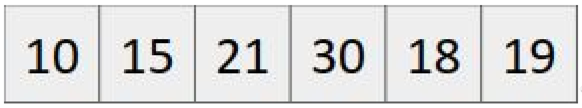
# Median of Running Stream

Problem
Stream of numbers are coming and we have to tell median after each input.

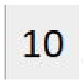
Example:
Input: Given an array



| | | | | | |
|---|---|---|---|---|---|
| 10 | 15 | 21 | 30 | 18 | 19 |

- <u>After first input</u>

Sorted array

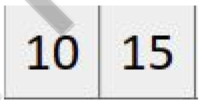


| |
|---|
| 10 |

Median: 10

- <u>After second input</u>

Sorted array

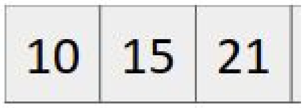


| | |
|---|---|
| 10 | 15 |

Median = (10+15)/2 = 12.5

- <u>After third input</u>

Sorted array

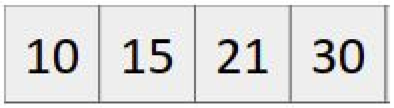


| | | |
|---|---|---|
| 10 | 15 | 21 |

Median = 15

- <u>After fourth input</u>

Sorted array



| | | | |
|---|---|---|---|
| 10 | 15 | 21 | 30 |

Median = (15+21)/2 = 18.5

- <u>After fifth input</u>

Sorted array

| | | | | |
|---|---|---|---|---|
| 10 | 15 | 18 | 21 | 30 |

Median = 18

- <u>After sixth input</u>

Sorted array

| | | | | | |
|---|---|---|---|---|---|
| 10 | 15 | 18 | 19 | 21 | 30 |

Median = (18+19)/2 = 18.5

Brute force approach

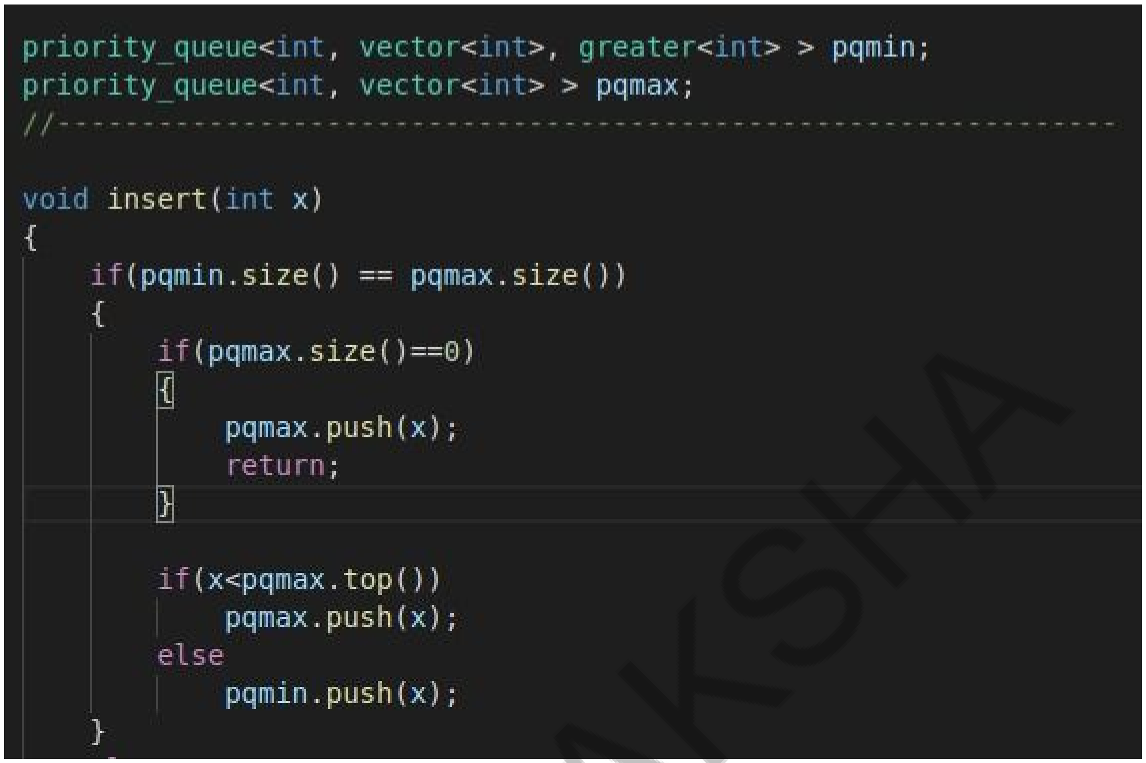
1. Maintain a vector (say temp).
2. After each input, push_back the element into temp and sort it.
3. If current size of temp is odd then output the middle element else output the average of middle two elements.

