Day 140/180 Stack Hard

1: Minimum of Maximum for every window size.

Given an integer array. The task is to find the minimum of the maximum of every window size in the array. Note: Window size varies from 1 to the size of the Array.

Ex: N = 7 arr[] = {10,20,30,50,10,70,30}
Output: {10,20,30,50,50,70,70}

Explanation:
Maximum of windows of size 1 are {10}, {20}, {30}, {50},{10}, {70} and {30}. Minimum of these maximum is 10.

Maximum of windows of size 2 are {20}, {30}, {50}, {50}, {70}, and {70}. Minimum of these maximum is 20.

The same logic as explained in the video

2: IPL Final

Logic: we have already balanced the parentheses problem, so, instead of storing the parentheses, we're storing the index value of it.
Now, whenever we're popping "(" that means we found out valid parentheses, so, at that point we have to record the answer.

We have to check also if there was a valid one before this. For that, we just have to check what is last invalid one is.

```
int findMaxLen(std::string s) {
    stack<int> st; // Stack to store indices of '(' characters
    int n = s.size();    // Length of the string
    int ans = 0;         // Initialize the maximum length of valid parentheses
substring

    for(int i = 0; i < n; i++) {
        if(s[i] == '(') { // If the current character is '('
            st.push(i);   // Push its index onto the stack
        } else { // If the current character is ')'
            if(st.empty()) { // If stack is empty, no matching '(' found
                st.push(i);  // Push the index of ')' onto the stack
            } else if(s[st.top()] == '(') { // If the top of the stack corresponds
to '('
                int p = st.top(); // Store the index of the matching '('
                st.pop();         // Pop the '(' index from the stack
                if(st.empty()) {  // If stack is empty after popping
                    ans = std::max(ans, i + 1); // Update ans with current index
                } else { // If stack is not empty
                    ans = std::max(ans, i - st.top()); // Update ans with the
difference between current index and top of stack
                }
            } else { // If the top of the stack corresponds to ')'
                st.push(i); // Push the index of ')' onto the stack
            }
        }
    }
    return ans; // Return the maximum length of valid parentheses substring
}
```