

```

#include <iostream>
#include <set>
#include <deque>
#include<unordered_map>
#include<unordered_set>

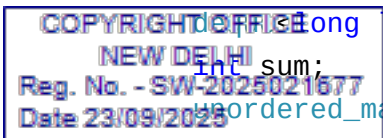
using namespace std;

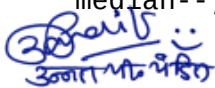
class MedianQueue {
private:
    multiset<long long> s;
    multiset<long long>::iterator median;
    deque<long long> q; // Keeps insertion order
    int sum;
    unordered_map<long long,int>freq;
    unordered_map<int,unordered_set<long long>>bucket;
    int mxFreq;
public:
    MedianQueue(){
        sum = 0;
        mxFreq = 0;
    }

    void insert(long long x) {
        s.insert(x);
        q.push_back(x); // Track insertion order
        sum += x;
        int f = ++freq[x];
        mxFreq = max(f,mxFreq);
        if(f > 1){
            bucket[f-1].erase(x);
        }
        bucket[f].insert(x);

        if (s.size() == 1) { // initially container is empty so set median
            to first and only element.
            median = s.begin();
        } else if (x < *median) {
            if (s.size() % 2 == 0)
                median--;
        }
    }
};

```




 अनामिका शर्मा

```

    } else {
        if (s.size() % 2 != 0)
            median++;
    }
}

```

```

void pop() {
    if (q.empty()) // can't pop empty container
        throw runtime_error("MedianQueue is empty");

```

```

    // Store size of s before removal.

```

```

    int n = s.size();

```

```

    long oldest = q.front(); // Fetching oldest inserted element
    in this container

```

```

    q.pop_front(); // Remove that from deque

```

```

    sum -= oldest;

```

```

    int f = --freq[oldest];

```

```

    bucket[f+1].erase(oldest);

```

```

    if(bucket[mxFreq].empty()){
        mxFreq--;
    }

```

```

//below code is for median pointer shifting

```

```

// Find the iterator for that oldest element in the multiset.

```

```

auto it = s.find(oldest);

```

```

if (it == s.end()) return; // just for safety, but oldest will
definitely present in the multiset

```

```

// Case 1: The element to be removed is exactly the median.

```

```

if (it == median) {

```

```

    // Erase the median and get the iterator to the next element.

```

```

    auto newMed = s.erase(it); // erase() returns iterator

```

```

following the erased element.

```

```

    if (s.empty()) return; // We've removed the only element.

```

```

    // For odd n, after removal new size becomes even.

```

```

    // Our invariant for even sizes is that median should point to

```

```

    median.

```





 Dr. Ravi Singh

```
// However, erase() gives the higher one, so we move one step back (if possible).
```

```
if (n % 2 == 1) {  
    if (newMed != s.begin())  
        median = prev(newMed);
```

```
    else
```

```
        median = newMed; // this step looks doubt ful and redundant, (for testing start taking array of size 3)
```

```
    }
```

```
// For even n, new size becomes odd, so taking newMed works as the median.
```

```
    else {
```

```
        median = newMed;
```

```
    }
```

```
// Case 2: The removed element is in the left partition (i.e. less than the median).
```

```
    else if (oldest < *median) {
```

```
        s.erase(it);
```

```
// For even n (before removal), new size is odd; the median should move right.
```

```
    if (n % 2 == 0)  
        median++;
```

```
    }
```

```
// Case 3: The removed element is in the right partition.
```

```
    else { // oldest > *median
```

```
        s.erase(it);
```

```
// For odd n (before removal), new size becomes even; median should move left.
```

```
    if (n % 2 == 1)  
        median--;
```

```
    }
```

```
}
```

```
long long peek() const {
```

```
    if (q.empty()) throw runtime_error("MedianQueue is empty");
```

```
    return q.front();
```

```
}
```



```
ng getMedian() const {
```

30/09/25
अनंत भंडारी

```

        if (s.empty()) // can't pop empty container
            throw runtime_error("MedianQueue is empty");
        return s.empty() ? -1 : *median;
    }
    long long getSum() const {
        if (s.empty()) // can't pop empty container
            throw runtime_error("MedianQueue is empty");
        return sum;
    }
    double getMean() const {
        if (s.empty()) // can't pop empty container
            throw runtime_error("MedianQueue is empty");
        return (sum/static_cast<int>(s.size()));
    }
    long long getMod() {
        if (s.empty()) // can't pop empty container
            throw runtime_error("MedianQueue is empty");
        return *bucket[mxFreq].begin();//return the first available element
        with max frequency
    }
    long long getMin() const {
        if (s.empty()) // can't pop empty container
            throw runtime_error("MedianQueue is empty");
        return *s.begin();
    }
    long long getMax() const {
        if (s.empty()) // can't pop empty container
            throw runtime_error("MedianQueue is empty");
        return *prev(s.end());
    }
    void dumpState() { //refresh the container to new fresh one
        s = multiset<long long>();
        q = deque<long long>();
        freq = unordered_map<long long,int>();
        bucket = unordered_map<int,unordered_set<long long>>();
        sum = 0;
        mxFreq = 0;
        median = s.end(); // Reset the median pointer since the set is now

```

COPYRIGHT OFFICE
 NEW DELHI
 Reg. No. - SW-2025021677
 Date 23/09/2025



30/09/25
 अनामिका सिंह

```
}

size_t size() const {
    return s.size();
}

bool empty() const { //to check if container is empty or not
    return s.empty();
}

};

// End of file
// Author: Shashank Vashistha
```

COPYRIGHT OFFICE
NEW DELHI
Reg. No. - SW-2025021677
Date 23/09/2025



अमित
अमिताभ अक्षर