**[10] Divide & Conquer –**

1) Merge Sort Algo -

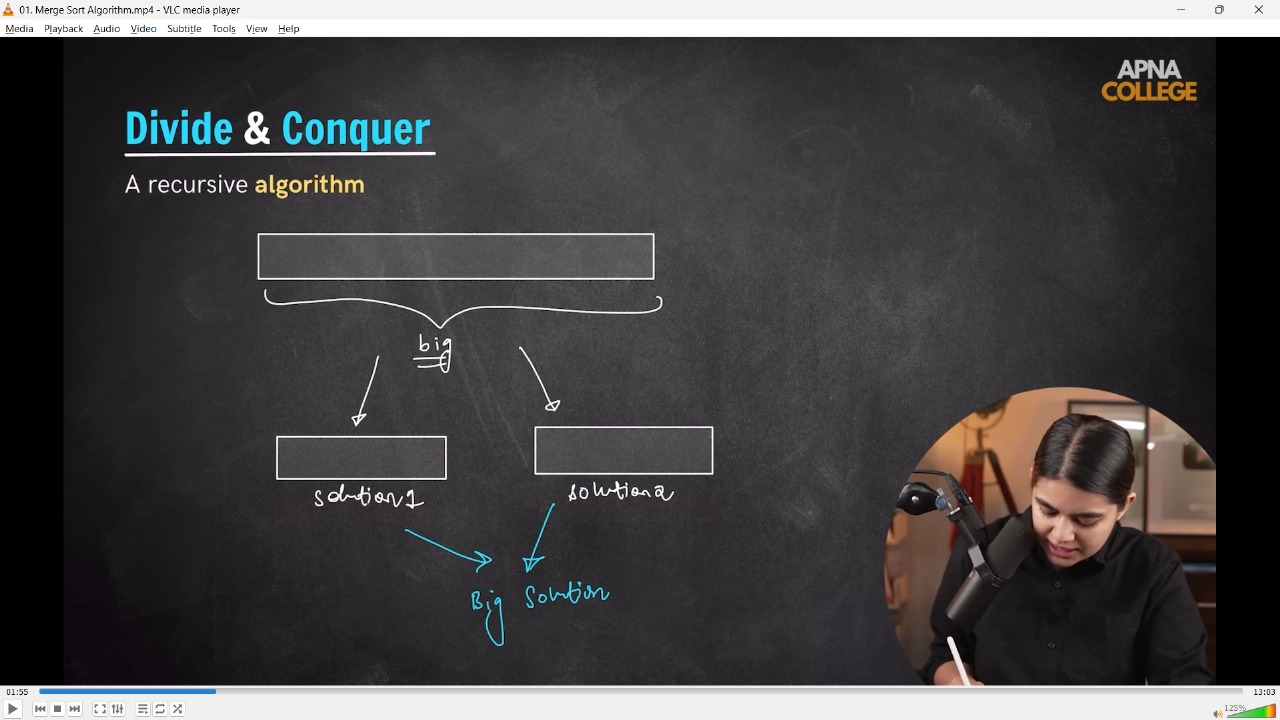
2) Quick Sort Algorithm -

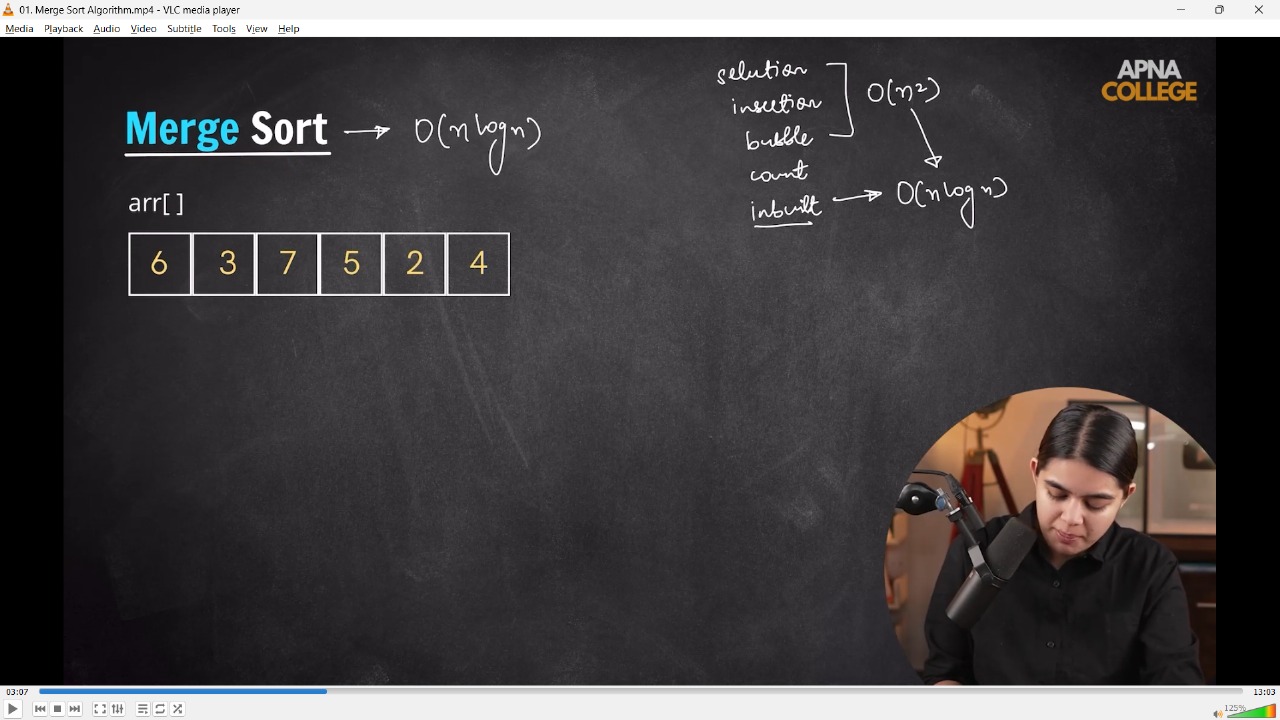
2.1) Quick Sort Worst Case Scenario -

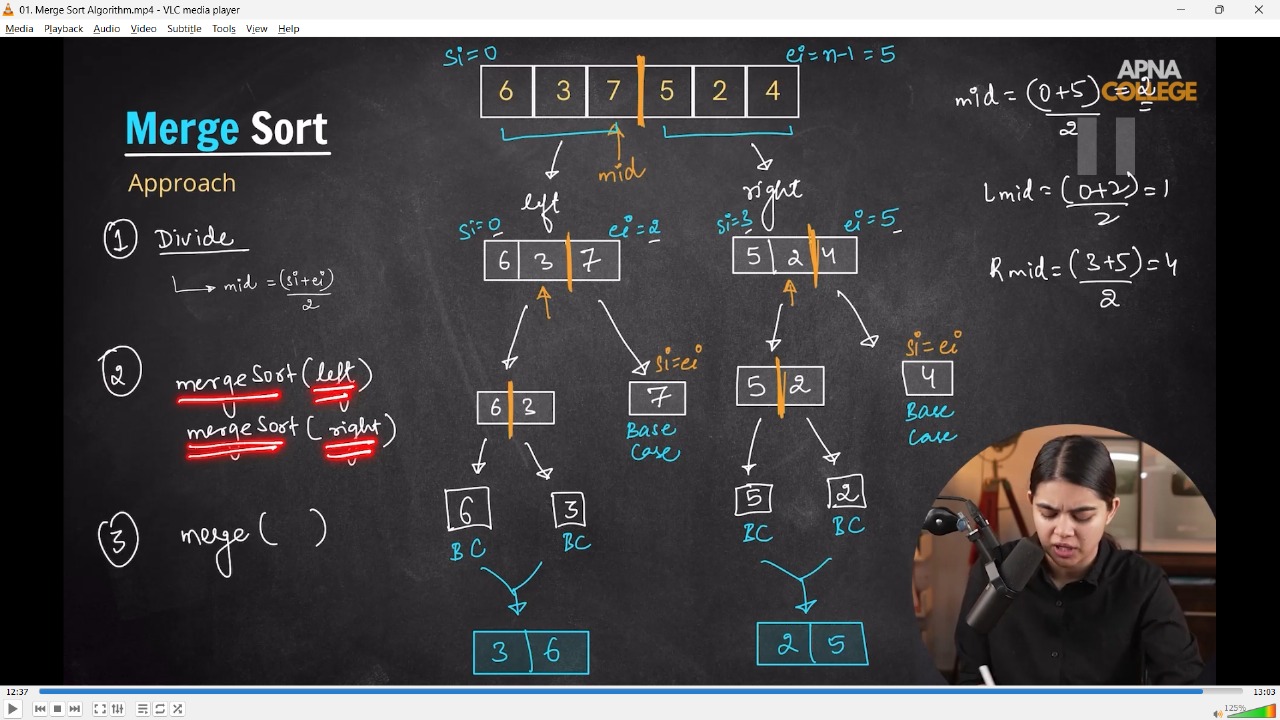
3) Searching In Rotated Array Problem -

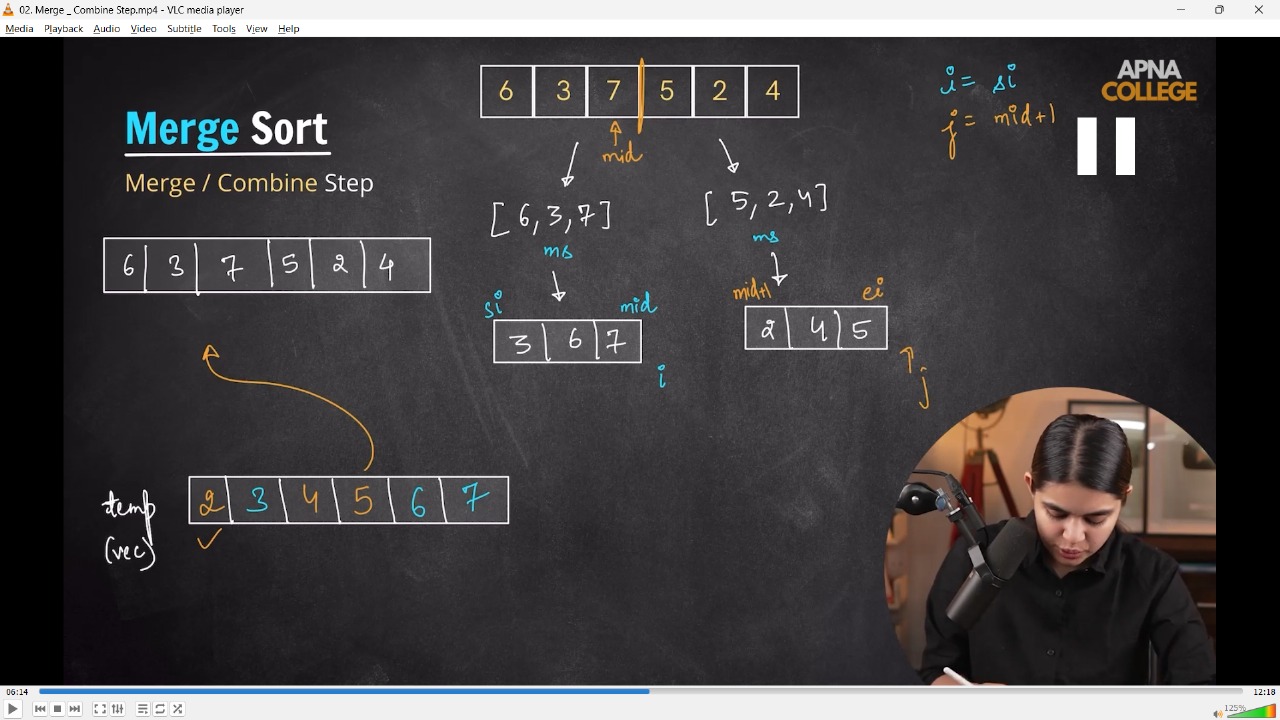
**10) Divide & Conquer –**

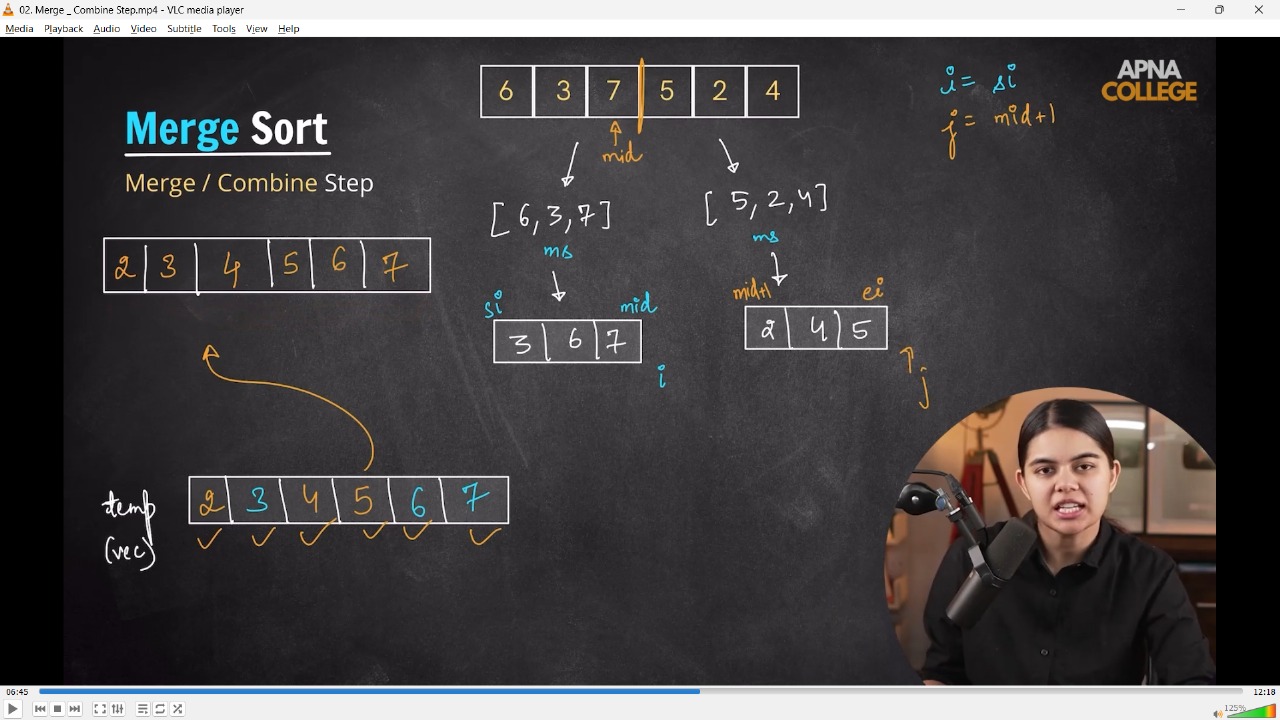
**//1) Merge Sort Algo -**

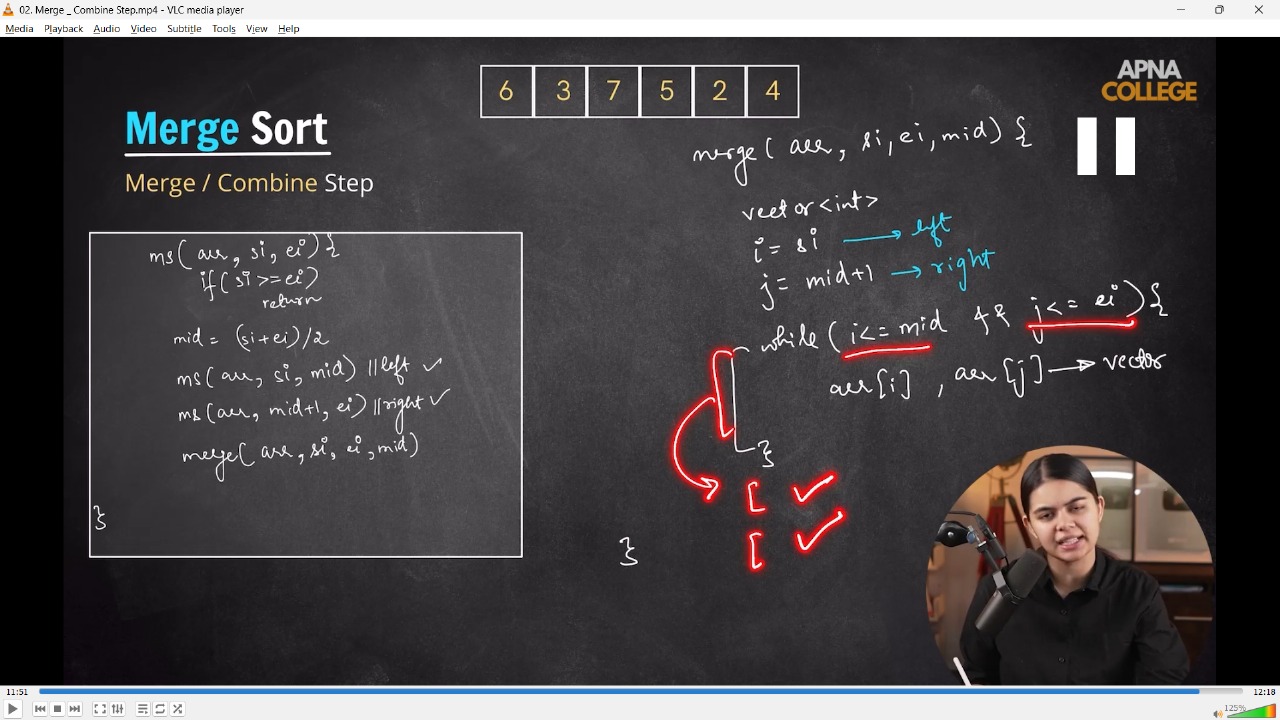


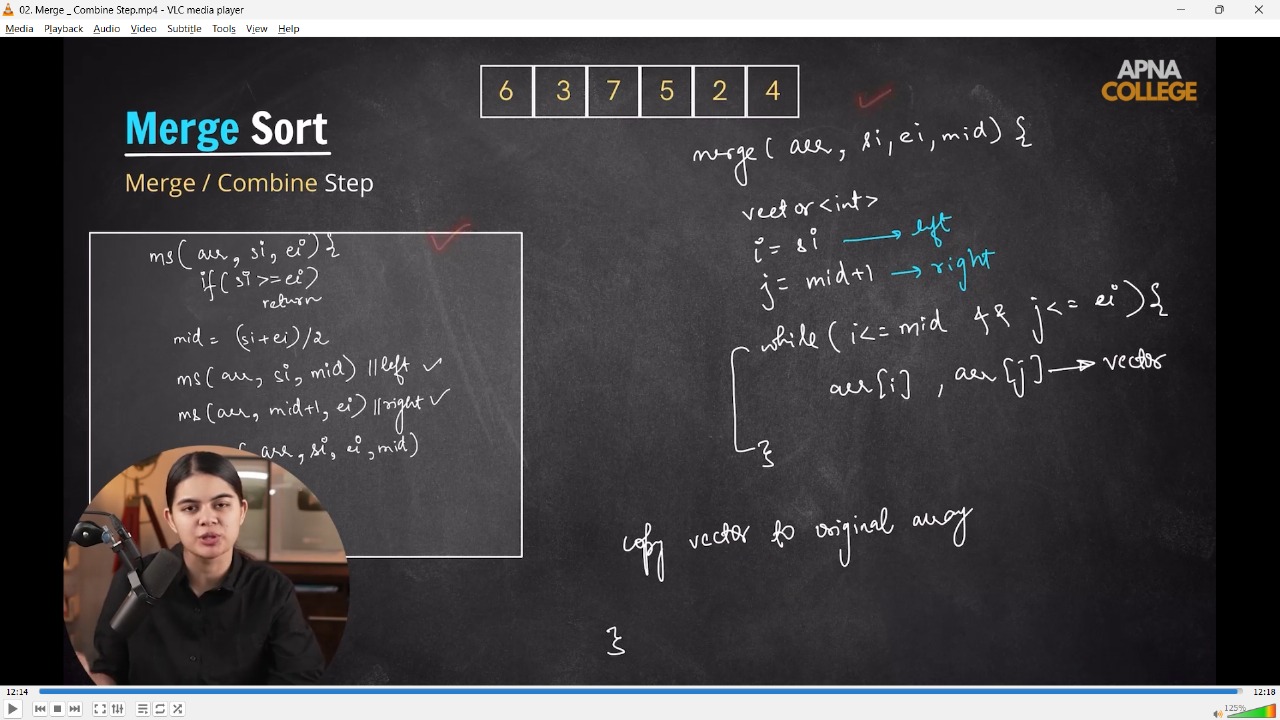


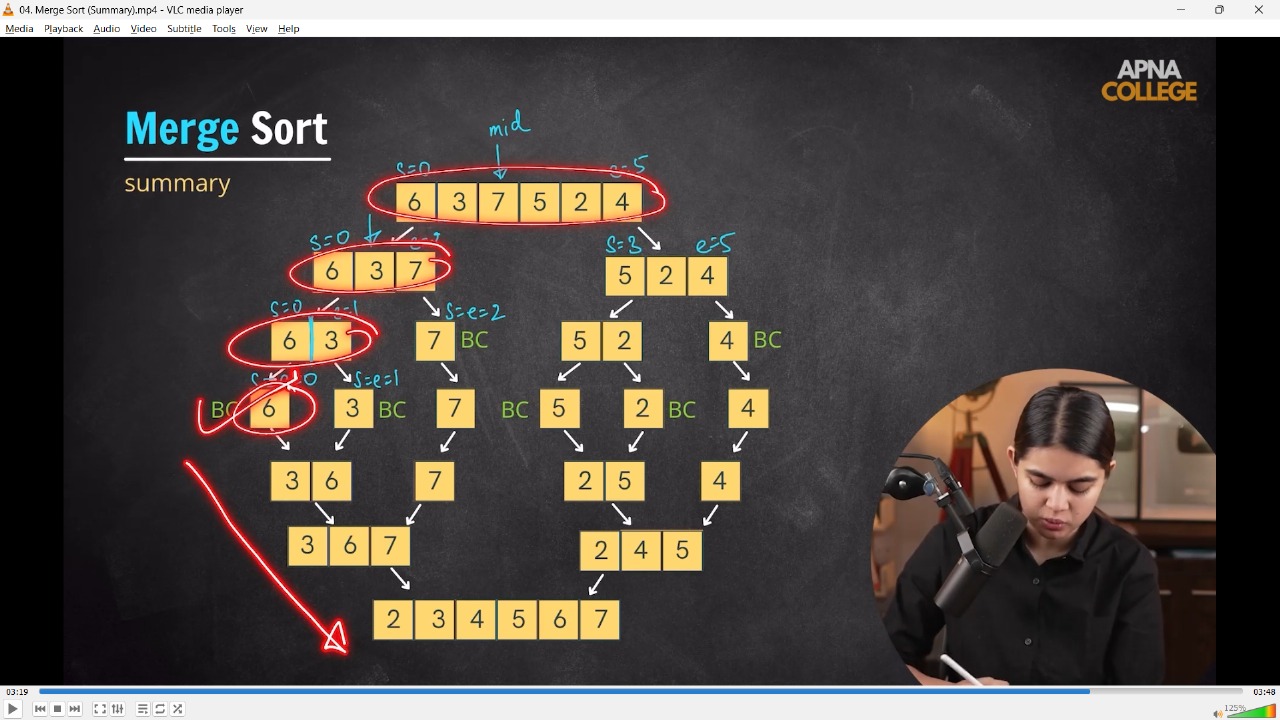


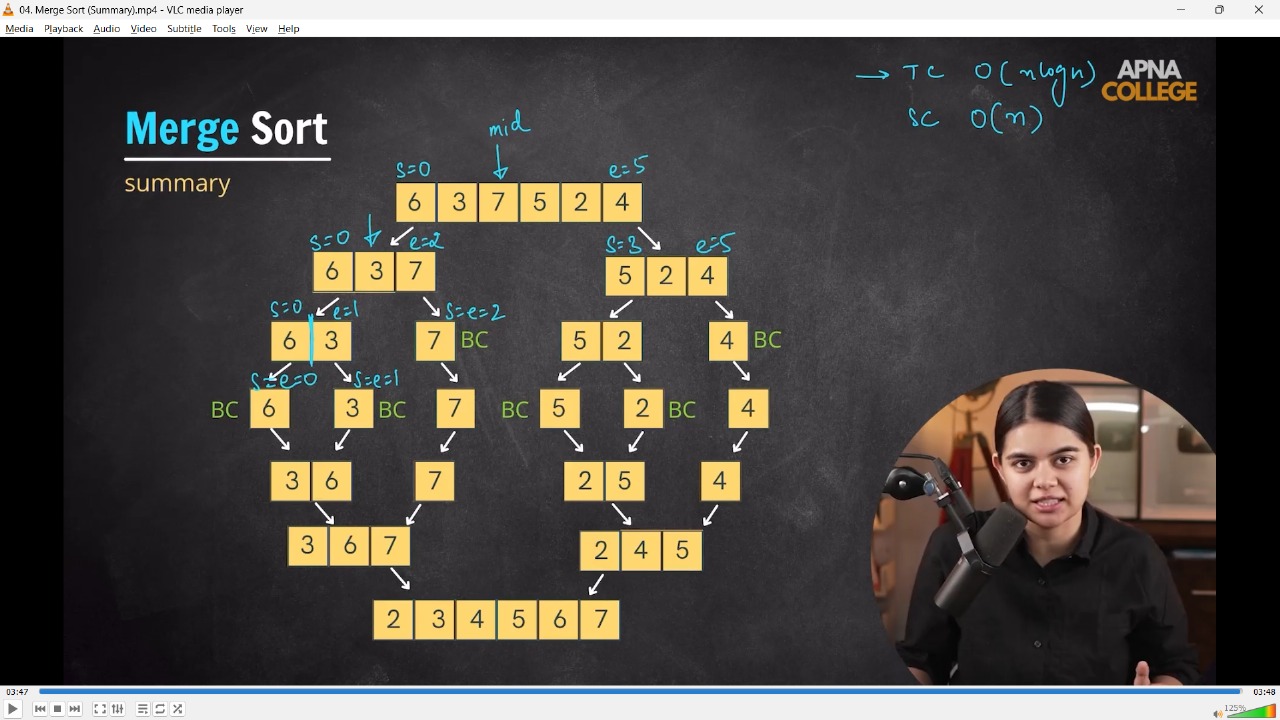












