24/12/2023
Wednesday

* **Merge K sorted Arrays –**

I/P – arr[][] = {{1, 3, 5},
{2, 4, 6},
{0, 7, 8, 9}}; K = 3

O/P – [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Approach –

= Create a min-heap with first element of all K arrays. Because smallest element yhi K arrays ke first element me se koi ek hoga.

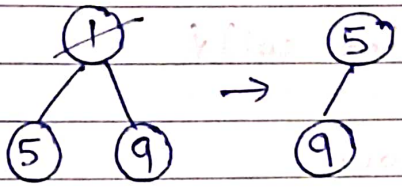
= While the min heap is not empty, fetch the top element of min heap.
Pop it..
Push that element in final ans.
Push the element next to top element in min heap if there it is.

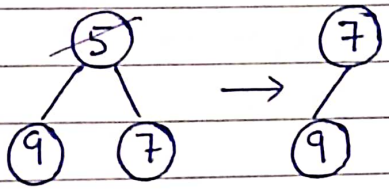
Note – Jis array me se element min heap se pop out hua hai, usi array ke next element ko min heap mei daalna hai.

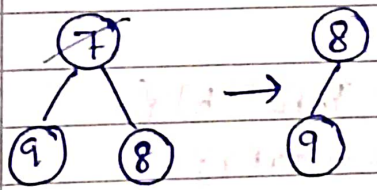
Uske liye hume us element ka row and column index pta hona caahiye. That's cuhy hum khud ka custom data type create krenge.

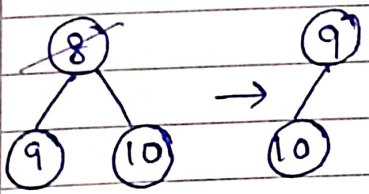
	0	1	2
Let - 0	9	12	17
1	1	7	10
2	5	8	14

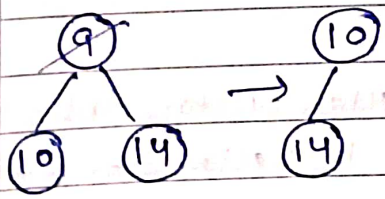
O/P - [1, 5, 7, 8, 9, 10, 12, 14, 17]

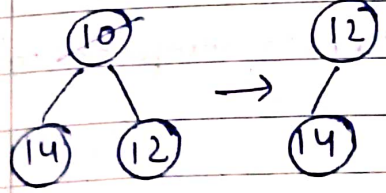
(1) 

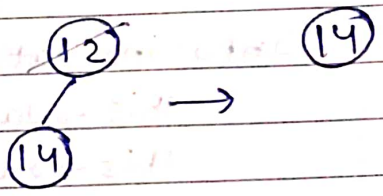
(2) 

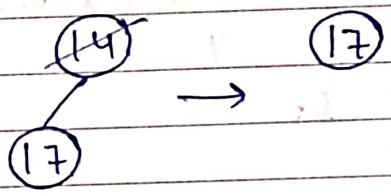
(3) 

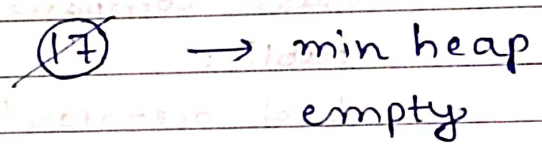
(4) 

(5) 

(6) 

(7) 

(8) 

(9) 

Code -

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

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

class compare{
public:
bool operator()(Info* a, Info* b){
        return a->data > b->data;
    }
};

class Solution{
public:

Info mergeKSortedArrays(vector<vector<int>>& arr,
                        int K, vector<int>&ans)
{
 priority_queue<Info*,vector<Info*>,compare> minHeap;
```

```
// Push first k elements in min heap
for(int row=0; row<k; row++){
    int element = arr[row][0];
    Info* temp = new Info(element, row, 0);
    minHeap.push(temp);
}

// Process rest of the elements
while(!minHeap.empty()){
    Info* temp= minHeap.top();
    minHeap.pop();

    int topData = temp->data;
    int topRow = temp->rowIndex;
    int topCol = temp -> colIndex;

    ans.push_back(topData);

    if(topCol+1 < arr[topRow].size()){
    Info* next= new Info(arr[topRow][topCol+1],
                         topRow, topCol+1);
    minHeap.push(next);
    }
  }
}

vector<int> mergeKArrays(vector<vector<int>>arr, int k)
{
    vector<int> ans;
    mergeKSortedArrays(arr, k, ans);
    return ans;
  }
};
```

* Comparator

for min Heap -

```
class compare{
public:
bool operator()(T a, T b){
        return a > b;
      }
};
```

for max Heap -

```
class compare{
public:
bool operator()(T a, T b){
        return a < b;
      }
};
```

Here, T is the data type.

= We need custom comparator here in priority queue so that we can compare custom data types by writing our own comparator.

* Merge K sorted lists

Given an array of k linked lists, each linked list is sorted in ascending order.
Merge all the linked lists into one sorted linked list and return it.

I/P - lists = [[1,4,5],[1,3,4],[2,6]]

O/P - [1,1,2,3,4,4,5,6]

Here, lists vector contains head of every linked list.

Approach -

= Process first K elements i.e. first element of given K linked lists. Insert them in min heap. Push krte time check kr lena khi linked list empty to nhi.

= create new linked list for output.

= while the min heap is not empty, fetch top element of min heap.
Pop it.
If it is first node of output linked list, then -
head = topNode; tail = topNode;
If it is any node other than first, then -
tail->next = topNode; tail = topNode;
Agr us particular LL mei tail->next non-null hai to use heap mei push krdo.

= Return head of output linked list.

Note - Jis LL mei se element pop hua hai, usi LL ke next element ko insert krna hai. Uske liye hume vo element or uska address pta hona caahiye or LL ki har node ye dono data store krti hai. That's why koi custom data type bnane ki need nhi hai.

Code -

