Merge K sorted arrays

code creating input

```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);

    ArrayList<ArrayList<Integer>> arr = new ArrayList<>();
    int k = scn.nextInt();
    for (int i = 0; i < k; i++) {
        int n = scn.nextInt();
        ArrayList<Integer> innerList = new ArrayList<>();
        for (int j = 0; j < n; j++) {
            int val = scn.nextInt();
            innerList.add(val);
        }
        arr.add( innerList );
    }

    // print
    for (int i = 0; i < arr.size(); i++) {
        for (int j = 0; j < arr.get(i).size() ; j++) {
            System.out.print( arr.get(i).get(j) + " " );
        }
        System.out.println();
    }
}
```

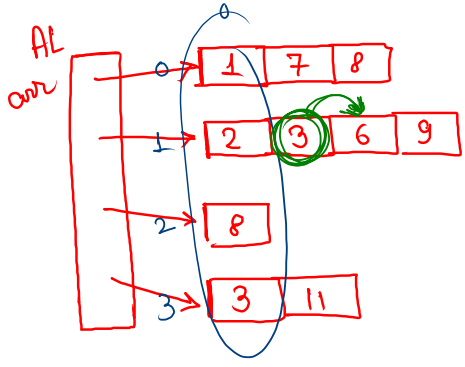
K = 4

n = 3 [1 7 8]

n = 4 [2 3 6 9]

n = 1 [8]

n = 2 [3 11]



psudo code

1) create PQ

2) put first element of each row
(store val, i, j)

3) keep iteration until PQ become empty

3.1) remove top element → print

3.2) add next of that element in PQ
(only if row is not exhausted)

code

```
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);

    ArrayList<ArrayList<Integer>> arr = new ArrayList<>();
    int k = scn.nextInt();
    for (int i = 0; i < k; i++) {
        int n = scn.nextInt();
        ArrayList<Integer> innerList = new ArrayList<>();
        for (int j = 0; j < n; j++) {
            int val = scn.nextInt();
            innerList.add(val);
        }
        arr.add( innerList );
    }

    // create PQ which will store val , i, j
    // assuming
    // 0th index : val
    // 1st index : i
    // 2nd index : j
    PriorityQueue<int[]> pq = new PriorityQueue<>((a, b) -> {
        return a[0] - b[0];
    });

    // put first elements of each row in PQ
    int m = arr.size();
    for (int i = 0; i < m; i++) {
        int[] temp = new int[3];
        temp[0] = arr.get(i).get(0);
        temp[1] = i;
        temp[2] = 0;

        pq.add( temp );

        // pq.add( new int[] { arr.get(i).get(0), i, 0 } );
    }

    // run until pq is not empty
    while ( !pq.isEmpty() ) {
        int[] rem = pq.poll();
        int val = rem[0];
        int i = rem[1];
        int j = rem[2];

        System.out.print(val + " ");

        // check if row is exhausted or not
        if ( j + 1 < arr.get(i).size() ) {
            pq.add( new int[] { arr.get(i).get(j + 1), i, j + 1 } );
        }
    }
}
```

$T.C = K \log(K)$

⇒ StringBuilder (dynamic string)

Syntax

```
StringBuilder sb = new StringBuilder();
```

- `sb.append(x);` — x = char or String
- `sb.deleteCharAt(idx)` — idx = index
- `sb.charAt(idx)` — return char at idx
- `sb.reverse()`

Longest Substring Without Repeating Characters 6

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

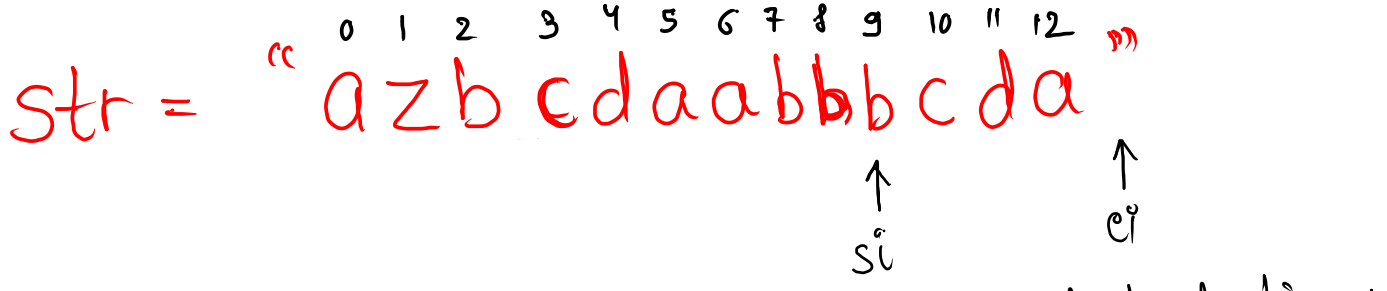
str = "azbcdaabbbcda" ; ans = 5

Note:- whenever we have a question of substring & we have to find something longestest or shortest then

brute force T.C $O(n^3 * n)$

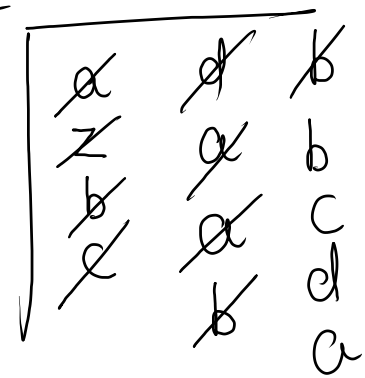
S.C = $O(n)$

sliding window



to check duplicacy

set



$ans = (ei - si + 1)$

---

ei will always expand my window

si " " collapse " "

Code

```java
public static void main(String[] args) {
    Scanner scn = new Scanner(System.in);
    String str = scn.nextLine();
    int ans = longestSubstringWithoutDuplicate(str);
    System.out.println(ans);
}
public static int longestSubstringWithoutDuplicate(String str) {
    HashSet<Character> set = new HashSet<>();
    int si = 0;
    int ei = 0;
    int ans = 0;
    while ( ei < str.length() ) {
        if ( set.contains( str.charAt(ei) ) == true ) {
            set.remove( str.charAt(si) );
            si++;
        } else {
            set.add( str.charAt(ei) );
            ei++;
        }
        ans = Math.max( ans, ei - si );
    }
    return ans;
}
```

T.C = O(n)

Longest Consecutive Sequence

arr = [4 , 102 , 3 , 101 , 1 , 2 , 100 , 6 , 5]
indices: 0 1 2 3 4 5 6 7 8

observation :- 1 2 3 4 5 6 100 101 102

map

int	vs	boolean
4	→	~~true~~ false
102	→	~~true~~ false
3	→	~~true~~ false
101	→	~~true~~ false
1	→	true
2	→	~~true~~ false
100	→	true
6	→	~~true~~ false
5	→	~~true~~ false

```java
public static int longestSequence(int[] arr, int n) {
    HashMap<Integer, Boolean> map = new HashMap<>();
    for (int i : arr) {
        map.put( i, true );
    }

    // check make every non starting point as false
    for (int val : arr) {
        if ( map.containsKey(val - 1) ) {
            map.put( val, false );
        }
    }

    // now check sequence for only those which are true
    int ans = 0;
    for (int i : arr) {
        if ( map.get(i) == true ) {

            int len = 1;
            int start = i;
            while ( map.containsKey( start + len ) ) {
                len++;
            }

            ans = Math.max(ans, len);
        }
    }
    return ans;
}
```

T.C = O(n)

S.C = O(n)