void Merge(int arr[], int si, int mid, int ei) // O(n)

{

    vector<int> temp;

    int i = si;

    int j = mid + 1;

    while (i <= mid && j <= ei)

    {

        if (arr[i] <= arr[j])

        {

            temp.push\_back(arr[i]);

            i++;

        }

        else

        {

            temp.push\_back(arr[j]);

            j++;

        }

    }

    while (i <= mid)

    {

        temp.push\_back(arr[i]);

        i++;

    }

    while (j <= ei)

    {

        temp.push\_back(arr[j]);

        j++;

    }

    // Coverting vector to an aOriginal Array

    for (int i = si, x = 0; i <= ei; i++)

    {

        arr[i] = temp[x++];

    }

}

void mergeSort(int arr[], int si, int ei) // O(log n), so total complexity - O(nlogn)

{

    if (si >= ei)

    {

        return;

    }

    int mid = si + (ei - si) / 2;

    mergeSort(arr, si, mid);     // LeftHalf

    mergeSort(arr, mid + 1, ei); // RightHalf

    Merge(arr, si, mid, ei);     // Conquer

}

void printArray(int arr[], int n)

{

    cout << "Sorted Array is - " << endl;

    for (int i = 0; i < n; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

}

int main()

{

    int arr[6] = {6, 3, 7, 5, 2, 4};

    int n = 6;

    mergeSort(arr, 0, n - 1);

    printArray(arr,n);