```
class compare{
public:
bool operator()(listNode* a, listNode* b){
        return a->val > b->val;
    }
};

class Solution{
public:

listNode* mergeklists(vector<listNode*>& lists){

priority_queue<listNode*, vector<listNode*>, compare> minHeap;
 // Push first k elements in min heap
for(int i=0; i< lists.size(); i++){
    listNode* listHead = lists[i];
    if(listHead) minHeap.push(listHead);
  }
 // creating new LL
listNode * head = NULL;
listNode * tail = NULL;
 // Process rest of the elements
while(!minHeap.empty()){
    listNode* topNode = minHeap.top();
    minHeap.pop();
```

```
        if (!head) {
            head = topNode;
            tail = topNode;
        }
        else {
            tail->next = topNode;
            tail = topNode;
        }
        if (tail->next) minHeap.push(tail->next);
    }

    return head;
    }
};
```

* **Smallest range covering elements from k lists -**

You have k lists of sorted integers in non-decreasing order. Find the smallest range that includes at least one number from each of the k lists.

I/P - nums = [[4,10,15,24,26],[0,9,12,20],[5,18,22,30]]

O/P - [20,24]

Approach -

- Push first k elements i.e. first element of k lists in min heap. Track minimum and maximum.
- Process rest of the elements. while min heap is not empty, fetch the top.
  Pop it.
  Check krlo kya nyi range chotti hai pehle wali range se. If yes, then update krdo.
  Now, push the rest of elements if they are in bound.
  - Push krte time check kr lena kbi maximum element to update ni hora. If yes, then update it.

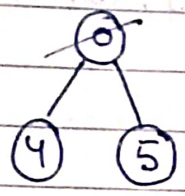
Note - Agr k lists me se koi ek bhi lists khtm ho jaye, then loop se bahr aa jao because range me har list ka atleast one element hona caahiye.

nums = 4, 10, 15, 24, 26 → L1
0, 9, 12, 20 → L2
5, 18, 22, 30 → L3

Possible Different ranges(L,R) = (0,5); (4,9); (5,10);
(9,18); (10,18); (12,18);
(15,20); (18,24); (20,24)

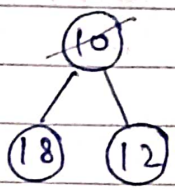
Smallest Range = [20,24]

(1)



L = 0
R = 5

(5)

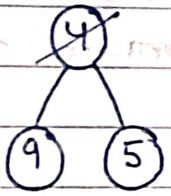


L = 10
R = 18

(9)

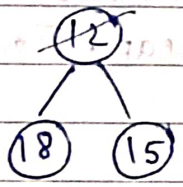
L = 20
R = 24

(2)



L = 4
R = 9

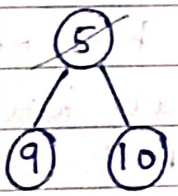
(6)



L = 12
R = 18

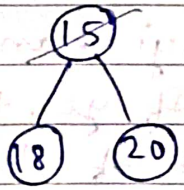
Yha pe list 2 khtm ho gyi break kr jao.

(3)



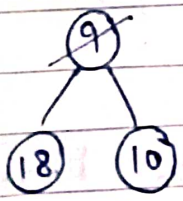
L = 5
R = 10

(7)



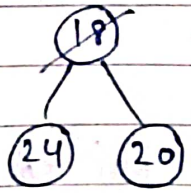
L = 15
R = 20

(4)



L = 9
R = 18

(8)



L = 18
R = 24

Code -

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

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

class compare{
public:
bool operator()(Info* a, Info* b){
    return a->data > b->data;
  }
};

class Solution{
public:

vector<int> smallestRange(vector<vector<int>>& nums)
{
priority_queue<Info*, vector<Info*>, compare> minHeap;
 int mini = INT_MAX;
 int maxi = INT_MIN;
```

```
// process first k elements
for (int i=0; i<nums.size(); i++){
    int element = nums[i][0];
    Info* temp = new Info(element, i, 0);
    minHeap.push(temp);
    mini = min(mini, element);
    maxi = max(maxi, element);
}

// store first range
int ansStart = mini;
int ansEnd = maxi;

while(!minHeap.empty()){
    Info* topNode = minHeap.top();
    minHeap.pop();

    int topData = topNode->data;
    int topRow = topNode->rowIndex;
    int topCol = topNode->colIndex;

    mini = topNode->data;

// check krlo kya new range chotti hai purani
// wali range se
    if((maxi-mini) < (ansEnd-ansStart)){
        // update krdo range ko
        ansStart = mini;
        ansEnd = maxi;
    }
```

```
if (topCol + 1 < nums[topRow].size()){
    int newElement= nums[topRow][topCol+1];
    //maxi ko update krdo
    maxi = max(maxi, newElement);
    Info * newInfo= new Info(newElement, topRow,
                                        topCol+1);
    minHeap.push(newInfo);
  }
else {
    break;
  }

//store final range of ans and return it
  vector<int>ans;
  ans.push_back(ansStart);
  ans.push_back(ansEnd);
  return ans;
  }
};
```