1. Merge K Sorted Arrays (GFG)

Output

Single sorted Array

1	2	3	4	5	6	7	8	9	10	11	12
0	1	2	3	4	5	6	7	8	9	10	11

13	14	15	16	17	18	19	20	21	22	23	24
12	13	14	15	16	17	18	19	20	21	22	23

Input

N = 6
K

n-size

arr1

1	5	9	13	17	21
0	1	2	3	4	5

arr2

2	6	10	14	18	22
0	1	2	3	4	5

arr3

3	7	11	15	19	23
0	1	2	3	4	5

arrK

4	8	12	16	20	24
0	1	2	3	4	5

ALGORITHMS

STEP1 Find First min Element of K-Arrays

Note when we merge two sorted Array

↳ To HUM phle First min element Find Karte Hai OR ussi Element Ko Ans Array me pahle push Karte Hai

Right - YES

↳ We find min-Element using min-Heap with Time complexity O(1)

EX1 **INPUT** N=4
K=3

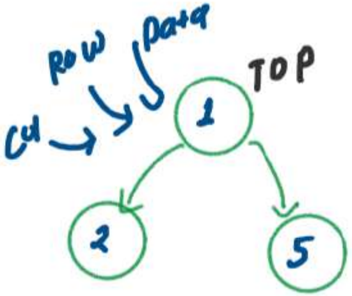
arr1	1	4	8	10
	0	1	2	3

arr2	2	3	6	9
	0	1	2	3

arr3	5	7	11	12
	0	1	2	3

Ⓘ CREATE MIN-HEAP USING First Elements of K-Arrays

Iteration1



Ⓘⓘ FIND MIN ELEMENT

MIN = 1 RIndex = 0 CIndex = 0

Ⓘⓥ Update TOP and Heapify it

TOP = 4

arr1	0	1	4	8	10
		0	1	2	3

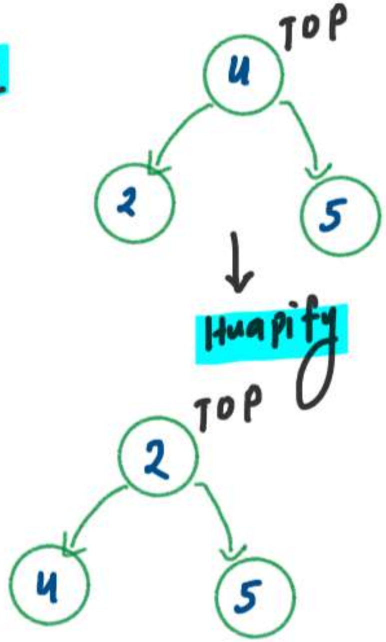
arr2	1	2	3	6	9
		0	1	2	3

arr3	2	5	7	11	12
		0	1	2	3

Ⓘⓘⓘ Push MIN into ANS ARRAY

ANS	1											
	0	1	2	3	4	5	6	7	8	9	10	11

Iteration2



Ⅱ FIND MIN ELEMENT

MIN = 2 RIndex=1 CIndex=0

Ⅳ Update TOP and Heapify it

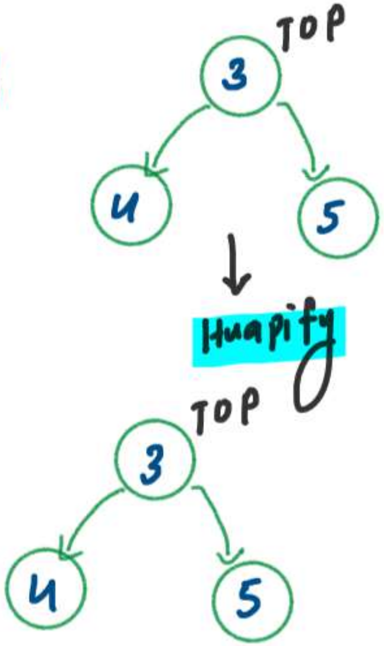
TOP= 3

	0	1	2	3
arr1 0	1	4	8	10
arr2 1	2	3	6	9
arr3 2	5	7	11	12

Ⅲ Push MIN into ANS ARRAY

	0	1	2	3	4	5	6	7	8	9	10	11
ANS	1	2										

Iteration3



Ⅱ FIND MIN ELEMENT

MIN = 3 RIndex= 1 CIndex= 1

Ⅳ Update TOP and Heapify it

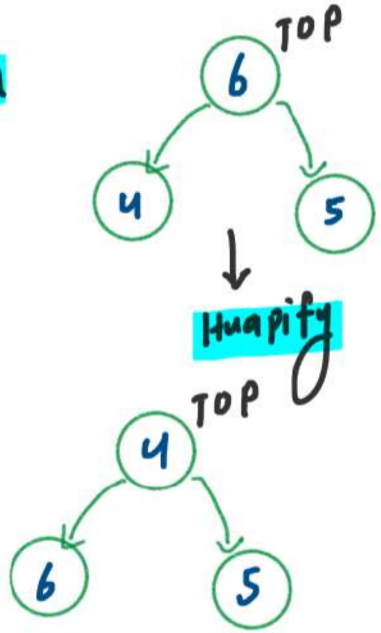
TOP= 6

arr1	0	1	4	8	10
		0	1	2	3
arr2	1	2	3	6	9
		0	1	2	3
arr3	2	5	7	11	12
		0	1	2	3

Ⅲ Push MIN into ANS ARRAY

ANS	1	2	3									
	0	1	2	3	4	5	6	7	8	9	10	11

Iteration 4



(II) FIND MIN ELEMENT

MIN = 4 RIndex = 0 CIndex = 1

(IV) Update TOP and Heapify it

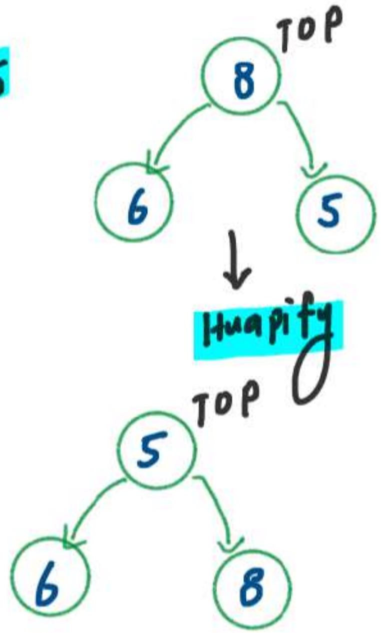
TOP = 8

arr1	0	1	4	8	10
		0	1	2	3
arr2	1	2	3	6	9
		0	1	2	3
arr3	2	5	7	11	12
		0	1	2	3

(III) Push MIN into ANS ARRAY

ANS	1	2	3	4								
	0	1	2	3	4	5	6	7	8	9	10	11

Iterations



(II) FIND MIN ELEMENT

MIN = 5 RIndex = 2 CIndex = 0

(IV) Update TOP and Heapify it

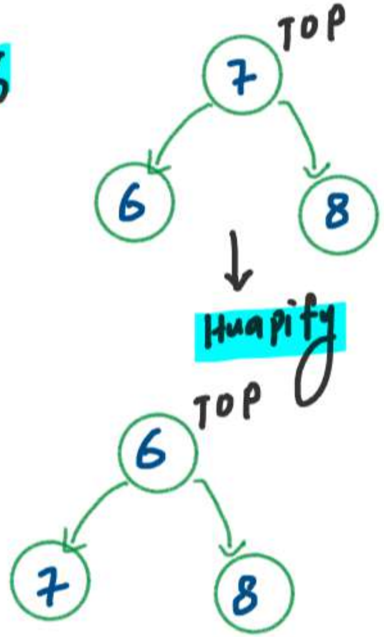
TOP = 7

arr1	0	1	4	8	10
		0	1	2	3
arr2	1	2	3	6	9
		0	1	2	3
arr3	2	5	7	11	12
		0	1	2	3

(III) Push MIN into ANS ARRAY

ANS	1	2	3	4	5							
	0	1	2	3	4	5	6	7	8	9	10	11

Iteration 6



(II) FIND MIN ELEMENT

MIN = 6 RIndex = 1 CIndex = 2

(IV) Update TOP and Heapify it

TOP = 9

arr1	0	1	4	8	10
		0	1	2	3

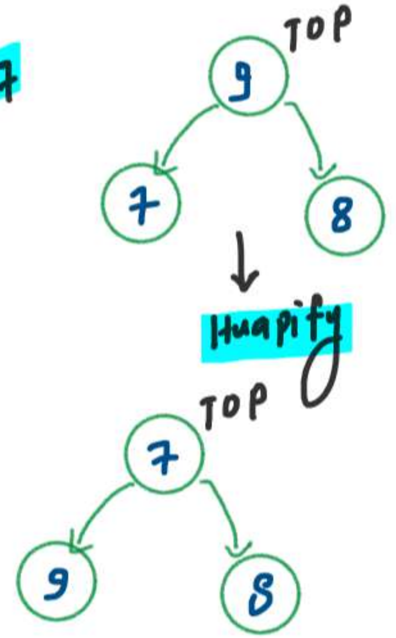
arr2	1	2	3	6	9
		0	1	2	3

arr3	2	5	7	11	12
		0	1	2	3

(III) Push MIN into ANS ARRAY

ANS	1	2	3	4	5	6						
	0	1	2	3	4	5	6	7	8	9	10	11

Iteration 7



Ⓘ FIND MIN ELEMENT

MIN = 7 RIndex = 2 CIndex = 1

Ⓘ UPdate TOP and Heapify it

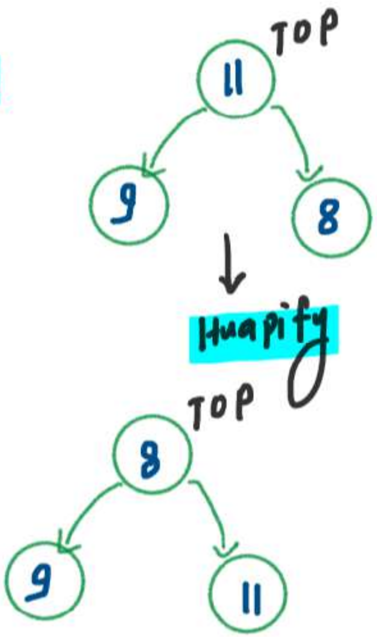
TOP = 11

	0	1	2	3
arr1 0	1	4	8	10
arr2 1	2	3	6	9
arr3 2	5	7	11	12

Ⓘ Push MIN into ANS ARRAY

	0	1	2	3	4	5	6	7	8	9	10	11
ANS	1	2	3	4	5	6	7					

Iteration8



(II) FIND MIN ELEMENT

MIN = 8 RIndex = 0 CIndex = 2

(IV) Update TOP and Heapify it

TOP = 10

arr1 0	1	4	8	10
	0	1	2	3

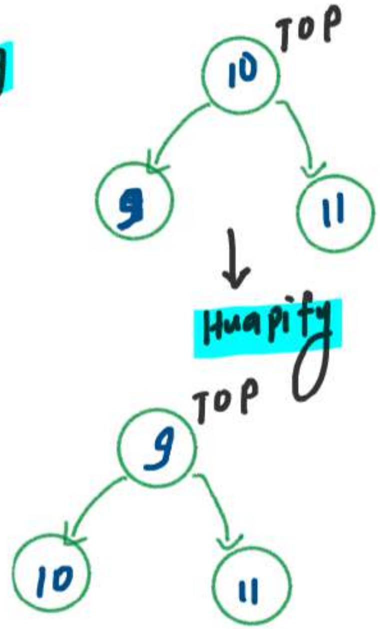
arr2 1	2	3	6	9
	0	1	2	3

arr3 2	5	7	11	12
	0	1	2	3

(III) Push MIN into ANS ARRAY

ANS	1	2	3	4	5	6	7	8				
	0	1	2	3	4	5	6	7	8	9	10	11

Iteration 9



Ⅱ FIND MIN ELEMENT

MIN = 9 RIndex= 1 CIndex= 3

Ⅳ Update TOP and Heapify it

TOP= 10

CIndex < N
3 < 3 X

(GALTI YAHNA PAR HOTI HAI)

