Link to the problem

https://leetcode.com/problems/find-median-from-data-stream/

Find the median of a number stream

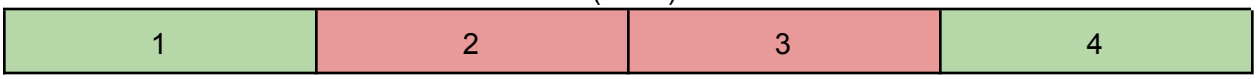
Understanding the problem

What does this problem want? To answer this question, we must first answer another question, what is the median of an array of numbers?

The median is the **middle number in a sorted array of numbers**. If there is an even number of numbers in the array, the median is the **average of the middle two numbers**.

Median = 3

Median = (2 + 3) / 2 = 2.5

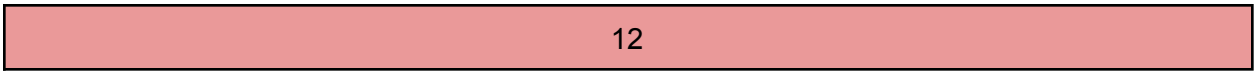


Another way of looking at the median is: Suppose we have N elements, the median is **the number that is greater than N/2 elements and less than N/2 elements.**

According to the question's requirements, we're going to keep receiving new numbers one by one, and every time we receive a number we want to calculate the median of all the numbers we've received up to that point in time.

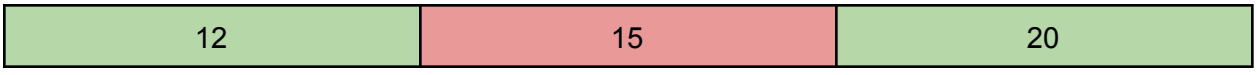
For example, suppose we receive the following sequence of numbers::

We received the number "12". Median = 12



We received the number "15". Median = (12 + 15) / 2 = 13.5

We received the number "20". Median = 15



And so on.

Coming up with an idea for a solution

This problem follows the 2 heaps pattern, we're going to be using 2 heaps to solve this. How does this pattern fit here? I'm going to discuss below how a max-heap fits, but the same logic works for a min-heap as well.

From the definition of a max-heap, we know that the top element in the heap is the largest element in that heap. Every other element in the heap is less than or equal to that element.

Suppose we have received "N" numbers from the stream so far. "N" is an odd number.

If I had two heaps, one max-heap and one min-heap and we filled the min-heap with a random collection of N/2 elements and filled the max-heap with the remaining (N/2 + 1) elements. The top element of the max-heap is guaranteed to be greater than every other element in that heap. In other words, it is guaranteed to be greater than N/2 elements. Of course, the same logic applies to the min-heap. From the earlier definition of a median, is this what we're looking for?

The answer is no. There is one last thing we need to pay attention to in this problem.

Making the solution work

The only thing that the max-heap effect guarantees is that the top element is greater than N/2 elements. Let's see how many numbers fit this condition in a normal, sorted array. These are all the elements that fulfill the guarantee of being greater than N/2 elements:

Every one of the purple elements has at least 3(=N/2) elements less than itself.

Meanwhile, the element we're looking for is just this one element:

Median = 4

This is why we need to change our algorithm. We want the median, which is the number that is greater than EXACTLY N/2 elements. There is only one number that fulfills such a tight bound and it is the median we're looking for.

So how do we change our algorithm to give us exactly this number and not any other number that simply fulfills the max-heap property?

Let's try to take an example step by step and manually guarantee that the number that is greater than N/2 elements and less than N/2 elements remains on top of the heap.

Going through an example

I'm going to use the following stream of numbers:
10 => 16 => 4 => 2 => 8 => 20

We're going to pick a primary heap to focus our attention on and a secondary heap. We can pick either heap as the primary/secondary heap. I'm going to choose the max-heap as my primary heap and the min-heap as my secondary heap. We're going to focus on:

1) Distributing elements equally into both heaps. The difference in heap sizes must always be either 0, or 1 in favor of the primary heap.
2) The top element of the max-heap has to be at all times greater than every element in its heap and less than every element in the min-heap. If the top element of the max-heap fails this condition we need to replace it with a number that fulfills the condition. We're going to fill any numbers less than it into the max-heap and any numbers greater than it into the min-heap.

