Find Median from Running Data Stream

Given that integers are read from a data stream. Find the median of elements read so far in an efficient way.

There are two cases for median on the basis of data set size.

- If the data set has an odd number then the middle one will be consider as median.
- If the data set has an even number then there is no distinct middle value and the median will be the arithmetic mean of the two middle values.

Example:

Input Data Stream: *5, 15, 1, 3*

Output: *5, 10,5, 4*

Explanation*:*

After reading 1st element of stream – 5 -> median = 5

After reading 2nd element of stream – 5, 15 -> median = (5+15)/2 = 10

After reading 3rd element of stream – 5, 15, 1 -> median = 5

After reading 4th element of stream – 5, 15, 1, 3 -> median = (3+5)/2 = 4

Input Data Stream*: 2, 2, 2, 2*

Output*: 2, 2, 2, 2*

Explanation*:*

After reading 1st element of stream – 2 -> median = 2

After reading 2nd element of stream – 2, 2 -> median = (2+2)/2 = 2

After reading 3rd element of stream – 2, 2, 2 -> median = 2

After reading 4th element of stream – 2, 2, 2, 2 -> median = (2+2)/2 = 2

Find Median from Running Data Stream using **Insertion Sort**:

*If we can sort the data as it appears, we can easily locate the median element. Insertion Sort is one such online algorithm that sorts the data appeared so far. At any instance of sorting, say after sorting i-th element, the first **i** elements of the array are sorted. The insertion sort doesn't depend on future data to sort data input till that point. In other words, insertion sort considers data sorted so far while inserting the next element. This is the key part of insertion sort that makes it an online algorithm.*

However, insertion sort takes $O(n^2)$ time to sort n elements. Perhaps we can use binary search on insertion sort to find the location of the next element in O(log n) time. Yet, we can't do data movement in O(log n) time. No matter how

efficient the implementation is, it takes polynomial time in case of insertion sort.

```
// C++ code to implement the approach

#include <bits/stdc++.h>
using namespace std;

// Function to find the median of stream of data
void streamMed(int A[], int n)
{
	// Declared two max heap
	priority_queue<int> g, s;

	for (int i = 0; i < n; i++) {
		s.push(A[i]);
		int temp = s.top();
		s.pop();

		// Negation for treating it as min heap
		g.push(-1 * temp);
		if (g.size() > s.size()) {
			temp = g.top();
			g.pop();
			s.push(-1 * temp);
		}
		if (g.size() != s.size())
			cout << (double)s.top() << "\n";
		else
```

```
                        cout << (double)((s.top() * 1.0
                                                - g.top() * 1.0)
                                                / 2)
                             << "\n";
        }
}

// Driver code
int main()
{
        int A[] = { 5, 15, 1, 3, 2, 8, 7, 9, 10, 6, 11, 4 };
        int N = sizeof(A) / sizeof(A[0]);

        // Function call
        streamMed(A, N);
        return 0;
}
```

Time Complexity: O(n * log n), All the operations within the loop (push, pop) take O(log n) time in the worst case for a heap of size N.
Auxiliary Space: O(n)