<https://medium.com/@zengruiwang/sliding-window-technique-360d840d5740>

Sliding window technique is useful for solving problems in array or string, especially it is considered as a technique that could reduce the time complexity from O(n²) to O(n).

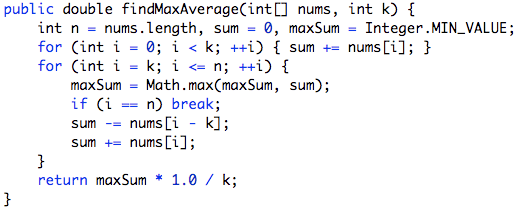
**There are two types of sliding window:**

1. Fixed window length k: the length of the window is fixed and it asks you to find something in the window such as the maximum sum of all windows, the maximum or median number of each window. Usually we need kind of variables to maintain the state of the window, some are as simple as a integer or it could be as complicated as some advanced data structure such as list, queue or deque.
2. Two pointers + criteria: the window size is not fixed, usually it asks you to find the subarray that meets the criteria, for example, given an array of integers, find the number of subarrays whose sum is equal to a target value.

**Some problems**

*1. Given an array consisting of n integers, find the contiguous subarray of given length k that has the maximum average value. And you need to output the maximum a`verage value.*

Solution: the problem falls into the first type of sliding window problems, since the length of the window is fixed. We find the first window with size k and maintain a variable curSum which equals the sum of all the elements within the current window and a global maxSum which is the maximum sum among all the windows we have examined. As we moving the window one step at a time from left to right, we subtract the leftmost element in the current window and add the next element of the array until we hit the end of the array. Finally, we return the maxSum / k which is the maximum average among all the windows of length k.



The above question is pretty straightforward and easy since we only needs to maintain a variable of the window’s state. As indicated in the description, some of the problem could be very tricky and involves more advanced data structures or approach to maintain the window’s state.

*2. Given an array 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. Your job is to output the median array for each window in the original array.*

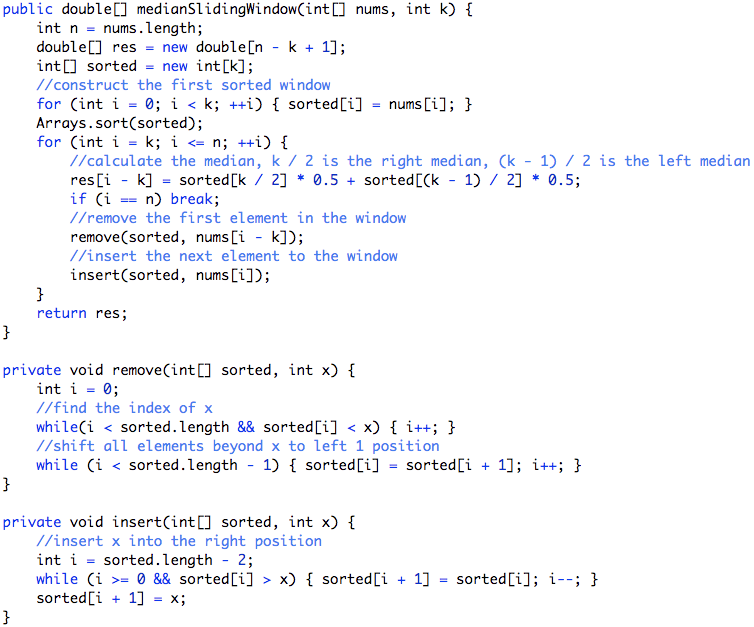
Solution: this problem also falls into the first type of sliding window technique, but it needs more thought on how we could maintain the state of the window. A naive approach would be as the window moves from left to right, we can sort the numbers within the window and find the median. The time complexity of it is O( (n — k) \* klogk). Notice, the naive approach does not maintain any state of the sliding window, every time we move the window, we sort the window like a new array of numbers. Thus, we maintain an array of size k which is always sorted of all the element within the window. Also notice, once we find the first window, and as we remove the oldest element from the window and add the next element, it is almost sorted, we just need to fix the oldest element removal and new element insertion.This could be further divided into: removal and insertion, which could be done in O(k) time complexity and the overall complexity could be reduced to O ((n — k) \* k).

The removal is easy, we only need to find the oldest element in the window, remove it and move all the elements on its right one step left. The insertion is easy too, just like insertion sort, we start from the end of the sorted part and move all the element that is larger than the target one step right.

There is an even harder problem below, but the idea and backbone are the same.

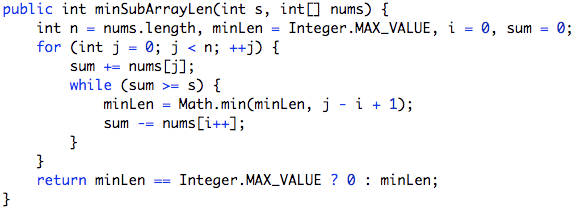
<https://leetcode.com/problems/sliding-window-maximum/description/>

As you compare the code with the previous problem, the part within the for loop is almost the same!



3. *Given an array of n positive integers and a positive integer s, find the minimal length of a contiguous subarray of which the sum ≥ s. If there isn’t one, return 0 instead.*

Solution: this problem falls into the second category, since the window size is not fixed but it asks you to find a subarray that meets the criteria *sum ≥ s* . To solve it, we can apply the two pointer technique to find the first window that meets the criteria and then moves the start pointer one step ahead until the criteria is no longer true, during which process, we also update a global min\_len variable which records the minimum subarray length that satisfies the criteria.



4. *Given a string, find the length of the longest substring T that contains at most k distinct characters.*

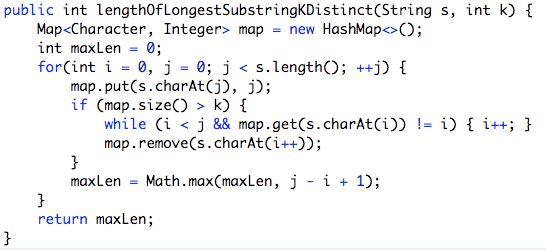
Solution: this problem also falls into the second category since it asks to find maximum length of substring that meets the criteria that contains at most k distinct characters. Again, we can apply two pointers to find the first window that meets the criteria, but how would we process to the next window? In other words, how would we determine the starting point of the next window? Let’s look at some examples:

[e,c,e,b,a], k = 2, the first window is [e, c, e], the next starting index should be 2

[e,c,c,b,a], k = 2, the first window is [e, c, c], the next starting index should be 1

Observation: we should remove the oldest element from the window.

Thus, we can maintain a Hash Table, where the key is the character and the value is the last index of in the string. Everytime the size of the hash table is larger than k when know that we need to remove the oldest element from the map and start from its next index. How do we find the oldest element? Since we update the index of a character every time we see it, thus we just need to scan from the start till the end and find the first map.get(s.charAt(i)) == i.



A even harder problem in this category:

<https://leetcode.com/problems/minimum-window-substring/description/>