215. Kth Largest Element in an Array

602 62

Description (/problems/kth-largest-element-in-an-array/description/) Hints (/problems/kth-largest-element-in-an-array/hints/) Submissions (

(/problems/kth-largest-element-in-an-array/discuss) > Solution explained

Share Subscribe Report Notes

Solution explained

86.0K

335

Last Edit: Feb 24, 2018, 12:07 AM

(/jmnarloch) jmnarloch (/jmna... 881

This problem is well known and quite often can be found in various text books.

You can take a couple of approaches to actually solve it:

- O(N lg N) running time + O(1) memory

The simplest approach is to sort the entire input array and then access the element by it's index (which is O(1)) operation:

```
public int findKthLargest(int[] nums, int k) {
        final int N = nums.length;
        Arrays.sort(nums);
        return nums[N - k];
}
```

- O(N lg K) running time + O(K) memory

Other possibility is to use a min oriented priority queue that will store the K-th largest values. The algorithm iterates over the whole input and maintains the size of priority queue.

```
public int findKthLargest(int[] nums, int k) {

    final PriorityQueue<Integer> pq = new PriorityQueue<>();
    for(int val : nums) {
        pq.offer(val);

        if(pq.size() > k) {
            pq.poll();
        }
    }
    return pq.peek();
}
```

- O(N) best case / O(N^2) worst case running time + O(1) memory

The smart approach for this problem is to use the selection algorithm (based on the partion method - the same one as used in quicksort).

```
public int findKthLargest(int[] nums, int k) {

        k = nums.length - k;
        int lo = 0;
        int hi = nums.length - 1;
        while (lo < hi) {
            final int j = partition(nums, lo, hi);
            if(j < k) {
                lo = j + 1;
            } else if (j > k) {
                hi = j - 1;
            } else {
                break;
            }
        }
        return nums[k];
    }

    private int partition(int[] a, int lo, int hi) {

        int i = lo;
        int j = hi + 1;
        while(true) {
            while(i < hi && less(a[++i], a[lo]));
            while(j > lo && less(a[lo], a[--j]));
            if(i >= j) {
                break;
            }
            exch(a, i, j);
        }
        exch(a, lo, j);
        return j;
    }

    private void exch(int[] a, int i, int j) {
        final int tmp = a[i];
        a[i] = a[j];
        a[j] = tmp;
    }

    private boolean less(int v, int w) {
        return v < w;
    }
```

O(N) guaranteed running time + O(1) space

So how can we improve the above solution and make it O(N) guaranteed? The answer is quite simple, we can randomize the input, so that even when the worst case input would be provided the algorithm wouldn't be affected. So all what it is needed to be done is to shuffle the input.

```
public int findKthLargest(int[] nums, int k) {

        shuffle(nums);
        k = nums.length - k;
        int lo = 0;
        int hi = nums.length - 1;
        while (lo < hi) {
            final int j = partition(nums, lo, hi);
            if(j < k) {
                lo = j + 1;
            } else if (j > k) {
                hi = j - 1;
            } else {
                break;
            }
        }
        return nums[k];
    }

private void shuffle(int a[]) {

        final Random random = new Random();
        for(int ind = 1; ind < a.length; ind++) {
            final int r = random.nextInt(ind + 1);
            exch(a, ind, r);
        }
    }
```

There is also worth mentioning the Blum-Floyd-Pratt-Rivest-Tarjan algorithm that has a guaranteed O(N) running time.

Comments: 65

Sort By

Type comment here... (Markdown is supported)

Preview Post

orangetabby (/orangetabby) ★ 76 May 23, 2015, 6:45 PM

The last ALG has average O(N) time, but worst case is O(N^2).

(/orangetabby)

76 Report post

jmnarloch (/jmnarloch) ★ 881 Jun 3, 2015, 5:40 AM

This is due the fact that the PriorityQueue is implemented as a Binary Heap, which in fact is nothing more then complete binary tree. So both inserting and removing the values through offer() and poll() methods have O(lg K) complexity and altogether since you doing this operation N times the total complexity is O(N lg K)

(/jmnarloch)

11 Report post

syftalent (/syftalent) ★ 485 Jun 20, 2015, 9:55 PM

I have a question for the O(n) solution. From my understanding, this algorithm is based on the quick sort. If there's a lot of duplication in the array, it will lower the performance. For instance, if there's only one number in the array, we only can eliminate one number in a recursion. Even randomize the partition can not solve that problem.
So my method to improve that algorithm is to split the array into three parts, which are <, = , > here's my code. Please tell me your opinion.

(/syftalent)

10 Read More Report post