arr1	0	1	4	8	10
		0	1	2	3

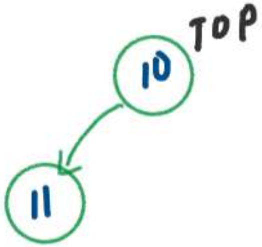
arr2	1	2	3	6	9
		0	1	2	3

arr3	2	5	7	11	12
		0	1	2	3

Ⅲ Push MIN into ANS ARRAY

ANS	1	2	3	4	5	6	7	8	9			
	0	1	2	3	4	5	6	7	8	9	10	11

Iteration 10



II FIND MIN ELEMENT

MIN = 10 RIndex = 0 CIndex = 3

IV Update TOP and Heapify it

TOP = 11

CIndex < N
3 < 3 X

(GALTI YANHA PAR HOTI HAI)

arr1	0	1	4	8	10
		0	1	2	3

arr2	1	2	3	6	9
		0	1	2	3

arr3	2	5	7	11	12
		0	1	2	3

III Push MIN into ANS ARRAY

ANS	1	2	3	4	5	6	7	8	9	10		
	0	1	2	3	4	5	6	7	8	9	10	11

Iteration11

II FIND MIN ELEMENT

MIN = 11   RIndex = 2   CIndex = 2

IV Update TOP and Heapify it

TOP = 12

	0	1	2	3
arr1 0	1	4	8	10

	0	1	2	3
arr2 1	2	3	6	9

	0	1	2	3
arr3 2	5	7	~~11~~	12

III Push MIN into ANS ARRAY

	0	1	2	3	4	5	6	7	8	9	10	11
ANS	1	2	3	4	5	6	7	8	9	10	11	

Iteration12

12 TOP ✗

Ⅱ FIND MIN ELEMENT

MIN = 12 RIndex=2 CIndex=3

Ⅳ Update TOP and Heapify it

TOP= TOP UPDATE NAHI KAR SAKTE HAI

Heap is Empty NOW AND CoIndex 3<3 ✗

FINAL OUTPUT

arr1	0	1	4	8	10
		0	1	2	3
arr2	1	2	3	6	9
		0	1	2	3
arr3	2	5	7	11	~~12~~
		0	1	2	3

Ⅲ Push MIN into ANS ARRAY

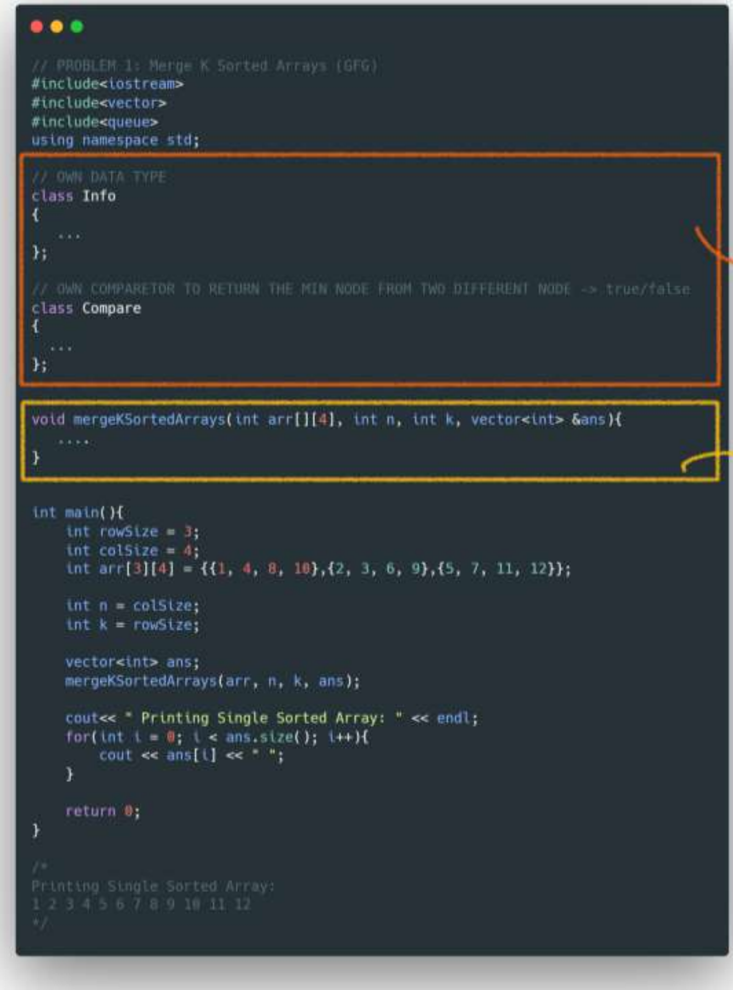
ANS	1	2	3	4	5	6	7	8	9	10	11	12
	0	1	2	3	4	5	6	7	8	9	10	11

(IV) Array main jis min value ko push kar rahe Hai uski next Element k dubara Heap ke TOP Element ko update karne ke liye Hume RowIndex, colIndex and data ki need Hogi Right → YES

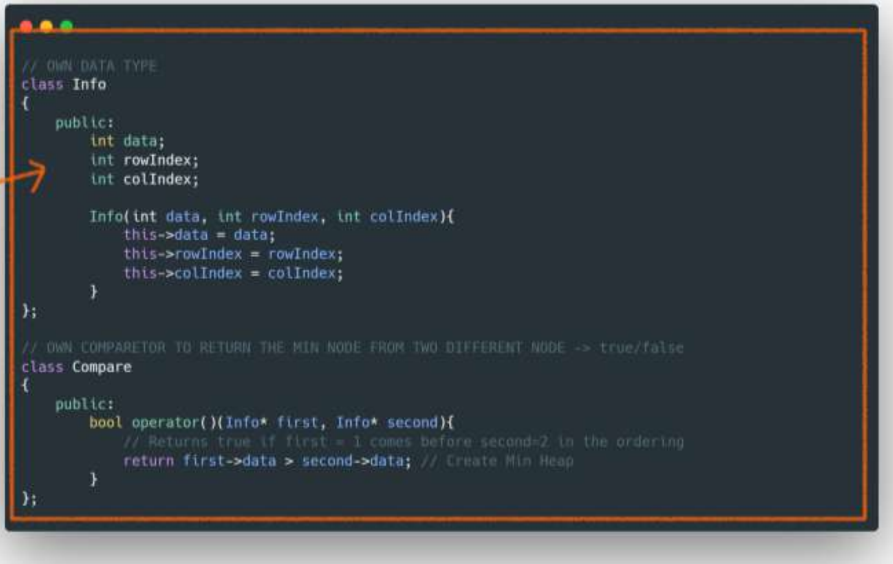
↳ To Hum Apna Khud ka Ek New data Type create kar lenge jisme yeh three properties milengi.

```
class Info {
    public:
            int data;
            int rowIndex;
            int colIndex;

            Info( int data,
                  int rowIndex,
                  int colIndex){
            this->data = data;
            this->rowIndex = rowIndex;
            this->colIndex = colIndex;
}       }
```

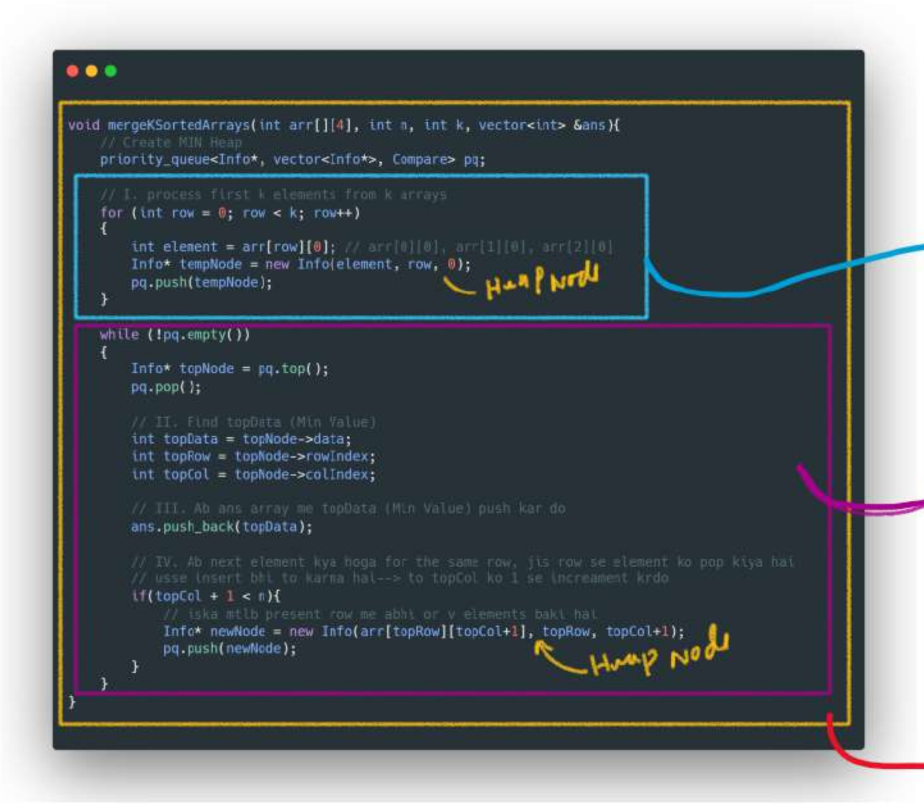
MIN Heap KA NODE Hai
Info
data
rowIndex
colIndex



```cpp
// PROBLEM 1: Merge K Sorted Arrays (GFG)
#include<iostream>
#include<vector>
#include<queue>
using namespace std;

// OWN DATA TYPE
class Info
{
   ...
};

// OWN COMPARETOR TO RETURN THE MIN NODE FROM TWO DIFFERENT NODE -> true/false
class Compare
{
  ...
};

void mergeKSortedArrays(int arr[][4], int n, int k, vector<int> &ans){
   ....
}

int main(){
    int rowSize = 3;
    int colSize = 4;
    int arr[3][4] = {{1, 4, 8, 10},{2, 3, 6, 9},{5, 7, 11, 12}};

    int n = colSize;
    int k = rowSize;

    vector<int> ans;
    mergeKSortedArrays(arr, n, k, ans);

    cout<< " Printing Single Sorted Array: " << endl;
    for(int i = 0; i < ans.size(); i++){
        cout << ans[i] << " ";
    }

    return 0;
}

/*
Printing Single Sorted Array:
1 2 3 4 5 6 7 8 9 10 11 12
*/
```



```cpp
// OWN DATA TYPE
class Info
{
    public:
        int data;
        int rowIndex;
        int colIndex;

        Info(int data, int rowIndex, int colIndex){
            this->data = data;
            this->rowIndex = rowIndex;
            this->colIndex = colIndex;
        }
};

// OWN COMPARETOR TO RETURN THE MIN NODE FROM TWO DIFFERENT NODE -> true/false
class Compare
{
    public:
        bool operator()(Info* first, Info* second){
            // Returns true if first = 1 comes before second=2 in the ordering
            return first->data > second->data; // Create Min Heap
        }
};
```



```cpp
void mergeKSortedArrays(int arr[][4], int n, int k, vector<int> &ans){
    // Create MIN Heap
    priority_queue<Info*, vector<Info*>, Compare> pq;

    // I. process first k elements from k arrays
    for (int row = 0; row < k; row++)
    {
        int element = arr[row][0]; // arr[0][0], arr[1][0], arr[2][0]
        Info* tempNode = new Info(element, row, 0);
        pq.push(tempNode);
    }
    while (!pq.empty())
    {
        Info* topNode = pq.top();
        pq.pop();

        // II. Find topData (Min Value)
        int topData = topNode->data;
        int topRow = topNode->rowIndex;
        int topCol = topNode->colIndex;

        // III. Ab ans array me topData (Min Value) push kar do
        ans.push_back(topData);

        // IV. Ab next element kya hoga for the same row, jis row se element ko pop kiya hai
        // usse insert bhi to karna hai--> to topCol ko 1 se increament krdo
        if(topCol + 1 < n){
            // iska mtlb present row me abhi or v elements baki hai
            Info* newNode = new Info(arr[topRow][topCol+1], topRow, topCol+1);
            pq.push(newNode);
        }
    }
}
```

Time Complexity

$$\left[\begin{array}{l}\text{Heap ki T.C.} = O(\log(K)) \\ \text{For loop ki T.C.} = O(K)\end{array}\right\} O(K * \log(K))$$

K = No. of Arrays

+

$$\left[\begin{array}{l}\text{While loop ki T.C.} = O(N) \\ \text{Heap ki T.C.} = O(\log(K))\end{array}\right\} O(N * \log(K))$$

N = Total Elements of All arrays

Overall T.C.

