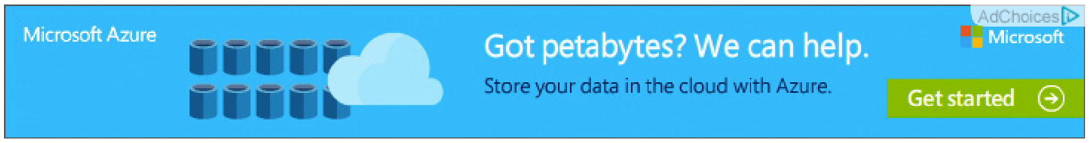
Interview Question: Find Median From Mega Number Of Integers


Microsoft Azure
Got petabytes? We can help.
Store your data in the cloud with Azure.
AdChoices
Microsoft
Get started

There is a file that contains 10G(1000000000) number of integers, please find the Median of these integers. you are given 2G memory to do this. Can anyone come up with an reasonable way? thanks!

algorithm

edited Aug 26 '10 at 7:09

asked Aug 26 '10 at 6:44
didxga
2,231 18 37

1 How big is an integer? Is this a 10G file that contains integers stored as text, or in binary format? – Will A Aug 26 '10 at 6:48

Is the count of integers known? – Abhinav Sarkar Aug 26 '10 at 7:06

I have updated my question, please review it. @Will A:the integer is as big as the computer can represent it. @abhin4v:yes as i have updated my question, its 10G(1000000000) – didxga Aug 26 '10 at 7:11

add a comment

7 Answers

Create an array of 8-byte longs that has 2^16 entries. Take your input numbers, shift off the bottom sixteen bits, and create a histogram.

Now you count up in that histogram until you reach the bin that covers the midpoint of the values.

Pass through again, ignoring all numbers that don't have that same set of top bits, and make a histogram of the bottom bits.

Count up through that histogram until you reach the bin that covers the midpoint of the (entire list of) values.

Now you know the median, in `O(n)` time and `O(1)` space (in practice, under 1 MB).

Here's some sample Scala code that does this:

```
def medianFinder(numbers: Iterable[Int]) = {
  def midArgMid(a: Array[Long], mid: Long) = {
    val cuml = a.scanLeft(0L)(_ + _).drop(1)
    cuml.zipWithIndex.dropWhile(_._1 < mid).head
  }
  val topHistogram = new Array[Long](65536)
  var count = 0L
  numbers.foreach(number => {
    count += 1
    topHistogram(number>>>16) += 1
  })
  val (topCount,topIndex) = midArgMid(topHistogram, (count+1)/2)
  val botHistogram = new Array[Long](65536)
  numbers.foreach(number => {
    if ((number>>>16) == topIndex) botHistogram(number & 0xFFFF) += 1
  })
  val (botCount,botIndex) =
    midArgMid(botHistogram, (count+1)/2 - (topCount-topHistogram(topIndex)))
  (topIndex<<16) + botIndex
}
```

and here it is working on a small set of input data:

```
scala> medianFinder(List(1,123,12345,1234567,123456789))
res18: Int = 12345
```

If you have 64 bit integers stored, you can use the same strategy in 4 passes instead.

edited Feb 14 '14 at 19:44
Billz
770 7 20

answered Aug 26 '10 at 15:06
Rex Kerr
101k 9 168 273

Smart. I like it. – Patrick Aug 26 '10 at 15:40

Nice! This one is better than mine, and with code! – ajduff574 Aug 26 '10 at 20:17

The complexities are really impressive. Kind of remind me the radix sort / bucket sort ideas. – Matthieu M. Aug 27 '10 at 9:36

1 +1. Works like a charm! I understood the concept of generating the histogram for top bits to get to the median bucket (sort of like counting sort), but what I couldn't understand is why ignoring the top bits while creating histogram for bottom bits, and then counting up to midpoint get us the right answer. Any explanation on that part would be much appreciated! – Kay Nov 12 '11 at 18:26