/\*

Sorted Array is -

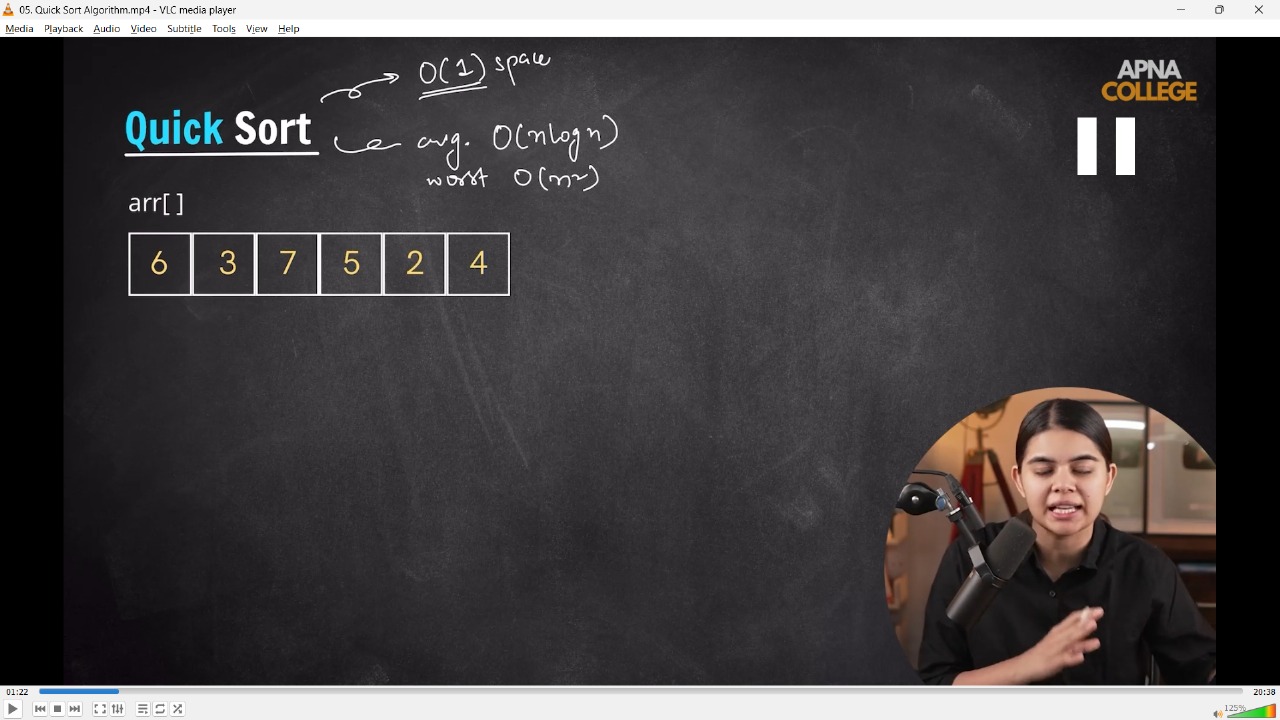
2 3 4 5 6 7

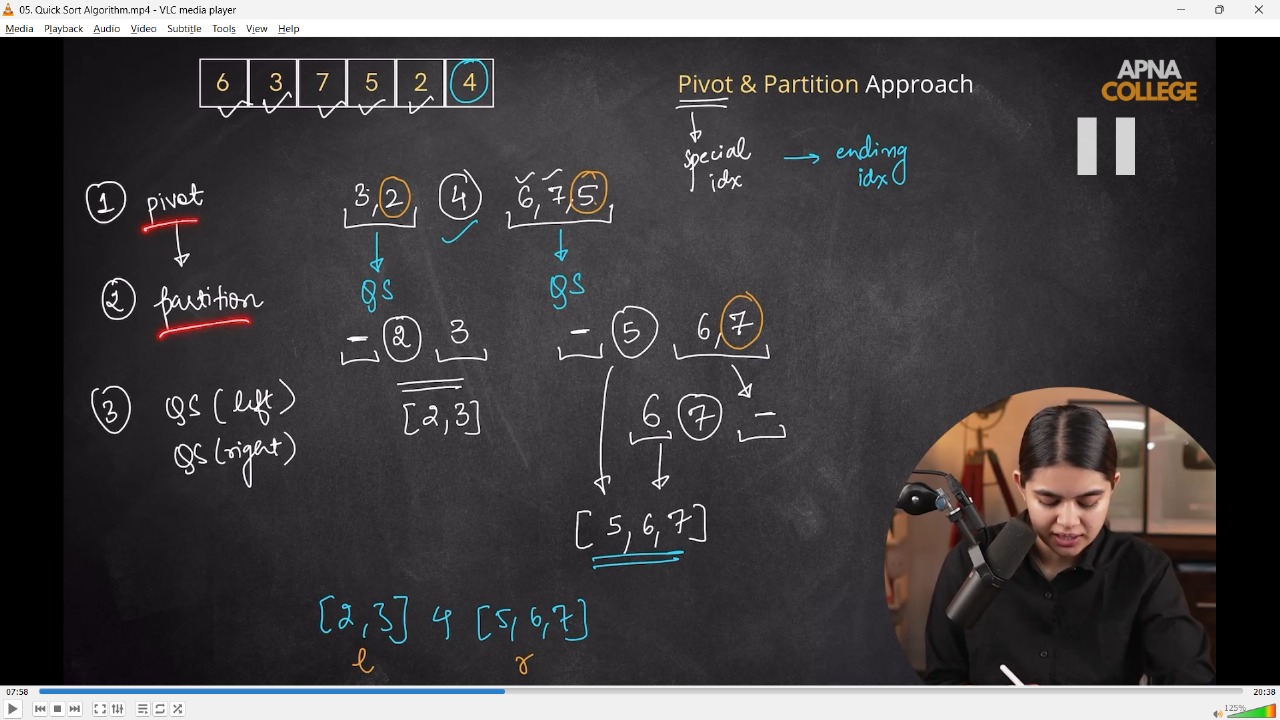
 \*/

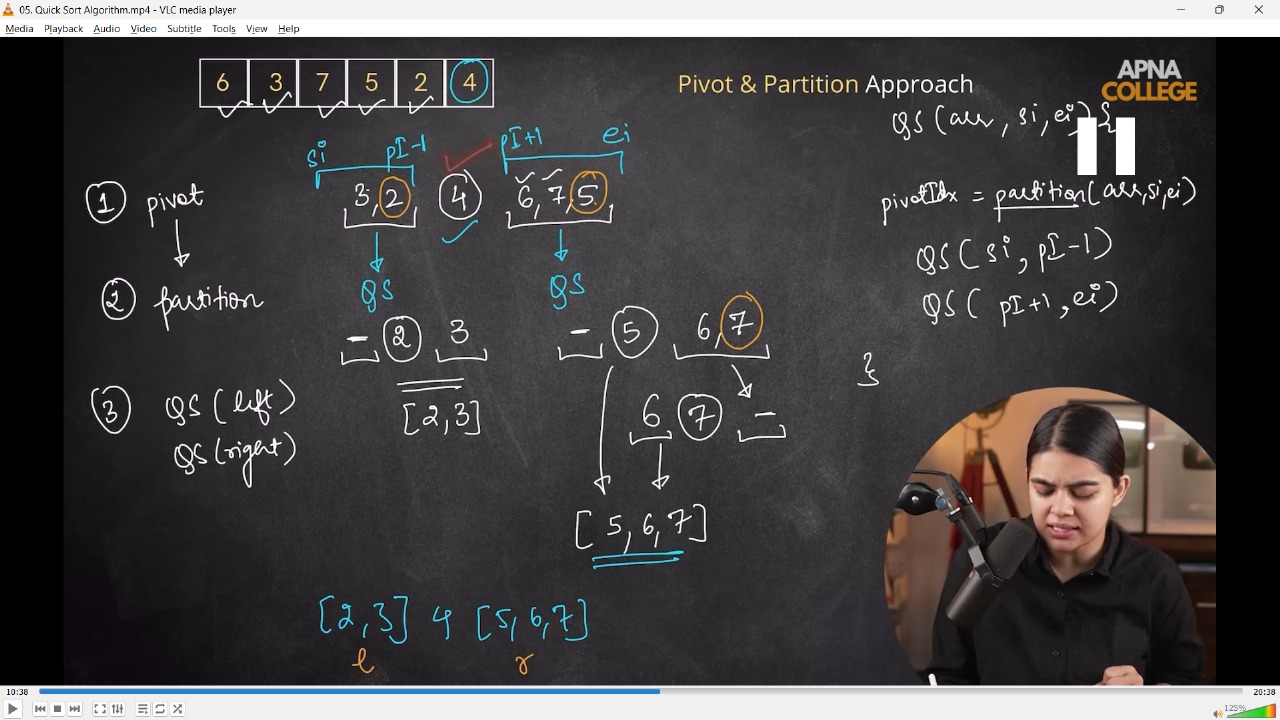
}

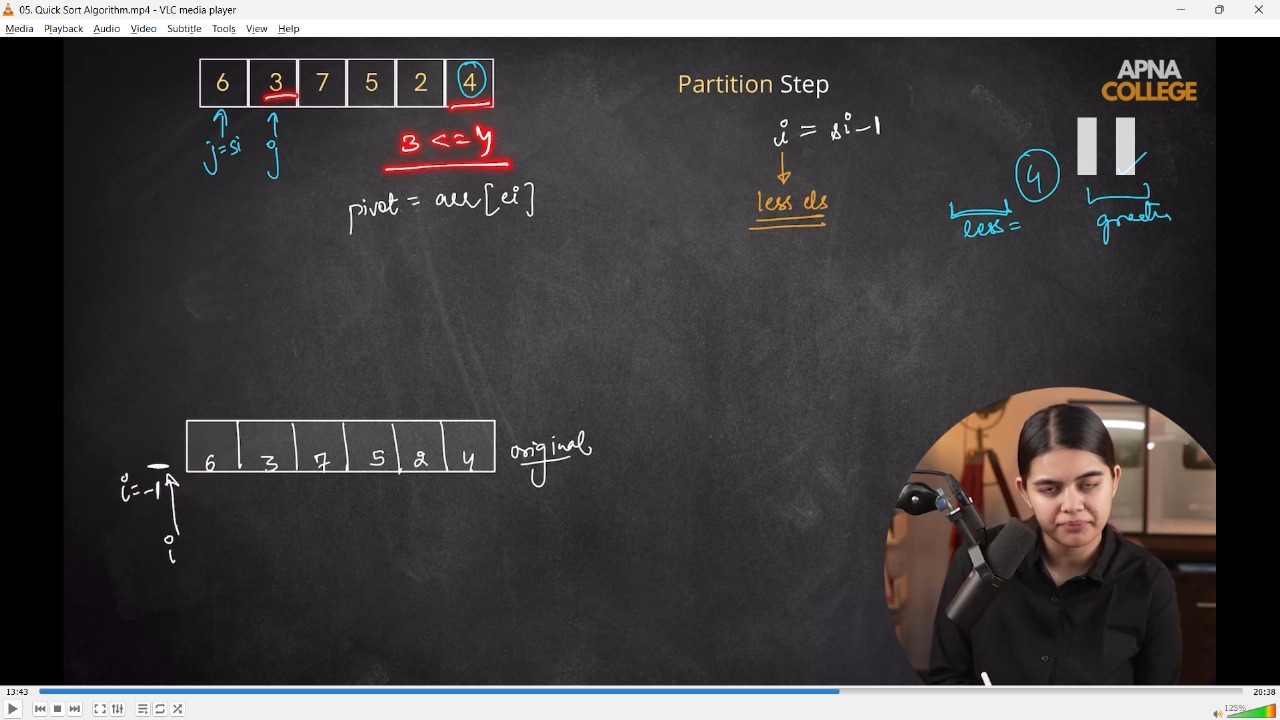
// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