$O(K * \log(K)) + O(N * \log(K))$

SPACE complexity

(MIN Heap) Priority queue ki S.C. = $O(K)$

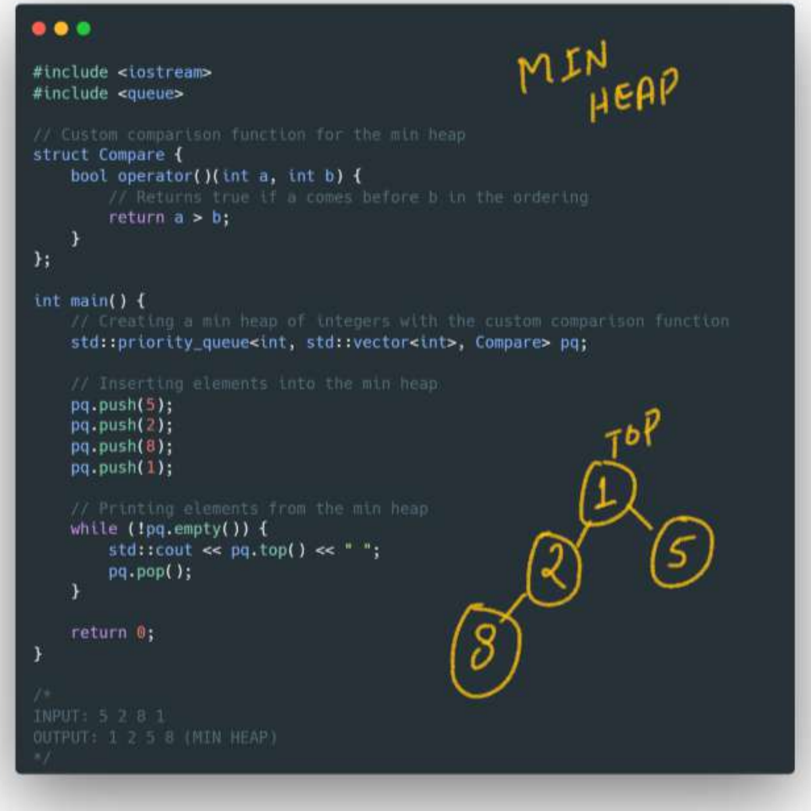
K = No. of arrays

(Ans) vector Array ki S.C. = $O(N)$

N = Total Elements of All arrays

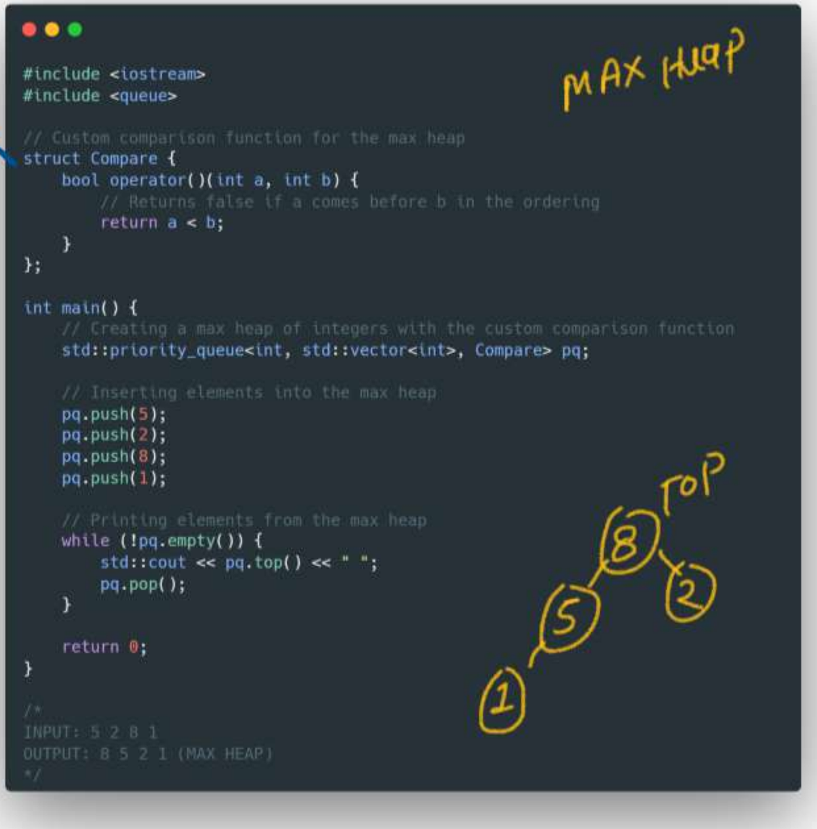
Overall S.C.

$O(K) + O(N)$



```
#include <iostream>
#include <queue>

// Custom comparison function for the min heap
struct Compare {
    bool operator()(int a, int b) {
        // Returns true if a comes before b in the ordering
        return a > b;
    }
};

int main() {
    // Creating a min heap of integers with the custom comparison function
    std::priority_queue<int, std::vector<int>, Compare> pq;

    // Inserting elements into the min heap
    pq.push(5);
    pq.push(2);
    pq.push(8);
    pq.push(1);

    // Printing elements from the min heap
    while (!pq.empty()) {
        std::cout << pq.top() << " ";
        pq.pop();
    }

    return 0;
}

/*
INPUT: 5 2 8 1
OUTPUT: 1 2 5 8 (MIN HEAP)
*/
```

Struct ke
place
par
class
v
use
kar
sakte
ho

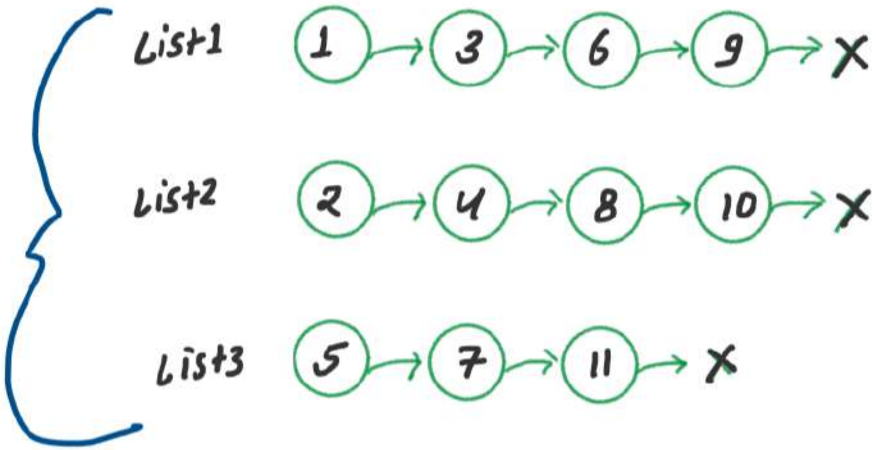


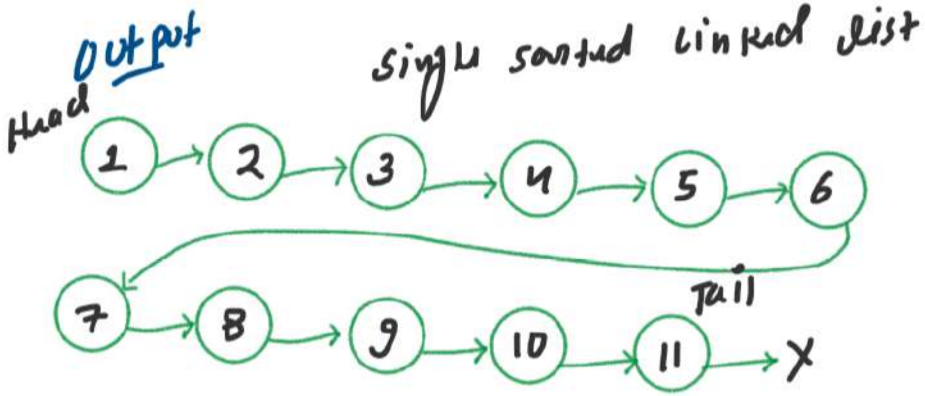
```
#include <iostream>
#include <queue>

// Custom comparison function for the max heap
struct Compare {
    bool operator()(int a, int b) {
        // Returns false if a comes before b in the ordering
        return a < b;
    }
};

int main() {
    // Creating a max heap of integers with the custom comparison function
    std::priority_queue<int, std::vector<int>, Compare> pq;

    // Inserting elements into the max heap
    pq.push(5);
    pq.push(2);
    pq.push(8);
    pq.push(1);

    // Printing elements from the max heap
    while (!pq.empty()) {
        std::cout << pq.top() << " ";
        pq.pop();
    }

    return 0;
}

/*
INPUT: 5 2 8 1
OUTPUT: 8 5 2 1 (MAX HEAP)
*/
```

2. Merge K Sorted Linked List (Leetcode-23)

Input

Lists [[1,3,6,9], [2,4,8,9], [5,7,11]]

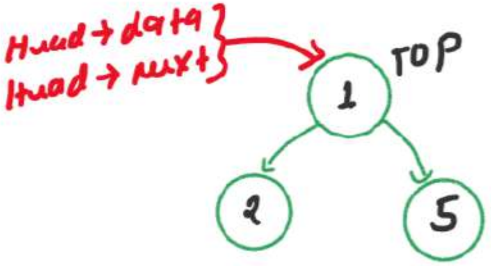




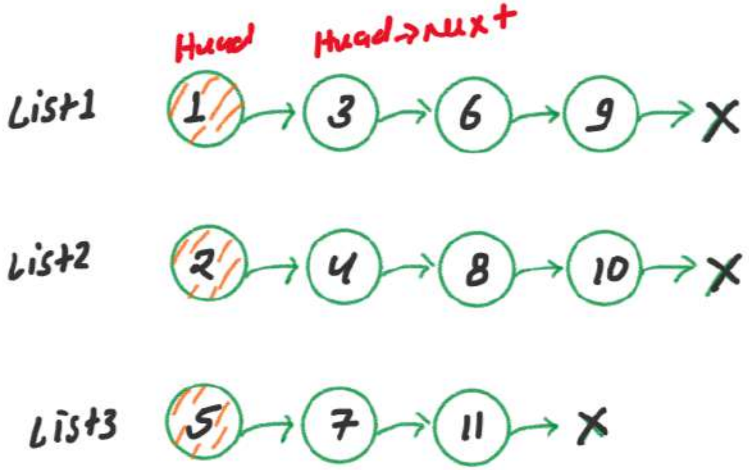
Algorithm

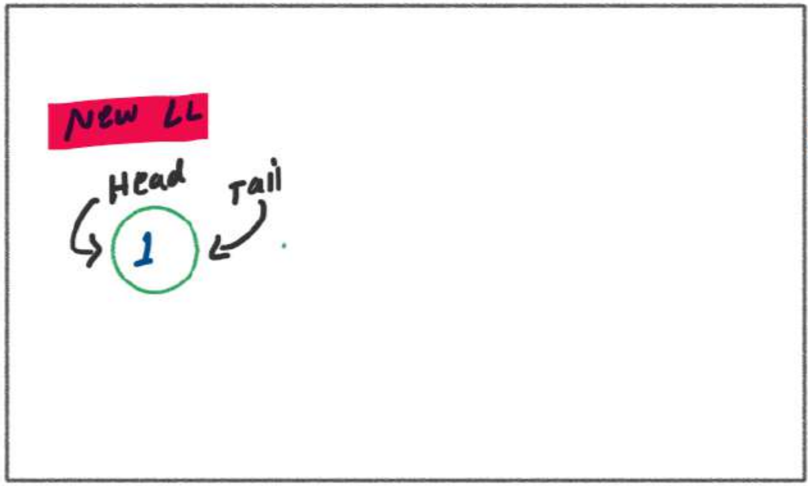
Ⅰ Make min Heap using First K-Nodes of all lists

iteration 1



K = Lists.size()
= 3





Ⅱ Create new linked list

Initially → Head = X Tail = X

Ⅲ Fetch TOP and push pop to New linked list

TOP = 1

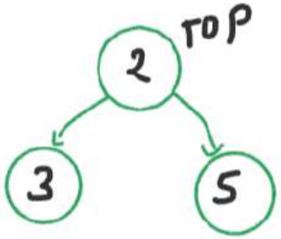
pop the TOP

Head = 1 and update the TOP if(Tail→next != Null)

