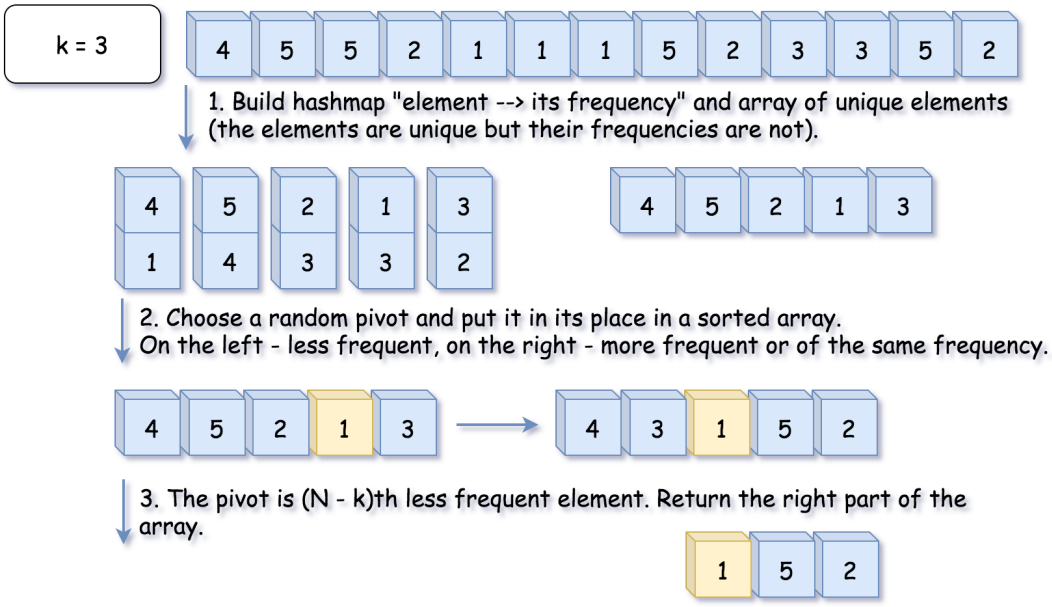
Thursday, 9 September 2021 1:08 PM

Approach 2: Quickselect (Hoare's selection algorithm)

Quickselect is a textbook algorthm typically used to solve the problems "find k*th* something": k*th* smallest, k*th*largest, k*th* most frequent, k*th* less frequent, etc. Like quicksort, quickselect was developed by Tony Hoare, and also known as *Hoare's selection algorithm*.
It has O(N)O(*N*) *average* time complexity and widely used in practice. It worth to note that its worth case time complexity is O(N2)O(*N*2), although the probability of this worst-case is negligible.
The approach is the same as for quicksort.
One chooses a pivot and defines its position in a sorted array in a linear time using so-called *partition algorithm*.
As an output, we have an array where the pivot is on its perfect position in the ascending sorted array, sorted by the frequency. All elements on the left of the pivot are less frequent than the pivot, and all elements on the right are more frequent or have the same frequency.
Hence the array is now split into two parts. If by chance our pivot element took N - k*th* final position, then k*k* elements on the right are these top k*k* frequent we're looking for. If not, we can choose one more pivot and place it in its perfect position.



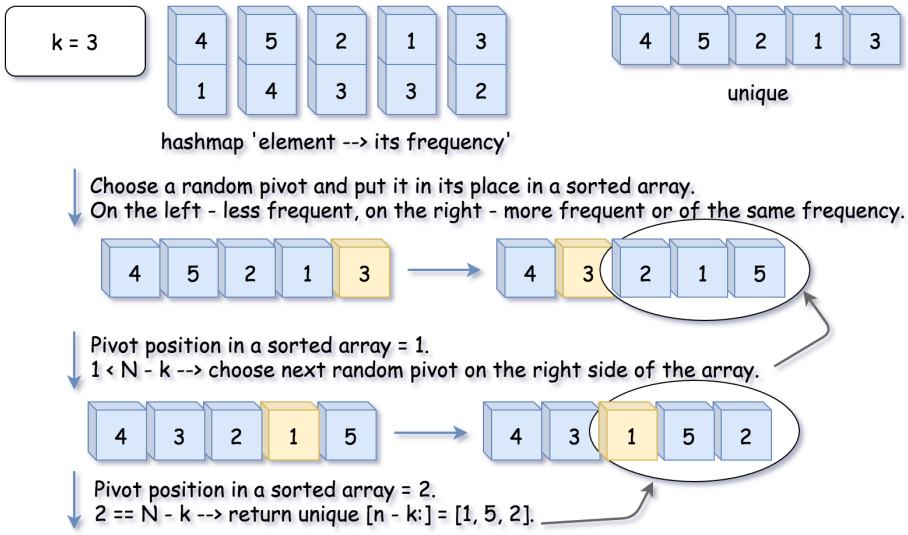
If that were a quicksort algorithm, one would have to process both parts of the array. That would result in O(Nlog N)O(*N*log*N*)time complexity. In this case, there is no need to deal with both parts since one knows in which part to search for N -

k*th* less frequent element, and that reduces the average time complexity to O(N) O(*N*).

Algorithm

The algorithm is quite straightforward :

- Build a hash map `element -> its frequency` and convert its keys into the array `unique` of unique elements. Note that elements are unique, but their frequencies are *not*. That means we need a partition algorithm that works fine with *duplicates*.
- Work with `unique` array. Use a partition scheme (please check the next section) to place the pivot into its perfect position `pivot_index` in the sorted array, move less frequent elements to the left of pivot, and more frequent or of the same frequency - to the right.
- Compare `pivot_index` and `N - k`.
    - If `pivot_index == N - k`, the pivot is `N - k`*th* most frequent element, and all elements on the right are more frequent or of the same frequency. Return these top k*k* frequent elements.
    - Otherwise, choose the side of the array to proceed recursively.



Lomuto's Partition Scheme

There is a zoo of partition algorithms. The most simple one is Lomuto's Partition Scheme, and so is what we will use in this article.

Here is how it works:

- Move pivot at the end of the array using swap.
- Set the pointer at the beginning of the array `store_index = left`.
- Iterate over the array and move all less frequent elements to the left `swap(store_index, i)`. Move `store_index` one step to the right after each swap.
- Move the pivot to its final place, and return this index.