cute interview question: the two smallest elements of an array

chris / 04.Sep.2008

As a consequence of the social awkwardness afforded to me by my nerd-like tendencies, I try to avoid human contact whenever possible. However, sometimes it's unavoidable. Like when my boss asks me to interview with a software development candidate (augh! why me?! screams my brain as I smile politely). And thus, I venture forth to face my hated nemesis; the awkward silence of death[1]. My best weapon against such a worthy adversary? Clever technical questions that are iterable in difficulty. Here's one I like to start with:

You have an unordered array of *n* integers, that might have been generated by:

```
array = Array.new
n = rand(1000)
(0 .. n - 1).each do
  array.push rand((1 << 32) - 1)
end
```

Find the two smallest.

They immediately reply "OMG sort it! And the grab the **smallest two elements**. Whee!".

That's a decent first intuition because it's such a brain dead simple solution; it works, and everyone knows why. But since this is an interview: simple, practical, readable, and maintainable solutions don't go as far as they might in the real world :).

So next I ask, exactly how integer comparisons are you doing?

And they reply, its `O(n log n)`.

Well… that doesn't really answer my question. So I ask them to come up with a solution that is linear in the # of comparisons, and require them to tell me exactly how many comparisons are being done.

Most people get this right away. Maintain a list of the **smallest two elements**. For each element `i`, check if `i` is less than the 2nd **smallest** element. If yes, check if `i` is less than the **smallest** element. If yes `i` is the new **smallest** and the old **smallest** element is now the second **smallest**, otherwise `i` is the new 2nd **smallest**. The zeroth element in the **array** does not require any comparisons, because, trivially, it is the **smallest**. The second element in the **array** requires only one comparison since it has only possible element to compare against. Here's a possible implementation:

```
if array[0] < array[1]
  a = array[0]
  b = array[1]
```

```
else
  a = array[1]
  b = array[0]
end
(2 .. n - 1).each do |i|
  if array[i] < b
    if array[i] < a
      b = a
      a = array[i]
    else
      b = array[i]
    end
  end
end
```

How many comparisons? Well, it depends. In the worst case, the **array** is sorted in descending order, and every element requires **two** comparisons. In the best case, the **array** is already sorted in ascending order, and every element requires one comparison (and no swapping occurs, the first **two elements** are the **two smallest**).

Worst: `2n - 3`
Best: `n - 1`

At this point, the interview candidate might be feeling warm and fuzzy inside. It is now time to crush those warm fuzzy feelings with a casual, "can you do better?", followed by a super-villain laugh.

Watch them squirm. Ask them about their thought process; what approaches are they considering? You will likely hear some chirping crickets.

The first hint I give is something along the lines of "intuitively, it seems possible to save a few comparisons by ruling out some particularly bad choices for the **smallest** and second **smallest elements**. Is it possible to weed out some bad choices?". In the worst case scenario, every element except the maximum is compared twice. If we simply compared each element to its neighbor, and discarded the higher of the **two**, we could prevent the second comparison from happening on integers with high values (relatively speaking).

If that doesn't help, I ask them to turn the **array** into an NCAA bracket for march madness. In the first round, integer at `array[0]` plays `array[1]`, 2 plays 3, 4 plays 5, etc. The winner (according to integer value) moves to the next round, and plays the winner of the neighboring comparison. Continue, until you have a winner, which is the **smallest** element in the **array**. You have basically built a binary tree out of the **array**. Building such a tree from an **array** clearly must touch every element and is `O(n)`, but I'm asking for the exact number of comparisons. The first round has `n / 2` comparisons, then `n / 4`, then `n / 8` and so forth, until there is one element left, which happens after `log n` rounds. How many comparisons is that? For a tournament of n **elements**, exactly `n - 1` have to lose in order for there to be one winner (after you lose one, you are discarded). That requires `n - 1` comparisons.

And then finally, how do you **find** the second **smallest** integer? The knee-jerk reaction is to point to the integer that we compared to the eventual winner in the final comparison. However, what if the **smallest** and second **smallest elements** were adjacent in the **array**? The second **smallest** element would have been eliminated in the first round of comparisons. The interesting thing to notice is that there is only one element that the second **smallest** element can lose to: the **smallest** element. So we know at some point that the second **smallest** element was compared to the **smallest**, and it lost. So taking our handy tree structure, we traverse the path of the **smallest** element, starting at the root node, and **find** the minimum valued integer among the **elements** that were compared to our newly anointed **smallest** element. That integer is our coveted second **smallest** element. There are exactly `log(n-1)` **elements** (1 per round) that the **smallest** element was compared to on its path to victory. So the exact number of comparisons is `n + log(n-1) - 1`. BTW, this solution is much cleaner if n is a power of **two**, so I make that assumption when explaining this optimization.

So… is that solution useful? No. Intuitive? Not in my experience. But I like it because it lies in stark contrast to the simplicity of the task at hand: **find** the **two smallest elements** in an **array**.

Does it help identify good software developers during an interview? I'm not sure. What do you think?

[1] (Actually, interviews aren't that bad; you get to meet some cool people with interesting insights & ideas, and they often get you out of meetings!)

(Edited to make me not sound like a dick).

2 Comments

zero comparisons, constant time:

```
a = rand((1 << 32) - 1)
b = rand((1 << 32) - 1)
```

(well, rand might be non-constant time or use a comparison)


someone / 04.Sep.2008 at 06:50am

> (well, rand might be non-constant time or use a comparison)

Mersenne_Twister.

Source code.


chris / 04.Sep.2008 at 02:10pm

Leave a Comment

name (required)

email (not published, used for gravatar)

website

SUBMIT COMMENT

Recent Entries

- seeqpod api client for ruby
- oh, you snarky developers!
- how do you identify a great software developer in an interview?
- use p4merge or DiffMerge with git mergetool on OS X
- sharing feeds with friends
- command line encryption and decryption with openssl, perfect for your aim logs :)
- sustaining a hacker lifestyle
- cute interview question: the **two smallest elements** of an **array**
- about me

Subscribe

Subscribe to logical.rand, exacting logic @ random.

Google

back to top