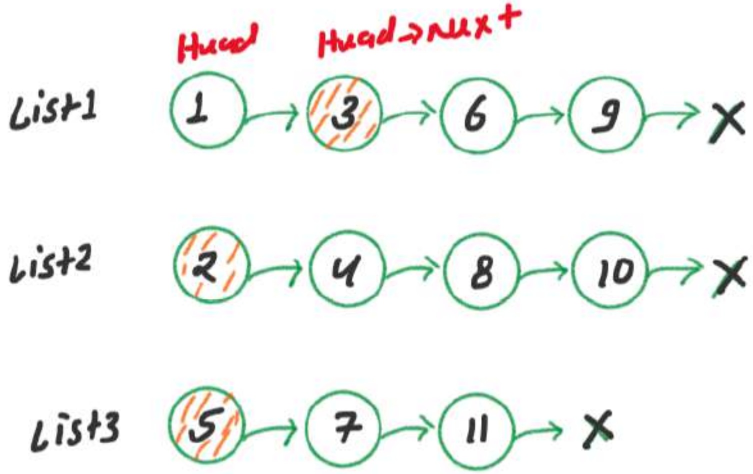
↳ TOP = 3

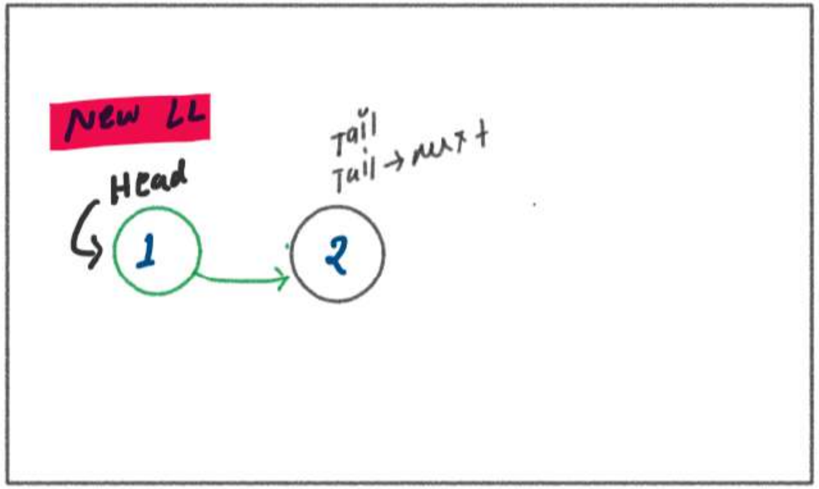
tail = 1

Iteration 2



K = Lists.size()
= 3





Ⅱ Create new linked list

Head = 1   Tail = 1

Ⅲ Fetch TOP and push pop to New linked list

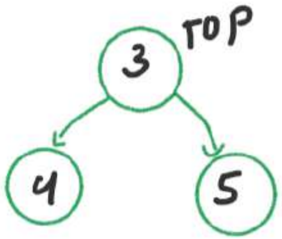
TOP = 2

pop the TOP

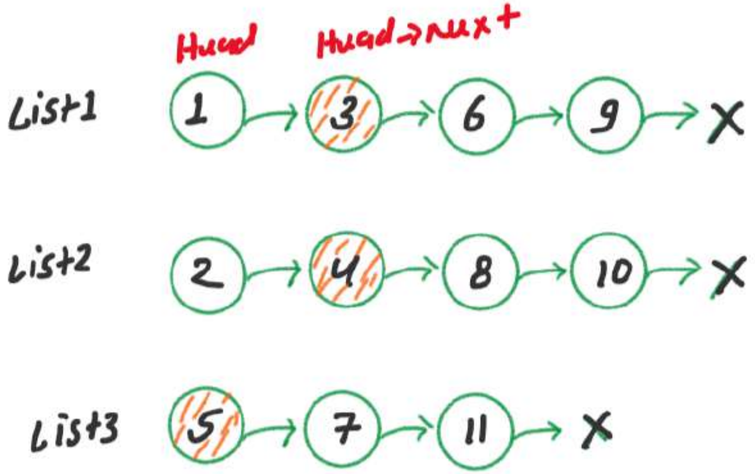
tail→next = 2 and update the TOP if(Tail→next != NULL)

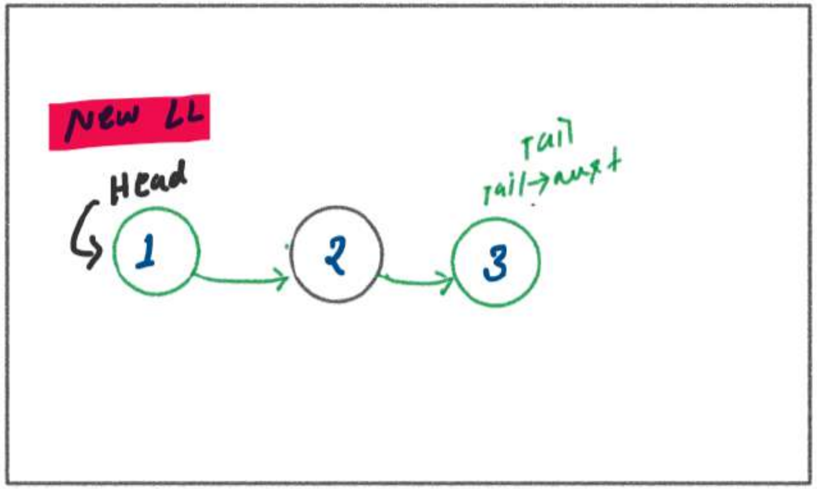
tail = 2   ↳ TOP = 4

Iteration3



K = Lists.size()
= 3





Ⅱ Create new linked list

Head = 1 Tail = 2

Ⅲ Fetch TOP and push pop to New linked list

TOP = 3

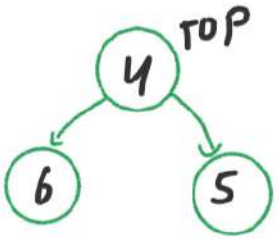
pop the TOP

tail→next = 3 and update the TOP if(Tail→next != NULL)

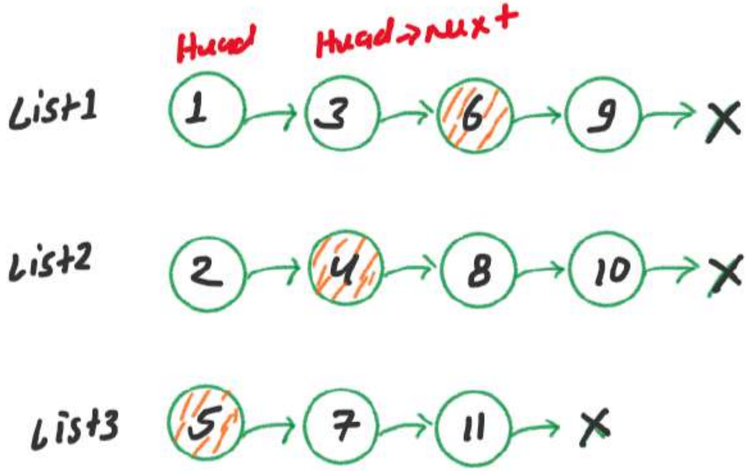
Tail = 3

↳ TOP = 6

iteration 4



K = lists.size()
= 3



Ⅱ Create new linked list

Head = 1   Tail = 3

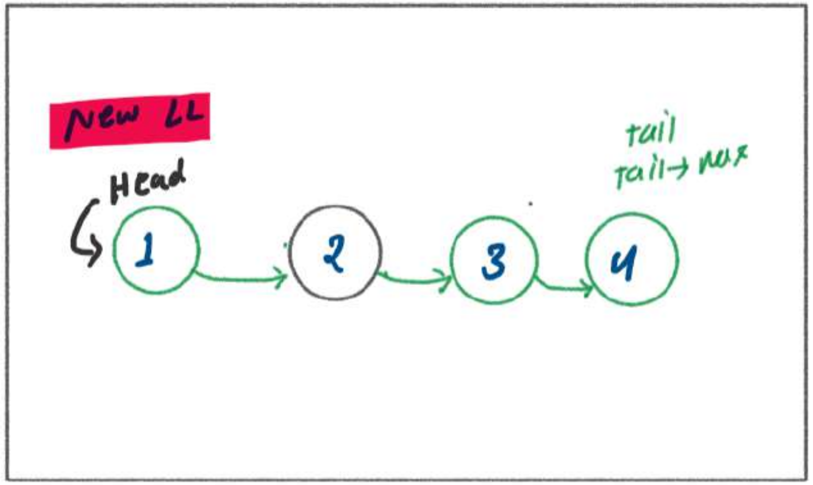
Ⅲ Fetch TOP and push pop to New linked list

TOP = 4

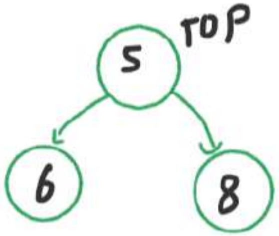
pop the TOP

tail→next = 4 and update the TOP if(Tail→next != NULL)