Step 1: Both heaps are empty right now.

Step 2: Receive the first number. The first number is "10". Since I'm focusing on the max-heap, I'm going to insert it into that heap. (The top element of each heap will be colored in blue.)

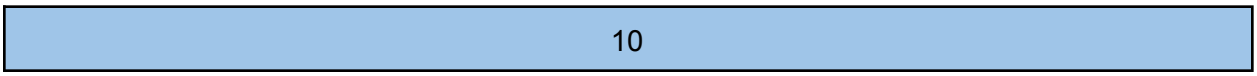
Max heap
Top element = 10.

Min heap
Heap is currently empty.

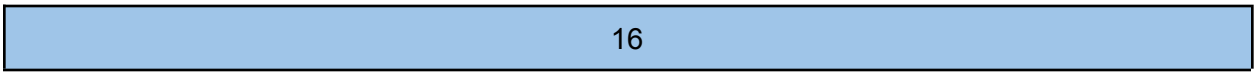
Median from both heaps = 10.

Step 3: Receive the second number. The second number is “16”. This number is greater than the top of the max-heap. Let’s insert it into the min-heap. Do the conditions still hold? Yes they do. No need to do anything else. 10 is still perfectly fine as the top of the primary heap.

Max heap
Top element = 10.



Min heap
Top element = 16.

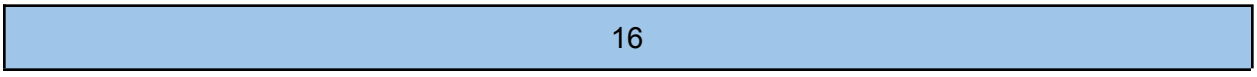


Median from both heaps = (10 + 16) / 2 = 13.

Step 4: Receive the third number. The third number is “4”. This number is less than the top of the max-heap. We can insert it there. The conditions still hold. We don’t need to do anything.

Max heap
Top element = 10.

Min heap
Top element = 16.



Median from both heaps = 10.

Step 5: Receive the fourth number. The fourth number is “2”. This number is less than the top of the max-heap. We can insert it there. However, after we insert it there, the difference in heap

sizes is now 2. We need to do something to fix this. We can fix this by balancing the heaps. We do this by popping the top element of the max-heap and inserting it into the min-heap. Now 4 serves as the new top of the primary heap, and it fulfills the conditions we're looking for.

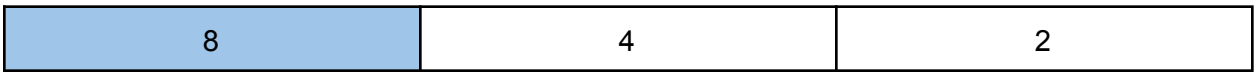
Max heap
Top element = 10.

Min heap
Top element = 10.

Median from both heaps = (4 + 10) / 2 = 7.

Step 6: Receive the fifth number. The fifth number is "8". This number is greater than the top of the max-heap. We must insert it into the min-heap. However, by doing this the difference in heap sizes will be 1 in favor of the min-heap, not the max-heap. We need to do something to fix this. We can do this by popping one element from the min-heap and adding it to the max-heap.

Max heap
Top element = 8.

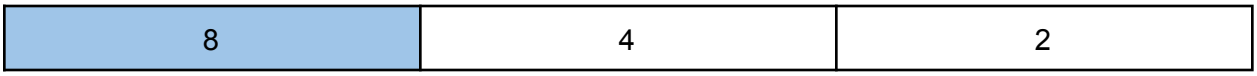


Min heap
Top element = 10.

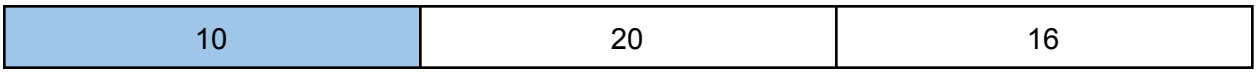
Median from both heaps = 8.

Step 7: Receive the final number. The final number is "20". This number is greater than the top of the max-heap. We must insert it into the min-heap. The conditions still hold, we don't need to do anything else.

Max heap
Top element = 8.



Min heap
Top element = 10.



Median from both heaps = (8 + 10) / 2 = 9.