Time Complexity: O($n^2$log(n))

Optimized approach (Using heaps)

1. Create two heaps. One MaxHeap to maintain elements of lower half and one MinHeap to maintain elements of higher half at any instant.
2. For every newly read element, insert it into either MaxHeap or MinHeap and calculate the median based on the following conditions:
    - If the size of MaxHeap is greater than the size of MinHeap and the element is greater than the top element of the MaxHeap then pop the top element from MaxHeap and insert into MinHeap and insert the new element to MaxHeap else insert the new element to MinHeap. Calculate the new median as the average of top elements of both MaxHeap and MinHeap.
    - If the size of MaxHeap is less than the size of MinHeap and the element is greater than the top element of MinHeap then pop the top element from MinHeap and insert into the MaxHeap and insert the new element to MinHeap else insert the new element to the MaxHeap. Calculate the new median as the average of top of elements of both max and MinHeap.
    - If the size of both heaps is the same, insert the element into the MaxHeap.

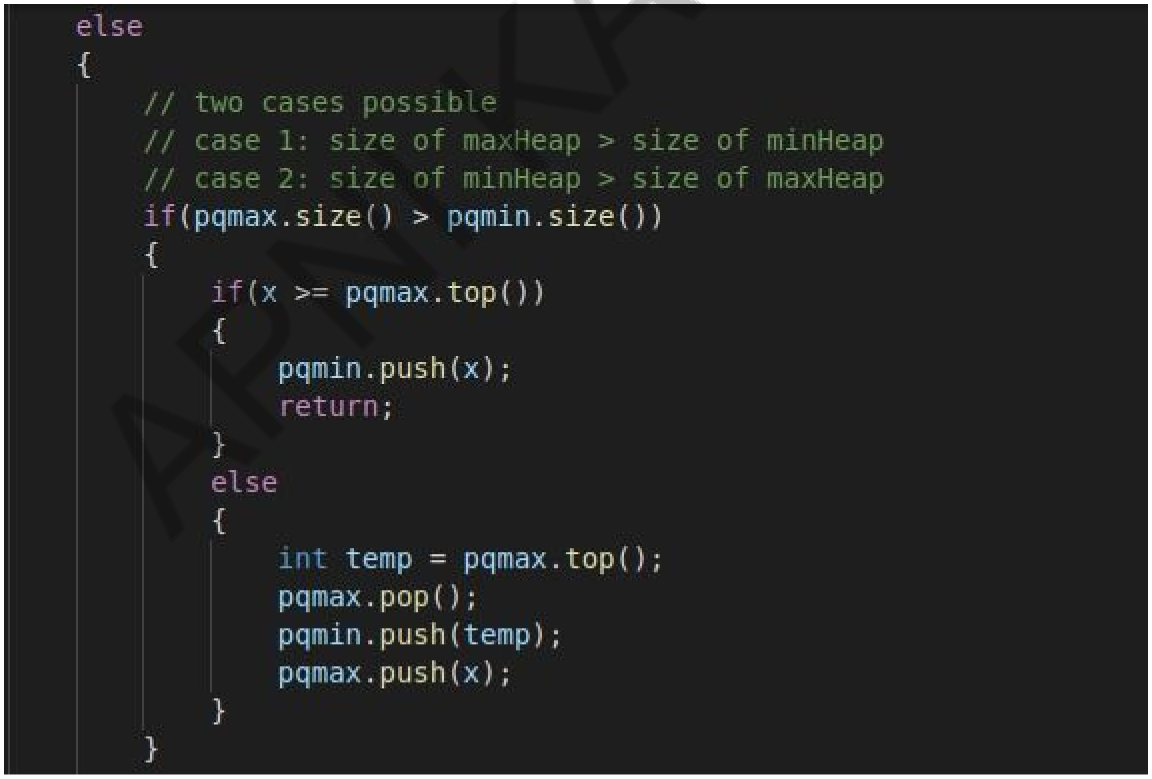
Code



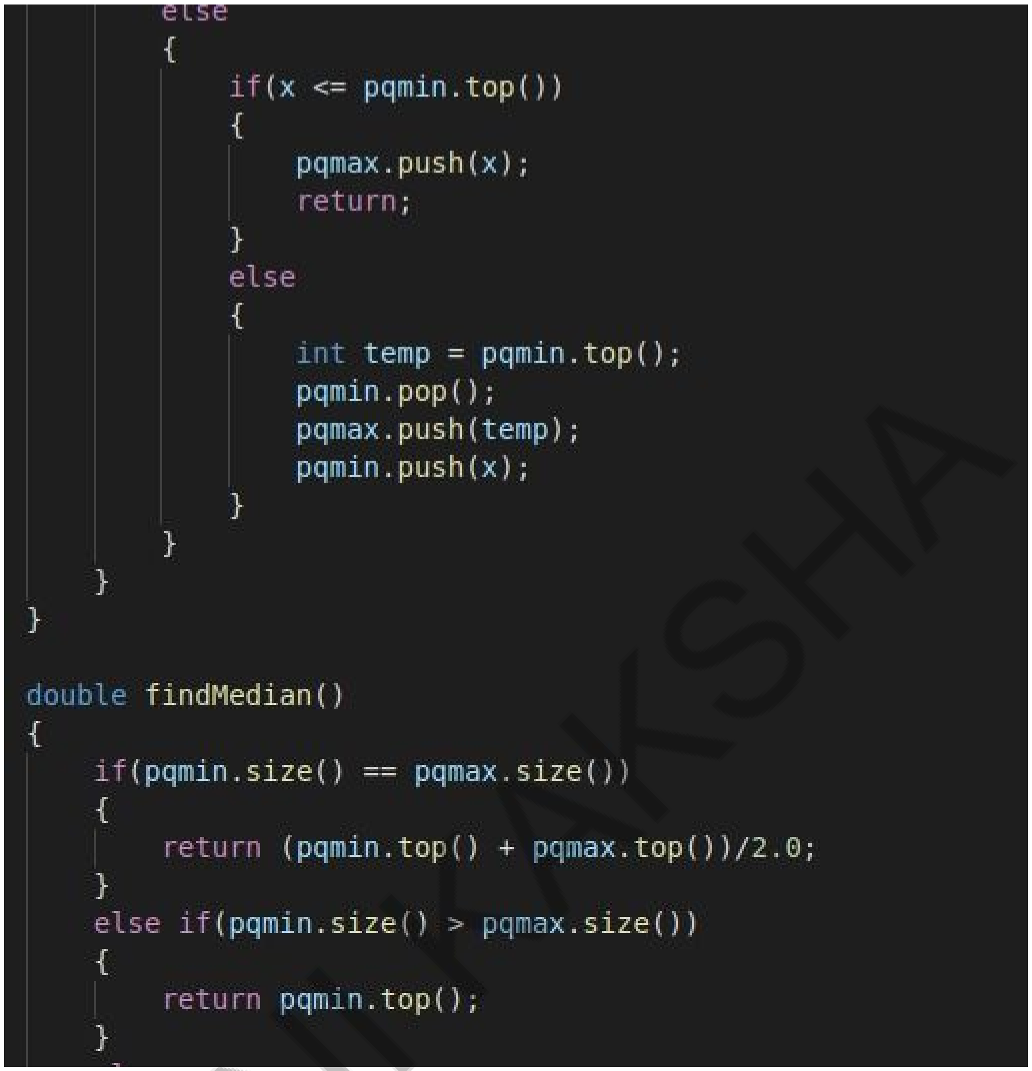
```cpp
priority_queue<int, vector<int>, greater<int> > pqmin;
priority_queue<int, vector<int> > pqmax;
//-----------------------------------------------------------------

void insert(int x)
{
    if(pqmin.size() == pqmax.size())
    {
        if(pqmax.size()==0)
        {
            pqmax.push(x);
            return;
        }

        if(x<pqmax.top())
            pqmax.push(x);
        else
            pqmin.push(x);
    }
```