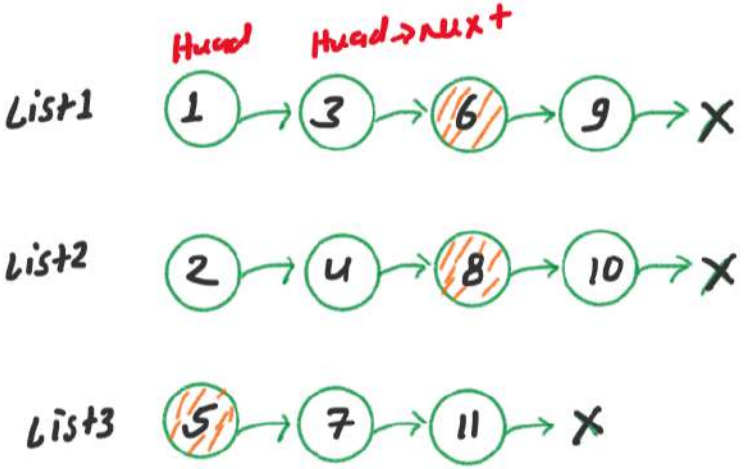
tail = 4   ↳ TOP = 8

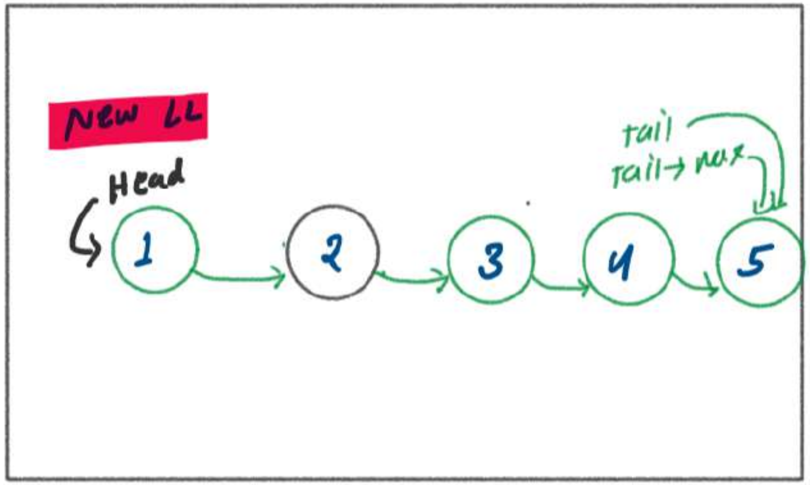


Iterations



K = Lists.size()
= 3

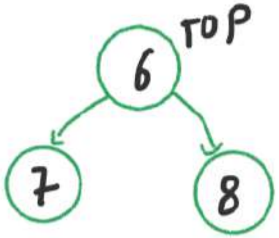




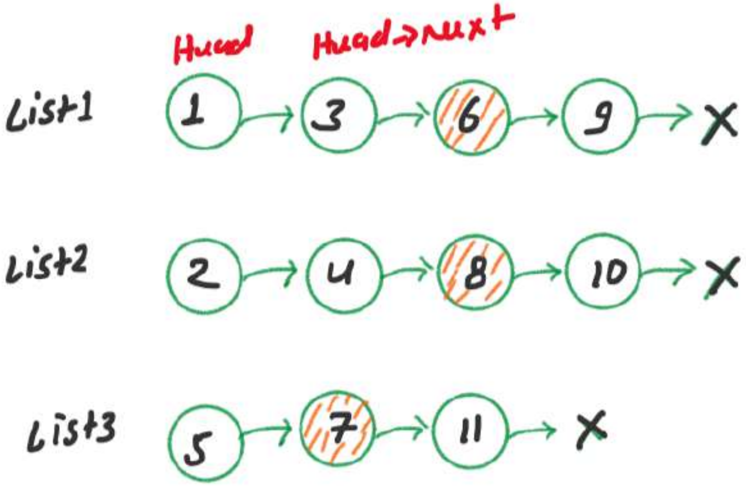
II Create new linked list
Head = 1   Tail = 4

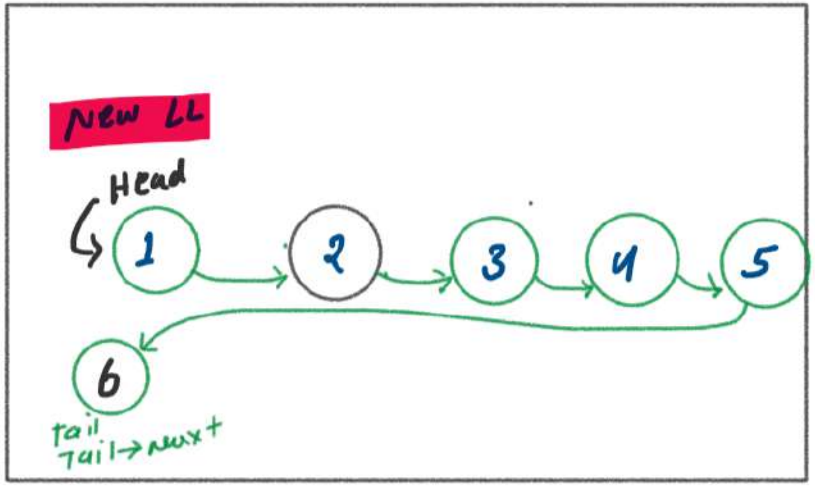
III Fetch TOP and push pop to New linked list
TOP = 5
pop the TOP
tail→next = 5   and update the TOP if(Tail→next != NULL)
tail = 5
↳ TOP = 7

Iteration5



K = Lists.size()
= 3

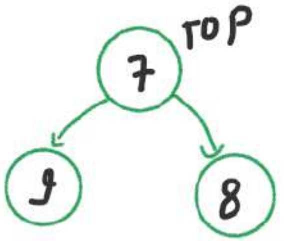




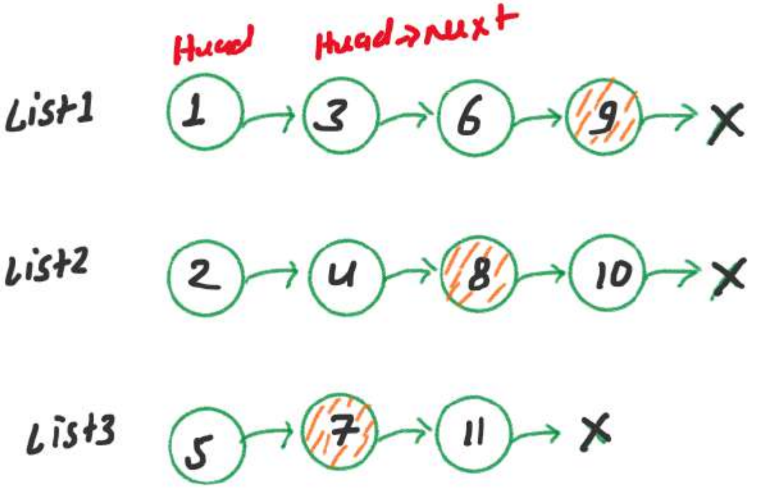
II) Create new linked list
Head = 1 Tail = 5

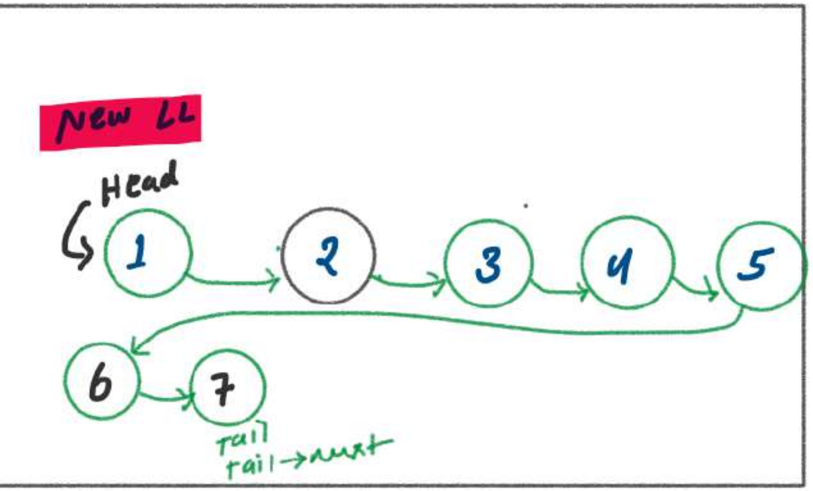
III) Fetch TOP and push pop to New linked list
TOP = 6
pop the TOP
tail→next = 6 and update the TOP if(Tail→next != NULL)
tail = 6 ↳ TOP = 9

Iteration7



K = Lists.size()
= 3





Ⅱ Create new linked list

Head = 1 Tail = 6

Ⅲ Fetch TOP and push pop to New linked list

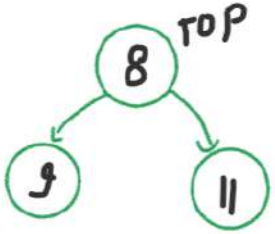
TOP = 7

pop the TOP

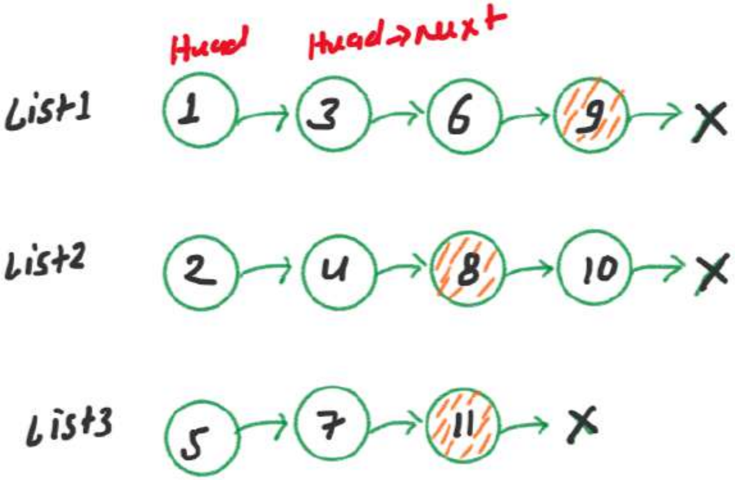
tail->next = 7 and update the TOP if(Tail->next != NULL)

tail = 7 ↳ TOP = 11

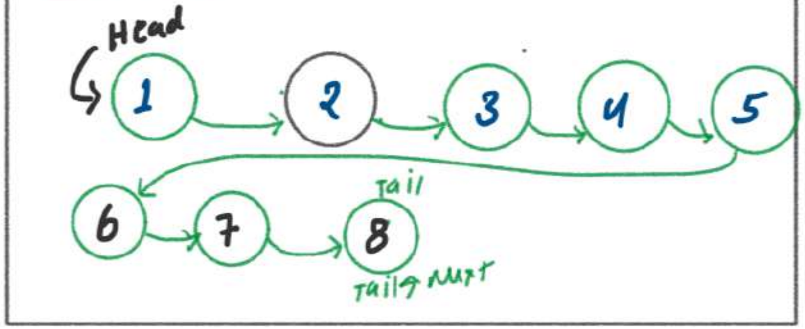
iteration8



K = Lists.size()
= 3



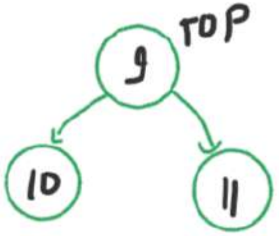
New LL



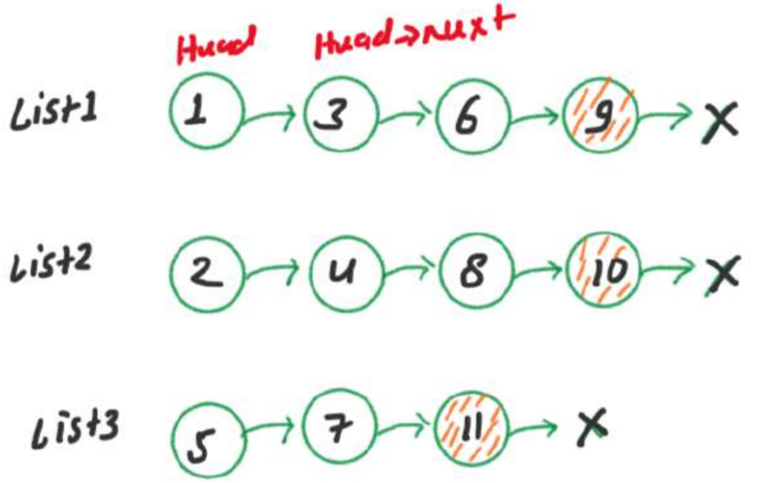
(II) Create new linked list
Head = 1 Tail = 7

