Nearly Sorted

solved by	Senan
Platform	GeeksForGeeks
difficulty	Medium
tags	Heaps Sorting
language	C++
solved on	@14/11/2024
link	https://www.geeksforgeeks.org/problems/nearly-sorted-1587115620/1
Completion	☑

Intuition

We are given a nearly sorted array where each element is at most `k` positions away from its target position in the sorted array. This means that every element in the array will lie within a sliding window of size `k` relative to its correct position. A min-heap (or priority queue with `greater<int>`) is ideal for this type of problem as it helps efficiently keep track of the smallest elements within this window.

Approach

1. **Min-Heap Usage**:

   We use a min-heap to maintain a window of `k+1` elements. The reason for using `k+1` is that, for any element in the array, its final sorted position will always be within `k` indices of its original position, and hence the smallest element within a window of size `k+1` will be the next element in the sorted order.

2. **Inserting Elements**:

   Iterate over the array and push each element into the heap. If the heap size exceeds `k`, we pop the smallest element from the heap and place it into the current position in the array. This ensures that we maintain a sorted order for elements that are `k` indices apart.

3. **Emptying the Heap**:

   Once we finish iterating over the array, there may still be elements left in the heap. Pop all remaining elements from the heap and place them in the array in sorted order.

Complexity

Time Complexity:

- Building and maintaining the heap takes **O(log k)** time per operation.
- For each element, we perform an insertion and possibly a removal, resulting in an overall complexity of **O(n log k)**, where `n` is the number of elements in the array.

Space Complexity:

- The heap will store at most `k+1` elements, so the space complexity is **O(k)**.

Code

```
class Solution {
  public:
    void nearlySorted(vector<int>& arr, int k) {
        priority_queue<int, vector<int>, greater<int>> pq;
        int j = 0;
        for(int i = 0; i<arr.size(); i++){
            if(pq.size()>k){
                arr[j++] = pq.top();
                pq.pop();
            }
            pq.push(arr[i]);
        }
        while(j<arr.size()){
            arr[j++] = pq.top();
            pq.pop();
        }
        return;
    }
};
```