Previous (/articles/powerful-integers/) Next (/articles/reverse-linked-list-ii/)

215. Kth Largest Element in an Array (/problems/kth-largest-element-in-an-array/)

Jan. 6, 2019 | 40.6K views

Average Rating: 4.35 (29 votes)

Find the **k**th largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

Example 1:

```
Input: [3,2,1,5,6,4] and k = 2
Output: 5
```

Example 2:

```
Input: [3,2,3,1,2,4,5,5,6] and k = 4
Output: 4
```

Note:
You may assume k is always valid, 1 ≤ k ≤ array's length.

Solution

Approach 0: Sort

The naive solution would be to sort an array first and then return kth element from the end, something like `sorted(nums)[-k]` on Python. That would be an algorithm of $\mathcal{O}(N \log N)$ time complexity and $\mathcal{O}(1)$ space complexity. This time complexity is not really exciting so let's check how to improve it by using some additional space.

Approach 1: Heap

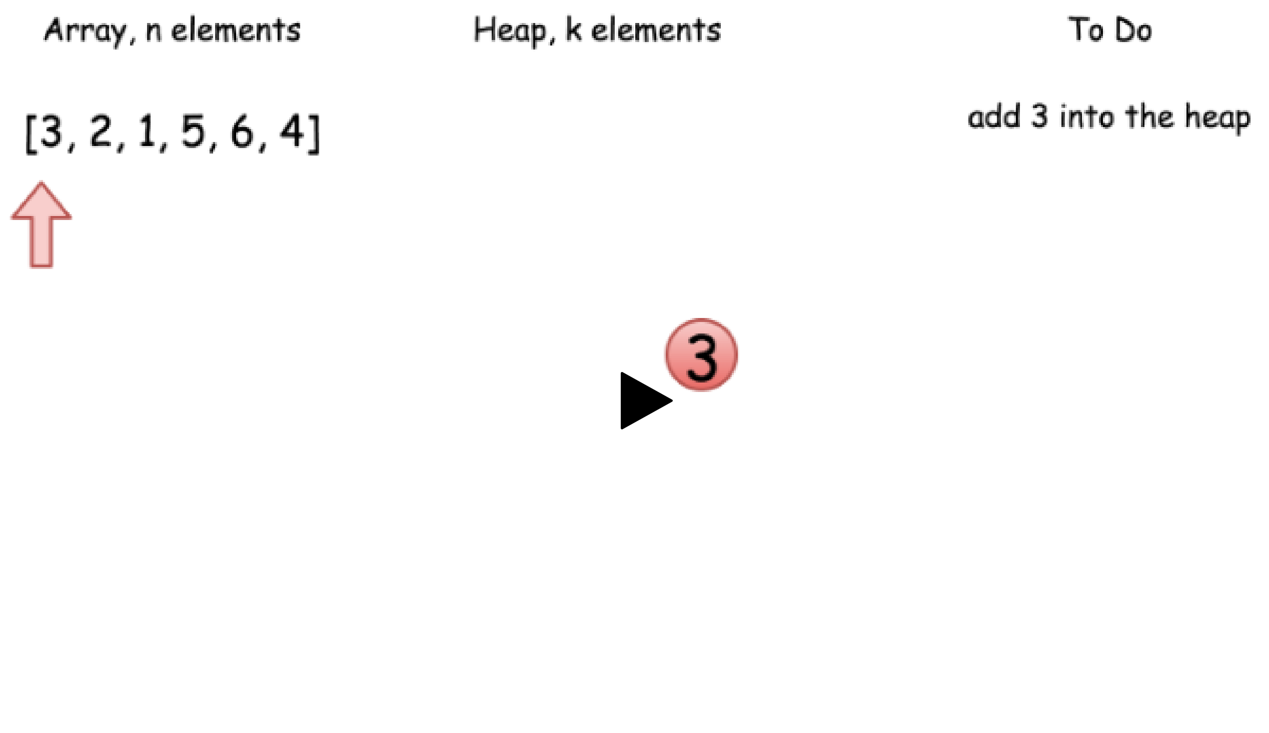
The idea is to init a heap "the smallest element first", and add all elements from the array into this heap one by one keeping the size of the heap always less or equal to `k`. That would results in a heap containing `k` largest elements of the array.

The head of this heap is the answer, i.e. the kth largest element of the array.

The time complexity of adding an element in a heap of size `k` is $\mathcal{O}(\log k)$, and we do it `N` times that means $\mathcal{O}(N \log k)$ time complexity for the algorithm.

In Python there is a method `nlargest` in `heapq` library which has the same $\mathcal{O}(N \log k)$ time complexity and reduces the code to one line.

This algorithm improves time complexity, but one pays with $\mathcal{O}(k)$ space complexity.



