

Problem: Given unsorted array, return kth largest element. Not kth distinct element.

k = 2,  
[5, 3, 1, 6, 4, 2] then [1, 2, 3, 4, 5, 6]  $\Rightarrow$  5  
k=2 k=1

k = 4  
[2, 3, 1, 2, 4, 2] then [1, 2, 2, 2, 3, 4]  $\Rightarrow$  1  
k=4 k=3 k=2 k=1

Intro:

- Verify Constraints
  - k larger than len(arr)? no
- Create Testcases
  - [5, 3, 1, 6, 4, 2] and k = 2  $\rightarrow$  5
  - [2, 3, 1, 2, 4, 2] and k = 4  $\rightarrow$  1
  - [3] and k = 1  $\rightarrow$  3

Brute Force:

- Brainstorming & Pattern Observations

after quicksort

[1, 2, 3, 4, 5, 6]  
k=2

[1, 2, 2, 2, 3, 4]  
k=4

Starting at end of array

if k == 1: return arr[-1]

i = len(arr) - 2

unique# = 1

prev = arr[-1]

while i >= 0:

if arr[i] != prev:

unique# += 1

if k == unique#:

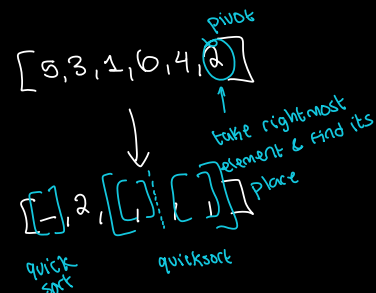
return arr[i]

prev = arr[i]

i -= 1

logic:  
moving everything  
to the left  
of pivot

Multiple  
Quicksorts: sorts in place  
(does not return new  
array)



2 was on  
most left

i - final resting place of where pivot  
should be.

j - scans / compares  
scanner w/ pivot  
(element @ j)

while i < len(arr) - 1:

if arr[j] < arr[pivot]

# swap j and i's elements

arr[i], arr[j] = arr[j], arr[i]

i += 1

j += 1

else j += 1

[5, 3, 1, 6, 4, 2]  
i < j:  
arr[i], arr[j] = arr[j], arr[i]  
i += 1  
j += 1

[5, 3, 1, 6, 4, 2]

[1, 3, 5, 6, 4, 2]

[1, 2, 5, 6, 4, 3]  
recursive recursive

3 2 1 5 6 4  
j  
3 < 4  
pivot 4

- Pseudocode
- Write code
- Run through testcases
- Analyze time and space complexity

Time:  $O(n \log n)$ : how long does it take to find the partition index? looks thru  $n$  times and then splitting tree

2x

Space:  $O(\log n)$ : split tree into two parts every time (2 recursive

#### Optimal:

- Brainstorming & Pattern Observations

Using HOARE'S QUICKSELECT ALGORITHM

- Pseudocode
- Write code
- Run through testcases
- Analyze time and space complexity

Time: partition function:  $O(n)$

quickSelect:

best:  $O(n) + O(n/2) + O(n/4) + \dots = O(n)$

worst: array sorted in descending order:

the array does not get split in half, always split on one huge array

$O(n^2)$

Space: reduced to tail recursion  $O(1)$ : we're not waiting on both the left and right branch recursion