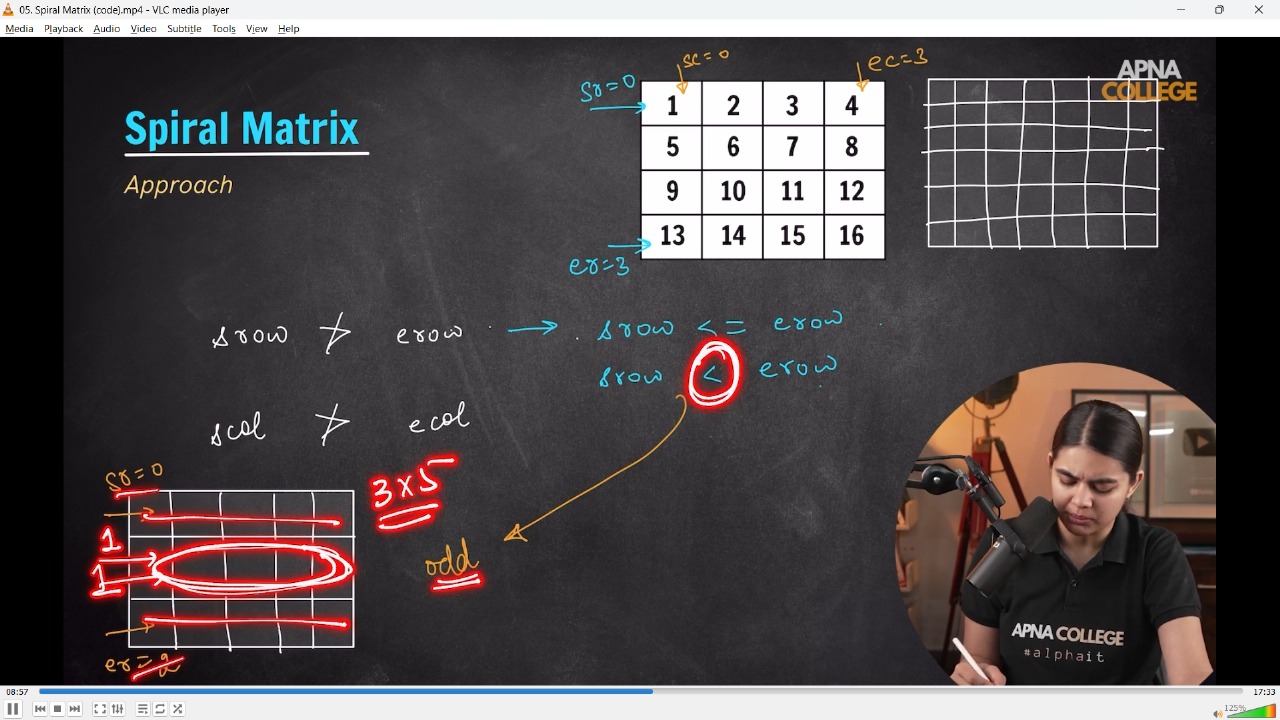
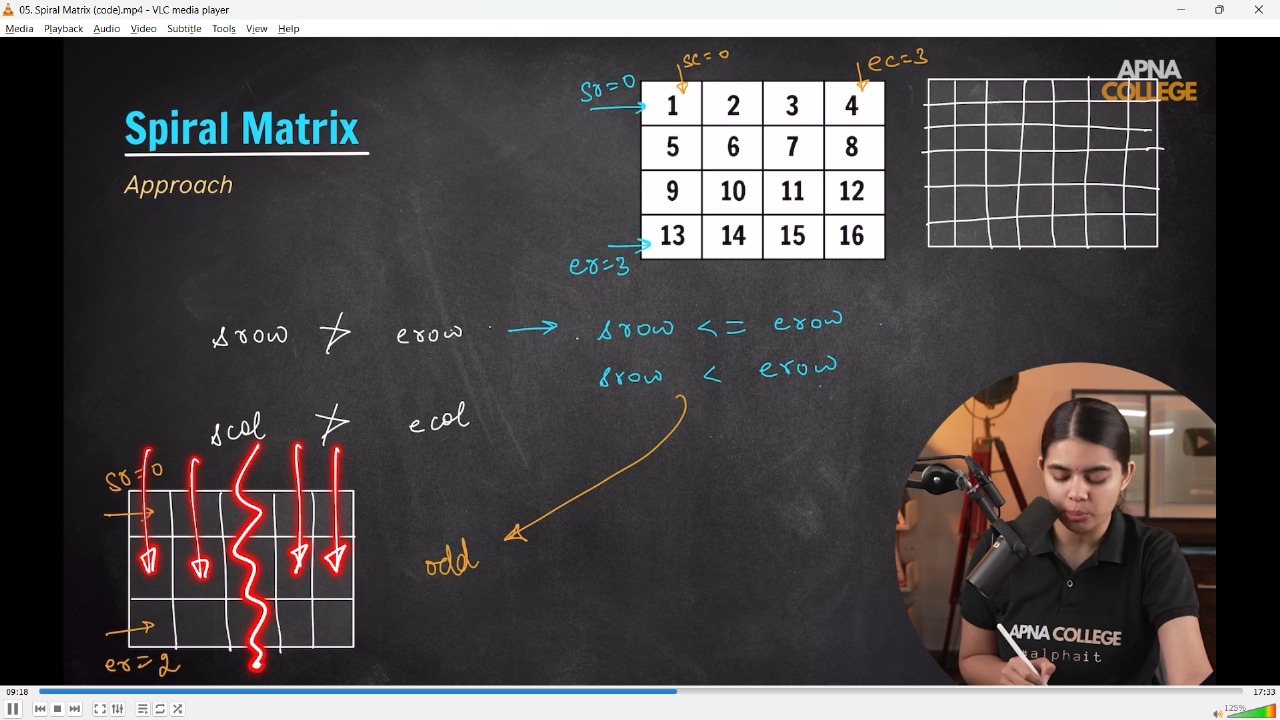
// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**2) Spiral matrix -**



//  void spiralMatrix(int mat[100][100], int n, int m)

//  {

//      int srow = 0, scol = 0;

//      int erow = n - 1, ecol = m - 1;

//     while (srow <= erow && scol <= ecol) // Odd Matrx -  jo value chhoti he vo decide kregi ki loop kitni baar run hoga. taaki kuchh rh na jaayee

//     {

//         // Top

//         for (int j = scol; j <= ecol; j++)

//         {

//             cout << mat[srow][j] << " ";

//         }

//         // Right

//         for (int i = srow + 1; i <= erow; i++)

