Master algorithms together on Binary Search! Create a room, invite your friends, and race to finish the problems.

Daily Coding Problem #306

Problem

This problem was asked by Palantir.

You are given a list of `N` numbers, in which each number is located at most `k` places away from its sorted position. For example, if `k = 1`, a given element at index 4 might end up at indices 3, 4, or 5.

Come up with an algorithm that sorts this list in `O(N log k)` time.

Solution

One approach we can take is to use a sliding window. At each position, we find the minimum of the next `k` elements. If this element is less than the left end of our window, we can perform a swap to reorder the array.

```
def sort_k(array, k):
    for i in range(len(array)):
        low = i
        window = array[i + 1 : i + k + 1]
        for j, item in enumerate(window, i + 1):
            if item < array[low]:
                low = j
```

```
            if array[low] < array[i]:
                array[i], array[low] = array[low], array[i]

    return array
```

Since we must iterate over `N` windows of size `k`, the time complexity of this algorithm will be `O(N * k)`. We will also require `O(k)` space to store the window at any point.

We can improve this by using a heap. Initially we can place the first `k` elements in a min heap. Then, for each new element in the list, we add it to the heap, and pop the lowest element in the heap to append to our result.

Finally, we will take the `k` leftover elements in the heap and add them from lowest to highest to the result.

```
import heapq

def k_sort(array, k):
    res = []

    heap = []
    for i in range(k):
        heapq.heappush(heap, array[i])

    for i in range(k, len(array)):
        heapq.heappush(heap, array[i])
        res.append(heapq.heappop(heap))

    while heap:
        res.append(heapq.heappop(heap))

    return res
```

Since each heap push and pop operation is `O(log K)`, and we will perform each of these `N` times, this algorithm will satisfy our time requirement of `O(N log k)`. The space complexity will be `O(N)`, since we must build up a new result of all the elements.

© Daily Coding Problem 2019

Privacy Policy

Terms of Service

Press