**Assumptions:**

1. Expecting only file stream as input stream
2. If we have a NaN within a window, we cannot calculate mean or max until it leaves the queue, so the position of NaN in stream should affect the calculation of mean and max as per the location of it in stream.
3. If we hit a NaN from stream, we need to recalculate old average and old Max if we depends on old mean and max values to calculate new max and new average.
4. Keeping separate queue for each window size, so we can process results for each window size while processing stream once.

**Space and Time complexity**

If we consider 'm' is the window size, this algorithm holds time complexity of O(m) in worse case scenario to find the max value out of each windows, so total time complexity to find max value from all windows is O(nm). If 'l' is the number of windowSizes we are passing to the program, this algorithm needs O(l\*(m+2)) space complexity.

Avg of each windows is calculated in O(n) with additional space of O(2).

**Improvements**

If we have option to pre-process data and ignore NaN records and calculate mean and max irrespective of order or position of NaN records in stream, we can considerably improves the algorithm performance in case of time and space complexity.

We can create a sorted queue using std::deque and can achieve time complexity of O(n) to find the maximum value out of each windows with amortized complexity of O(1) for deletion and insertion. This approach is explained in below article:[https://people.cs.uct.ac.za/~ksmith/articles/sliding*window*minimum.html#sliding-window-minimum-algorithm](https://people.cs.uct.ac.za/~ksmith/articles/sliding_window_minimum.html#sliding-window-minimum-algorithm)

**Window length change to 10,000 or 1 Million elements**

1. My current program can work without any change for any window Size. It can accept any number of window sizes as input parameter.
2. Check the feasibility to clean up data by removing NaN records.
3. Definitely try to implement the improvement suggested above so we can decrease the time complexity to find the maximum to O(N)

**Median evaluation**

I have 2 approaches in mind:

1. Use std::nth\_element algorithm over the the std::deque and find the median, if size of queue grows this approach may not be suitable and this may not be a scalable solution.
2. Create a minimum and maximum heap to find the median, details are explained in this stackoverflow answer: <http://stackoverflow.com/questions/10657503/find-running-median-from-a-stream-of-integers>