**Kadane’s Algortuhms & Subarray, Subsequence & Substring Concepts –**

#include <bits/stdc++.h>

// #include <iostream>

// #include<algorithm>

// #include<climits>

// #include<string>

// #include<cctype>

// #include<vector>

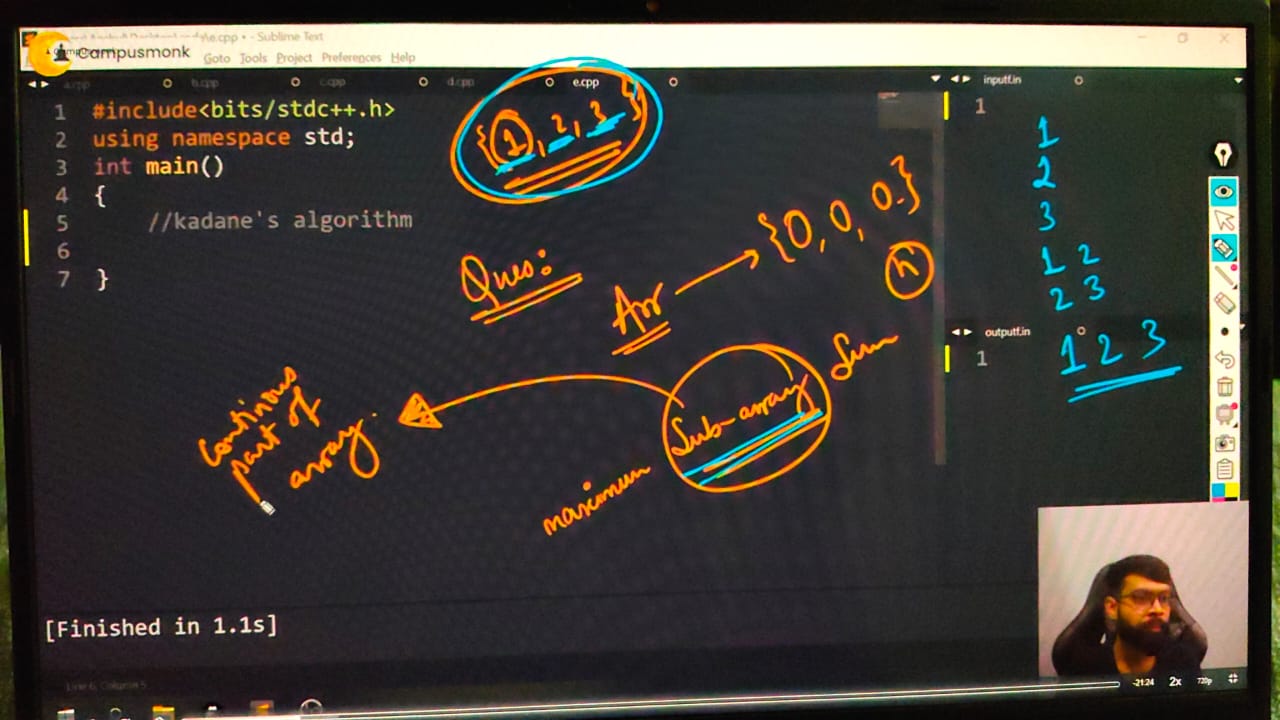
// #include<set>

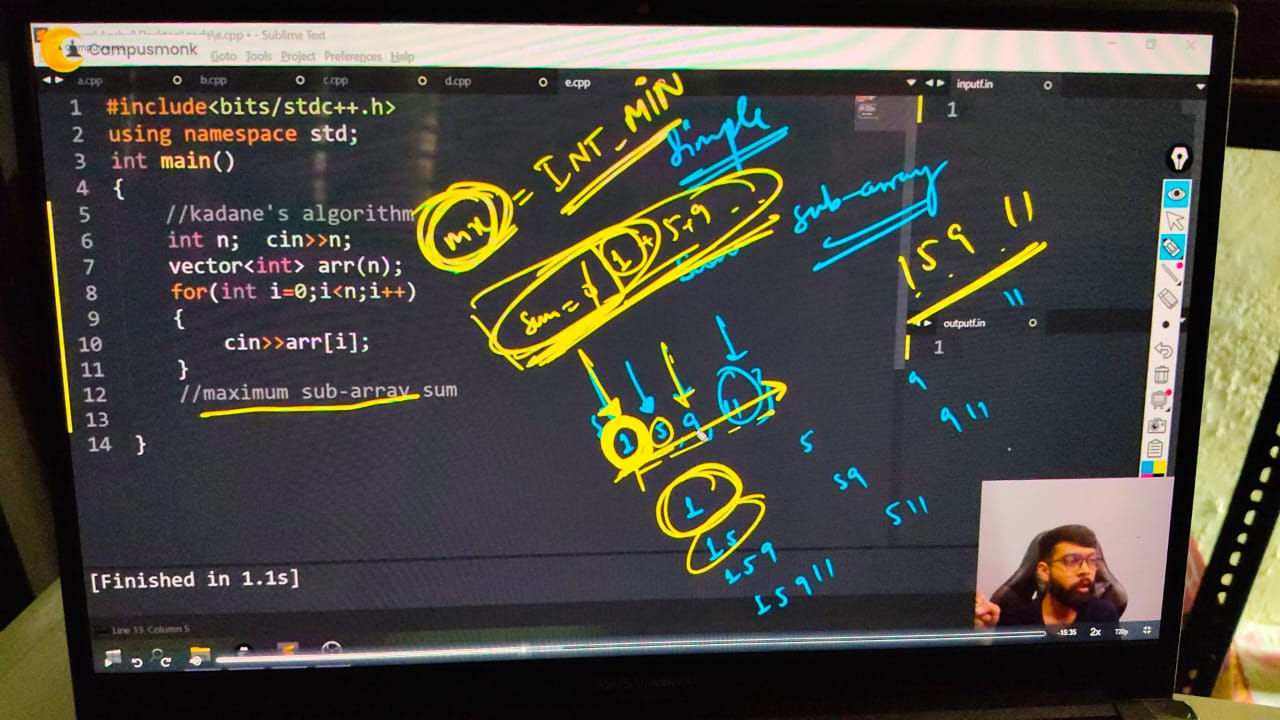
using namespace std;