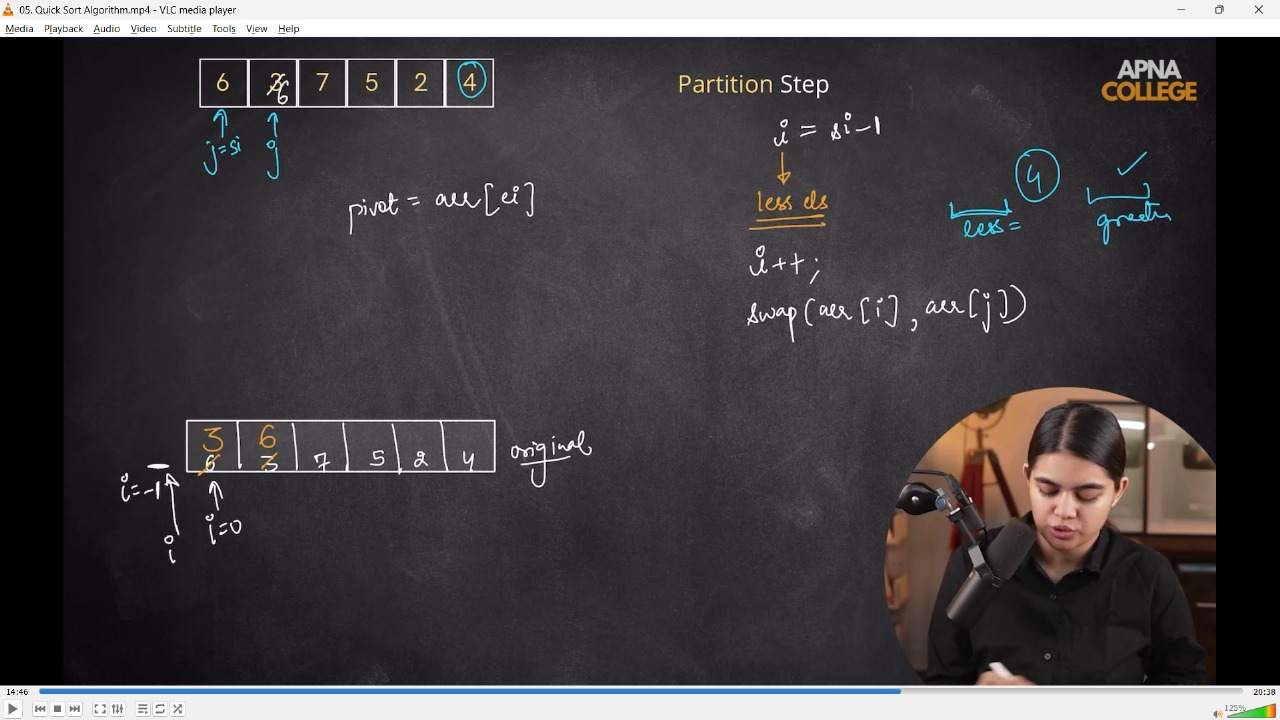
**//2) Quick Sort Algorithm -**



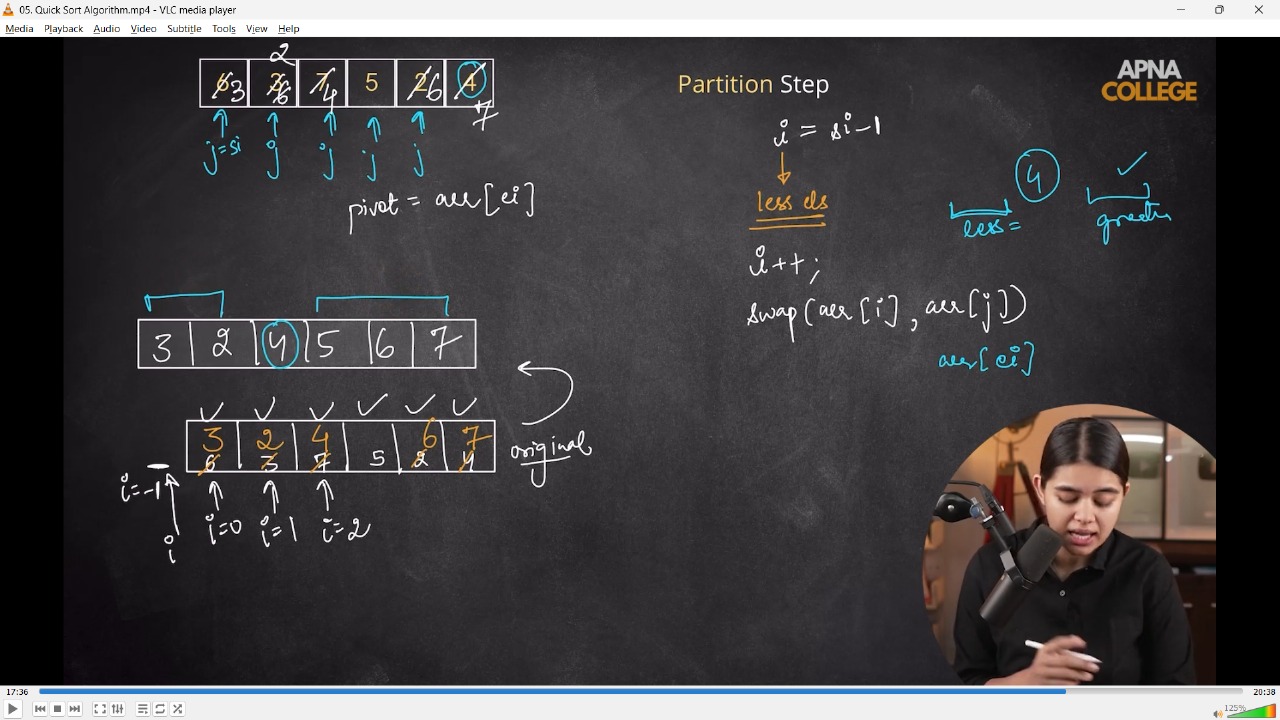


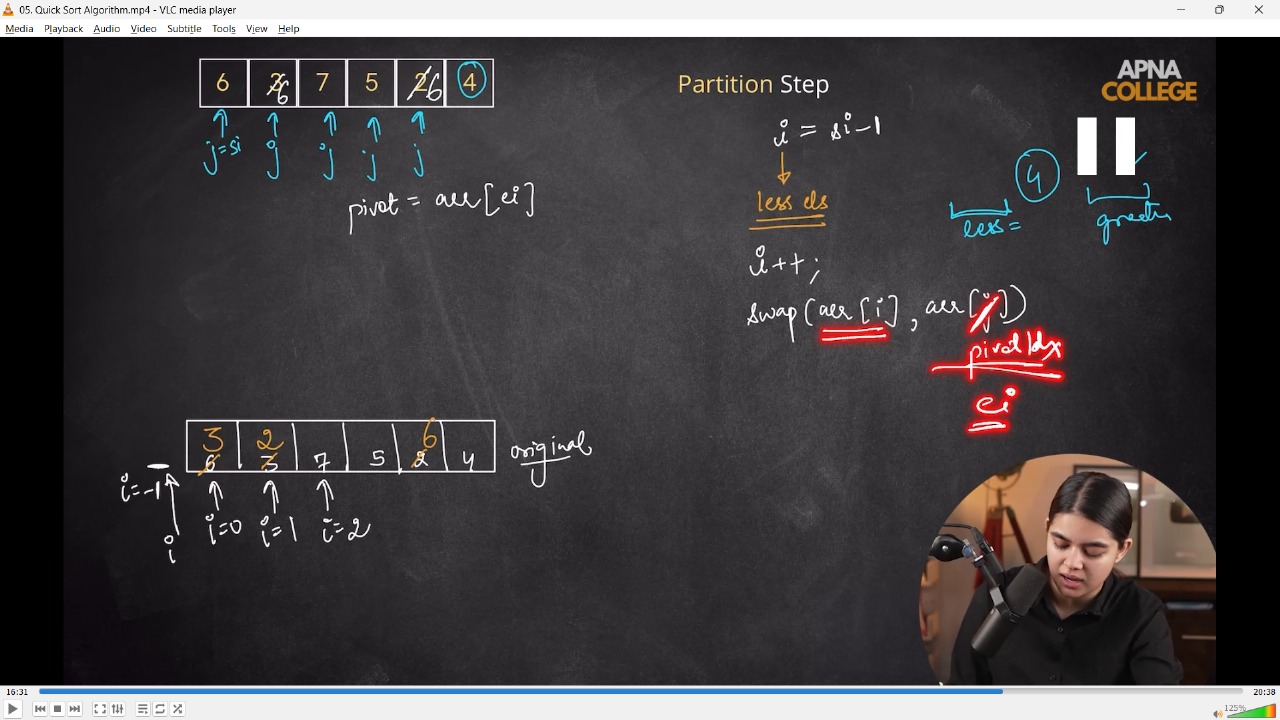


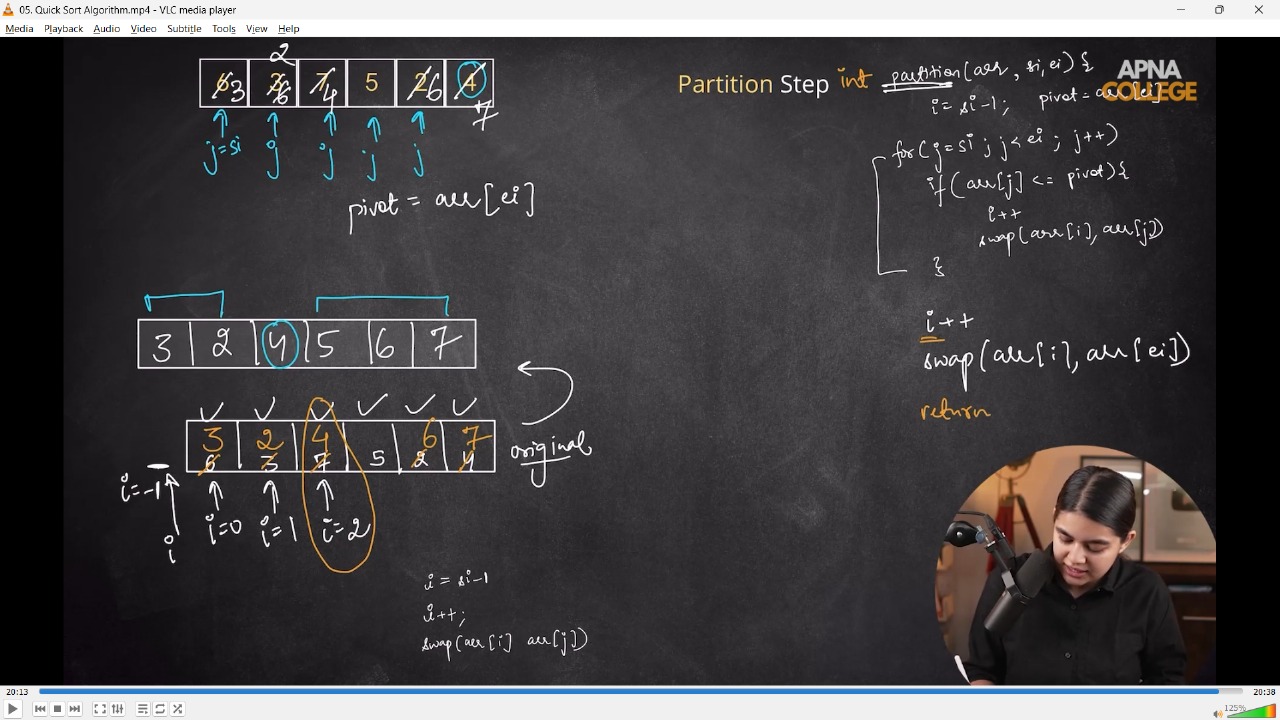












int partition(int arr[], int si, int ei)

{

    int i = si - 1;

    int pivot = arr[ei];

    for (int j = si; j < ei; j++)

    {

        if (arr[j] <= pivot)

        {

            i++;

            swap(arr[i], arr[j]);

        }

    }

    i++;

    swap(arr[i], arr[ei]);

    // pivotindex = i

    return i;

}

void QuickSort(int arr[], int si, int ei)//O(n\*logn)

{

    if (si >= ei)

    {

        return;

    }

    int pivotIdx = partition(arr, si, ei);

    QuickSort(arr, si, pivotIdx - 1); // Left Half

    QuickSort(arr, pivotIdx + 1, ei); // Right Half

}

void printArr(int arr[], int n)

{

    for (int i = 0; i < n; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

}

int main()

{

    int arr[6] = {6, 3, 7, 5, 2, 4};

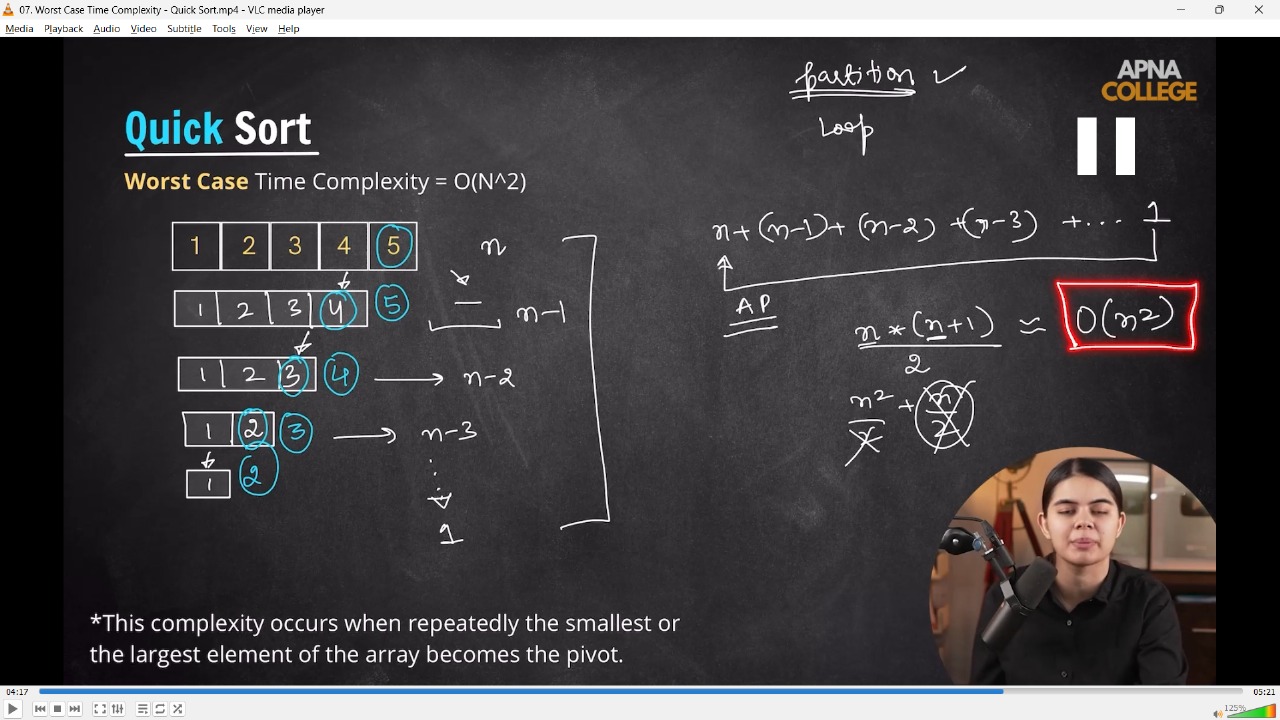
    int n = 6;