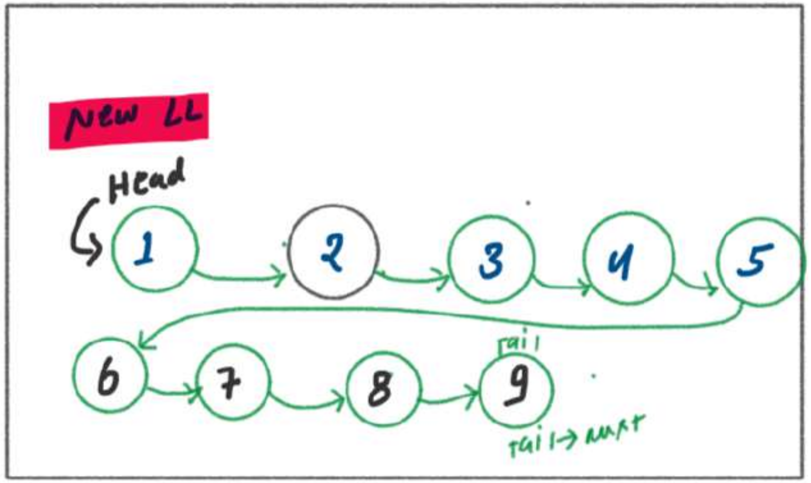
(III) Fetch TOP and push pop to New linked list
TOP = 8
pop the TOP
tail→next = 8 and update the TOP if(Tail→next != NULL)
tail = 8 ↳ TOP = 10

Iterating



$K = \text{Lists.size()}$
$= 3$

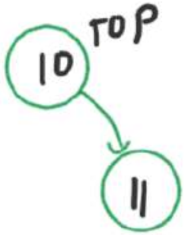




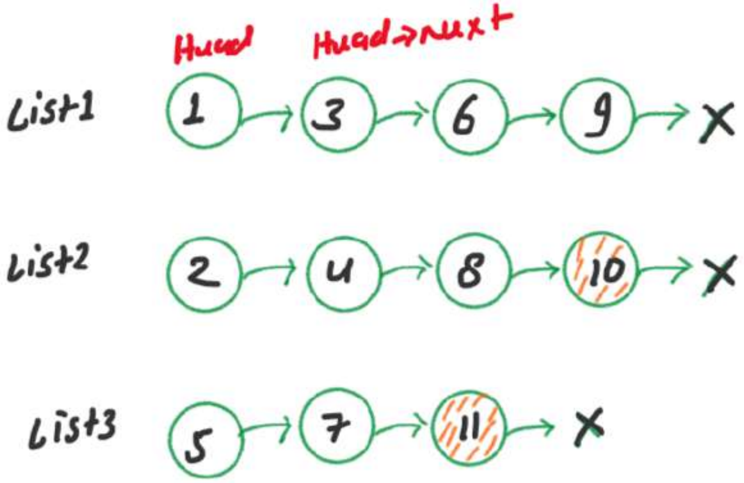
(II) Create new linked list
Head = 1   Tail = 8

(III) Fetch TOP and push pop to New linked list
TOP = 9
pop the TOP
tail→next = 9   and update the TOP if(Tail→next != NULL)   TRUE
tail = 9
↳ TOP = TOP update nahi hoga

iteration10



K = Lists.size()
= 3



II Create new Linked list

Head = 1 Tail = 9

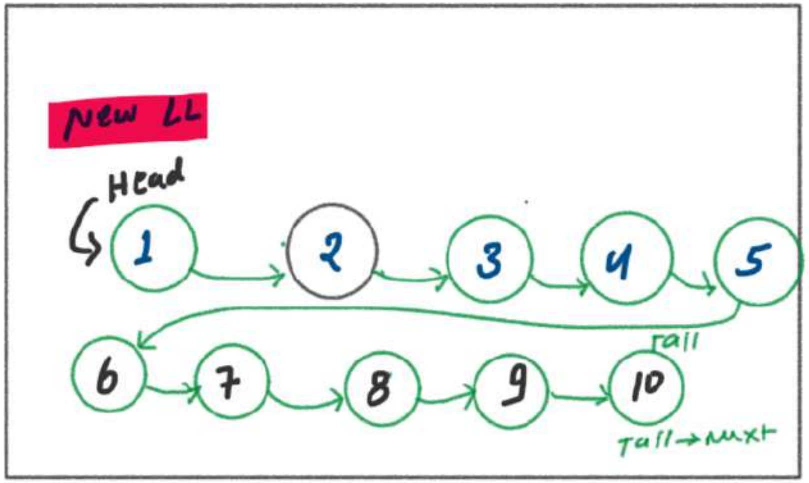
III Fetch TOP and push pop to New Linked list

TOP = 10

pop the TOP

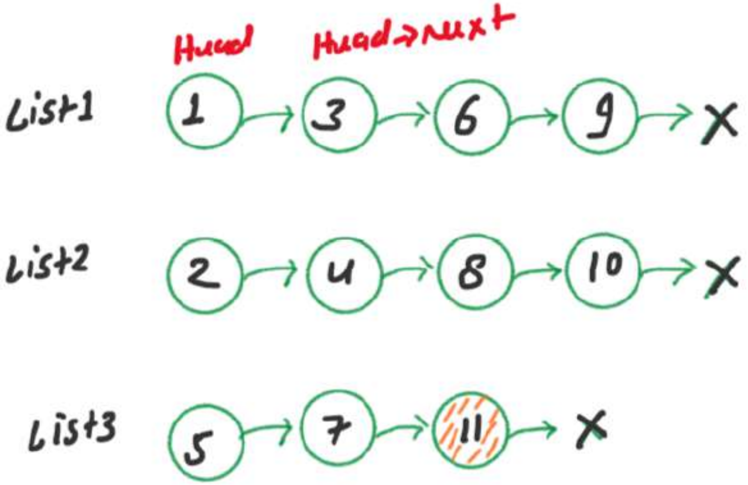
tail→next = 10 and update the TOP if(Tail→next != NULL) TRUE

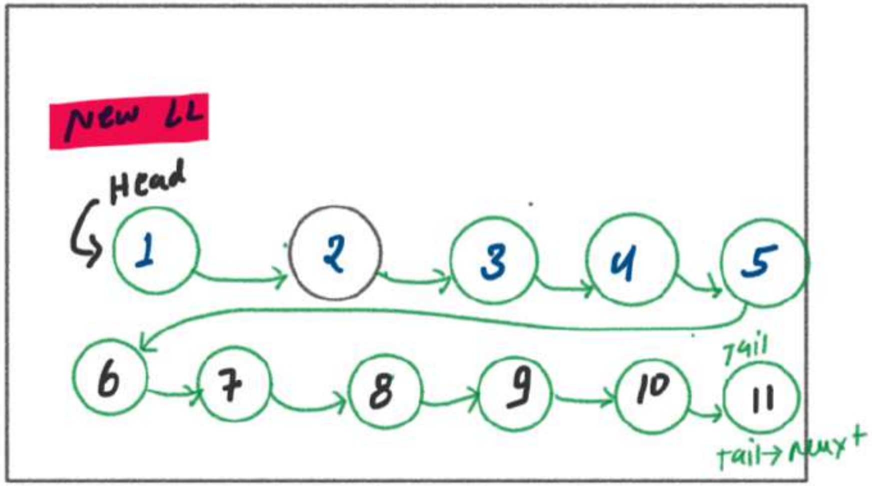
tail = 10 ↳ TOP = TOP update Nani Hoga



iteration11

K = Lists.size()
= 3





(II) Create new linked list

Head = 1 Tail = 10

(III) Fetch TOP and push pop to New linked list

TOP = 11

pop the TOP and update the TOP if(Tail→next != Null) TRUE

tail→next = 11

Tail = 11

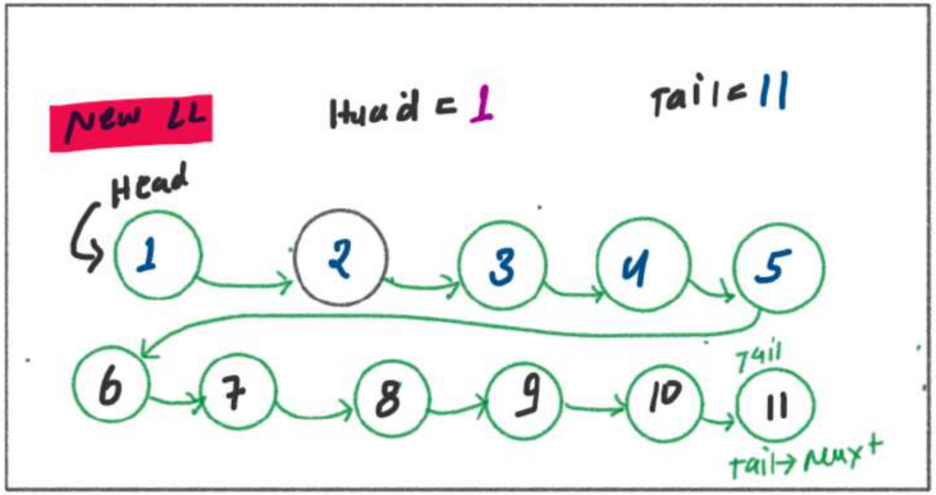
→ TOP = TOP update Nahi Hoga

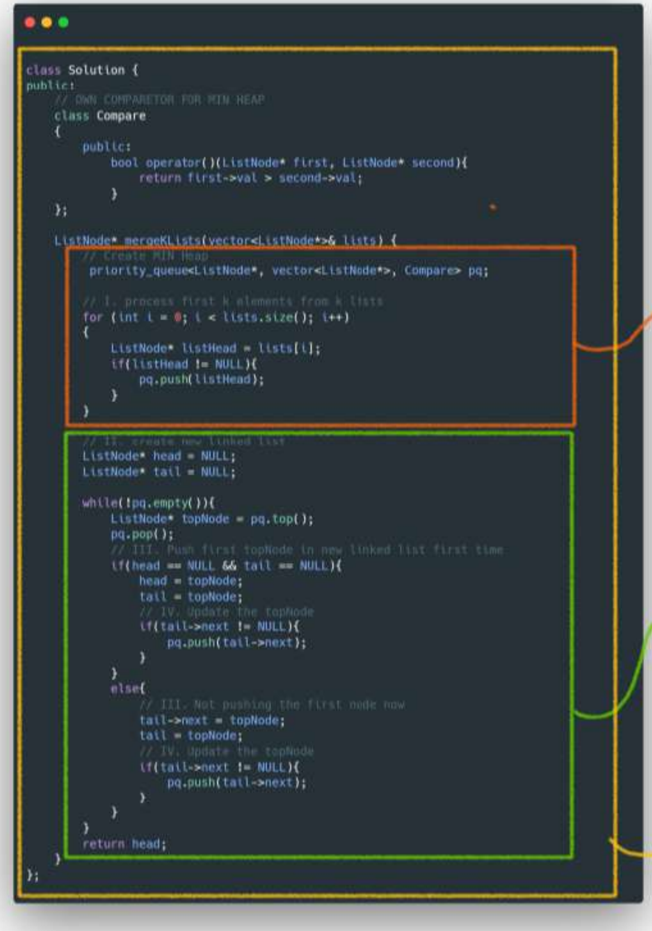
Iteration 2

Heap is now EMPty

↳ STOP

Final Output





```cpp
class Solution {
public:
    // OWN COMPARETOR FOR MIN HEAP
    class Compare
    {
        public:
            bool operator()(ListNode* first, ListNode* second){
                return first->val > second->val;
            }
    };

    ListNode* mergeKLists(vector<ListNode*>& lists) {
        // Create MIN Heap
         priority_queue<ListNode*, vector<ListNode*>, Compare> pq;

        // I. process first k elements from k lists
        for (int i = 0; i < lists.size(); i++)
        {
            ListNode* listHead = lists[i];
            if(listHead != NULL){
                pq.push(listHead);
            }
        }
        // II. create new linked list
        ListNode* head = NULL;
        ListNode* tail = NULL;

        while(!pq.empty()){
            ListNode* topNode = pq.top();
            pq.pop();
            // III. Push first topNode in new linked list first time
            if(head == NULL && tail == NULL){
                head = topNode;
                tail = topNode;
                // IV. Update the topNode
                if(tail->next != NULL){
                    pq.push(tail->next);
                }
            }
            else{
                // III. Not pushing the first node now
                tail->next = topNode;
                tail = topNode;
                // IV. Update the topNode
                if(tail->next != NULL){
                    pq.push(tail->next);
                }
            }
        }
        return head;
    }
};
```

Time Complexity

For loop ki T.C. = O(K)

MIN Heap ki T.C. = O(log K) → T.C. = O(K * log K)

where
K = Total Linked lists

Merge Hone ke T.C.