//         {

//             cout << mat[i][ecol] << " ";

//         }

//         // Bottom

//         for (int j = ecol - 1; j >= scol; j--)

//         {

//             if (srow == erow) // Middle waali condition(row). yadi middle line printed already then joneed to print it aggain

//             {

//                 break;

//             }

//             cout << mat[erow][j] << " ";

//         }

//         // Left

//         for (int i = erow - 1; i > srow; i--)

//         {

//             if (srow == erow) // Middle waali condition(col). yadi middle line printed already then joneed to print it aggain

//             {

//                 break;

//             }

//             cout << mat[i][scol] << " ";

//         }

//         srow++, erow--;

//         scol++, ecol--;

//     }

//     cout << endl;

// }

// int main()

// {

//     int n, m;

//     cout << "no. of rows & colms are - " << endl;

//     cin >> n >> m;

//     int mat[100][100];

//     // Taking input & printing the values

//     cout << "Now, enter the array elements - " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             cin >> mat[i][j];

//         }

//     }

//     cout << "So, the entered 2D Array si - " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             cout << mat[i][j] << " ";

//         }

//         cout << endl;

//     }

//     cout << "Now, the spiral matrix moment for this array is - " << endl;

//     spiralMatrix(mat, n, m);

//     /\*

//     no. of rows & colms are -

//     4 4

//     Now, enter the array elements -

//     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

//     So, the entered 2D Array si -

//     1 2 3 4

//     5 6 7 8

//     9 10 11 12

//     13 14 15 16

//     Now, the spiral matrix moment for this array is -

//     1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10

//     no. of rows & colms are -

//     5 5

//     Now, enter the array elements -

//     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25

//     So, the entered 2D Array si -

//     1 2 3 4 5

//     6 7 8 9 10

//     11 12 13 14 15

//     16 17 18 19 20

//     21 22 23 24 25

//     Now, the spiral matrix moment for this array is -

//     1 2 3 4 5 10 15 20 25 24 23 22 21 16 11 6 7 8 9 14 19 18 17 12 13

// no. of rows & colms are -

// 3 4

// Now, enter the array elements -

// 1 2 3 4 5 6 7 8 9 10 11 12

// So, the entered 2D Array si -

// 1 2 3 4

// 5 6 7 8

// 9 10 11 12

// Now, the spiral matrix moment for this array is -

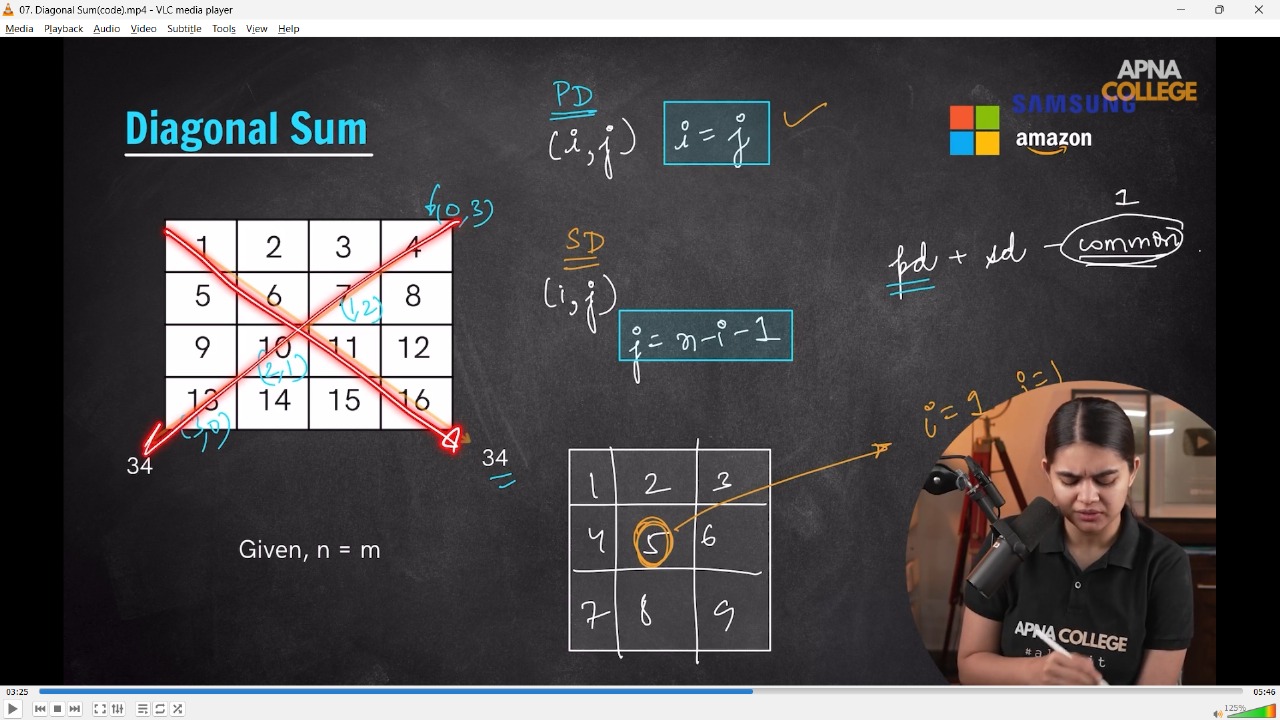
// 1 2 3 4 8 12 11 10 9 5 6 7

//      \*/

// }

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**3) Diagonal Sum of Matrix -**



// void diagonalSum(int mat[100][100], int n, int m)

// {

//     int SumofDiagonal1 = 0;

//     int SumofDiagonal2 = 0;

//     int totalSum = 0;

//     int totalDiff = 0;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             if (i == j)

//             {

//                 SumofDiagonal1 += mat[i][j];

//             }

//             if (i + j == n - 1)

//             {

//                 SumofDiagonal2 += mat[i][j];

//             }

//         }

//         totalSum = SumofDiagonal1 + SumofDiagonal2;

//         totalDiff = SumofDiagonal1 - SumofDiagonal2;

//     }

//     cout << "So, the diagonalSum is - " << totalSum << endl;

//     cout << "and simialrly, the diagonaldiff is - " << totalDiff << endl;

//     /\* For avoiding complexity from O(n^2)to O(n)

//     for(int i=0; i<n; i++)

//     {

//     sum+=mat[i][i];

//     if(i!=n-i-1)

//     {

//     sum+=mat[i][n-i-1];

//     }

//     }

//      \*/

// }

// int main()

// {

//     int n, m;

