Finding a single number in a list [duplicate]

Asked 16 years, 3 months ago Modified 10 years, 3 months ago

Viewed 25k times

40

This question already has answers here:

How to find the only number in an array that doesn't occur twice [duplicate] (5 answers)

Closed 9 years ago.

What would be the best algorithm for finding a number that occurs only once in a list which has all other numbers occurring exactly twice.

So, in the list of integers (lets take it as an array) each integer repeats exactly twice, except one. To find that one, what is the best algorithm.

algorithm puzzle

Share

Improve this question

Follow

edited Sep 25, 2008 at 14:38

Motti

114k ● 56 ● 194 ● 273

asked Aug 29, 2008 at 20:03

Vaibhav
11.4k ● 11 ● 53 ● 71

11 Answers

Sorted by: Highest score (default)

140

The fastest (O(n)) and most memory efficient (O(1)) way is with the XOR operation.

In C:

```
int arr[] = {3, 2, 5, 2, 1, 5, 3};

int num = 0, i;

for (i=0; i < 7; i++)
    num ^= arr[i];

printf("%i\n", num);
```

This prints "1", which is the only one that occurs once.

This works because the first time you hit a number it marks the num variable with itself, and the second time it unmarks num with itself (more or less). The only one that remains unmarked is your non-duplicate.

Share Improve this answer Follow

edited Sep 25, 2008 at 17:09

answered Aug 29, 2008 at 20:43

Kyle Cronin
79k ● 45 ● 151 ● 167

3 this is the "best" solution as long as you can actually XOR the items. Meaning, it depends on the data type. I am not sure if you can do this or not if the items are strings. of course, in that case it can be solved with one more layer of abstraction... – csmba Sep 11, 2008 at 6:37

There are ways of XORing strings by XORing the individual chars - you would just have to have a temporary variable as large as the largest string. What wouldn't work is trying to XOR a linked list or some other complicated data structure, but this problem is simply about integers. – Kyle Cronin Sep 11, 2008 at 13:54

1 Clever solution, but I think that negative numbers might screw it up a little. You could potentially XOR in a mask which entirely threw off the rest of your masking results. – Daniel Spiewak Sep 25, 2008 at 17:14

5 A negative number is a bitfield just like a positive number. XOR doesn't care – Airsource Ltd Sep 25, 2008 at 21:17

1 @NickJohnson: What you need is not that the hash is "cryptographically secure", but that it is "perfect" or "two-way" or "unique". You need to reliably be able to get back to the object from the hash. – Thomas Ahle Dec 26, 2011 at 23:18

20

By the way, you can expand on this idea to very quickly find *two* unique numbers among a list of duplicates.

Let's call the unique numbers a and b. First take the XOR of everything, as Kyle suggested. What we get is a^b. We know a^b != 0, since a != b. Choose any 1 bit of a^b, and use that as a mask -- in more detail: choose x as a power of 2 so that x & (a^b) is nonzero.

Now split the list into two sublists -- one sublist contains all numbers y with y&x == 0, and the rest go in the other sublist. By the way we chose x, we know that a and b are in different buckets. We also know that each pair of duplicates is still in the same bucket. So we can now apply ye olde "XOR-em-all" trick to each bucket independently, and discover what a and b are completely.

Bam.

Share Improve this answer Follow

answered Aug 29, 2008 at 21:31

Tyler
28.9k ● 12 ● 93 ● 108

Love this . If will be super helpful if every alg questions come with this kind of expansion. – wanghq Jul 26, 2015 at 15:55

11

O(N) time, O(N) memory

HT= Hash Table

HT.clear() go over the list in order for each item you see

```
if(HT.Contains(item)) -> HT.Remove(item)
else
ht.add(item)
```

at the end, the item in the HT is the item you are looking for.

Note (credit @Jared Updike): This system will find all Odd instances of items.

---

comment: I don't see how can people vote up solutions that give you NLogN performance. in which universe is that "better" ? I am even more shocked you marked the accepted answer s NLogN solution...

I do agree however that if memory is required to be constant, then NLogN would be (so far) the best solution.

Share Improve this answer Follow

edited Aug 29, 2008 at 21:10

answered Aug 29, 2008 at 20:08

csmba
4,083 ● 3 ● 33 ● 42

4

Kyle's solution would obviously not catch situations were the data set does not follow the rules. If all numbers were in pairs the algorithm would give a result of zero, the exact same value as if zero would be the only value with single occurance.

If there were multiple single occurance values or triples, the result would be errouness as well.

Testing the data set might well end up with a more costly algorithm, either in memory or time.