→ While Loop T.C. = O(N)

MIN Heap T.C. = O(log K) → T.C. = O(N * log K)

where
N = All Nodes of Each Linked Lists

Overall T.C. = O(K * log K) + O(N * log K)

= O(N * log K)

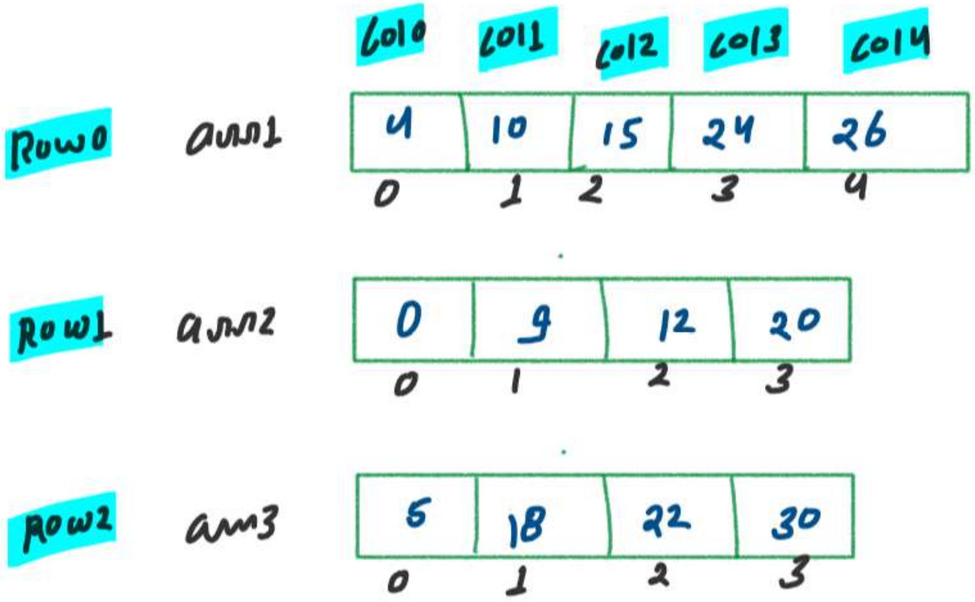
SPACE complexity

↳ MIN Heap is taking space $O(K)$

Where K is Number of linked lists

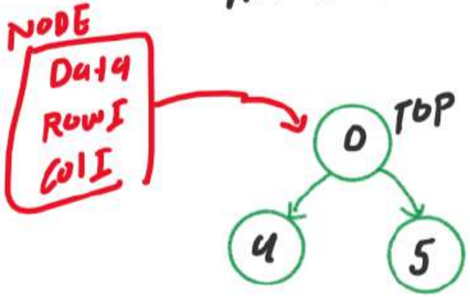
3. Smallest Range in K Lists (Leetcode-632)

```
Input: NUMS = [
                [4,10,15,24,26],
                [0,9,12,20],
                [5,18,22,30]
              ]

Output: [20,24]

Explanation:
List 1: [4, 10, 15, 24,26], 24 is in range [20,24].
List 2: [0, 9, 12, 20], 20 is in range [20,24].
List 3: [5, 18, 22, 30], 22 is in range [20,24].
```



Algorithm

(I) Create MIN Heap of First K-Elements of All lists



K = Total Row Size
= 3

			Col0	Col1	Col2	Col3	Col4
Row0	arr1	0	4	10	15	24	26
			0	1	2	3	4
Row1	arr2	1	0	9	12	20	
			0	1	2	3	
Row2	arr3	2	6	18	22	30	
			0	1	2	3	

MIN = 0
MAX = 5

Range [MIN , MAX]
AnsStart = 0
AnsEND = 5

Iteration 1

(II) Fetch TOP and POP

TOP = 0

update min = 0

update the Ans (Range) when $[(maxi - MIN) < (AnsEnd - AnsStart)]$ ↙ For smaller Range

$5 - 0 < 5 - 0$

$5 < 5$ ✗

AnsStart = 0
AnsEND = 5

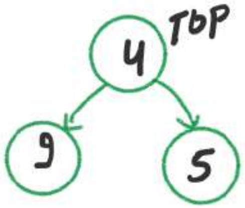
(III) Insert New Element

NewElement = 9

ColI = 1
RowI = 1

update maxi = 9

Iteration 2



NewRange

MIN = 4
MAX = 9

Old Range

Range [MIN , MAX]
AnsStart = 0
AnsEND = 5

		Col0	Col1	Col2	Col3	Col4
Row0	arr1	4	10	15	24	26
		0	1	2	3	4
Row1	arr2	0	9	12	20	
		0	1	2	3	
Row2	arr3	5	18	22	30	
		0	1	2	3	

(III) Insert New Element

NewElmnt = 10

colI = 1

RowI = 0

update maxi = 10

(II) Fetch TOP and pop

TOP = 4

update min = 4

update the Ans (Range) when [(maxi - MIN) < (AnsEnd - AnsStart)] ← For smaller Range

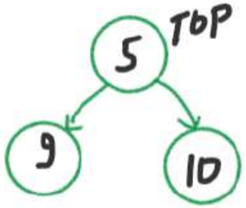
(9 - 4) < (5 - 0)

5 < 5 ✗

AnsStart = 0

AnsEND = 5

Iteration 3



New Range

MIN = 5
MAX = 10

Old Range

Range [MIN , MAX]
AnsStart = 0
AnsEND = 5

		Col0	Col1	Col2	Col3	Col4
Row0	arr1	4	10	15	24	26
		0	1	2	3	4
Row1	arr2	0	9	12	20	
		0	1	2	3	
Row2	arr3	5	18	22	30	
		0	1	2	3	

(III) Insert New Element

NewElement = 18

ColI = 1
RowI = 2

Update maxi = 18

(II) Fetch TOP and POP

TOP = 5

Update min = 5

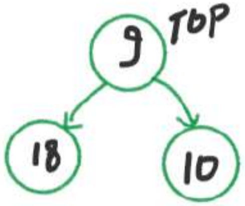
Update the Ans (Range) when [(maxi − MIN) < (AnsEnd − AnsStart)] ← For smaller Range

10 − 5   5 − 0

5 < 5 ✗

AnsStart = 0
AnsEND = 5

Iteration 4



New Range

MIN = 9
MAX = 18

Old Range

Range [MIN , MAX]
AnsStart = 0
AnsEND = 5

		Col0	Col1	Col2	Col3	Col4
Row0	arr1	4	10	15	24	26
		0	1	2	3	4
Row1	arr2	0	9	12	20	
		0	1	2	3	
Row2	arr3	5	18	22	30	
		0	1	2	3	

(III) Insert New Element

NewElement = 12

colI = 2
RowI = 1

✗
(12 > 18)

update maxi = 18

(II) Fetch TOP and pop

TOP = 9

update min = 9

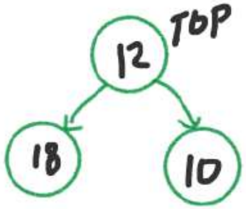
update the Ans (Range) when [(maxi − MIN) < (AnsEnd − AnsStart)] ← For smaller Range

18 − 9 < 5 − 0

9 < 5 ✗

AnsStart = 0
AnsEND = 5

Iteration 5



New Range

MIN = 12
MAX = 18

Old Range

Range[MIN , MAX]

AnsStart = 0
AnsEND = 5

		Col0	Col1	Col2	Col3	Col4
Row0	arr1	4	10	15	24	26
		0	1	2	3	4
Row1	arr2	0	9	12	20	
		0	1	2	3	
Row2	arr3	5	18	22	30	
		0	1	2	3	

(III) Insert New Element

NewElement = 20

colI = 3
RowI = 0

Update maxi = 20 $(20 > 18)$ ✓

(II) Fetch TOP and POP

TOP = 12

update min = 12

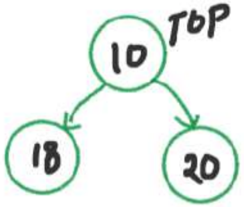
update the Ans (Range) when $[(maxi - MIN) < (AnsEnd - AnsStart)]$ ← For smaller Range

$18 - 12 < 5 - 0$

$6 < 5$ ✗

AnsStart = 0
AnsEND = 5

Iteration 6



New Range

MIN = 10
MAX = 20

Old Range

Range[MIN , MAX]
AnsStart = 0
AnsEND = 5

		Col0	Col1	Col2	Col3	Col4
Row0	arr1	4	10	15	24	26
		0	1	2	3	4
Row1	arr2	0	9	12	20	
		0	1	2	3	
Row2	arr3	5	18	22	30	
		0	1	2	3	

(III) Insert New Element

NewElement = 15

ColI = 2

RowI = 1

(15 > 20) ✗

Update maxi = 20

(II) Fetch TOP and POP

TOP = 10

Update min = 10

Update the Ans (Range) when [(maxi − MIN) < (AnsEnd − AnsStart)] ← For smaller Range

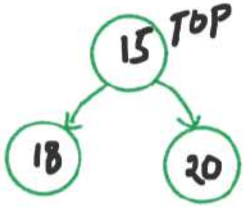
$20 - 10 < 5 - 0$

$10 < 5$ ✗

AnsStart = 0

AnsEND = 5

[Item 7]



New Range

MIN = 15
MAX = 20

Old Range

Range[MIN , MAX]
AnsStart = 0
AnsEND = 5

		col0	col1	col2	col3	col4
Row0	arr1	4	10	15	24	26
		0	1	2	3	4
Row1	arr2	0	9	12	20	
		0	1	2	3	
Row2	arr3	6	18	22	30	
		0	1	2	3	

(III) Insert New Element

NewElement = 24

colI = 3
RowI = 0

(24 > 20) ✓

Update maxi = 24

(II) Fetch TOP and POP

TOP = 15

Update min = 15

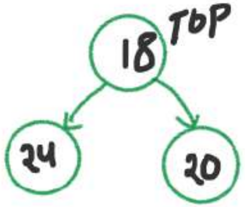
Update the Ans (Range) when [(maxi − MIN) < (AnsEnd − AnsStart)] ← For smaller Range

20 − 15 < 5 − 0

5 < 5 ✗

AnsStart = 0
AnsEND = 5

Iteration 8



New Range

MIN = 18
MAX = 24

Old Range

Range [MIN , MAX]
AnsStart = 0
AnsEND = 5

		Col0	Col1	Col2	Col3	Col4
Row0	arr1	4	10	15	24	26
		0	1	2	3	4
Row1	arr2	0	9	12	20	
		0	1	2	3	
Row2	arr3	5	18	22	30	
		0	1	2	3	

(III) Insert New Element

NewElement = 22

ColI = 2
RowI = 2

$(22 \overset{\times}{>} 24)$

update maxi = 24

(II) Fetch TOP and pop

TOP = 18

update min = 18

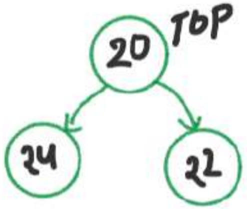
update the Ans (Range) when $[(maxi - MIN) < (AnsEnd - AnsStart)]$ ← For smaller Range

$24 - 18 < 5 - 0$

$6 < 5$ ✗

AnsStart = 0
AnsEND = 5

Iteration 9



New Range

MIN = 20
MAX = 24

Old Range

Range [MIN , MAX]
AnsStart = 0
AnsEND = 5

		Col0	Col1	Col2	Col3	Col4
Row0	arr1	4	10	15	24	26
		0	1	2	3	4
Row1	arr2	0	9	12	20	
		0	1	2	3	
Row2	arr3	5	18	22	30	
		0	1	2	3	

