Find smallest range containing elements from k lists

Given k sorted lists of integers of size n each, find the smallest range that includes at least element from each of the k lists. If more than one smallest ranges are found, print any one of them.

Example :

```
Input:
K = 3
arr1[] : [4, 7, 9, 12, 15]
arr2[] : [0, 8, 10, 14, 20]
arr3[] : [6, 12, 16, 30, 50]
Output:
The smallest range is [6 8]
Explanation: Smallest range is formed by
number 7 from first list, 8 from second
list and 6 from third list.


Input:
k = 3
arr1[] : [4, 7, 30]
arr2[] : [1, 2]
arr3[] : [20, 40]
The smallest range is [2 20]
```

The idea is to use min heap. Below are the steps –

1. Create a min heap of size k and insert first elements of all k lists into the heap.
2. Maintain two variables min and max to store minimum and maximum values present in the heap at any point. Note min will always contain value of the root of the heap.
3. Repeat following steps
    - Get minimum element from heap (minimum is always at root) and compute the range.
    - Replace heap root with next element of the list from which the min element is extracted. After replacing the root, heapify the tree. Update max if next element is greater. If the list doesn't have any more elements, break the loop.

Below is C++ implementation of above approach –

```
// C++ program to finds out smallest range that includes
// elements from each of the given sorted lists.
#include<iostream>
#include<limits.h>
using namespace std;

#define N 5

// A min heap node
struct MinHeapNode
{
    int element; // The element to be stored
    int i; // index of the list from which the element is taken
    int j; // index of the next element to be picked from list
};

// Prototype of a utility function to swap two min heap nodes
void swap(MinHeapNode *x, MinHeapNode *y);

// A class for Min Heap
class MinHeap
{
    MinHeapNode *harr; // pointer to array of elements in heap
    int heap_size; // size of min heap
public:
```

```
public:
    // Constructor: creates a min heap of given size
    MinHeap(MinHeapNode a[], int size);

    // to heapify a subtree with root at given index
    void MinHeapify(int );

    // to get index of left child of node at index i
    int left(int i) { return (2*i + 1); }

    // to get index of right child of node at index i
    int right(int i) { return (2*i + 2); }

    // to get the root
    MinHeapNode getMin() { return harr[0]; }

    // to replace root with new node x and heapify() new root
    void replaceMin(MinHeapNode x) { harr[0] = x; MinHeapify(0); }
};

// Constructor: Builds a heap from a given array a[] of given size
MinHeap::MinHeap(MinHeapNode a[], int size)
{
    heap_size = size;
    harr = a; // store address of array
    int i = (heap_size - 1)/2;
    while (i >= 0)
    {
        MinHeapify(i);
        i--;
    }
}

// A recursive method to heapify a subtree with root at
// given index. This method assumes that the subtrees
// are already heapified
void MinHeap::MinHeapify(int i)
{
    int l = left(i);
    int r = right(i);
    int smallest = i;
    if (l < heap_size &&
        harr[l].element < harr[i].element)
        smallest = l;
    if (r < heap_size &&
        harr[r].element < harr[smallest].element)
        smallest = r;
    if (smallest != i)
    {
        swap(harr[i], harr[smallest]);
        MinHeapify(smallest);
    }
}

// This function takes an k sorted lists in the form of
// 2D array as an argument. It finds out smallest range
// that includes elements from each of the k lists.
void findSmallestRange(int arr[][N], int k)
{
    // Create a min heap with k heap nodes. Every heap node
    // has first element of an list
    int range = INT_MAX;
    int min = INT_MAX, max = INT_MIN;
    int start, end;

    MinHeapNode *harr = new MinHeapNode[k];
    for (int i = 0; i < k; i++)
    {
        harr[i].element = arr[i][0]; // Store the first element
        harr[i].i = i; // index of list
        harr[i].j = 1; // Index of next element to be stored
                       // from list

        // ...
```

```
        // store max element
        if (harr[i].element > max)
            max = harr[i].element;
    }

    MinHeap hp(harr, k); // Create the heap

    // Now one by one get the minimum element from min
    // heap and replace it with next element of its list
    while (1)
    {
        // Get the minimum element and store it in output
        MinHeapNode root = hp.getMin();

        // update min
        min = hp.getMin().element;

        // update range
        if (range > max - min + 1)
        {
            range = max - min + 1;
            start = min;
            end = max;
        }

        // Find the next element that will replace current
        // root of heap. The next element belongs to same
        // list as the current root.
        if (root.j < N)
        {
            root.element = arr[root.i][root.j];
            root.j += 1;

            // update max element
            if (root.element > max)
                max = root.element;
        }

        // break if we have reached end of any list
        else break;

        // Replace root with next element of list
        hp.replaceMin(root);
    }

    cout << "The smallest range is " << "["
         << start << " " << end << "]" << endl;;
}

// Driver program to test above functions
int main()
{
    int arr[][N] = {
                    {4, 7, 9, 12, 15},
                    {0, 8, 10, 14, 20},
                    {6, 12, 16, 30, 50}
                    };
    int k = sizeof(arr)/sizeof(arr[0]);

    findSmallestRange(arr, k);

    return 0;
}
```

Output :

```
The smallest range is [6 8]
```

Time Complexity: The while loop inside findSmallestRange() function can run maximum n*k times. In every iteration of loop, we call heapify which takes O(Logk) time. Therefore, the time complexity is O(nk Logk).