// int main()

// {

**// Qun 1 - Print Maximum sum of SubArray. Coondition - Sub-array must not be empty –**





// int n;

// cout << "size of array - " << endl;

// cin >> n;

// vector<int> arr(n);

// cout << "Give elements of array  - " << endl;

// for (int i = 0; i < n; i++)

// {

//     cin >> arr[i];

// }

// // Maximum subarray sum -

// int ans = INT\_MIN;

// for (int i = 0; i < n; i++)

// {

//     int sum = 0;

//     for (int j = i; j < n; j++)

//     {

//         sum += arr[j];

//         ans = max(ans, sum);

//     }

// }

// cout<<ans<<endl;

/\*

size of array -

5

Give elements of array  -

1 4 -7 11 5

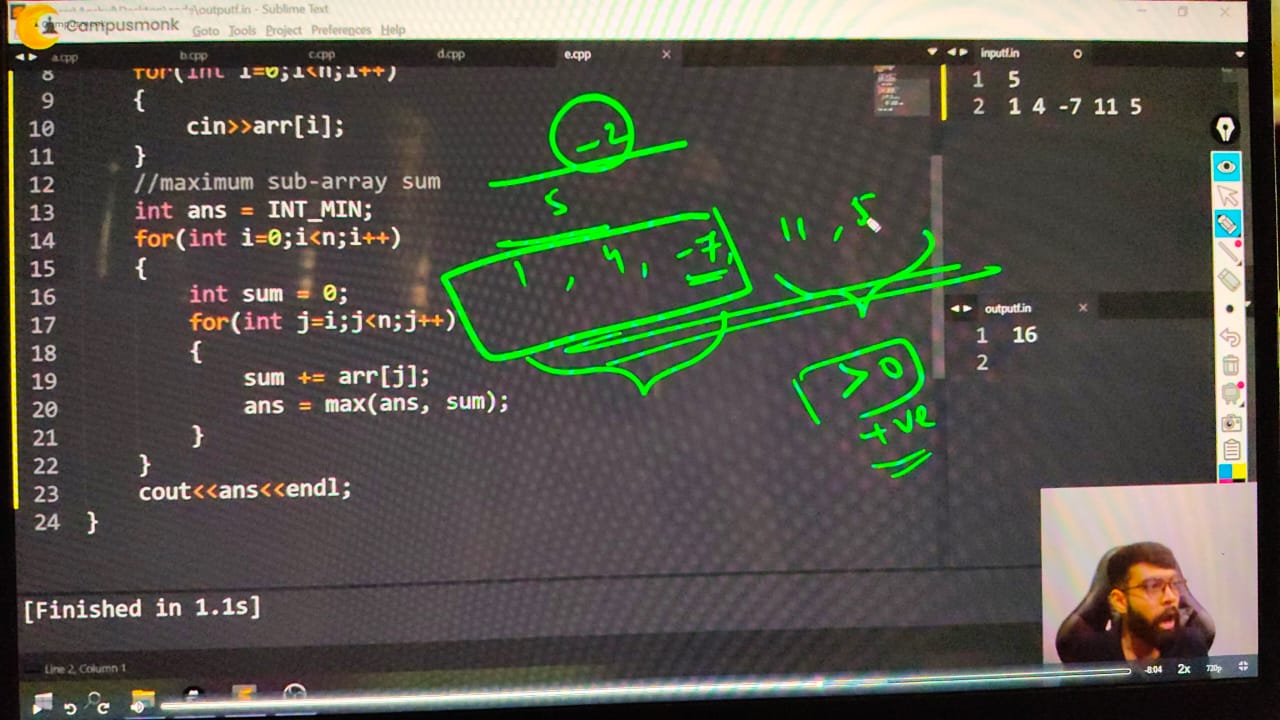
16

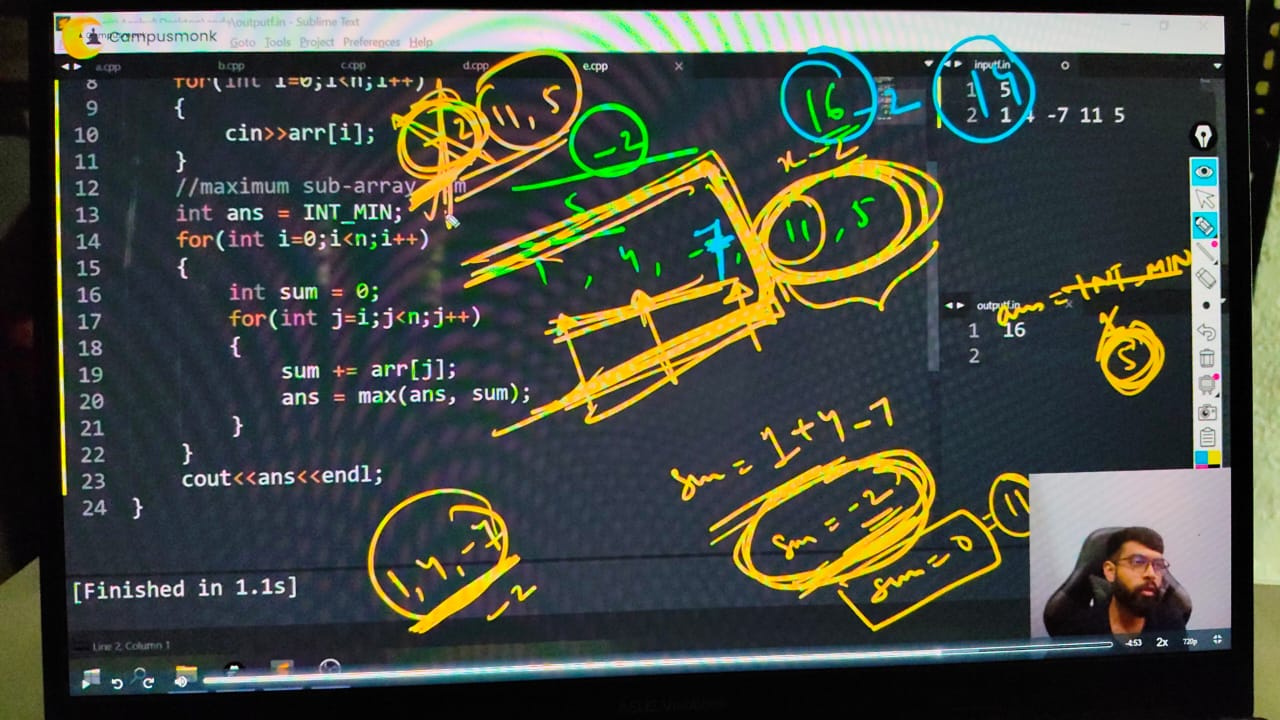
T.C - here the time complexity is - O(n\*n = n^2)

 \*/

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