1 / 11

Java Python Copy

```java
class Solution {
    public int findKthLargest(int[] nums, int k) {
        // init heap 'the smallest element first'
        PriorityQueue<Integer> heap =
            new PriorityQueue<Integer>((n1, n2) -> n1 - n2);

        // keep k largest elements in the heap
        for (int n: nums) {
          heap.add(n);
          if (heap.size() > k)
            heap.poll();
        }

        // output
        return heap.poll();
  }
}
```

Articles >

Complexity Analysis

Time complexity : $\mathcal{O}(N \log k)$.
Space complexity : $\mathcal{O}(k)$ to store the heap elements.

Approach 2: Quickselect

This textbook algorthm (https://en.wikipedia.org/wiki/Quickselect) has $\mathcal{O}(N)$ average time complexity. Like quicksort, it was developed by Tony Hoare, and is also known as *Hoare's selection algorithm*.

The approach is basically the same as for quicksort. For simplicity let's notice that `k` th largest element is the same as `N - k` th smallest element, hence one could implement `k` th smallest algorithm for this problem.

First one chooses a pivot, and defines its position in a sorted array in a linear time. This could be done with the help of *partition algorithm*.

> To implement partition one moves along an array, compares each element with a pivot, and moves all elements smaller than pivot to the left of the pivot.

As an output we have an array where pivot is on its perfect position in the ascending sorted array, all elements on the left of the pivot are smaller than pivot, and all elements on the right of the pivot are larger or equal to pivot.

Hence the array is now split into two parts. If that would be a quicksort algorithm, one would proceed recursively to use quicksort for the both parts that would result in $\mathcal{O}(N \log N)$ time complexity. Here there is no need to deal with both parts since now one knows in which part to search for `N - k` th smallest element, and that reduces average time complexity to $\mathcal{O}(N)$.

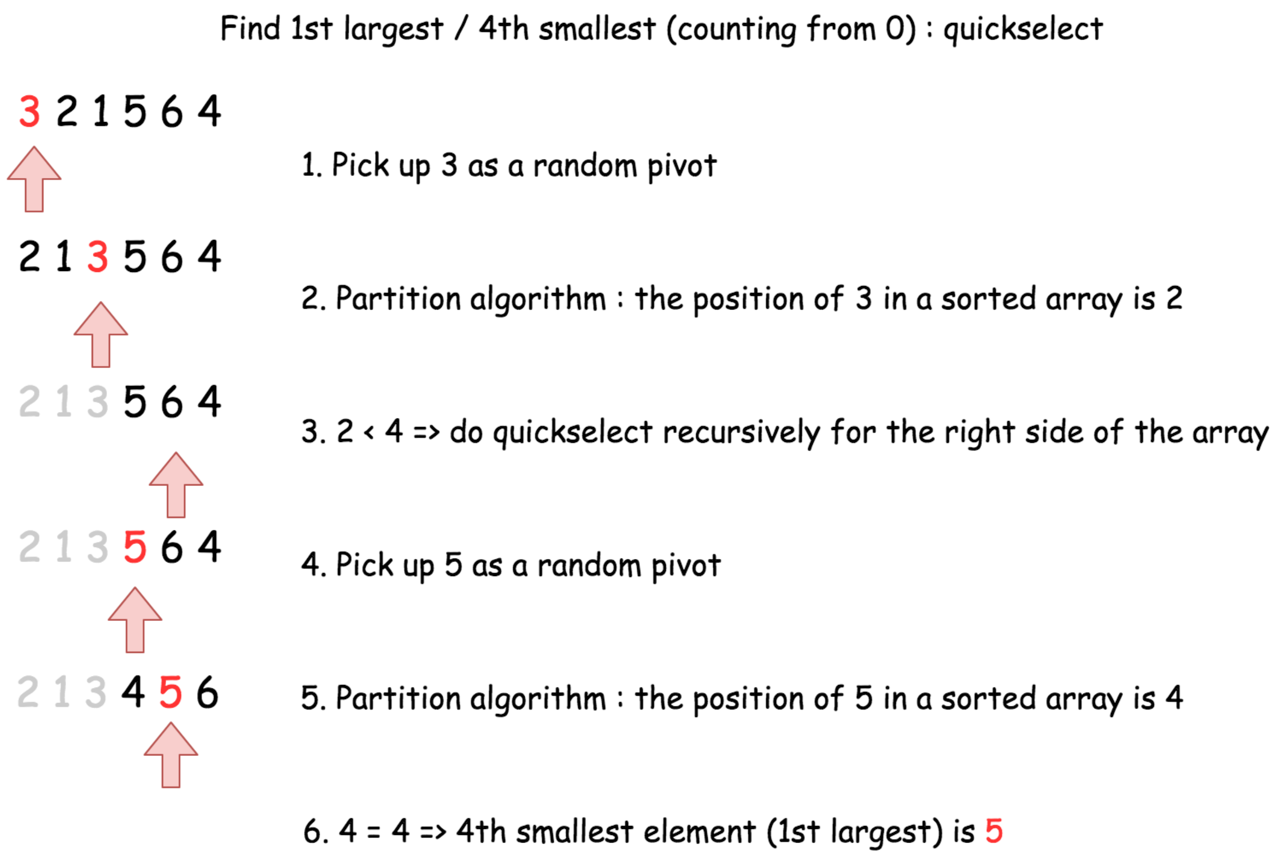
Finally the overall algorithm is quite straightforward :

Choose a random pivot.

