Arrays (programming) Theoretical Computer Science Algorithms +1

What is the fastest algorithm to find the largest number in an unsorted array?

Answer Request ▾ Follow 161 Comments 6+ Downvote

91 Answers

Máté Kovács, has been programming longer than speaking English
Answered Oct 15, 2015 · Upvoted by Daniel R. Page, Theoretical Computer Scientist

Assumptions: we mean 'fastest' in the asymptotic sense, by 'number' we mean integer, and algorithms are deterministic.

TL;DR: The fastest algorithm looks at every element once, keeping track of the largest one seen so far.

Details

Intuition tells us that any correct algorithm **must** look at every element in the array, that's why we can't do better. Let's find an actual proof.

Notation: GET(A, i) is shorthand for the operation of reading the element at index i in the array A.

We can show that for any program p which performs fewer than n GETs on some array A of size n to compute max(A), there's some array A' on which it fails to compute max(A').

First, we run p on A, and record the set of indices for which p did a GET(A, i). Then we pick an index k (0 <= k < n) that's not in that set. Note that there must be at least one such index, because p used fewer than n GETs. Finally, we construct the array A' by copying A then setting the element at index k to max(A) + 1.

The behavior of p cannot depend on elements of the input which it ignores, so p(A') = p(A) = max(A). But max(A') = max(A) + 1 by construction, which means that p fails on A'.

60.7k Views · 85 Upvotes

Upvote 85 Downvote

Add a comment... Recommended All

There's more on Quora...

Pick new people and topics to follow and see the best answers on Quora.

Update Your Interests

Related Questions

- What would be fastest algorithm for determining if a number is in a set or not?
- Is the fastest algorithm in searching the biggest number in an unsorted array be heapsort?
- What is the fastest algorithm to find the largest number in an unsorted array with multiple processors?
- How do I write an algorithm to find the nth highest number in an array?
- What is the most efficient sorting algorithm for finding largest number among 20 numbers?
- How can we return the max K numbers from an unsorted integer array?
- There is an array with 60% sorted and 40% unsorted, which algorithm will give fastest accurate sorted output?
- What is the fastest Algorithm to multiply large numbers?
- How can I get the three largest numbers from 2000 unsorted distinct numbers with least compare operations under worst condition? My idea is to...
- What is the minimum number of comparisons required to find the pair of integers, in an array of n unsorted integers, whose absolute difference...

+ Ask New Question

More Related Questions

Question Stats

160 Public Followers

1,577,600 Views

Last Asked Feb 25

1 Merged Question

Edits

Ken Alverson, professional bit twiddler
Updated Oct 7, 2015 · Upvoted by Remco Gerlich, 30 years of programming experience, Python/Django expert

The other answers are correct, the problem as stated is O(N). However, consider that the problem is Embarrassingly parallel .

You could easily assign four threads to look at one quarter of the array, each, and return the largest number they find. After they all complete, you can select the largest of the four and that is the final answer.

If you have terabytes of data, instead of dividing the problem into multiple threads, you could divide it into multiple computers (which could further subdivide into threads).

There are inefficiencies in this approach, of course. If the data is local to a machine, processing it there (with such a simple operation as this) will be faster than transferring it across the network for others to do some of the work...you are I/O bound. Even on a single machine, you won't actually get 4x speed from four threads as they fight for memory bandwidth. But you should get some speedup.

39.9k Views · 75 Upvotes

Upvote 75 Downvote

Armand Welsh
This is still o(n) processing. Consider that unless you have full control over all the cor...

Tim Farage, Professor and Graduate Advisor in the CS Dept. at UT Dallas
Updated Apr 14, 2016 · Upvoted by Daniel R. Page, Theoretical Computer Scientist

I've been a senior programmer at three companies, and I've taught computer programming at the undergraduate and graduate levels.

Peter de Vroede is correct. The only way to find the largest value in an unsorted array is to look at every value in the array.

Think of it this way. If I gave you 100 index cards with a random number on each card, how would you find the largest number? You'd have to go though all the cards.

So if you had N cards, you have to look at all N of them. So the time complexity is O(N).

The type of algorithm that does this called a linear search algorithm.

PLEASE NOTE: Some answers given here say that parallel processing would find the max value faster. This is true. But no matter how many processors you have, you will still need to 'look' at each value. So the number of operations is still O(N).

To see this think about having a random number on each of 200 index cards. If you are trying to find the max value, you have to look at all of the 200 cards. Now, suppose you enlist a friend to help. You give your friend half of the cards, and each of you finds the max in their half. Compare these and you'll have your answer. Yes, it will take half the time, but both you and your friend will look at 100 cards, so all 200 cards are being looked at. Thus the number of operations is still O(N).

Go through the first half of the values in the array, and keep track of the greatest value in the first half, H. Then go through the second half of the array. When (and if) you find a value greater than H, stop. Let's call this G.

What is the probability that G is the greatest value in the array?

In order for this to happen, G must be in the second half of the array. Since the array values are random, there is a 50% chance of this.

And then if the *second* greatest value of the array is in the first half, this would guarantee that we'd pick the actual greatest value.

The probability that the second greatest value is in the first half is also 50%. So the probability that both the second greatest value is in the first half, and that the greatest value is in the second half is 50% x 50% = 25%.

So this algorithm gives us at least a 25% of finding the greatest value in the array.

Not impressed? When I first saw this, I was.

By tweaking this, you can increase your chances of finding the greatest value to about 37%. And you only have to look at the first third of the array to get this probability.

Admit it. That's interesting.

Want the gory details? Check out this Wikipedia entry: The Secretary Problem .

1.3m Views · 2,845 Upvotes

Upvote 2.8k Downvote

Kurt Guntheroth
In parallel, using n/2 processors, you can compute the maximum in log2(n) time. In t...

Promoted by Association for Computing Machinery

Write more than code. Write the future. Join ACM today.
Join a global network of your peers in making a positive impact on the computing profession and society.

Learn more at acm.org

Top Stories from Your Feed