    QuickSort(arr, 0, n - 1);

    printArr(arr, n); // 2 3 4 5 6 7

}

// **2.1) Quick Sort Worst Case Scenario**



/\*📒 In the Worst Case of Quick Sort, the complxity will be O(n^2) and the

Worst case occurs when either in INCREASING Order Or in DECREASING Order

So, the Pivot element will be at last , so there will be an AP

\*/

int partition(int arr[], int si, int ei)

{

    int i = si - 1;

    int pivot = arr[ei];

    for (int j = si; j < ei; j++)

    {

        if (arr[j] <= pivot)

        {

            i++;

            swap(arr[i], arr[j]);

        }

    }

    i++;

    swap(arr[i], arr[ei]);

    // pivotindex = i

    return i;

}

void QuickSort(int arr[], int si, int ei) // O(n\*logn)

{

    if (si >= ei)

    {

        return;

    }

    int pivotIdx = partition(arr, si, ei);

    QuickSort(arr, si, pivotIdx - 1); // Left Half

    QuickSort(arr, pivotIdx + 1, ei); // Right Half

}

void printArr(int arr[], int n)

{

    for (int i = 0; i < n; i++)

    {

        cout << arr[i] << " ";

    }

    cout << endl;

}

int main()

{

    int arr[6] = {1, 2, 3, 4, 5, 6};

    int n = 6;

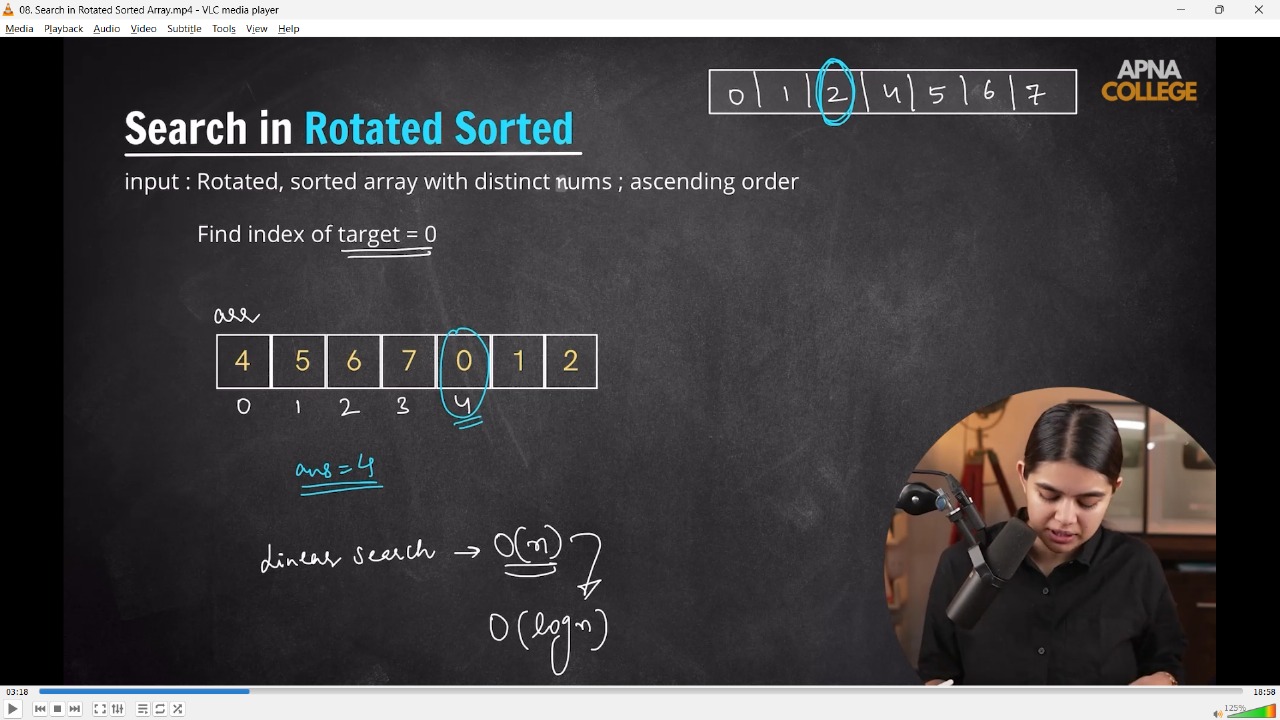
    QuickSort(arr, 0, n - 1);

