26/05/2023

Q1

Ex→ $k = 3$, $n = 4$

arr1 → {1, 3, 5, 7}
arr2 → {2, 4, 6, 8}
arr3 → {0, 9, 10, 11}

1) Insert k elements in heap (first element from all the arrays).

heap →

~~0~~
1
2

ans → {0}

heap →

~~1~~
2
9

ans → {0, 1}

heap →

~~2~~
3
9

ans → {0, 1, 2}

heap →

~~3~~
4
9

ans → {0, 1, 2, 3}

heap →

~~4~~
5
9

ans → {0, 1, 2, 3, 4}

heap →

~~5~~
6
9

ans → {0, 1, 2, 3, 4, 5}

Similarly we can achieve the final merged array.

Note → Along with the element, the array from which this element belongs and also its index will be stored in the heap.

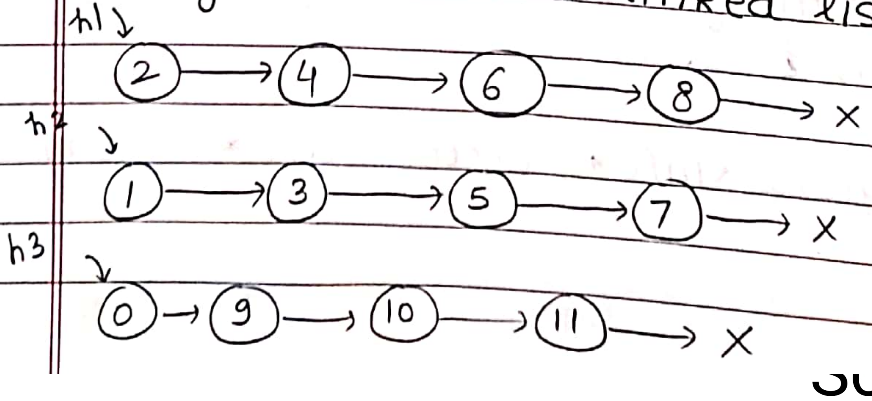
Code

```
class info {
        public :
              int data;
              int row;
              int col;
               info (int d, int r, int c){
                     data = d;
                     row = r;
                     col = c;
              }
};
class compare {
         public :
              bool operator () (info *a, info *b)
                          return a->data > b->data;
};

vector <int> mergekSortedArrays (int arr [][4],
int k, int n) {
  // Creation of min heap
  priority_queue <info*, vector <info*>, compare>
  minHeap;
 // Insert first element of k arrays
for (int i=0; i<k; i++) {
        info * temp = new info (arr[i][0], i, 0);
        minHeap.push (temp);
}
vector <int> ans;
While (! minHeap.empty ()) {
      // Find top element
      info * temp = minHeap.top ();
      int topElement = temp → data;
      int topRow = temp → row;
      int topCol = temp → col;
      minHeap.pop ();
    // Remove top element & insert in ans vector
    ans.push_back (topElement);
    // Is next index a valid index ?
    if (topCol +1 < n) {
            info * newInfo = new info (arr[
            topRow][topCol+1], topRow, topCol+1);
            minHeap.push (newInfo);
     }
}
return ans;
}
```

Time complexity = O(n k log k)

Q2 Merge k sorted linked list



Initially ans linked has both head and tail as NULL.

heap →

~~0~~
1
2

0

heap →

~~1~~
2
9

0, 1

heap →

~~2~~
3
9

0, 1, 2
⋮

Hence this also has similar approach as that of merge k sorted arrays.

Code

```
ListNode * mergekLists (vector<ListNode *>
lists) {
   // Min heap creation
   priority_queue <ListNode * , vector <
   ListNode * >, compare >minHeap;
  //Find no. of lists
   int k = lists. size ();
```

```
    // No linked list present
    if (k = = 0)
         return NULL;
   // Insert first node's pointer in minHeap
   for (int i=0; i< k; i++) {
          if (lists [i] != NULL)
                 minHeap. push (lists [i]);
  }
  // Create head & tail for ans list.
  ListNode* head = NULL;
  ListNode* tail = NULL;
  while (! minHeap. empty()) {
     // Fetch top element of heap
     ListNode * temp = minHeap. top();
     minHeap. pop();
    // Insert top elements in ans vector
    // Inserting first element ?
     if (head = = NULL) {
            head = temp;
            tail = temp;
           // Insert further elements
            if (tail → next != NULL)
                 minHeap. push (tail → next);
    }
    else {
        // Not inserting first element
        tail → next = temp;
        tail = temp;
        // Insert further elements
        if (tail → next != NULL)
              minHeap. push (tail → next);
  }
```

```
    }
    return head;
}
```

↱ Can make mistake (Amazon round 3)

Q3 Smallest range in k list.

i/p →

1	2	3	4	5	6	7	8	9

Suppose that we pick first element from every list. Then find maximum and minimum. Then we can say that atleast one element from every list will be there in this range. But we can't tell that this is smallest range or not.

1
4
7 } mini = 1
maxi = 7

[1,7] ⇒ Here in this range atleast one element of every list will be present.

[3,7] is also a range which is much smaller than [1,7].

Dry run

{1,2,3}
{1,2,3}
{1,2,3}

1
1
1 } maxi = 1
mini = 1 | Range = 0

2
1
1 } maxi = 2
mini = 1 | Range = 2 - 1 = 1

2 maxi = 2 Range = 1
2 mini = 1
1

2 maxi = 2 Range = 0
2 mini = 2
2

3 maxi = 3 Range = 1
2 mini = 2
2

3 maxi = 3 Range = 1
3 mini = 2
2

3 maxi = 3 Range = 0
3 mini = 3
3

Now we are basically moving ahead as range can be decreased either by increasing mini or decreasing maxi. By moving forward we are increasing the mini. Hence we will have to use the min heap and this question can't be done with the help of max heap.

Note → We can only move forward & not backward as it is a singly linked list.

We have stop when any of the linked list is finished.

Code

```
vector <int> smallestRange (vector <vector <int>> &
nums) {

      int mini = INT_MAX;
      int maxi = INT_MIN;
      //min heap creation  → same as info class
      priority_queue <node*, vector <node*>,
      compare>minHeap;
     // insert first elements of all k lists
      int k = nums.size();
     for (int i=0 ; i<k ; i++) {
            int element = nums[i][0];
            maxi = max (maxi, element);
            mini = min (mini, element);
            minHeap.push (new node (element, i, 0));
   }
   int ansStart = mini;
   int ansEnd = maxi;
   while (!minHeap.empty()) {
          //fetch top element
          node * top = minHeap.top();
          int topElement = top->data;
          int topRow = top->row;
          int topCol = top->col;
        // pop top element
        minHeap.pop();
       //top element would be minimum in min heap
       mini = topElement;
      // Check for smaller range
      if (maxi-mini < ansEnd-ansStart){
                ansStart = mini;
            ansEnd = maxi ;
        }
       //Element exist or not
       if (topCol + 1  < nums [topRow].size()){
            //exists
            // create a new node & insert
            // update maxi
```

Can forget

```
     * maxi = max(maxi, nums[topRow][topCol+1]);
         node * newNode = new node (nums [topRow
          [topCol +1], topRow , topCol +1) ;
         minHeap.push (newNode);
      }
      else { //notexist
              break;
      }
      }
      vector <int> ans;
      ans.push_back (ansStart);
      ans.push_back (ansEnd);
      return ans;
     }
```