@Kay - You don't ignore the top bits--you pick only those that have the *same top bits as the central bin*. So you're really creating a histogram for the whole number, but you don't need to represent the part that is in common. – Rex Kerr Nov 12 '11 at 18:30

show **3** more comments

You can use the Medians of Medians algorithm.

answered Aug 26 '10 at 7:17
starblue
31.2k 8 51 100

+1, the factor of 5 difference between 10G and 2G sounds like this is the expected answer. – Ants Aasma Aug 26 '10 at 22:42

@Ants Aasma, 10G integers is usually 40GB which is 20x of 2GB or RAM. However Medians of Medians can still work. – grokus Aug 27 '10 at 1:46

Ah yeah, so it is. I originally misread that as 10GB of integers. – Ants Aasma Aug 27 '10 at 10:42

It should be noted that this gives only an **approximation** of the median. Taking the list in the Wikipedia example and sorting them, the middle values are 49 and 50 (so you could define the median to be 49.5) and NOT 47 as highlighted in the example there. – Andre Holzner Aug 28 '10 at 21:25

@Andre Holzer, it is a selection algorithm that uses an approximation of the median to find, for example, the exact median in O(n). The example on Wikipedia is about how to find that approximation, the pivot for the pseudo code above it. – Ishtar Aug 30 '10 at 12:46

add a comment

If the file is in text format, you may be able to fit it in memory just by converting things to integers as you read them in, since an integer stored as characters may take more space than an integer stored as an integer, depending on the size of the integers and the type of text file. EDIT: You edited your original question; I can see now that you can't read them into memory, see below.

If you can't read them into memory, this is what I came up with:

1. Figure out how many integers you have. You may know this from the start. If not, then it only takes one pass through the file. Let's say this is S.
2. Use your 2G of memory to find the x largest integers (however many you can fit). You can do one pass through the file, keeping the x largest in a sorted list of some sort, discarding the rest as you go. Now you know the x-th largest integer. You can discard all of these except for the x-th largest, which I'll call x1.
3. Do another pass through, finding the next x largest integers *less than* x1, the least of which is x2.
4. I think you can see where I'm going with this. After a few passes, you will have read in the (S/2)-th largest integer (you'll have to keep track of how many integers you've found), which is your median. If S is even then you'll average the two in the middle.

edited Aug 26 '10 at 7:18

answered Aug 26 '10 at 7:13
ajduff574
1,185 11 18

@ajduff574:as i have updated my question, there are 10G(1000000000) integers – didxga Aug 26 '10 at 7:16

1 +1 Impressive. But I foresee a problem dealing with large amounts of repeated numbers. Imagine that you are halfway through a pass through the file and your 2G sorted memory array fills up. Throughout the second half of your pass, you do not encounter any numbers that enable you to evict elements from your 2G array, but you encounter a lot of numbers that are exactly equal to the smallest element in the array (x1). You reach the end, discard your list, begin the next pass, and realize that you don't know which of the x1's you have discarded previously and which you have not. – advait Aug 26 '10 at 7:29

2 A possible solution might be not to use x1 in this case and use x1+1, thereby discarding everything in your 2G array that is greater than or equal to x1+1. However, you might reach a point where your entire 2G array becomes homogeneous. What then? You can't discard any numbers! – advait Aug 26 '10 at 7:31

2 Good catch! I think you could also keep track of how many times the smallest number (x) has appeared in your array. If it appears n times, then on the next pass you find the largest numbers <= to x, discarding the first n that you find that are equal to x. If it appears only once, this is the same as above. Even if the whole list is the same, this should still work. – ajduff574 Aug 26 '10 at 7:38

1 This has to be repeated n/2k times in the worst case. So if you use a sorted list implementation, that can do search, insert and delete in O(log n), the time complexity is O(n<sup>2</sup>/k log k), where k is the number of integers in the sorted list. That means if k comes close to n it tends to be O(nlog n) and in the other case if k is a lot smaller than n it tends to be O(n<sup>2</sup>). In the given case k is close to n/2. So its nearly O(nlog n). Median of Medians algorithm is O(n) but it maybe needs to save the partitions to the disk, so i think this should probably be faster. +1 – rudi-moore Aug 26 '10 at 14:09

show **2** more comments

Make a pass through the file and find count of integers and minimum and maximum integer value.