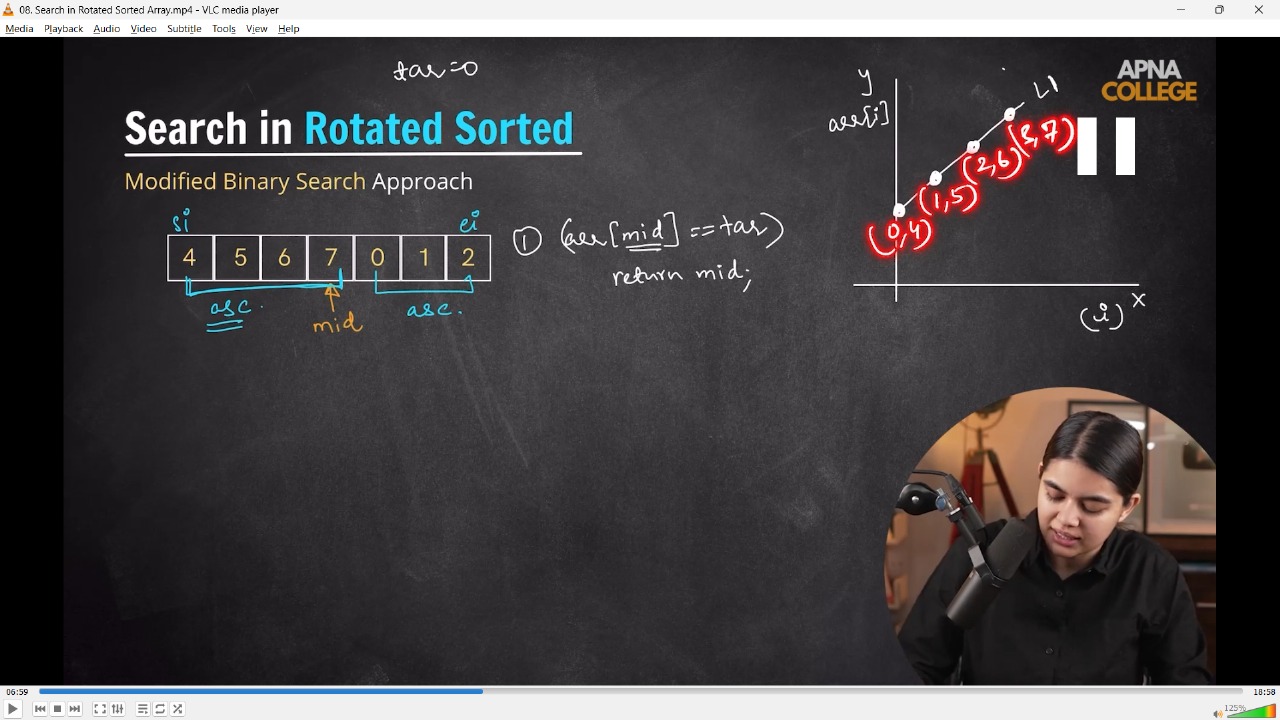
    printArr(arr, n); // 1 2 3 4 5 6

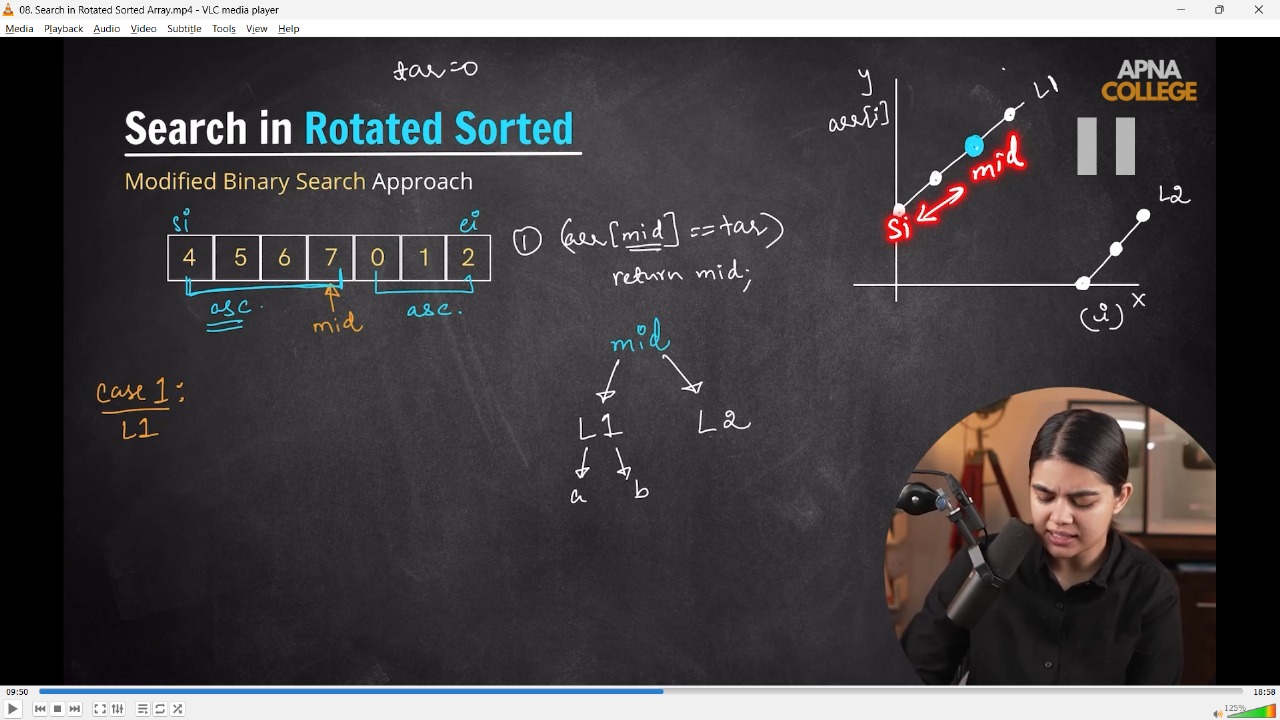
}

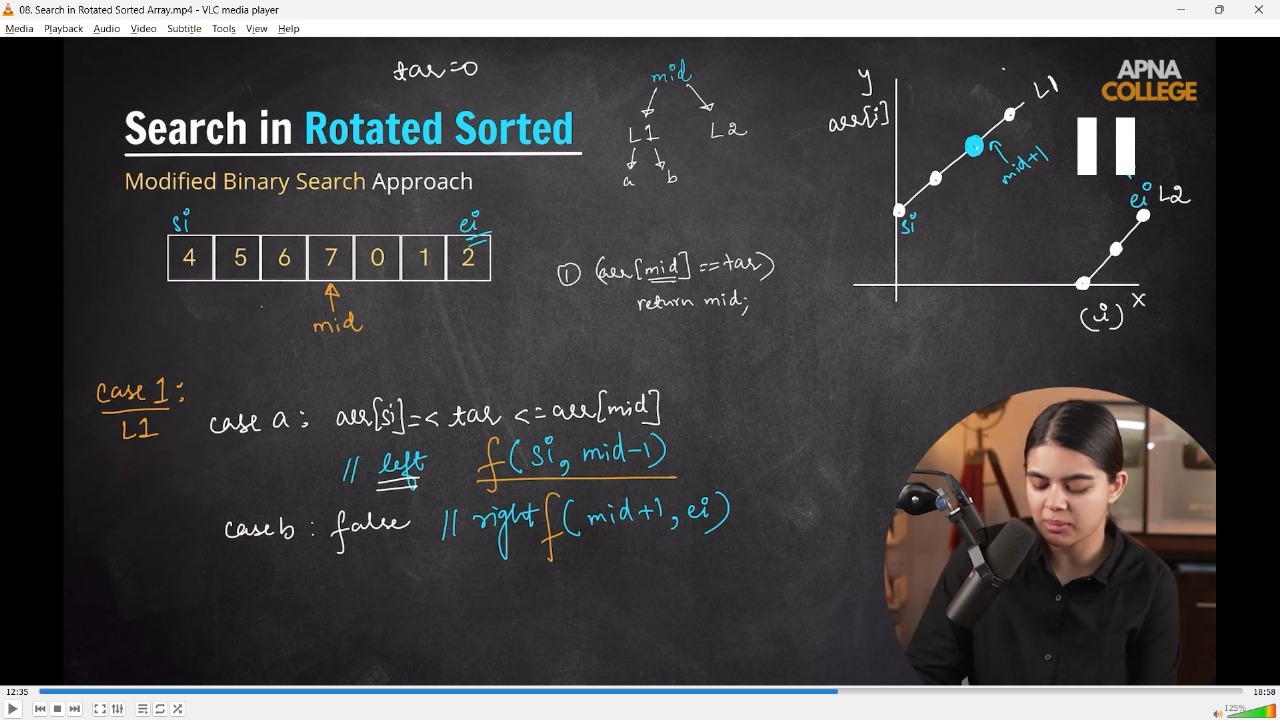
// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

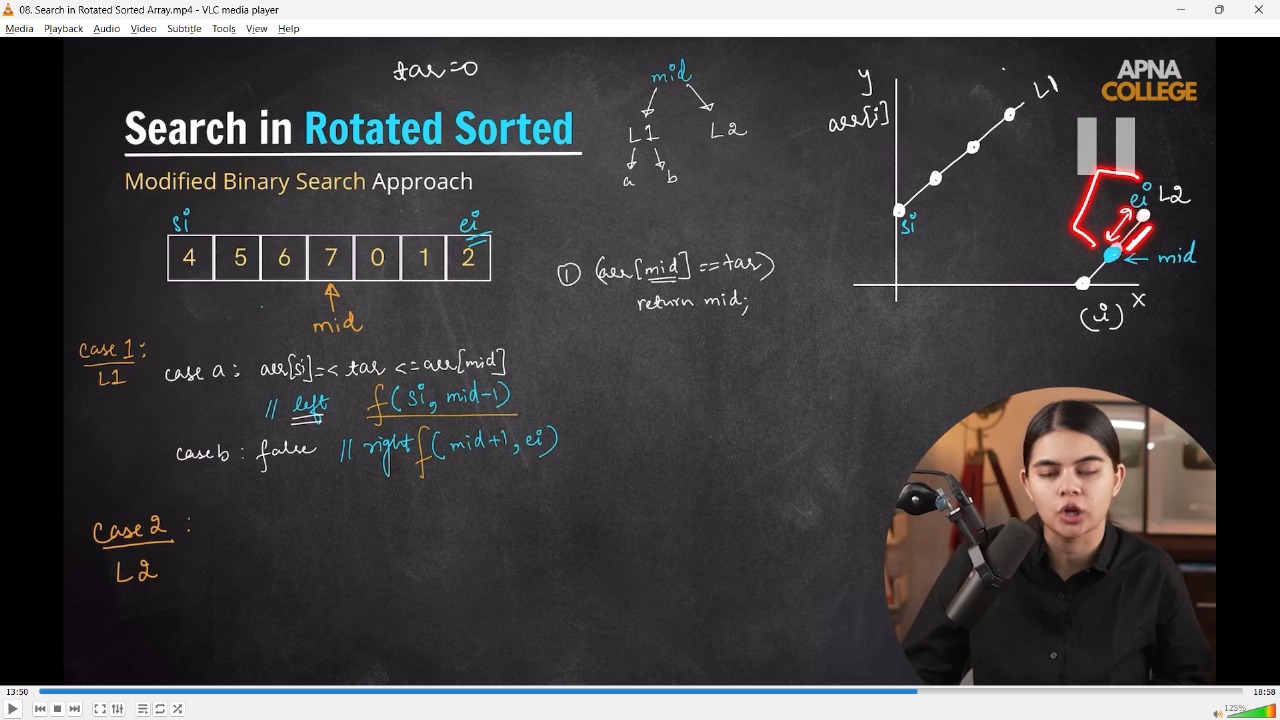
//**3) Searching In Rotated Array Problem**

