SortKSortedArray in C++

```
#include <iostream>
#include <queue>
using namespace std;

void sort(int arr[], int n, int k) {
    // Create a min-heap (priority_queue) to store the first k+1 elements
    priority_queue<int, vector<int>, greater<int>> pq;

    // Insert the first k+1 elements into the min-heap
    for (int i = 0; i <= k && i < n; i++) {
        pq.push(arr[i]);
    }

    // Process the remaining elements
    int index = 0;
    for (int i = k + 1; i < n; i++) {
        // Pop the smallest element from the min-heap and store it in arr
        arr[index++] = pq.top();
        pq.pop();

        // Push the current element into the min-heap
        pq.push(arr[i]);
    }

    // Pop and store the remaining elements from the min-heap
    while (!pq.empty()) {
        arr[index++] = pq.top();
        pq.pop();
    }
}

int main() {
    int arr[] = {2, 4, 1, 9, 6, 8};
    int k = 3;
    int n = sizeof(arr) / sizeof(arr[0]);

    sort(arr, n, k);

    // Print sorted array
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Input:

- arr[] = {2, 4, 1, 9, 6, 8}
- k = 3
- n = 6

🧠 **Understanding the Flow:**

1. Initialize a **min-heap** (using priority_queue with greater<int>).
2. Push the first k + 1 = 4 elements into the heap: [2, 4, 1, 9]
3. Pop the smallest from the heap and replace in arr (heapify and continue).
4. Keep pushing the next element and popping from the heap until all elements are processed.
5. At the end, empty the remaining heap into the array.

🔍 **Dry Run Table:**

Step	Min-Heap (Top = Min)	Array Update (arr[])
Init	[1, 2, 4, 9]	—
Pop	1 → arr[0] = 1	[**1**, _, _, _, _, _]
Push 6 → Heap = [2, 6, 4, 9]	—	
Pop	2 → arr[1] = 2	[1, **2**, _, _, _, _]
Push 8 → Heap = [4, 6, 9, 8]	—	
Pop	4 → arr[2] = 4	[1, 2, **4**, _, _, _]
No more to push	—	
Pop	6 → arr[3] = 6	[1, 2, 4, **6**, _, _]
Pop	8 → arr[4] = 8	[1, 2, 4, 6, **8**, _]
Pop	9 → arr[5] = 9	[1, 2, 4, 6, 8, **9**]

✅ **Final Output:**

1 2 4 6 8 9

1 2 4 6 8 9