**// Kadane's ALgorithms –**





// So, for decreasing this much complexity using Kadane's ALgotihms - Acc. ot this aalgorithm don;t addd to those elements which is negative , leave the sum zero, and incldue only possitive numbers sum

// int n;

// cout << "vector size - ";

// cin >> n;

// vector<int> arr(n);

// cout << "What are the vector elements - " << endl;

// for (int i = 0; i < n; i++)

// {

//     cin >> arr[i];

// }

// // For max sum array using Kadane's algorthms -

// int ans = INT\_MIN;

// int sum = 0;

// for (int i = 0; i < n; i++)

// {

//     sum += arr[i];

//     ans = max(ans, sum);

//     if (sum < 0)

//     {

//         sum = 0;

//     }

// }

// cout<<"hence the maximum sum can be print as - "<<ans<<endl;

/\*

vector size - 5

What are the vector elements -

1 4 -7 11 5

16

vector size - 5

What are the vector elements -

1 4 -3 11 5

hence the maximum sum can be print as - 18

T.C - O(n) - no any unnecessary loops

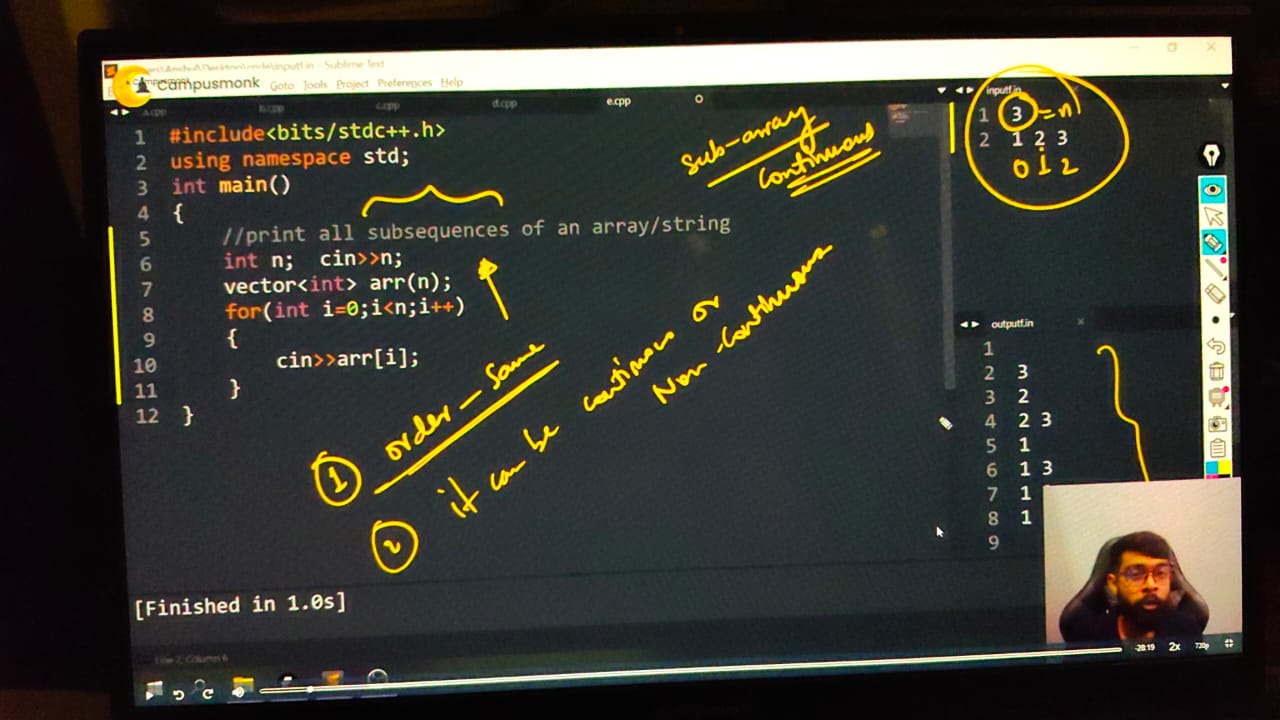
S.C - O(1) - constant because of only 2 variable declared