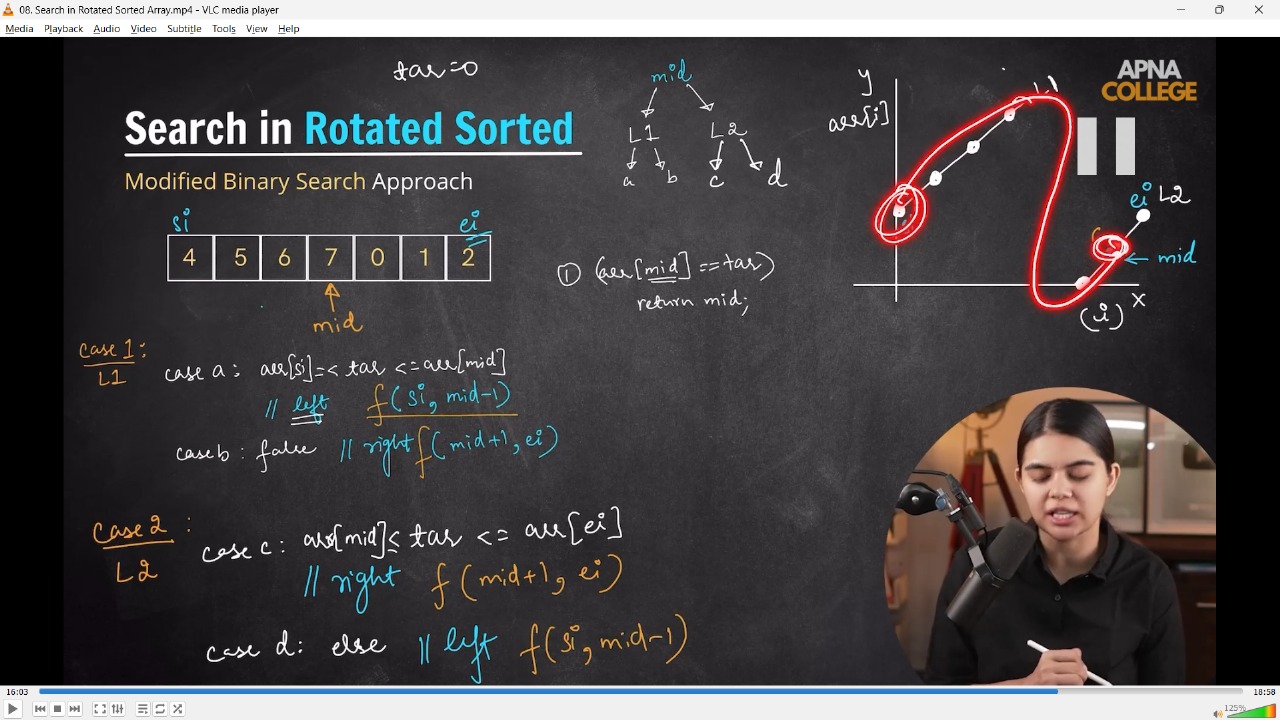


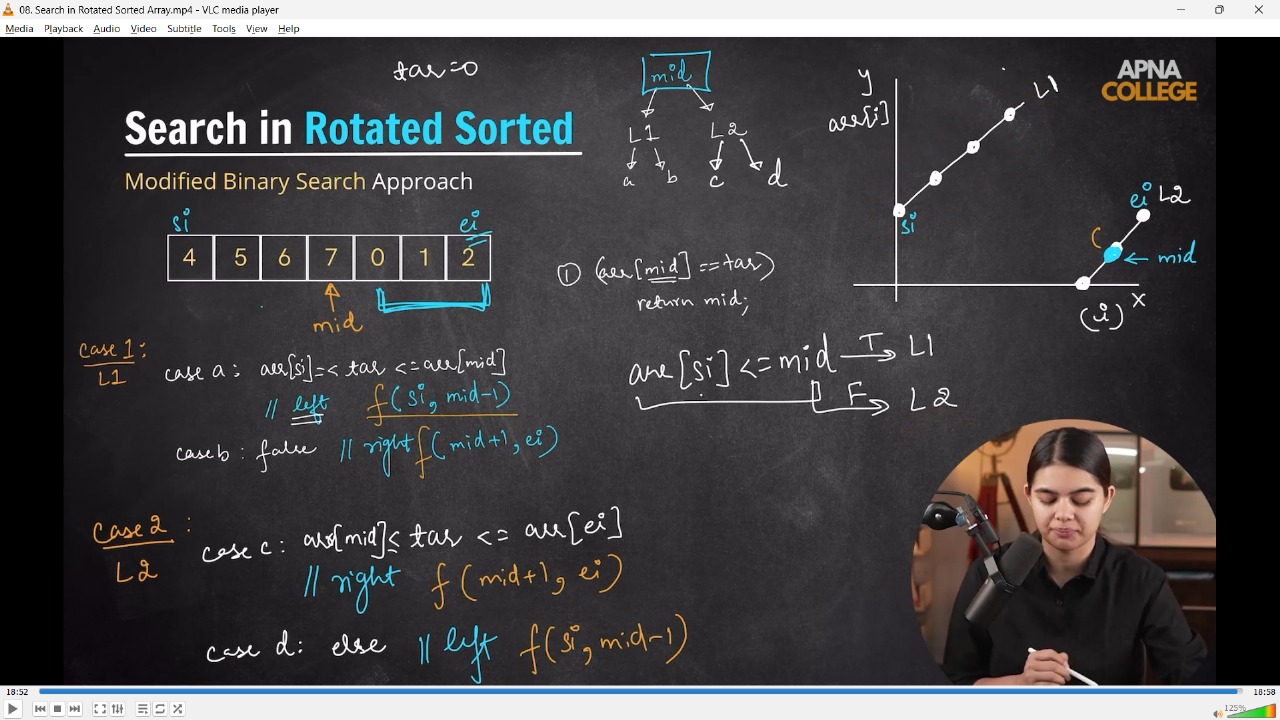


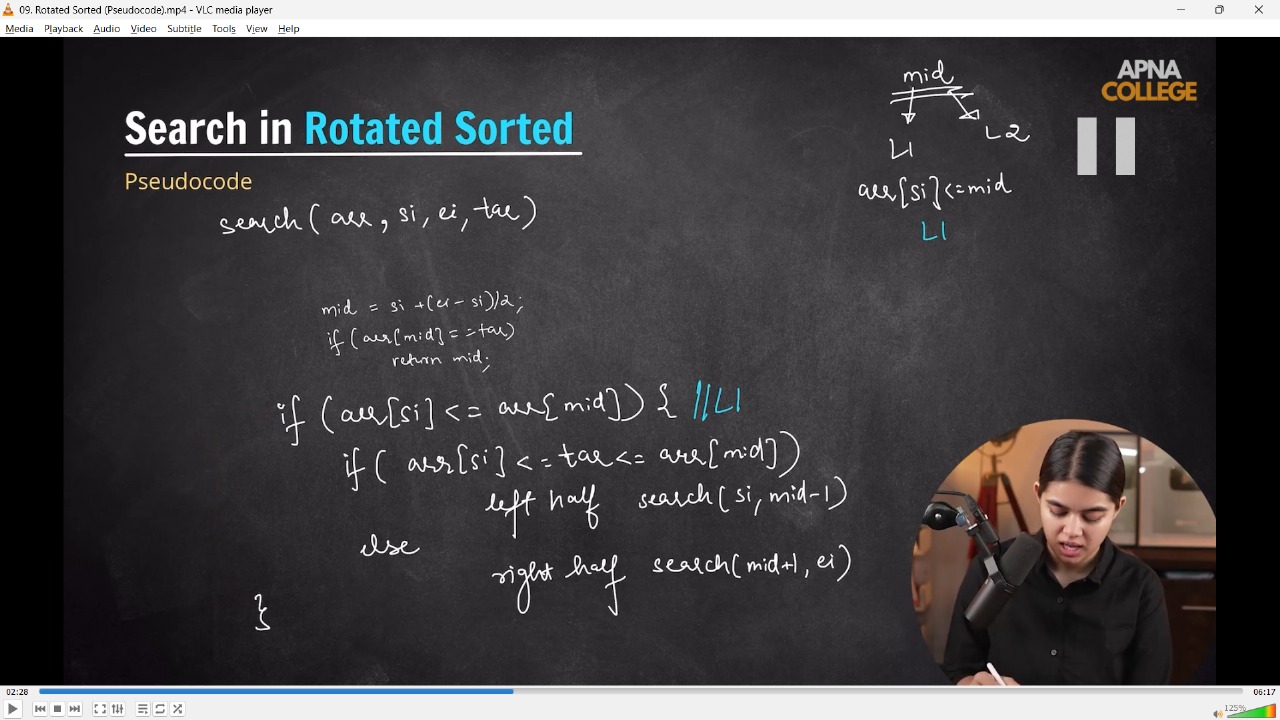


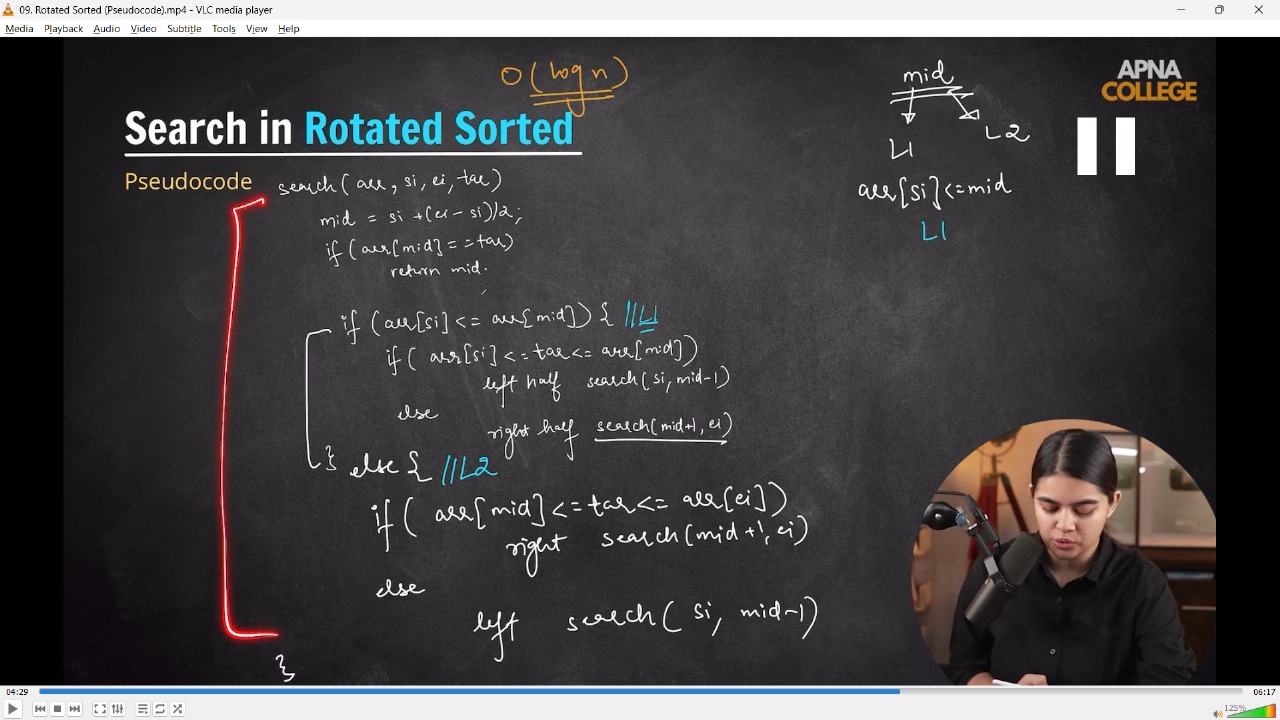


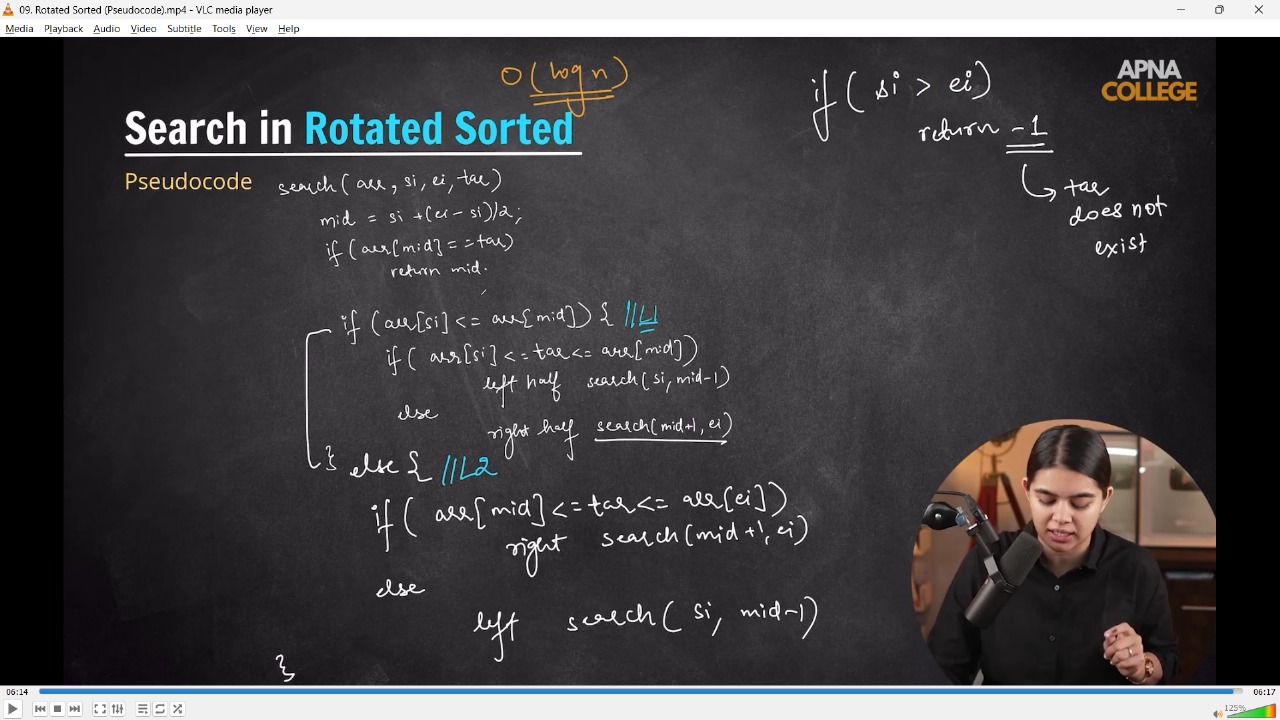












int search(int arr[], int si, int ei, int target)

{

    if (si > ei)

    {

        return -1;

    }

    int mid = si + (ei - si) / 2;

    if (arr[mid] == target)

    {

        return mid;

    }

    if (arr[si] <= arr[mid]) // it's for L1

    {

        if (arr[si] <= target && target <= arr[mid])

        {

            // For left Half

            return search(arr, mid + 1, ei, target);

        }

        else

        {

            // For Right half

            return search(arr, mid + 1, ei, target);

        }

    }

    else

    {

        // Now it's for L2

        if (arr[mid] <= target && target <= arr[ei])

        {

            // For Right Half

            return search(arr, mid + 1, ei, target);

        }

        else

        {

            // For left half

            return search(arr, si, mid - 1, target);

        }

    }

}

int main()

{

    int arr[7] = {4, 5, 6, 7, 0, 1, 2};

    int n = 7;

    cout << "idx : " << search(arr, 0, n - 1, 0) << endl; // idx : -1

    cout << "idx : " << search(arr, 0, n - 1, 2) << endl; // idx : 6

    cout << "idx : " << search(arr, 0, n - 1, 7) << endl; // idx : 3

// T.C - O(logn)

}

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