//     cout << "no. of rows & colms are - " << endl;

//     cin >> n >> m;

//     int mat[100][100];

//     // Taking input & printing the values

//     cout << "Now, enter the array elements - " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             cin >> mat[i][j];

//         }

//     }

//     cout << "So, the entered 2D Array si - " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             cout << mat[i][j] << " ";

//             // }

//             cout << endl;

//         }

//         diagonalSum(mat, n, m);

//     }

// /\*

// no. of rows & colms are -

// 4 4

// Now, enter the array elements -

// 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

// So, the entered 2D Array si -

// 1 2 3 4

// 5 6 7 8

// 9 10 11 12

// 13 14 15 16

// So, the diagonalSum is - 68

// and simialrly, the diagonaldiff is - 0

// no. of rows & colms are -

// 3 3

// Now, enter the array elements -

// 1 2 3 4 5 6 7 8 9

// So, the entered 2D Array si -

// 1 2 3

// 4 5 6

// 7 8 9

// So, the diagonalSum is - 30

// and simialrly, the diagonaldiff is - 0

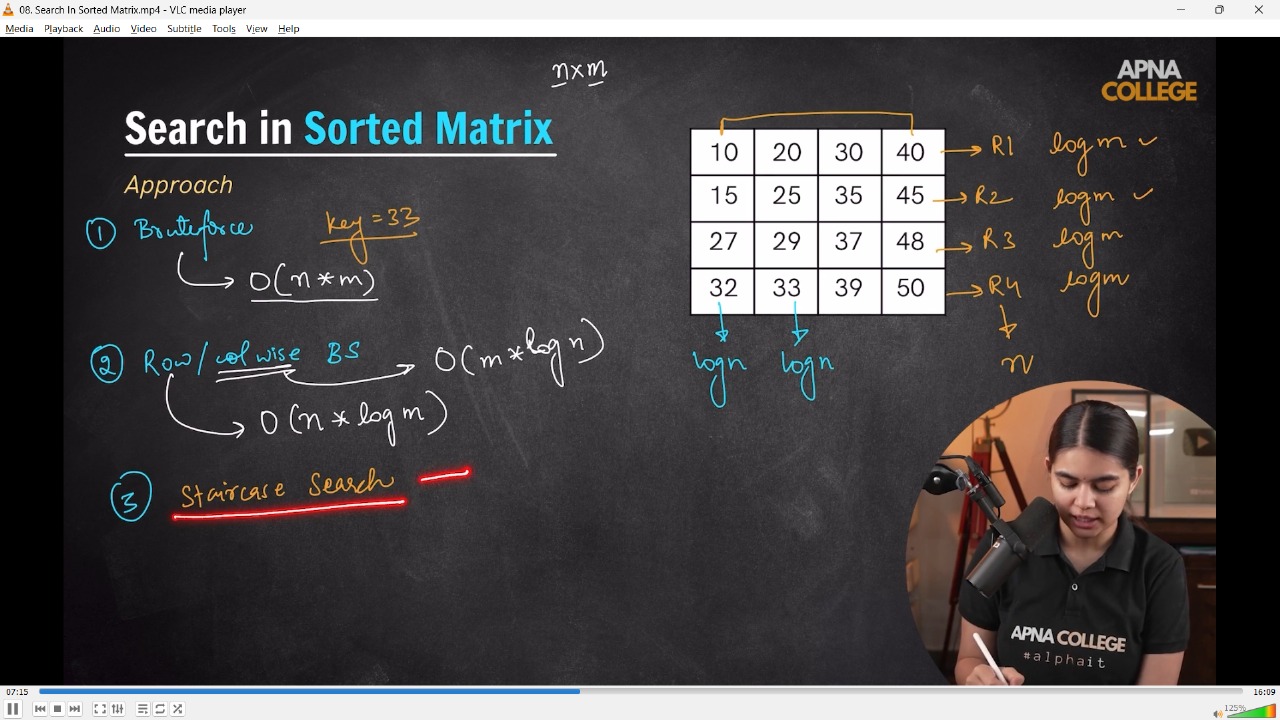
// T.C  - O(n^2)

//  \*/

// }

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**4) Search for targeted eleemnt in Sorted Matrix -**



// **4.1) - Brute Force Approch - most simple method. T.C - O(n\*m)**

// int targetElementinMatrix(int mat[100][100], int n, int m, int target)

// {

//     // Printing the sum of entered matrix -

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             if (mat[i][j] == target)

//             {

//                 cout << "Yes it's present and the address is: row no-  " << i << " & column no-  " << j << " having targeted value-  " << mat[i][j];

//                 return 0;

//             }

//         }

//     }

//     cout << "Sorry, Element Not FOund Bruh.." << endl;

// }

// int main()

// {

//     int n, m;

//     cout << "no. of rows & colms are - " << endl;

//     cin >> n >> m;

//     int mat[100][100];

//     // Taking input & printing the values

//     cout << "Now, enter the array elements - " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             cin >> mat[i][j];

//         }

//     }

//     cout << "So, the entered 2D Array si - " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             cout << mat[i][j] << " ";

//         }

//         cout << endl;

//     }

//     int target;

//     cout << "So, what's the targeted element - " << endl;

//     cin >> target;

//     targetElementinMatrix(mat,n,m,target);

// /\*

// no. of rows & colms are -

// 3 3

// Now, enter the array elements -

// 1 2 3 4 5 6 7 8 9

// So, the entered 2D Array si -

// 1 2 3

// 4 5 6

// 7 8 9

// So, what's the targeted element -

// 6

// Yes it's present and the address is: row no-  1 & column no-  2 having targeted value-  6

//  \*/

// }

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**// 4.2) - USing Binary Search - 2nd Optimum - Row Wise - O(n\*logn), Colm Wise - O(m\*logn)**

**// rOW major approach for seareching targeted element in the sorted matrix -**

// bool binarySearchInMatrix(int mat[100][100], int n, int m, int target)

// {

//     int start = 0;

//     int end = n \* m - 1;

//     while (start <= end)

//     {

//         int mid = start + (end - start) / 2;

//         int row = mid / m;

//         int col = mid % m;

//         if (mat[row][col] == target)

//         {

//             cout << "Yes, it's present at row " << row << " & column " << col << " having targeted value - " << mat[row][col] << endl;

//             return true;

//         }

//         else if (mat[row][col] < target)

//         {

//             start = mid + 1;

//         }

//         else

//         {

//             end = mid - 1;

//         }

//     }

//     cout << "Sorry, Element Not Found Bruh.." << endl;

//     return false;

