# Find The Maximum of A Sliding Window of Size K LeetCode

This program addresses the problem of finding the maximum element within each sliding window of size 'k' in an input array 'nums'.

**Approach 1: Function to find the maximum element in each sliding window of size 'k' using brute force approach.**

**Idea:** In this approach, we calculate the maximum element for each sliding window by iterating through each window and finding the maximum element within that window.

**Algorithm:**

1. Initialize a vector 'ans' to store the maximum elements for all windows.
2. Calculate the number of sliding windows, which is 'n - k + 1', where 'n' is the size of the input array 'nums'.
3. Iterate through each window: a. Initialize 'maxVal' to the first element in the window. b. Iterate through the elements within the current sliding window (from 'i' to 'i + k - 1') and update 'maxVal' if a larger element is found. c. Store 'maxVal' in the 'ans' vector for the current window.
4. Return the 'ans' vector containing maximum elements for all windows.

**Time Complexity: The time complexity of this approach is O(n * k), where 'n' is the size of the input array and 'k' is the window size.**

**Space Complexity: The space complexity is O(1) as it doesn't use additional data structures.**

**Approach 2: Function to find the maximum element in each sliding window of size 'k' using deque-based approach.**

**Idea:** In this approach, we maintain a deque to efficiently track the indices of maximum elements within the current sliding window.

**Algorithm:**

1. Initialize an empty deque 'maxDeque' to store indices of maximum elements.
2. Initialize an empty vector 'ans' to store the maximum elements for all windows.
3. Iterate through the elements of the input array 'nums': a. Remove elements from the front of 'maxDeque' that are out of the current window (indices less than 'i - k + 1'). b. Remove elements from the back of 'maxDeque' that are smaller than the current

element. c. Add the current index 'i' to 'maxDeque'. d. Once the window size reaches 'k', add the maximum element (at the front of 'maxDeque') to 'ans'.

4. Return the 'ans' vector containing maximum elements for all windows.

**Time Complexity: The time complexity of this approach is O(n), where 'n' is the size of the input array.**

**Space Complexity: The space complexity is O(k) as 'maxDeque' stores at most 'k' indices.**