(III) Insert New Element

R[1][4] X

col1 = 3+1 = 4

→ STOP ⇒ Loop Ko Break Kardo

(II) Fetch Top and Pop

TOP = 20

update min = 20

update the Ans (Range) when [(maxi − MIN) < (AnsEnd − AnsStart)] ← For smaller Range

24 − 20 < 5 − 0

4 < 5 ✓

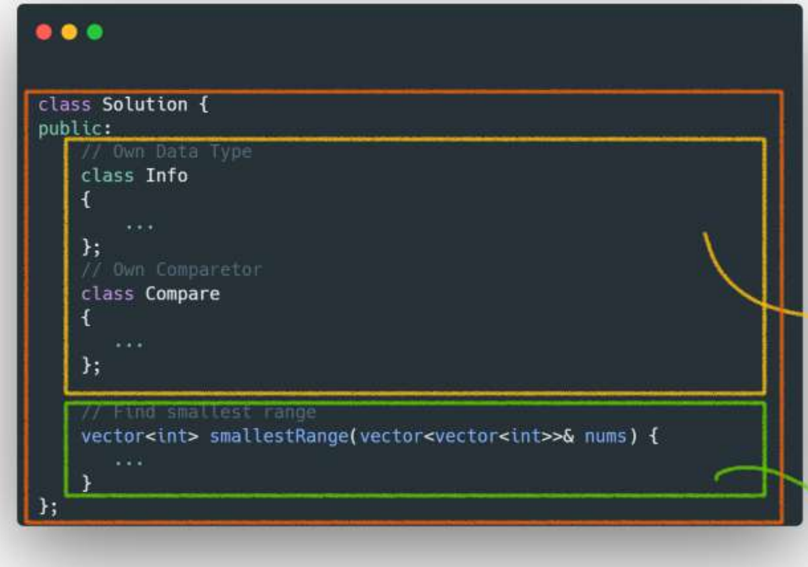
AnsStart = 20
AnsEND = 24

Final output →

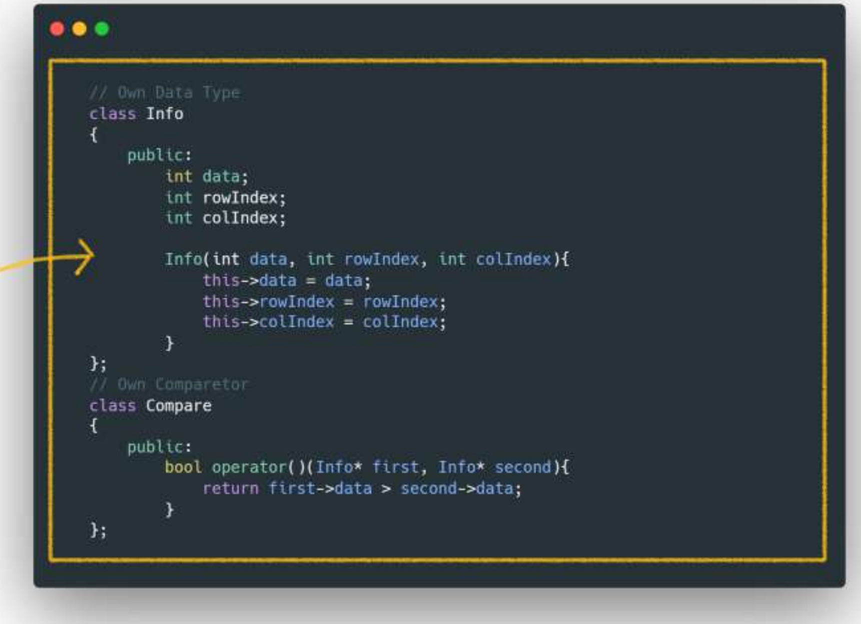
Range [20 24]
Range[MIN , MAX]
AnsStart= 20
AnsEND= 24

Iss Range me jo Bhi Elements Honge wo Har EK List me Present Honge.

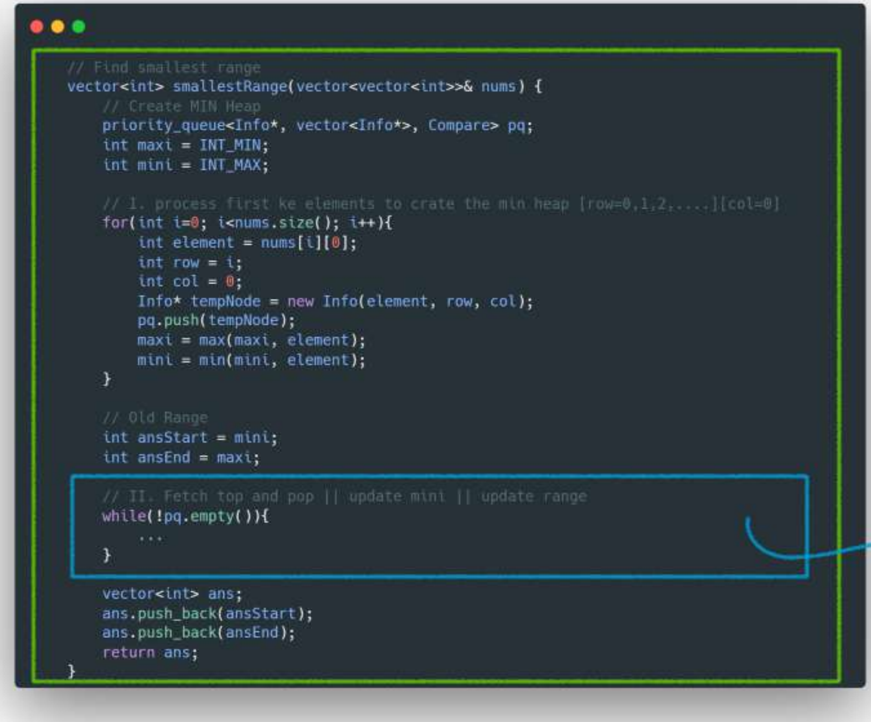
		col0	col1	col2	col3	col4	
Row0	arr1	4	10	15	24	26	20 21 22 23 **24**
		0	1	2	3	4	
Row1	arr2	0	9	12	20		**20** 21 22 23 24
		0	1	2	3		
Row2	arr3	5	18	22	30		20 21 **22** 23 24
		0	1	2	3		



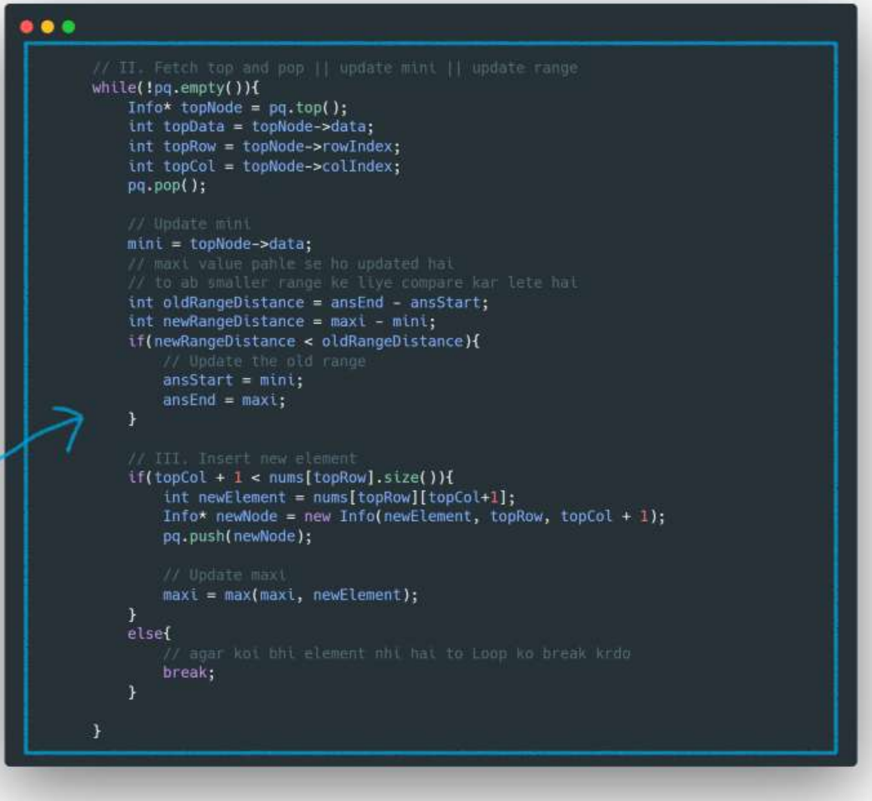
```
class Solution {
public:
    // Own Data Type
    class Info
    {
        ...
    };
    // Own Comparetor
    class Compare
    {
        ...
    };
    // Find smallest range
    vector<int> smallestRange(vector<vector<int>>& nums) {
        ...
    }
};
```



```
// Own Data Type
class Info
{
    public:
        int data;
        int rowIndex;
        int colIndex;

        Info(int data, int rowIndex, int colIndex){
            this->data = data;
            this->rowIndex = rowIndex;
            this->colIndex = colIndex;
        }
};
// Own Comparetor
class Compare
{
    public:
        bool operator()(Info* first, Info* second){
            return first->data > second->data;
        }
};
```



```cpp
// Find smallest range
vector<int> smallestRange(vector<vector<int>>& nums) {
    // Create MIN Heap
    priority_queue<Info*, vector<Info*>, Compare> pq;
    int maxi = INT_MIN;
    int mini = INT_MAX;

    // I. process first ke elements to crate the min heap [row=0,1,2,....][col=0]
    for(int i=0; i<nums.size(); i++){
        int element = nums[i][0];
        int row = i;
        int col = 0;
        Info* tempNode = new Info(element, row, col);
        pq.push(tempNode);
        maxi = max(maxi, element);
        mini = min(mini, element);
    }

    // Old Range
    int ansStart = mini;
    int ansEnd = maxi;

    // II. Fetch top and pop || update mini || update range
    while(!pq.empty()){
        ...
    }

    vector<int> ans;
    ans.push_back(ansStart);
    ans.push_back(ansEnd);
    return ans;
}
```



```cpp
    // II. Fetch top and pop || update mini || update range
    while(!pq.empty()){
        Info* topNode = pq.top();
        int topData = topNode->data;
        int topRow = topNode->rowIndex;
        int topCol = topNode->colIndex;
        pq.pop();

        // Update mini
        mini = topNode->data;
        // maxi value pahle se ho updated hai
        // to ab smaller range ke liye compare kar lete hai
        int oldRangeDistance = ansEnd - ansStart;
        int newRangeDistance = maxi - mini;
        if(newRangeDistance < oldRangeDistance){
            // Update the old range
            ansStart = mini;
            ansEnd = maxi;
        }

        // III. Insert new element
        if(topCol + 1 < nums[topRow].size()){
            int newElement = nums[topRow][topCol+1];
            Info* newNode = new Info(newElement, topRow, topCol + 1);
            pq.push(newNode);

            // Update maxi
            maxi = max(maxi, newElement);
        }
        else{
            // agar koi bhi element nhi hai to Loop ko break krdo
            break;
        }

    }
```

T.C.

For loop ki T.C. = $O(K)$

Heap ki T.C. = $O(\log K)$

↳ T.C. = $O(K * \log K)$

where

K = Row / Numbers of lists

+

Range Find

while loop ki T.C. = $O(N)$

Heap ki T.C. = $O(\log K)$

↳ T.C. = $O(N * \log K)$

where

N = Total Number of Elements of each lists

overAll T.C. ⇒ $O(K * \log K) + O(N * \log K)$

⇒ $O(N * \log K)$

Heap takes $O(K)$ space

↳ S.C. = $O(K)$