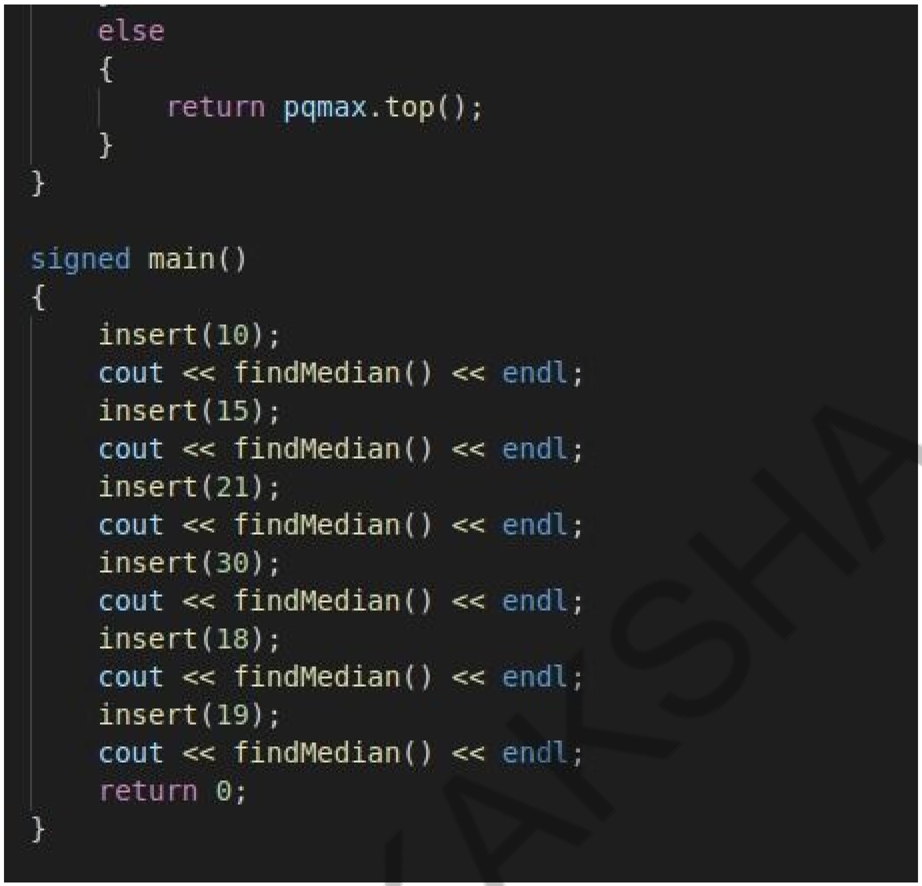


```cpp
    else
    {
        // two cases possible
        // case 1: size of maxHeap > size of minHeap
        // case 2: size of minHeap > size of maxHeap
        if(pqmax.size() > pqmin.size())
        {
            if(x >= pqmax.top())
            {
                pqmin.push(x);
                return;
            }
            else
            {
                int temp = pqmax.top();
                pqmax.pop();
                pqmin.push(temp);
                pqmax.push(x);
            }
        }
```



```
        else
        {
            if(x <= pqmin.top())
            {
                pqmax.push(x);
                return;
            }
            else
            {
                int temp = pqmin.top();
                pqmin.pop();
                pqmax.push(temp);
                pqmin.push(x);
            }
        }
    }
}

double findMedian()
{
    if(pqmin.size() == pqmax.size())
    {
        return (pqmin.top() + pqmax.top())/2.0;
    }
    else if(pqmin.size() > pqmax.size())
    {
        return pqmin.top();
    }
```



```
    else
    {
        return pqmax.top();
    }
}

signed main()
{
    insert(10);
    cout << findMedian() << endl;
    insert(15);
    cout << findMedian() << endl;
    insert(21);
    cout << findMedian() << endl;
    insert(30);
    cout << findMedian() << endl;
    insert(18);
    cout << findMedian() << endl;
    insert(19);
    cout << findMedian() << endl;
    return 0;
}
```