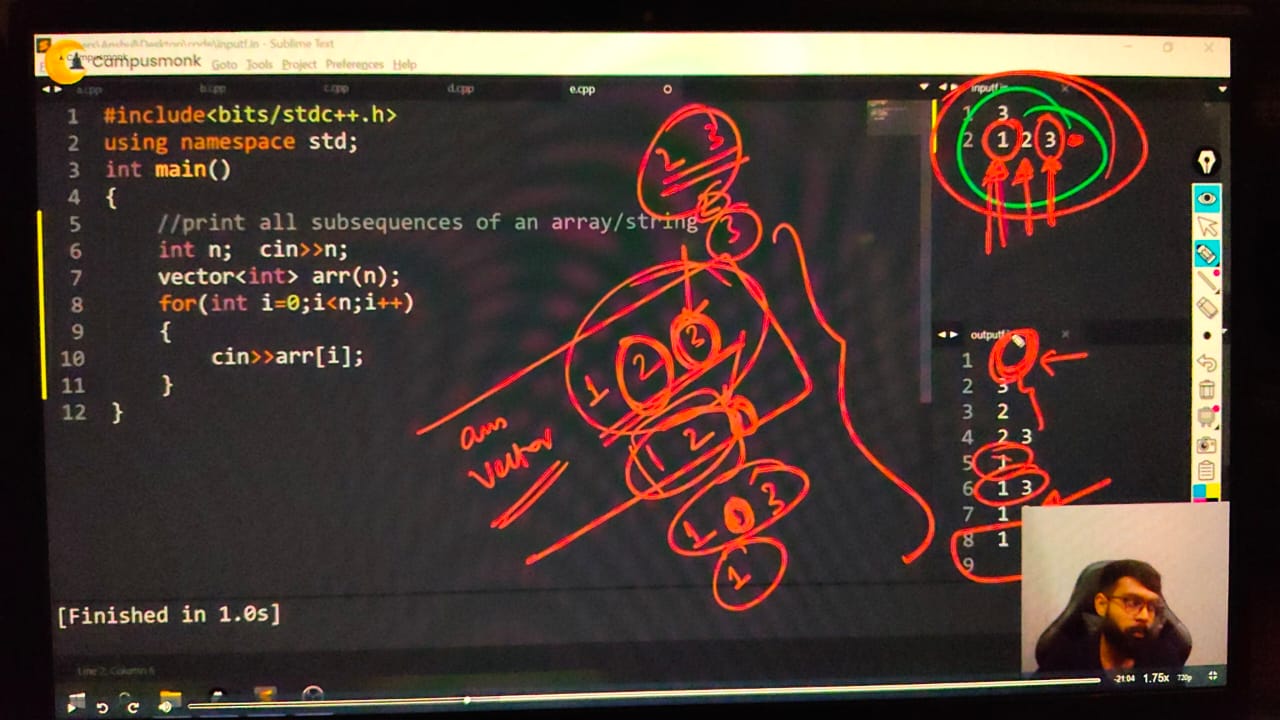
 \*/

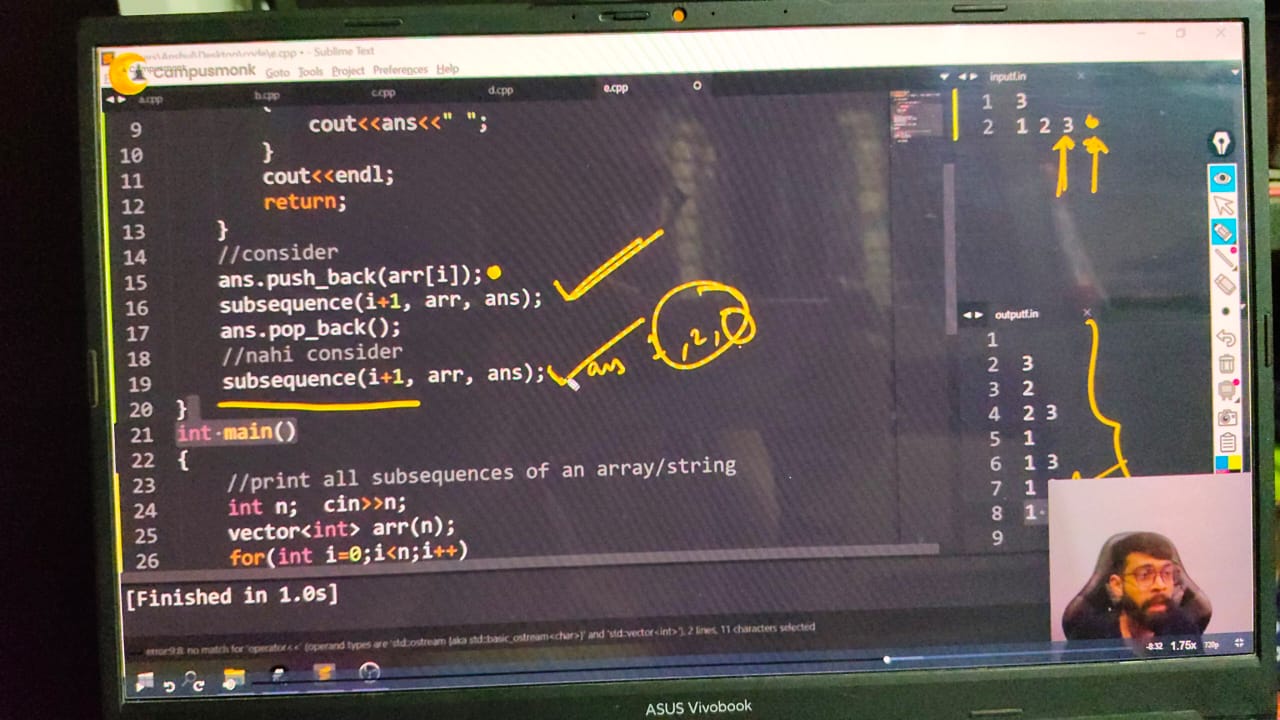
// }

