Previous Next

215. Kth Largest Element in an Array

Jan. 6, 2019 | 131.4K views

Average Rating: 4.11 (62 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

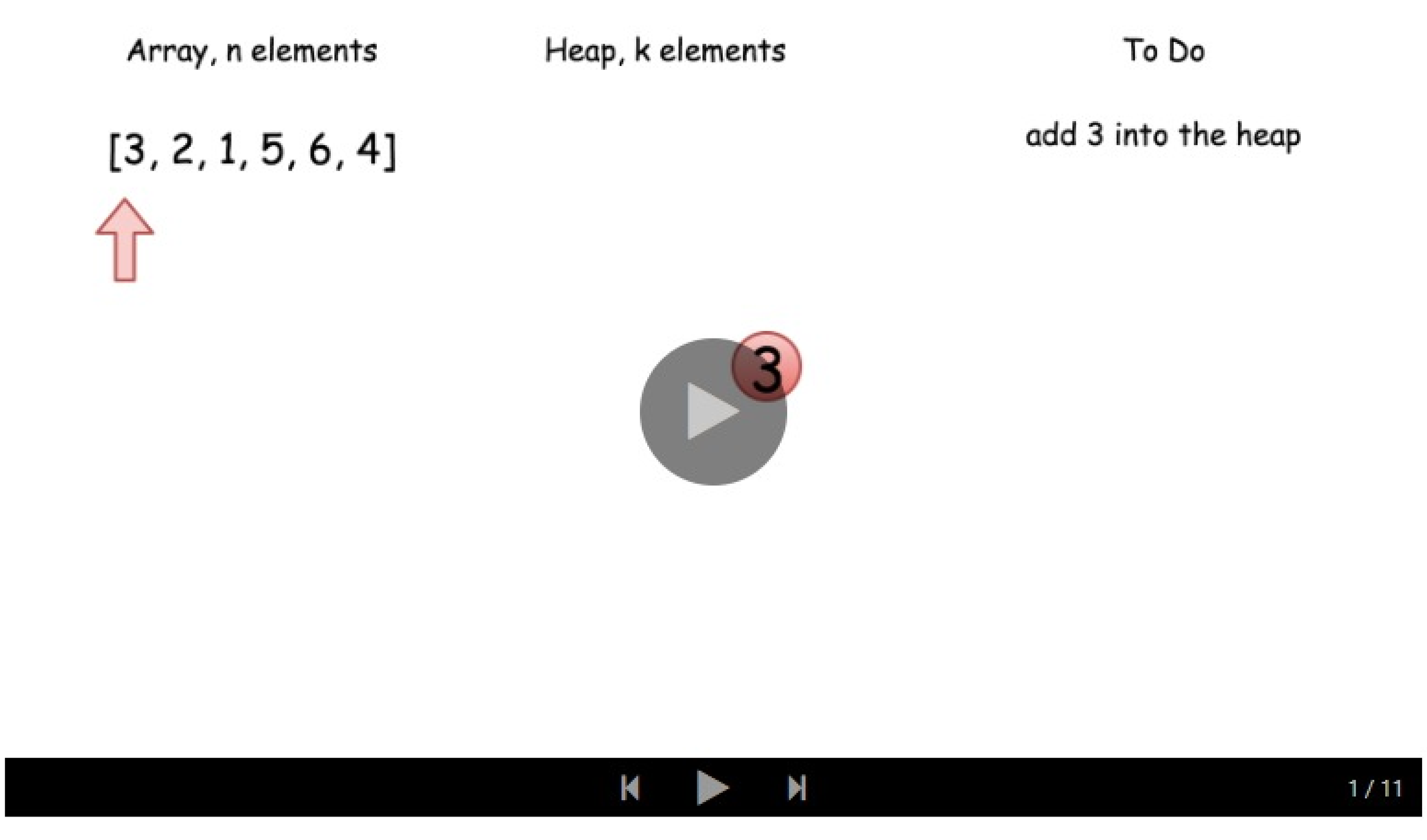
The idea is to init a heap "the smallest element first", and add all elements from the array into this heap one by one keeping the size of the heap always less or equal to `k`. That would results in a heap containing `k` largest elements of the array.

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.



Java | Python | Copy

```python
class Solution:
    def findKthLargest(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
        """
        return heapq.nlargest(k, nums)[-1]
```

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 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.