// }

// int main()

// {

//     int n, m;

//     cout << "No. of rows & columns are - " << endl;

//     cin >> n >> m;

//     int mat[100][100];

//     cout << "Now, enter the array elements - " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             cin >> mat[i][j];

//         }

//     }

//     cout << "So, the entered 2D Array is - " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             cout << mat[i][j] << " ";

//         }

//         cout << endl;

//     }

//     int target;

//     cout << "So, what's the targeted element - " << endl;

//     cin >> target;

//     binarySearchInMatrix(mat, n, m, target);

// /\*

// No. of rows & columns are -

// 3 3

// Now, enter the array elements -

// 1 2 3 4 5 6 7 8 9

// So, the entered 2D Array is -

// 1 2 3

// 4 5 6

// 7 8 9

// So, what's the targeted element -

// 8

// Yes, it's present at row 2 & column 1 having targeted value - 8

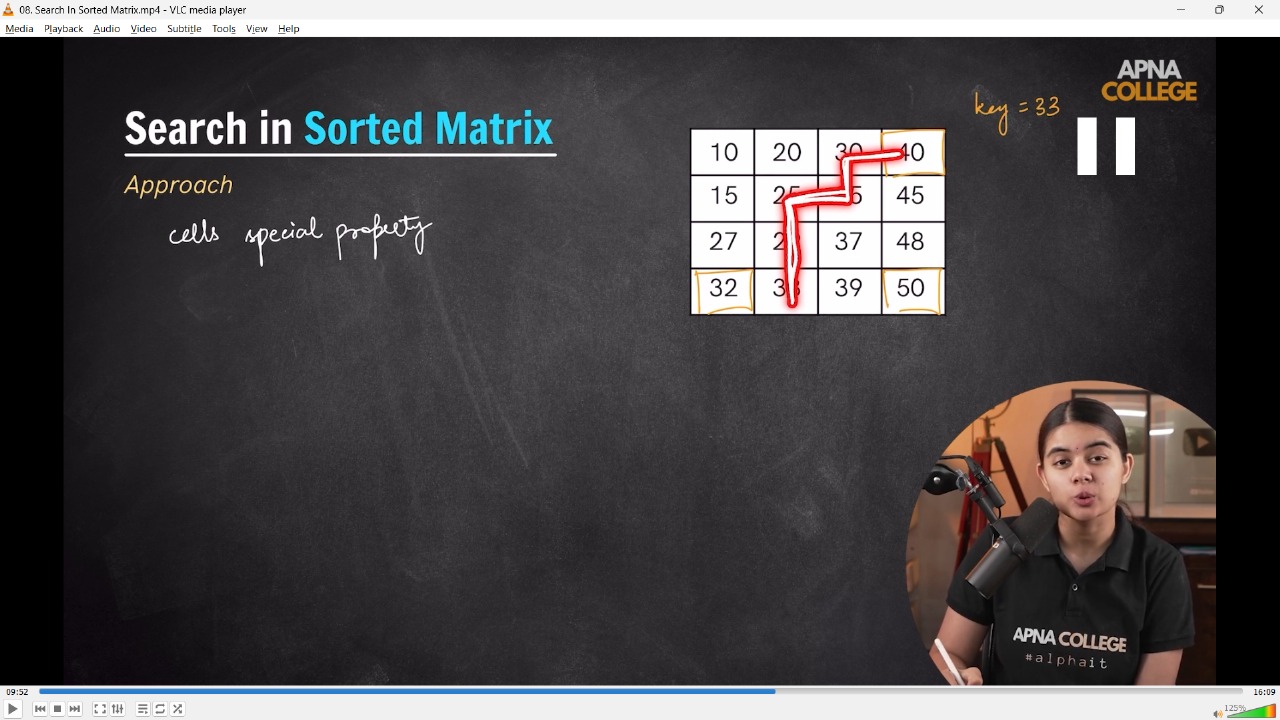
//  \*/

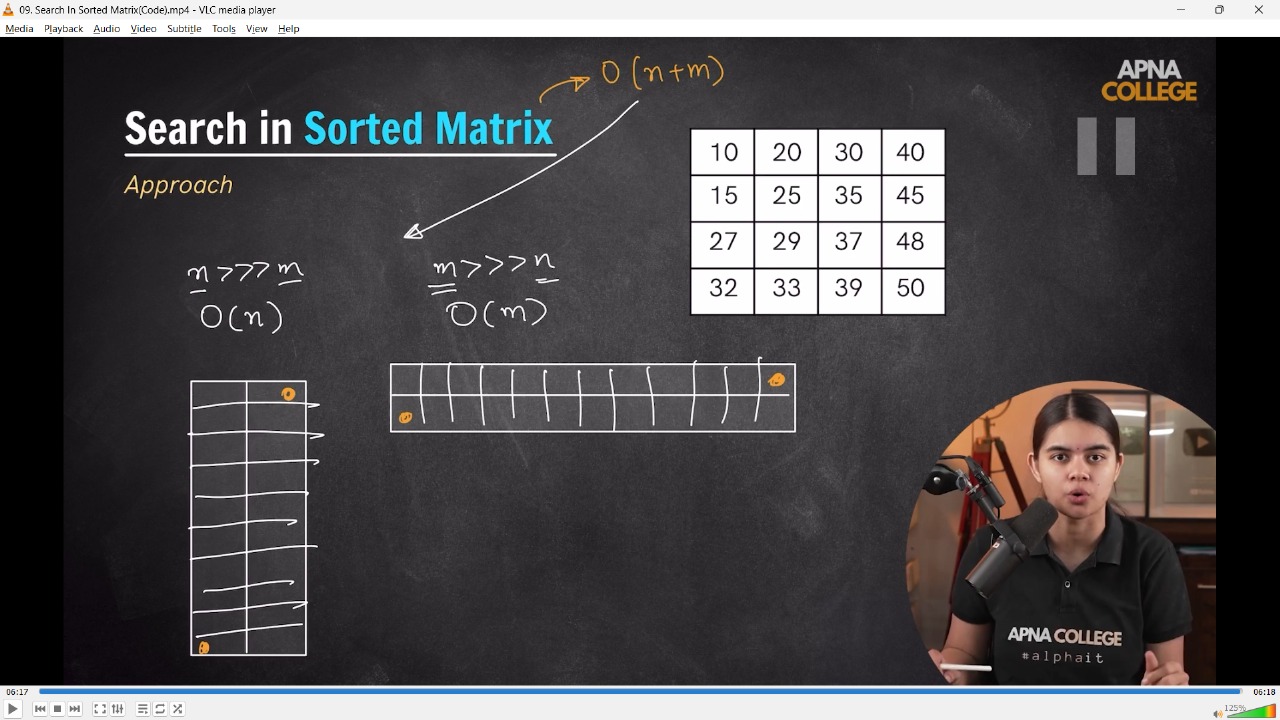
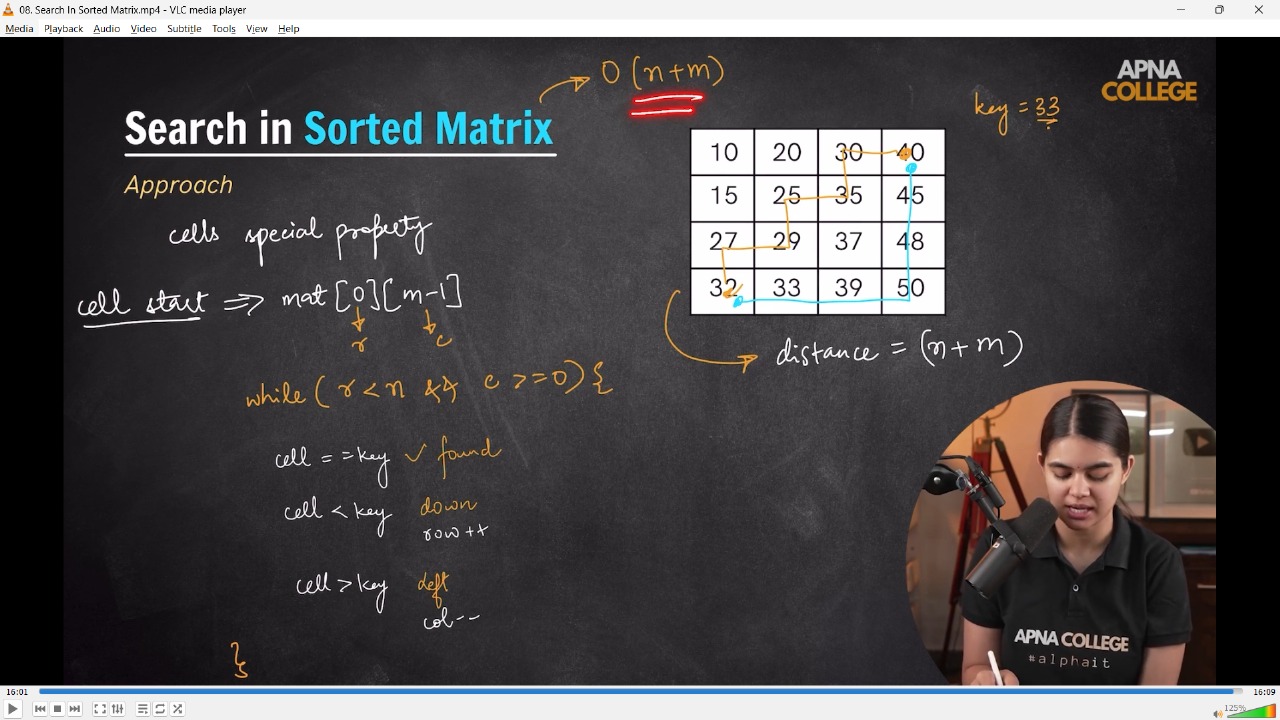
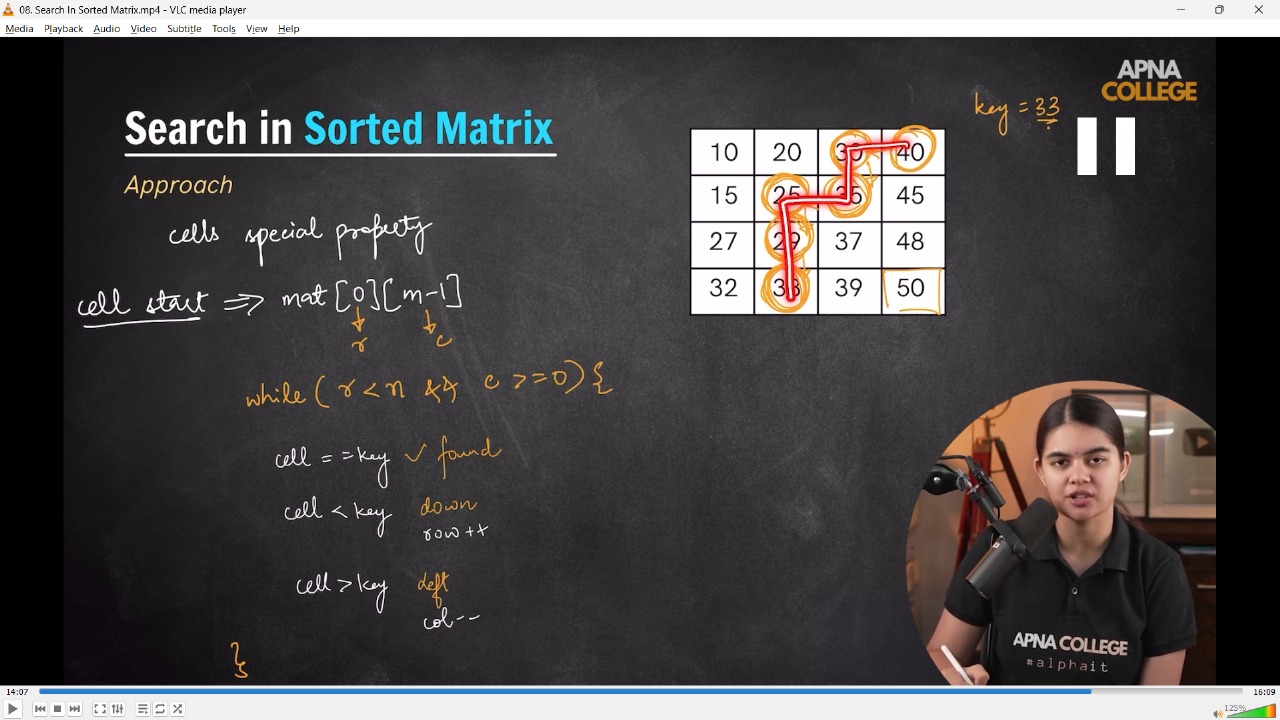
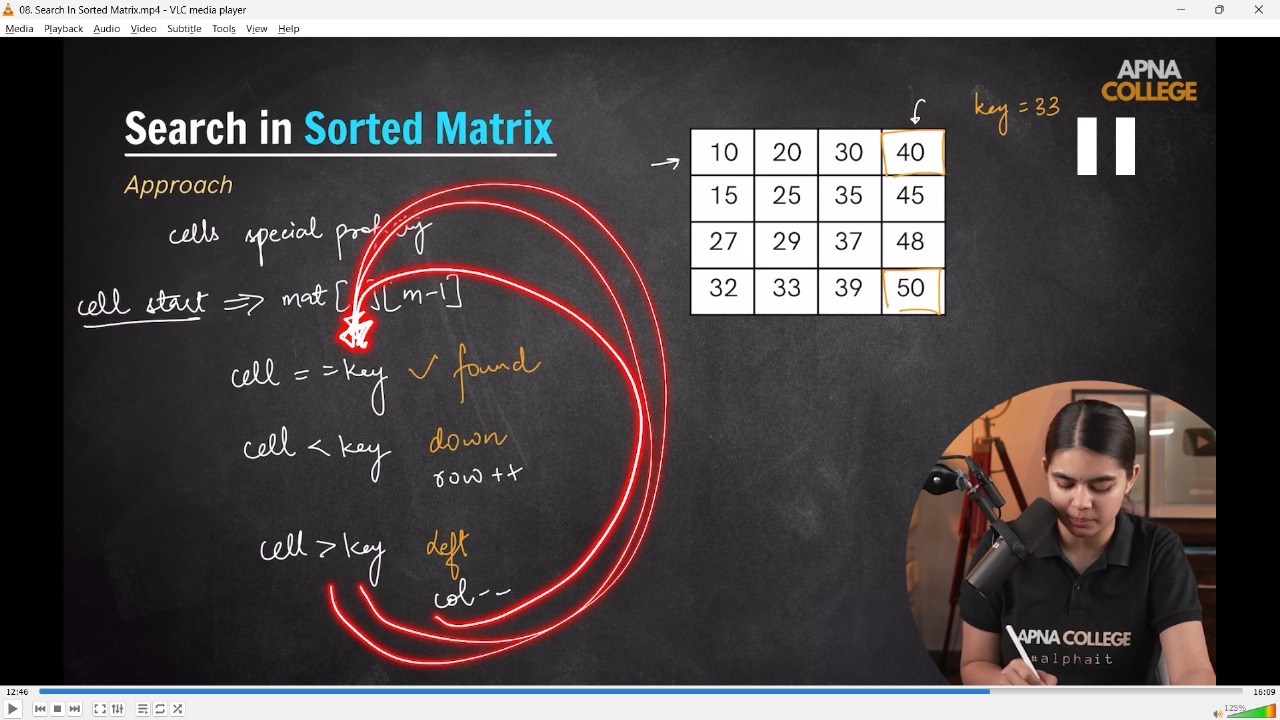
// }