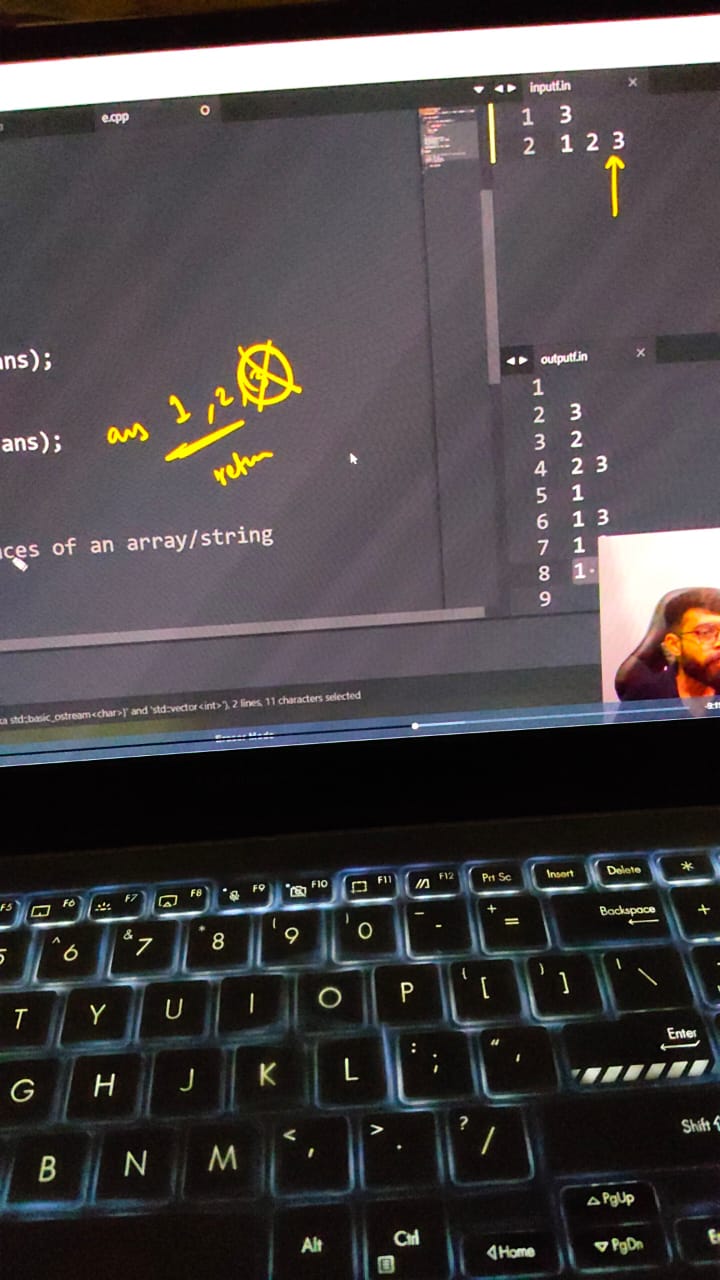
// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