Use a partition algorithm to place the pivot into its perfect position `pos` in the sorted array, move smaller elements to the left of pivot, and larger or equal ones - to the right.

Compare `pos` and `N - k` to choose the side of array to proceed recursively.

> ! Please notice that this algorithm works well even for arrays with duplicates.


Find 1st largest / 4th smallest (counting from 0) : quickselect
3 2 1 5 6 4
1. Pick up 3 as a random pivot
2 1 3 5 6 4
2. Partition algorithm : the position of 3 in a sorted array is 2
2 1 3 5 6 4
3. 2 < 4 => do quickselect recursively for the right side of the array
2 1 3 5 6 4
4. Pick up 5 as a random pivot
2 1 3 4 5 6
5. Partition algorithm : the position of 5 in a sorted array is 4
6. 4 = 4 => 4th smallest element (1st largest) is 5

Java Python Copy

Articles >

```java
import java.util.Random;
class Solution {
  int [] nums;

  public void swap(int a, int b) {
    int tmp = this.nums[a];
    this.nums[a] = this.nums[b];
    this.nums[b] = tmp;
  }


  public int partition(int left, int right, int pivot_index) {
    int pivot = this.nums[pivot_index];
    // 1. move pivot to end
    swap(pivot_index, right);
    int store_index = left;

    // 2. move all smaller elements to the left
    for (int i = left; i <= right; i++) {
      if (this.nums[i] < pivot) {
        swap(store_index, i);
        store_index++;
      }
    }

    // 3. move pivot to its final place
    swap(store_index, right);
```

Time complexity : $\mathcal{O}(N)$ in the average case, $\mathcal{O}(N^2)$ in the worst case.

Space complexity : $\mathcal{O}(1)$.

Analysis written by @liaison (https://leetcode.com/liaison/) and @andvary (https://leetcode.com/andvary/)

Rate this article:

Previous (/articles/powerful-integers/)

Next (/articles/reverse-linked-list-ii/)

Comments: 15

Sort By

Type comment here... (Markdown is supported)

Preview Post

chintant94 (chintant94) ★ 9 January 14, 2019 10:50 PM

How about creating a Min-heap(takes O(N) and then perform Extract-Min(log N) k times, so complexity of O(N + kLogN) ?

9 Share Reply

SHOW 4 REPLIES

czc404 (czc404) ★ 3 January 8, 2019 10:20 PM

According to this article (https://www.geeksforgeeks.org/kth-smallestlargest-element-unsorted-array-set-3-worst-case-linear-time/), there is an O(N) time solution, even in the worst case.

3 Share Reply

SHOW 3 REPLIES

likeabbas (likeabbas) ★ 7 January 6, 2019 11:06 AM

Here's a Θ(N)+O(M) time complexity and Θ(M) space complexity solution in Golang using Bucket Sort where M is the highestNum - lowestNum numbers in the array

Analysis:

Read More

3 Share Reply

kremebrulee (kremebrulee) ★ 24 June 14, 2019 12:17 PM

[mention:(display:andvary)(type:username)(id:andvary)]
int pivot_index = left + random_num.nextInt(right - left);

^the above line doesn't seem correct. The argument to nextInt is exclusive so "right" would never get picked.

2 Share Reply

tom53 (tom53) ★ 2 July 31, 2019 6:32 AM

Space complexity of the Implementation of Approach2 is not O(1) due to the recursive calls.

1 Share Reply

SHOW 1 REPLY

usta06 (usta06) ★ 3 July 29, 2019 11:44 AM

Can someone tell me what is going on inside parenthesis, thanks!

```
PriorityQueue<Integer> heap = new PriorityQueue<Integer>((n1, n2) -> n1 - n2);
```

1 Share Reply

SHOW 2 REPLIES

eugenepark3 (eugenepark3) ★ 1 July 8, 2019 11:41 AM

can someone explain to me why is the space complexity for quickselect O(1)?

1 Share Reply

oscardoudou (oscardoudou) ★ 28 January 6, 2019 9:57 PM

Following changes won't improve any complexity, it does help beats more though.
One major change is use default pq without lamba expression
One minor change is pq.peek()

Articles >

1 Share Reply

SHOW 2 REPLIES

v1s1on (v1s1on) 198 March 5, 2019 9:28 PM

Got to love C++:

```
int findKthLargest(vector<int>& nums, int k) {
        nth_element(begin(nums), end(nums) - k, end(nums));
        return nums[nums.size()-k];
```

Read More

1 Share Reply

SHOW 1 REPLY

ltsiros (ltsiros) 2 March 21, 2019 2:56 AM

Quickselect's quadratic worst case can be avoided by shuffling the input array ;)

0 Share Reply

SHOW 4 REPLIES

< 1 2 >