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**// 4.3) - Staircase Search technique - O(n+m)**

**// Case 1 - when considering the Top Right Most**





// Case 1 - when considering the Top Right Most

// bool searchStaircase(int mat[100][100], int n, int m, int key)

// {

//     int i = 0, j = m - 1;

//     while (i < n && j >= 0)

//     {

//         if (mat[i][j] == key)

//         {

//             cout << "found at cell (" << i << " , " <<  j  << ")" << endl;

//             return true;

//         }

//         else if (mat[i][j] > key)

//         {

//             // as the small value so in the left

//             j--;

//         }

//         else

//         {

//             // as greater value so in the right

//             i++;

//         }

//     }

//     cout << "key not found" << endl;

//     return false;

// }

// int main()

// {

//     int n, m;

//     cout << "no. of rows & colms are - " << endl;

//     cin >> n >> m;

//     int mat[100][100];

//     // Taking input & printing the values

//     cout << "Now, enter the array elements - " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             cin >> mat[i][j];

//         }

//     }

//     cout << "So, the entered 2D Array si - " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             cout << mat[i][j] << " ";

//         }

//         cout << endl;

//     }

//     int key;

//     cout << "Mention. the value you want to search for - " << endl;

//     cin >> key;

//     searchStaircase(mat, n, m, key);

// /\*

// no. of rows & colms are -

// 4 4

// Now, enter the array elements -

// 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

// So, the entered 2D Array si -

// 1 2 3 4

// 5 6 7 8

// 9 10 11 12

// 13 14 15 16

// Mention. the value you want to search for -

// 12

// found at cell (2 , 3)

// T.C - O(n+m)

//  \*/

// }

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

// Case 2 - when cosidering the Down Left Corner -

// bool searchStaircase(int mat[100][100], int n, int m, int key)

// {

//     int i = n - 1, j = 0;

//     while (i >= 0 && j < m)

//     {

//         if (mat[i][j] == key)

//         {

//             cout << "found at cell (" << i << " , " << j << ")" << endl;

//             return true;

//         }

//         else if (mat[i][j] > key)

//         {

//             // as the small value so in the left

//             i--;

//         }

//         else

//         {

//             // as greater value so in the right

//             j++;

//         }

//     }

//     cout << "key not found" << endl;

//     return false;

// }

// int main()

// {

//     int n, m;

//     cout << "no. of rows & colms are - " << endl;

//     cin >> n >> m;

//     int mat[100][100];

//     // Taking input & printing the values

//     cout << "Now, enter the array elements - " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             cin >> mat[i][j];

//         }

//     }

//     cout << "So, the entered 2D Array is - " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         for (int j = 0; j < m; j++)

//         {

//             cout << mat[i][j] << " ";

//         }

//         cout << endl;

//     }

//     int key;

//     cout << "Mention. the value you want to search for - " << endl;

//     cin >> key;

//     searchStaircase(mat, n, m, key);

//     return 0;

//     /\*

//     no. of rows & colms are -

//     4

//     4

//     Now, enter the array elements -

//     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

//     So, the entered 2D Array is -

//     1 2 3 4

//     5 6 7 8

//     9 10 11 12

//     13 14 15 16

//     Mention. the value you want to search for -

//     12

//     found at cell (2 , 3)

//     no. of rows & colms are -

//     4 4

//     Now, enter the array elements -

//     1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

//     So, the entered 2D Array is -

//     1 2 3 4

//     5 6 7 8

//     9 10 11 12

//     13 14 15 16

//     Mention. the value you want to search for -

//     75

//     key not found

//     // T.C - O(n+m)