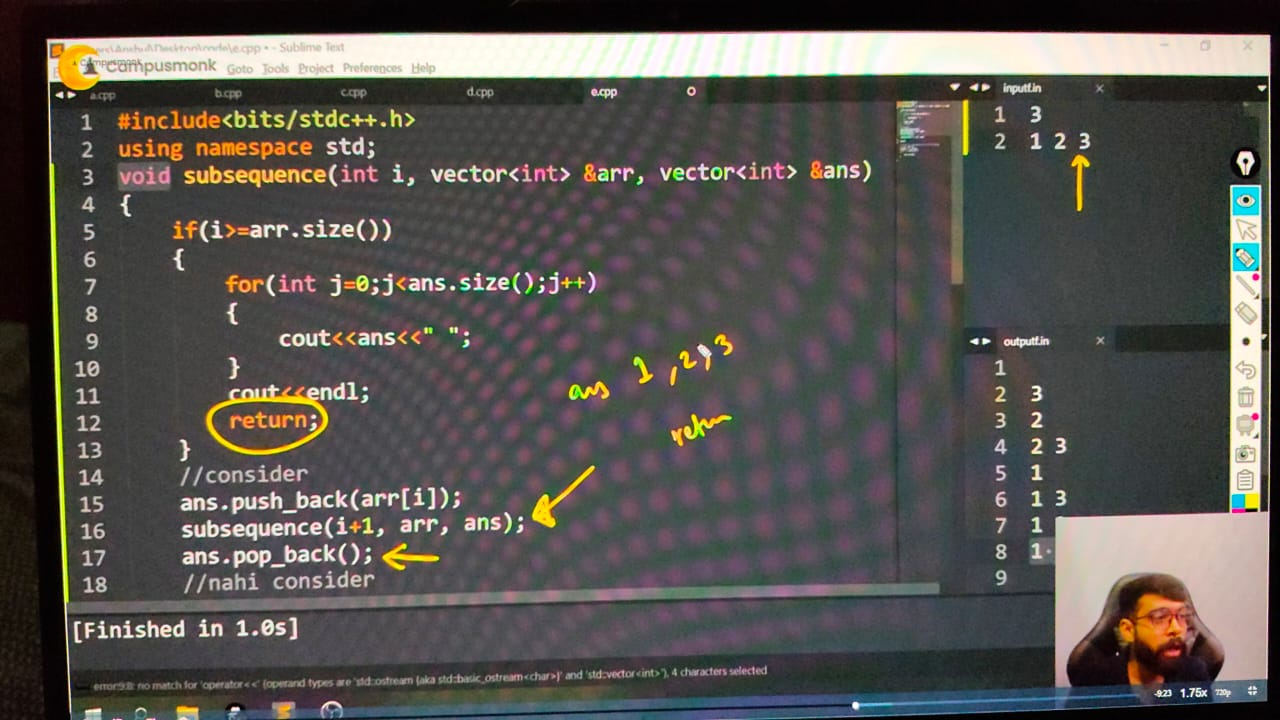
**//  Qun 2 - Print all subsequences of an Array or String**











/\*  Subsequence  - ALl the array eleemnts possible with the same order as well as it can be contigeous

Ex - [1,2,3] - {Empty}, {1},{1 2},{1 3},{1 2 3},{2},{ 2 3},{3}

backtracking - In one case we select something but in the another case we re not selectin g its a Bactracking approach

\*/

// void subsequence(int i, vector<int> &arru, vector<int> &ans)

// {

//     if (i >= arru.size())

//     {

//         for (int j = 0; j < ans.size(); j++)

//         {

//             cout << ans[j] << " ";

//         }

//         cout << endl;

//         return;

//     }

//     // element to be consider

//     ans.push\_back(arru[i]);

//     subsequence(i + 1, arru, ans);

//     ans.pop\_back();

//     // nahi consider krna he

//     subsequence(i + 1, arru, ans);

// }

// int main()

// {

//     int n;

//     cout << "What's your array seize" << endl;

//     cin >> n;

//     vector<int> arru(n);

//     cout << "Mention the elememnts of array you want dude " << endl;

//     for (int i = 0; i < n; i++)

//     {

//         cin >> arru[i];

//     }

//     vector<int> ans;

//     subsequence(0, arru, ans);

//     /\*

//     AGr condition di h ki emepty hoga ki nahi to uske according code likhene. If menetion that Subsequence not be empty then different, if mention can be empty then difeferent

//     so, jb saare elemnets me condition aa rhi ho ki consider krna he ki nhi then rmeeber always use recursion

//     \*/

// /\*

// What's your array seize

// 3

// Mention the elememnts of array you want dude

// 1 2 3

// 1 2 3

// 1 2

// 1 3

// 1

// 2 3

// 2

// 3

// \*/

// }

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

**// Qun - print all subsrtings of a givcen string -**

void subsequence(int i, string &s, string &ans)

{

    if (i >= s.size())

    {

        cout << ans << endl;

        return;

    }

    // Include current character

    ans.push\_back(s[i]);

    subsequence(i + 1, s, ans);

    // Exclude current character

    ans.pop\_back();

    subsequence(i + 1, s, ans);

}

int main()

{

    string s;

    cout << "What is your statement - " << endl;

    getline(cin, s);  // Use getline to accept full statement

    string ans = "";

    subsequence(0, s, ans);

    return 0;