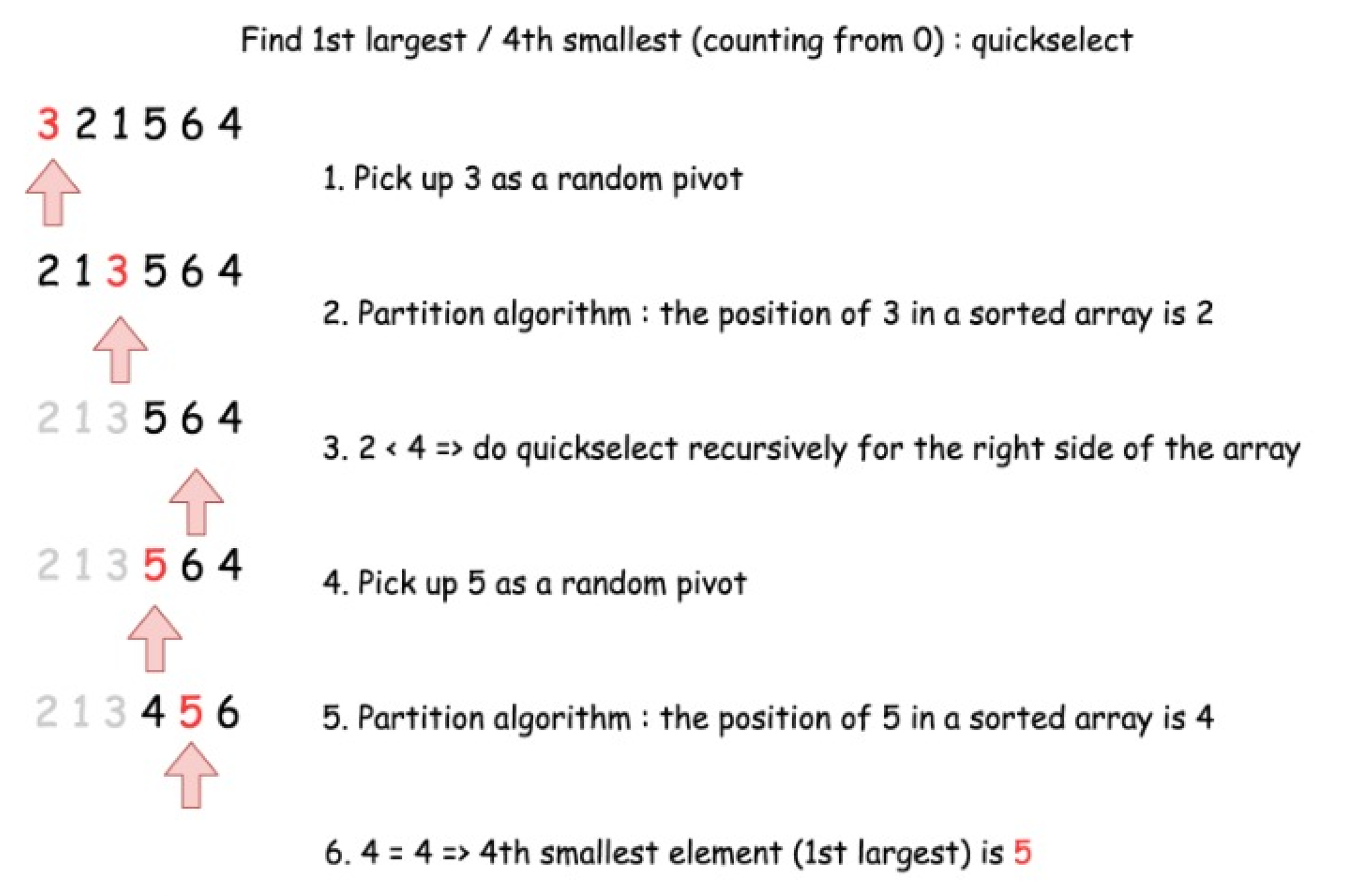
As an output we have an array where pivot is on its perfect position in the ascending sorted array, all elements on the left of the pivot are smaller than pivot, and all elements on the right of the pivot are larger or equal to pivot.

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.



Java | Python | Copy

```python
class Solution:
    def findKthLargest(self, nums, k):
        """
        :type nums: List[int]
        :type k: int
        :rtype: int
        """
        def partition(left, right, pivot_index):
            pivot = nums[pivot_index]
            # 1. move pivot to end
            nums[pivot_index], nums[right] = nums[right], nums[pivot_index]

            # 2. move all smaller elements to the left
            store_index = left
            for i in range(left, right):
                if nums[i] < pivot:
                    nums[store_index], nums[i] = nums[i], nums[store_index]
                    store_index += 1

            # 3. move pivot to its final place
            nums[right], nums[store_index] = nums[store_index], nums[right]

            return store_index

        def select(left, right, k_smallest):
            """
            Returns the k-th smallest element of list within left..right
```

- Time complexity : $\mathcal{O}(N)$ in the average case, $\mathcal{O}(N^2)$ in the worst case.
- Space complexity : $\mathcal{O}(1)$.

Rate this article:

Previous Next

Comments: 53

Sort By

Type comment here... (Markdown is supported)

Preview Post

RogerBuoy ★ 46 January 15, 2019 12:20 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) ?
44 Share Reply
SHOW 14 REPLIES

my3m ★ 558 December 6, 2019 1:44 AM
I used quick select and my solution is in the bottom 10%
so much for optimizations
16 Share Reply
SHOW 5 REPLIES

dmgcodevil ★ 36 August 29, 2019 10:20 PM
The 1st largest is 6, no?
10 Share Reply
SHOW 2 REPLIES

tom53 ★ 28 July 31, 2019 7:02 PM
Space complexity of the Implementation of Approach2 is not O(1) due to the recursive calls.

Iterative version, space:O(1)

```
class Solution:
```

Read More
10 Share Reply
SHOW 2 REPLIES

likeabbas ★ 11 January 7, 2019 12:36 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
6 Share Reply
SHOW 3 REPLIES

kremebrulee ★ 52 June 15, 2019 12:47 AM
[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.

Read More
5 Share Reply

sungwonesports ★ 14 October 5, 2019 4:00 AM
can someone explain why approach 2 works for duplicates?
4 Share Reply
SHOW 3 REPLIES

eugenepark3 ★ 5 July 9, 2019 12:11 AM
can someone explain to me why is the space complexity for quickselect O(1)?
4 Share Reply
SHOW 4 REPLIES

czc404 ★ 2 January 9, 2019 11:50 AM
According to this article, there is an O(N) time solution, even in the worst case.
2 Share Reply
SHOW 3 REPLIES

oscardoudou ★ 51 January 7, 2019 11:27 AM
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()
2 Share Reply
SHOW 2 REPLIES

‹ 1 2 3 4 5 6 ›