Take midpoint of min and max, and get count, min and max for values either side of the midpoint - by again reading through the file.

partition count > count => median lies within that partition.

Repeat for the partition, taking into account size of 'partitions to the left' (easy to maintain), and also watching for min = max.

Am sure this'd work for an arbitrary number of partitions as well.

answered Aug 26 '10 at 7:13
Will A
17.1k 19 43

Impressive, seems to me this should take N*log(N) time, and use O(1) memory (with an extremely low constant). – avl_sweden Apr 21 '13 at 11:54

Very nice! @avl_sweden - note that it's not really N*log(N), because the log isn't on the number of elements in the array, it's on the number range ! so basically, for 64bit integers, it's N*log(2^64), aka 64*N :) – ZeDuS Oct 19 '13 at 18:33

add a comment

1. Do an on-disk external mergesort on the file to sort the integers (counting them if that's not already known).
2. Once the file is sorted, seek to the middle number (odd case), or average the two middle numbers (even case) in the file to get the median.

The amount of memory used is adjustable and unaffected by the number of integers in the original file. One caveat of the external sort is that the intermediate sorting data needs to be written to disk.

Given `n` = number of integers in the original file:

- Running time: `O(nlogn)`
- Memory: `O(1)` , adjustable
- Disk: `O(n)`

edited Nov 11 '11 at 22:34
usr
85.2k 16 92 167

answered Aug 26 '10 at 7:14
Chris Schmich
17.1k 2 40 64

add a comment

Check out Torben's method in here:http://ndevilla.free.fr/median/median/index.html. It also has implementation in C at the bottom of the document.

answered Aug 26 '10 at 7:33
spbfox
703 3 7

add a comment

My best guess that probabilistic median of medians would be the fastest one. Recipe:

1. Take next set of N integers (N should be big enough, say 1000 or 10000 elements)
2. Then calculate median of these integers and assign it to variable X_new.
3. If iteration is not first - calculate median of two medians:

   X_global = (X_global + X_new) / 2

4. When you will see that X_global fluctuates not much - this means that you found approximate median of data.

But there some notes :

- question arises - Is median error acceptable or not.
- integers must be distributed randomly in a uniform way, for solution to work

EDIT: I've played a bit with this algorithm, changed a bit idea - in each iteration we should sum X_new with decreasing weight, such as:

X_global = k*X_global + (1.-k)*X_new :

k from [0.5 .. 1.], and increases in each iteration.

Point is to make calculation of median to converge fast to some number in very small amount of iterations. So that very approximate median (with big error) is found between 100000000 array elements **in only 252 iterations !!!** Check this C experiment:

```
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#define ARRAY_SIZE 100000000
#define RANGE_SIZE 1000

// probabilistic median of medians method
// should print 5000 as data average
// from ARRAY_SIZE of elements
int main (int argc, const char * argv[]) {
    int iter = 0;
    int X_global = 0;
    int X_new = 0;
    int i = 0;
    float dk = 0.002;
    float k = 0.5;
    srand(time(NULL));

    while (i<ARRAY_SIZE && k!=1.) {
        X_new=0;
        for (int j=i; j<i+RANGE_SIZE; j++) {
            X_new+=rand()%10000 + 1;
        }
        X_new/=RANGE_SIZE;

        if (iter>0) {
            k += dk;
            k = (k>1.)? 1.:k;
            X_global = k*X_global+(1.-k)*X_new;

        }
        else {
            X_global = X_new;
        }
```

Opps seems i'm talking about mean, not median. If it is so, and you need exactly median, not mean - ignore my post. In any case mean and median are very related concepts.

Good luck.

edited Aug 26 '10 at 12:35

answered Aug 26 '10 at 9:48
Agnius Vasiliauskas
4,319 2 20 44

add a comment

Not the answer you're looking for? Browse other questions tagged algorithm or ask your own question.