<https://leetcode.com/problems/sliding-window-maximum/>

**Sliding Window Maximum**

**You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right.**

**You can only see the k numbers in the window. Each time the sliding window moves right by one position.**

**Return the max sliding window.**

Example 1:

Input: nums = [1,3,-1,-3,5,3,6,7], k = 3

Output: [3,3,5,5,6,7]

Explanation:

Window position Max

--------------- -----

[1 3 -1] -3 5 3 6 7 3

1 [3 -1 -3] 5 3 6 7 3

1 3 [-1 -3 5] 3 6 7 5

1 3 -1 [-3 5 3] 6 7 5

1 3 -1 -3 [5 3 6] 7 6

1 3 -1 -3 5 [3 6 7] 7

Example 2:

Input: nums = [1], k = 1

Output: [1]

Constraints:

1 <= nums.length <= 105

-104 <= nums[i] <= 104

1 <= k <= nums.length

**Method 1: (Brute Force)**

Number of windows is (n-k) and window size is k

Run loop (n-k) times and in each round find maximum.

Time Complexity: O(n\*k)*[]*

Space Complexity: O(1)

vector<int> maxSlidingWindow(vector<int>& nums, int k) {

vector<int> res;

int j, max, n=nums.size();

for (int i = 0; i <= n - k; i++) {

max = nums[i];

for (j = 1; j < k; j++) {

if (nums[i + j] > max)

max = nums[i + j];

}

res.push\_back(max);

}

return res;

}

**Method 2: (Using deque)**

Use deque to store only useful elements (useful in computing max)

Deque=[Max, 2ndmax, 3max…..]

If new element is greater than dq.front : pop\_front and push-front new element till back reached.

Time Complexity: O(n) *[]*

Space Complexity: O(k) *[]*

vector<int> maxSlidingWindow(vector<int>& nums, int k) {

        vector<int> res;

        int n=nums.size();

        deque<int> dq;

        for (int i = 0; i <k; i++) {

            while(!dq.empty() && dq.front()<nums[i])

                dq.pop\_front();

            dq.push\_front(nums[i]);

        }

        res.push\_back(dq.back());

        for (int i = k; i <n; i++) {

            if(dq.back()==nums[i-k])

                dq.pop\_back();

            while(!dq.empty() && dq.front()<nums[i])

                dq.pop\_front();

            dq.push\_front(nums[i]);

            res.push\_back(dq.back());

        }

        return res;

    }