Csmba's solution does show some errouness data (no or more then one single occurence value), but not other (quadrouples). Regarding his solution, depending on the implementation of HT, either memory and/or time is more then O(n).

If we cannot be sure about the correctness of the input set, sorting and counting or using a hashtable counting occurances with the integer itself being the hash key would both be feasible.

Share Improve this answer Follow

answered Sep 3, 2008 at 13:14
Ralph M. Rickenbach
13.1k ● 5 ● 30 ● 49

1 @malach Kyle's proposal solves exactly what the problem statement says. It makes no sense to write an O(nlogn) solution which protects from invalid data if O(n) solution exists and problem statement doesn't mention the possibility that data are wrong. Anyway, here is one article that explains the solution in a little bit more words from information theory point of view: [sysexpand.com/?path=exercises/number-appearing-once-in-array]() – Zoran Horvat Dec 17, 2013 at 23:07

1

I would say that using a sorting algorithm and then going through the sorted list to find the number is a good way to do it.

And now the problem is finding "the best" sorting algorithm. There are a lot of sorting algorithms, each of them with its strong and weak points, so this is quite a complicated question. The Wikipedia entry seems like a nice source of info on that.

Share Improve this answer

Follow

answered Aug 29, 2008 at 20:11

Farinha

18.1k ● 22 ● 66 ● 80

1

Implementation in Ruby:

```
a = [1,2,3,4,123,1,2,.........]
t = a.length-1
for i in 0..t
   s = a.index(a[i])+1
   b = a[s..t]
   w = b.include?a[i]
   if w == false
       puts a[i]
   end
end
```

Share Improve this answer

Follow

answered Sep 14, 2014 at 15:17

SuperNova

27.3k ● 7 ● 96 ● 69

0

You need to specify what you mean by "best" - to some, speed is all that matters and would qualify an answer as "best" - for others, they might forgive a few hundred milliseconds if the solution was more readable.

"Best" is subjective unless you are more specific.

That said:

Iterate through the numbers, for each number search the list for that number and when you reach the number that returns only a 1 for the number of search results, you are done.

Share Improve this answer Follow

answered Aug 29, 2008 at 20:07

Jason Bunting
58.9k ● 15 ● 104 ● 94

0

Seems like the best you could do is to iterate through the list, for every item add it to a list of "seen" items or else remove it from the "seen" if it's already there, and at the end your list of "seen" items will include the singular element. This is O(n) in regards to time and n in regards to space (in the worst case, it will be much better if the list is sorted).

The fact that they're integers doesn't really factor in, since there's nothing special you can do with adding them up... is there?

Question

I don't understand why the selected answer is "best" by any standard. O(N*lgN) > O(N), and it changes the list (or else creates a copy of it, which is still more expensive in space and time). Am I missing something?

Share Improve this answer Follow

edited Aug 29, 2008 at 20:24

answered Aug 29, 2008 at 20:12

levand
8,480 ● 3 ● 42 ● 55

0

Depends on how large/small/diverse the numbers are though. A radix sort might be applicable which would reduce the sorting time of the O(N log N) solution by a large degree.

Share Improve this answer

Follow

answered Aug 29, 2008 at 20:33

chakrit
61.5k ● 25 ● 136 ● 163

0

The sorting method and the XOR method have the same time complexity. The XOR method is only O(n) if you assume that bitwise XOR of two strings is a constant time operation. This is equivalent to saying that the size of the integers in the array is bounded by a constant. In that case you can use Radix sort to sort the array in O(n).

If the numbers are not bounded, then bitwise XOR takes time O(k) where k is the length of the bit string, and the XOR method takes O(nk). Now again Radix sort will sort the array in time O(nk).

Share Improve this answer

Follow

answered Sep 1, 2008 at 6:21

mercurio

-1

You could simply put the elements in the set into a hash until you find a collision. In ruby, this is a one-liner.

```
def find_dupe(array)
  h={}
  array.detect { |e| h[e]||(h[e]=true; false) }
end
```

So, `find_dupe([1,2,3,4,5,1])` would return 1.

This is actually a common "trick" interview question though. It is normally about a list of consecutive integers with one duplicate. In this case the interviewer is often looking for you to use the Gaussian sum of *n*-integers trick e.g. `n*(n+1)/2` subtracted from the actual sum. The textbook answer is something like this.

```
def find_dupe_for_consecutive_integers(array)
  n=array.size-1   # subtract one from array.size
because of the dupe
  array.sum - n*(n+1)/2
end
```

Share Improve this answer

Follow

answered Aug 29, 2008 at 20:27

hoyhoy

6,351 ● 7 ● 40 ● 36