/\*

What is your statement -

abc

abc

ab

ac

a

bc

b

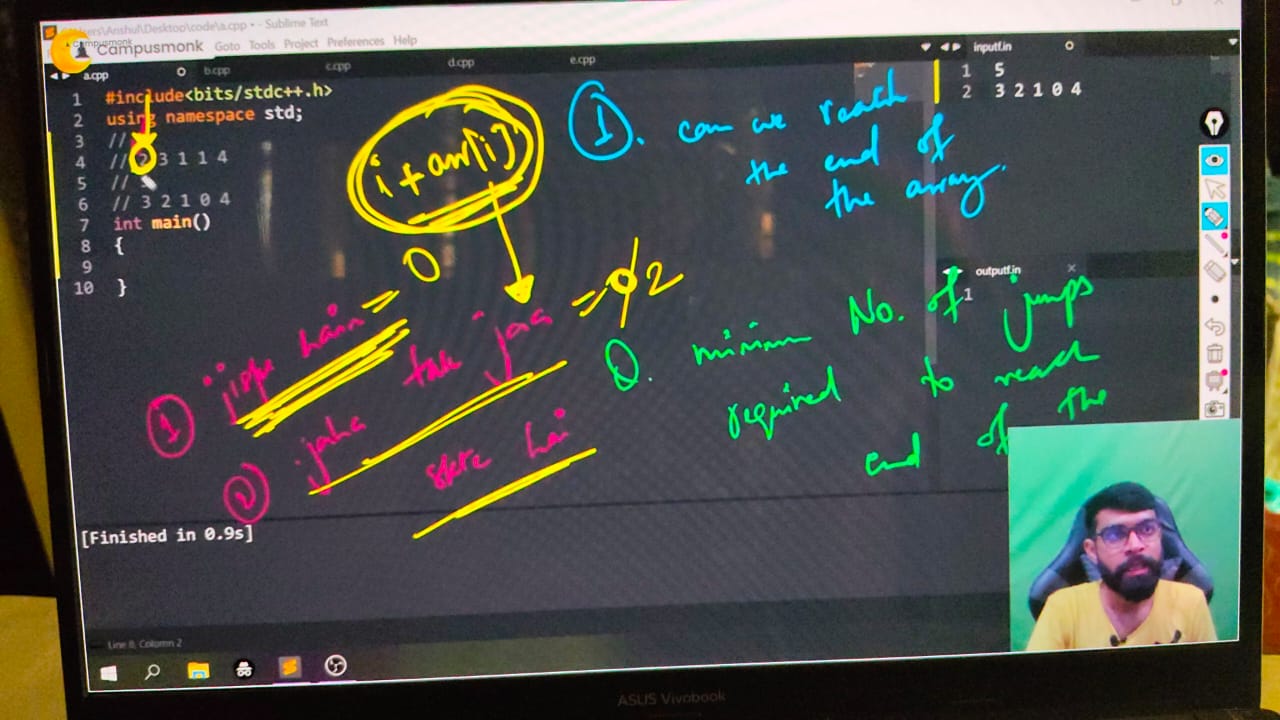
c

\*/

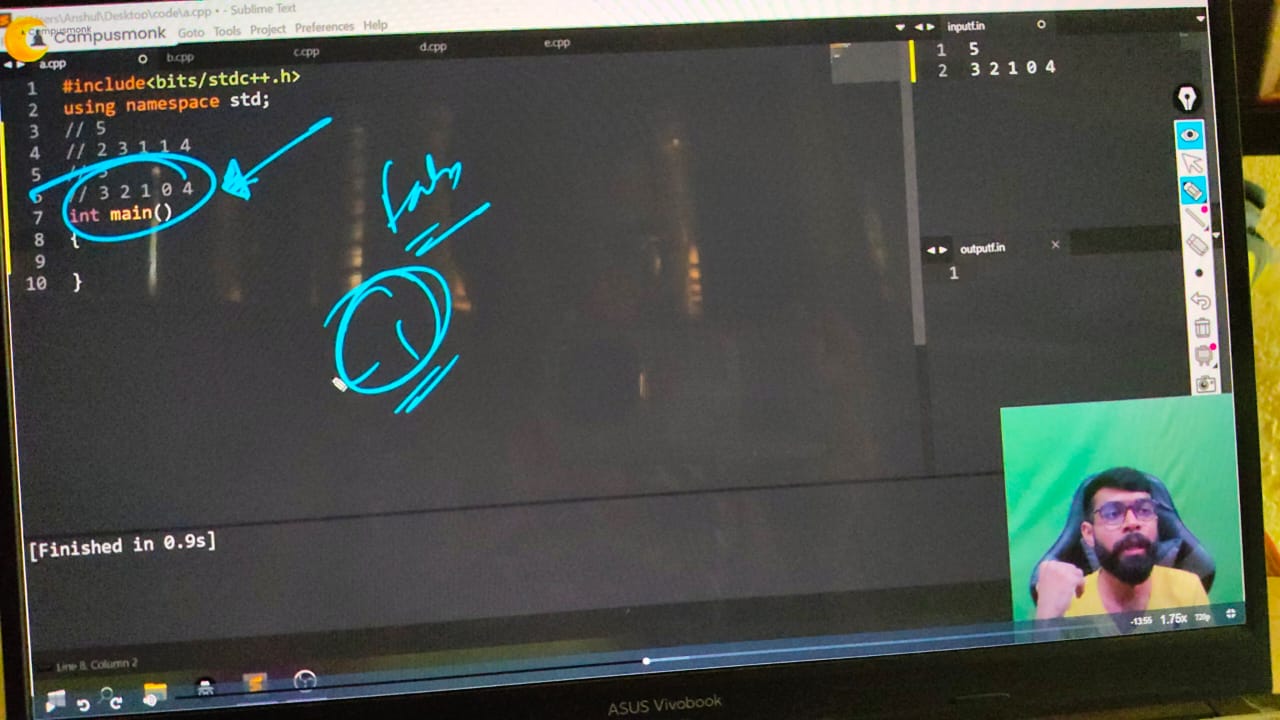
}

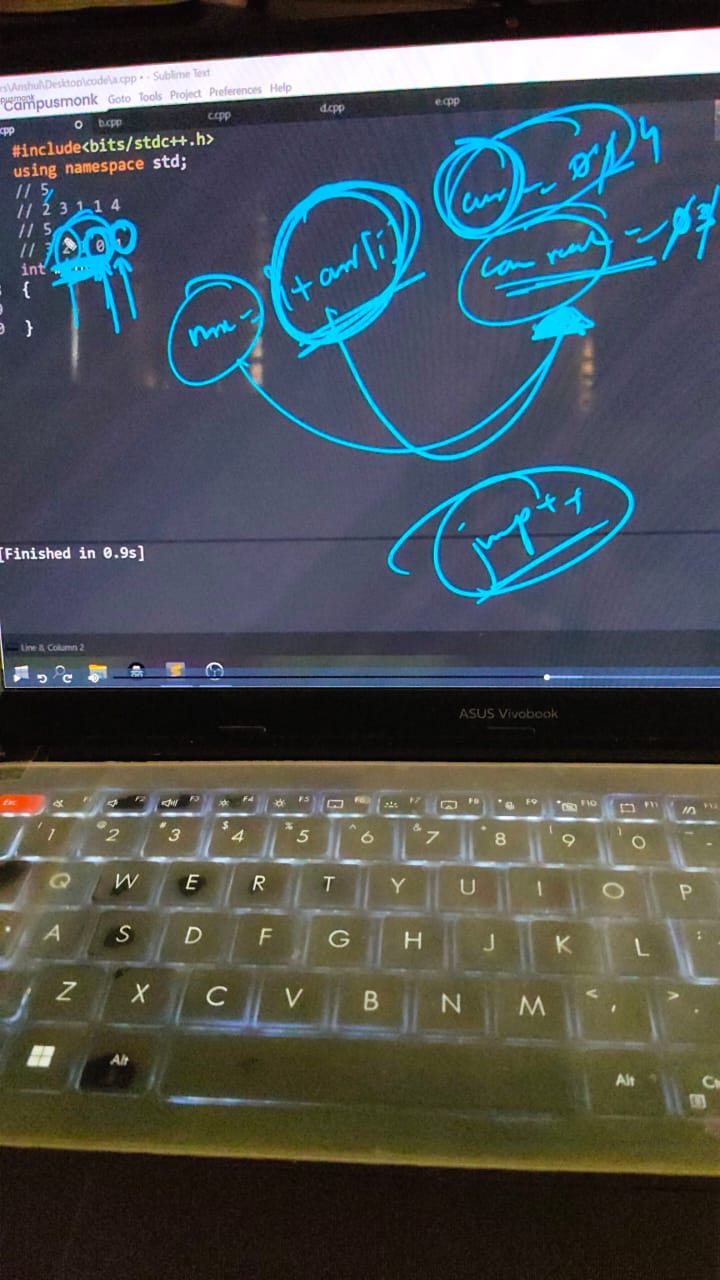
// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**// Qun 3 - Minimum no, of jumps required to reach at the end of the array?  OR Can we reach at the end of the array**









int main()

{

    int n;

    cout << "Menttion the size of array - " << endl;

    cin >> n;

    int arr[n];

    cout << "what are the elements of array - " << endl;

    for (int i = 0; i < n; i++)

    {

        cin >> arr[i];

    }

    int curr = 0, can\_reach = 0;

    int jumps = 0;

    while (can\_reach < n - 1)

    {

        int mx = 0;

        for (int i = curr; i <= can\_reach; i++)

        {

            mx = max(mx, i + arr[i]);

        }

        jumps++;

        curr = can\_reach + 1;

        can\_reach = mx;

        if (can\_reach < curr)

        {

            cout << "No" << endl;

            return 0;

        }

    }

    cout << "YES" << endl;

    cout << jumps << endl;

    /\*

    Menttion the size of array -

    5

    what are the elements of array -

    3 2 1 0 4

    NO

    Menttion the size of array -

    5

    what are the elements of array -

    2 3 1 1 4

    2

    \*/

    // T.C - 2 times travelling for 2 variables so O(2\*n) = O(n) & S.C = O(1)

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

}

**//Qun 4 - Stack: Valid Paranethesis Problem  -**

// bool isValid(string s)

// {

//     stack<char> st;

//     for (int i = 0; i < s.size(); i++)

//     {

//         // oOpening Brackets

//         if (s[i] == '(' || s[i] == '[' || s[i] == '{')

//         {

//             st.push(s[i]);

//         }

//         // Closing - yaha order m,atter kr rha he jo bracket lastr me open huah h vo phle close bhi to hona hi chahiye menas Last Open First Close , so using simple stack aspproaches

//         else

//         {

//             if (st.empty())

//                 return false;

//             if (s[i] == ')')

//             {

//                 if (st.top() != '(')

//                     return false;

//             }

//             else if (s[i] == ']')

//             {

//                 if (st.top() != '[')

//                     return false;

//             }

//             else

//             {

//                 if (st.top() != '{')

//                     return false;

//             }

//             st.pop();

//         }

//     }

//     if (st.size() != 0)

//         return false;

//     return true;

// }

// int main()

// {

//     string s;

//     cout << "Mention the string - " << endl;

//     cin >> s;

//     cout << isValid(s) << endl;

// }

/\*

Mention the string -

([{}])

1

Mention the string -

[{()}](

0

Mention the string -

(){}[]()

1

T.C - O(n) as n size of string

S.C. - O(n) as n size of string

\*/

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