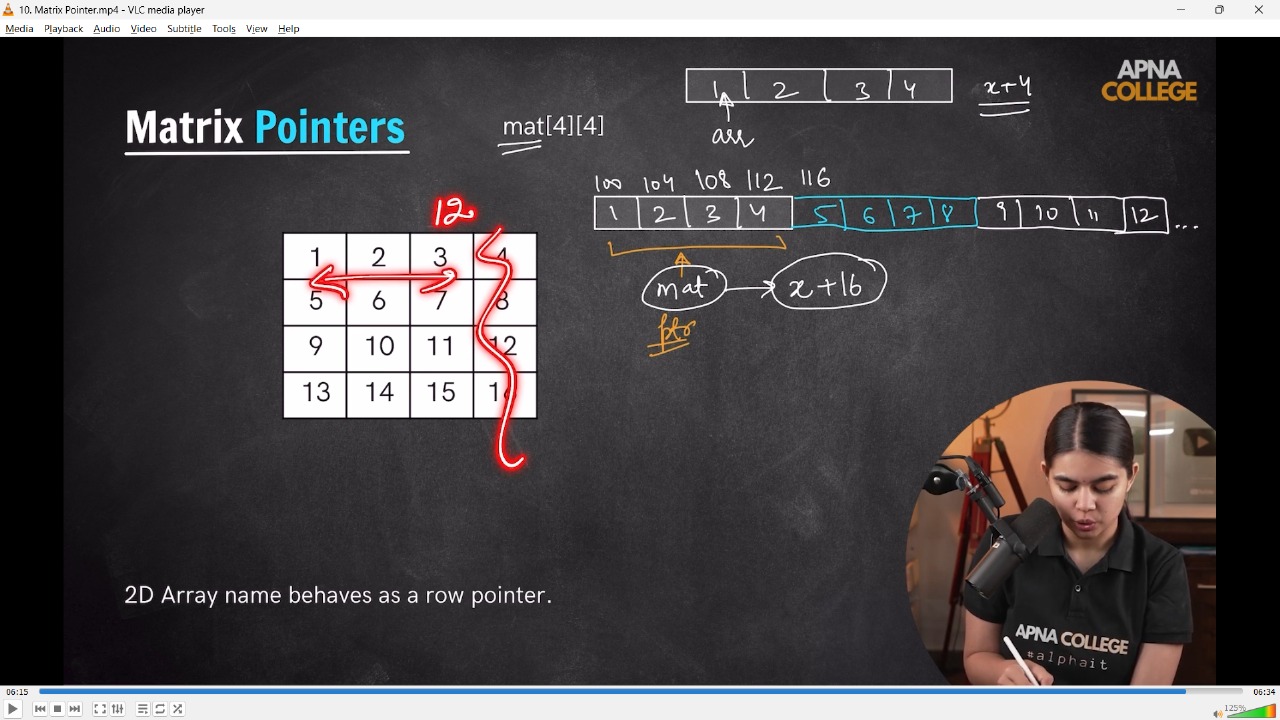
//     Special Cases -  for TC. When  n>>>m then TC will bE o(n) as row dominating and simlarly when m>>>n then TC eill be O(m) as colms dominating

