**Problem**: Given an extremely busy server which receives thousands of requests per second.  
It is desired to find the number of requests received in the last second, minute and hour.  
What algorithm and data-structures can be used to do this as accurately as possible?

**If 100% accuracy is required:**

Have a linked-list of all requests and 3 counts - for the last hour, the last minute and the last second.

You will have 2 pointers into the linked-list - for a minute ago and for a second ago.

An hour ago will be at the end of the list. Whenever the time of the last request is more than an hour before the current time, remove it from the list and decrement the hour count.

The minute and second pointers will point to the first request that occurred after a minute and a second ago respectively. Whenever the time of the request is more than a minute / second before the current time, shift up the pointer and decrement the minute / second count.

When a new request comes in, add it to all 3 counts and add it to the front of the linked-list.

Requests for the counts would simply involve returning the counts.

All of the above operations are constant time.

**If less than 100% accuracy is acceptable:**

Since the above is already constant time, you can't get anything to be much quicker than that. However, the space-complexity could be a bit much, depending on how many requests per second you would typically get; you can reduce this by sacrificing slightly on accuracy as follows:

Have a linked-list as above, but only for the last second. Also have the 3 counts.

Then have a circular array of 60 elements indicating the counts of each of the last 60 seconds. Whenever a second passes, subtract the last (oldest) element of the array from the minute count and add the last second count to the array.

Have a similar circular array for the last 60 minutes.

Loss of accuracy: The minute count can be off by all the requests in a second and the hour count can be off by all the requests in a minute.

Obviously this won't really make sense if you only have one request per second or less. In this case you can keep the last minute in the linked-list and just have a circular array for the last 60 minutes.

There are also other variations on this - the accuracy to space used ratio can be adjusted as required.

Why not just use a circular array? We have 3600 elements in that array.

index = 0;

Array[index % 3600] = count\_in\_one\_second.

++index;

if you want last second, return the last element of this array. if you want last minute, return the sum of last 60 elements. if you want last hour, return the sum of the whole array (3600 elements).

**Solution**: Some of the first solutions that come to mind are:

1. Maintain 3 counters, one each for last hour, second and minute.  
   This is quickly rejected because there is no way to remove the count of requests which fall out of the window of last hour, minute and second.
2. Maintain 3 lists, one each for last hour, second and minute  
   This does solve the problem with the above counters, but since the number of requests is huge, a massive synchronization will be required for each thread adding its request to three lists.  
   Given thousands of requests per second, synchronization alone will slow down the entire lists' updation process.

**Best approach**: A good solution is one which allows concurrent updates and does not eat up too much memory.  
With this idea in mind, the following can be a good solution:  
  
1) Create an array ***AtomicInteger frequencies[1000];*** to store the number of requests received per second.  
  
2) This array will store frequencies of requests for the current second i.e. from HH:MM:SS to HH:MM:SS+1 time  
  
3) We will store current second in some variable, say ***currentSecond***  
This can be retrieved in Java as:

Calendar calendar = Calendar.getInstance();

**int** currentSecond = calendar.get([Calendar.SECOND](http://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html#SECOND));

4) ***frequencies*** array counts are incremented as:

**int** newSecond = calendar.get([Calendar.SECOND](http://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html#SECOND));

**if** (newSecond != currentSecond)

{

**synchronized** (...)

{

**int** requestsPerSecond = sumFrequenciesInTheSecond (frequencies);

frequencies = **new** AtomicInteger[1000];

}

}

// frequencies points to current second at this point

**int** requestMillisOfSecond = calendar.get([Calendar.MILLISECOND](http://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html#MILLISECOND));

frequencies[requestMillisOfSecond]++;

5) So we are able to get the frequencies per second by just using 1000 AtomicIntegers.  
  
6) To get requests per minute, all we need to do is add the seconds in a minute.  
This can be done by storing just 60 seconds of data, i.e. just 60 integers.  
  
7) Similarly, once we have minutes' data, we can get hours' data by adding 60 minutes.  
Again there is no need to store more than 60 minutes of data.  
Whenever 60 minutes complete for the current hour, we sum up those minutes, store the hour's requests per second and reset the minutes array.  
  
8) Thus, all we need is the following:

AtomicInteger frequencies[1000];

**int** secondsFrequencies[60]; // Every second gets its value by summing frequencies array

**int** minutesFrequencies[60]; // Every minute gets its value by summing secondsFrequencies array

**int** hoursFrequencies[60]; // Every hour gets its value by summing minutesFrequencies array