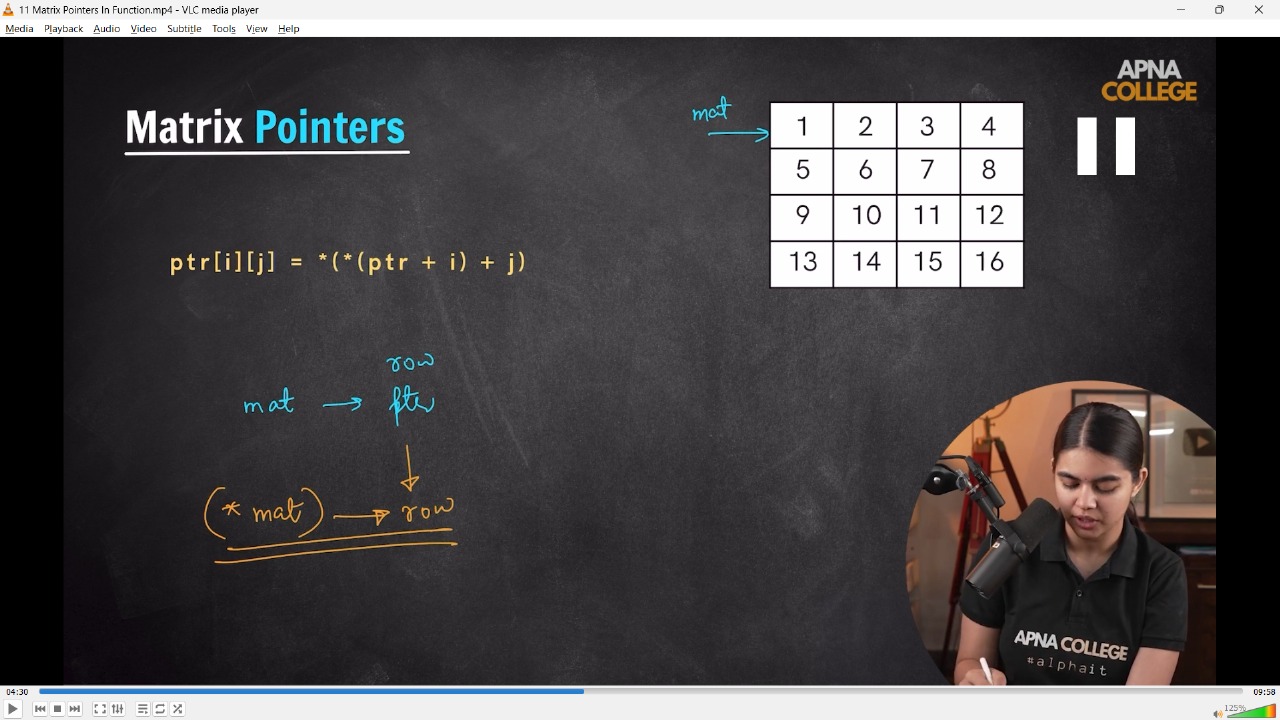
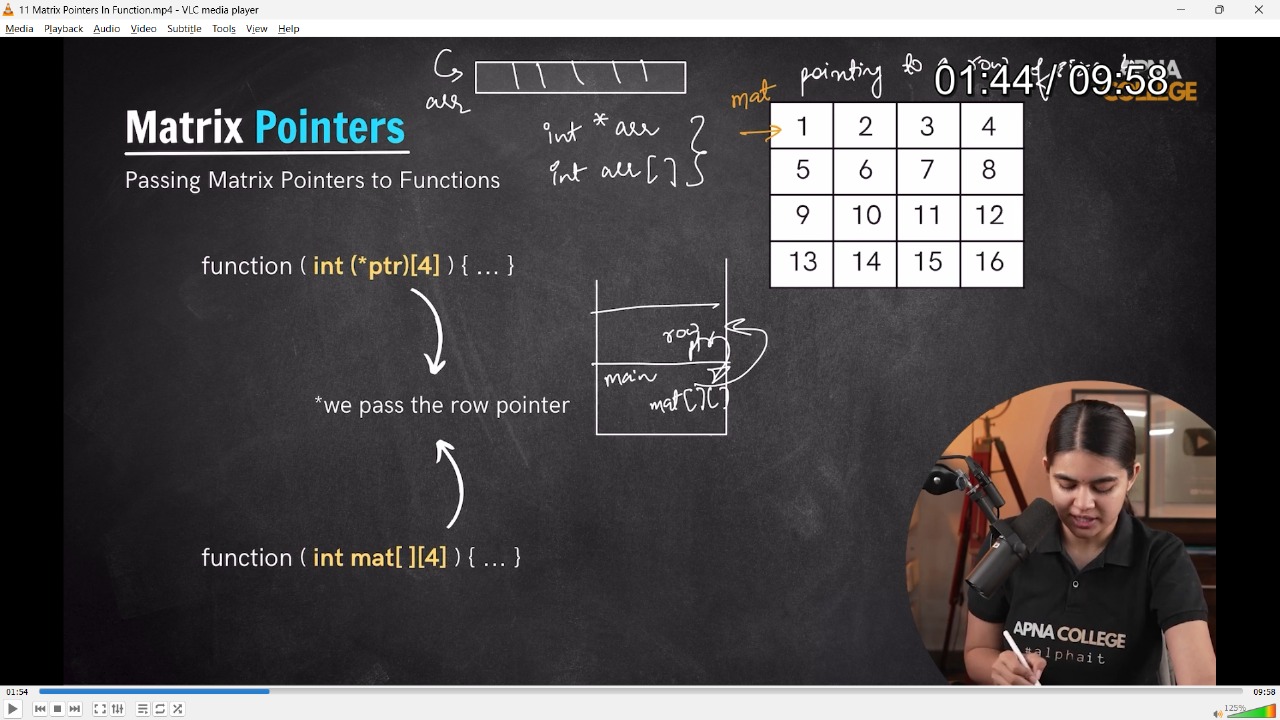
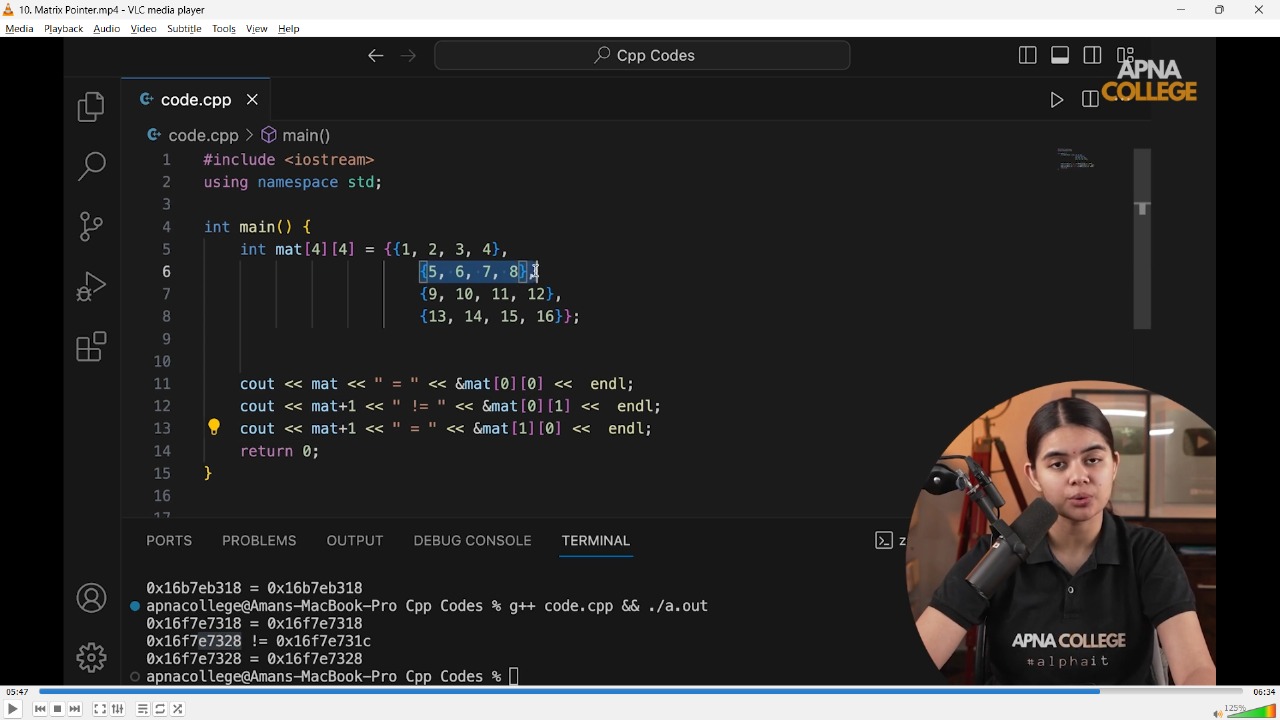
//      \*/

// }

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**//5)Important pointer concept for matrix –**





// void func(int mat[][4], int n, int m)

// {

//     cout << "0th row ptr" << mat << endl;//0th row ptr0x61fee0

//     cout << "1st row ptr" << mat + 1 << endl;//1st row ptr0x61fef0

//     cout << "2nd row ptr" << mat + 2 << endl;//2nd row ptr0x61ff00

//     cout << endl;

//     cout << "0th row value is - " << \*mat << endl;//0th row value is - 0x61fee0

//     cout << "1st row value is - " << \*(mat + 1) << endl;//1st row value is - 0x61fef0

//     cout << "2nd row value is - " << \*(mat + 2) << endl;//2nd row value is - 0x61ff00

//     cout << endl;

//     cout << \*(\*(mat + 2) + 2) << endl;//11

// }

// void func2(int (\*mat)[4], int n, int m)

// {

// }

// int main()

// {

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

//     /\*

//     Above mentioned both the way of passing matrix into

//     user defined function are same. Any of the above can be used

//      \*/

//     func(mat, 4, 4);

//     /\*

//     0th row ptr0x61fee0

//     1st row ptr0x61fef0

//     2nd row ptr0x61ff00

//      \*/

// }

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_