Agenda:

1. **Meeting Rooms**
2. **Sort the nearly sorted array**
3. **Merge K sorted arrays**
4. **Minimum Distance Equal Pair**
5. **Minimum Window Substring**

Meeting Rooms

You are given an array of meeting time intervals where each interval is represented as [start, end] (start time is less than the end time). Your task is to find the minimum number of conference rooms required to schedule all meetings without overlap.

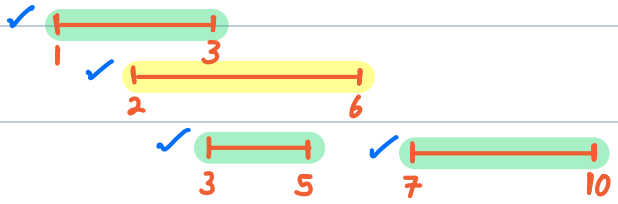
Example 1:

Input: [[0, 30], [5, 10], [15, 20]]
Output: 2

Explanation:

- The first meeting (0, 30) starts at 0 and ends at 30.
- The second meeting (5, 10) overlaps with the first meeting, so a second room is required.
- The third meeting (15, 20) can reuse a room after the second meeting ends, resulting in a total of 2 rooms.

Self try → 10 min



Ans = 2

$0 <= start < end <= 10^6$

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

+1

0 1 2 2 2 1 0 1 1 1 0 0 → Ans = 2

```
for i → 0 to (N-1) {
    s = B[i][0]    e = B[i][1]
    rooms[s]++
    rooms[e]--
}

ans = 0
for i → 1 to 10^6 {
    rooms[i] += rooms[i-1]
    ans = max(ans, rooms)
}

return ans
```

TC = O(N + Range) (Range → 10^6) SC = O(Range)

Q → Sort the nearly sorted array i.e every element is at max K distance away from its position in sorted order.

|i − sorted position|

	0	1	2	3	4	5	6	7	8	
sorted →	11	13	20	22	31	45	48	50	60	
A =	13	22	31	45	11	20	48	60	50	K = 4

Sol 1 → Use sorting algo. TC = O(N log(N))

Self try → 10 min 7:55 AM

Sol 2 →

	0	1	2	3	4	5	6	7	8	
A =	13	22	31	45	11	20	48	60	50	K = 4

Smallest element will be between index 0 − K.

min Heap (size K+1)

0	1	2	3	4	5	6	7	8
11	13	20	22	31	45	48	50	60

~~13~~ ~~22~~ ~~31~~

~~45~~ ~~11~~ ~~20~~

~~48~~ ~~60~~ ~~50~~

```
for i → 0 to K {
    insert (A[i])
}

j = 0

for i → (K+1) to (N-1) {
```

```
        A[j] = extractMin()
        j++
        insert(A[i])
    }

while (j < N) {
        A[j] = extractMin()
        j++
}
```

TC = O(N log(K))

SC = O(K)

Q → Merge K-sorted arrays.

A = [2 3 11 15 20]

B = [1 5 7 9]

C = [0 2 4]

D = [3 4 5 6]

Sol 1 → Merge 2 sorted arrays (K-1) times.

Let say len of each array ~ N.

TC → 2N + 3N + 4N + ... + KN

$= N\left(\frac{K(K+1)}{2} - 1\right) \approx O(K^2 N)$

minHeap (size K)

(A[i], A, i)

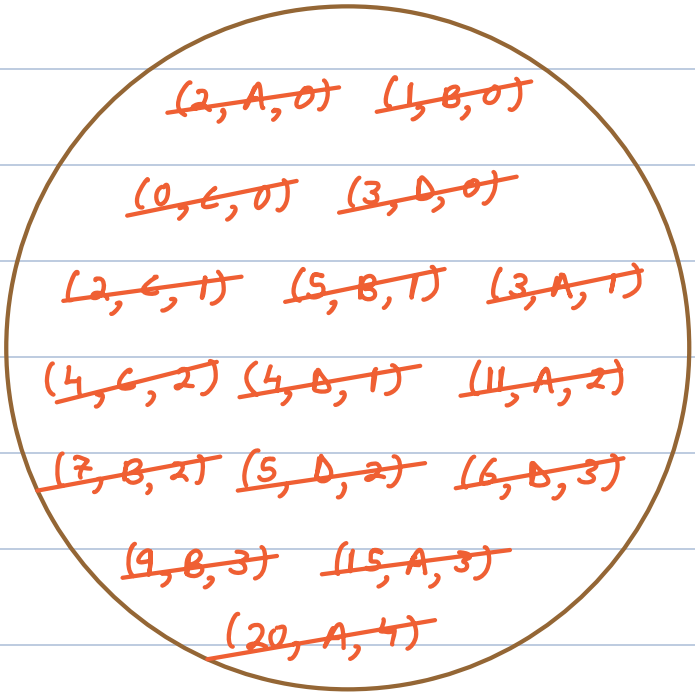
Ans → [0 1 2 2 3 3 4 4 5 5 6 7 9 11 15 20]

SC = O(K)

Let say len of each array ~ N.

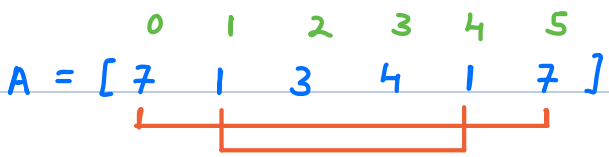
⇒ Total # elements = KN

TC = O(KN log(K))



Minimum Distance Equal Pair

Given an array A, find a pair of indices (i, j) such that A[i] = A[j] and the absolute difference |i - j| is minimised. In simpler terms, you need to find two equal elements in the array that are the closest to each other and return the minimum distance between them.



Ans = 4 - 1 = 3

Self try → 10 min 8:55 AM

Bruteforce → ∀ i, j if A[i] == A[j] find |i-j| & take min.

TC = O(N²) SC = O(1)

0	1	2	3	4	5
7	1	3	4	1	7

A = [7 1 3 4 1 7]

✓ ✓ ✓ ✓ ✓ ✓

HashMap

A[i] → latest index.

ans = ~~∞~~ 4 - 1 = 3

7 → ~~0~~ 5
1 → ~~1~~ 4
3 → 2
4 → 3

ans = ∞

```
for i → 0 to (N-1) {
    if (hm.containsKey(A[i])) {
        ans = min(ans, i - hm.get(A[i]))
    }
    hm.put(A[i], i)
}
if (ans == ∞) return -1
return ans
```

TC = O(N)

SC = O(N)

Minimum Window Substring

Given two strings s and t, find the minimum window in s which contains all characters of t (including duplicates). If no such window exists, return an empty string "".

s = "a d b e c d e b a n c"

t = "a b c b"

Ans = 7

"b e c d e b a"

Bruteforce → ∀ substring of s check if all characters of t are present.

|s| = N $\frac{N(N+1)}{2}$

1) store freq ∀ char in 't' (array / map).

2) store freq ∀ char in substring.

3) ∀ char freq(substring) >= freq(t)

⇒ valid substring to be potential ans.

Sol → Dynamic Sliding Window ⇒ to calculate freq of chars in s.

s = "a d b e c d e b a n c" // N

t = "a b c b" // M

1) freq_t = [1 2 1 0 0 0]
0 1 2 3 4 5 ...

```
for i → 0 to (M-1) {
    freq_t[ t[i] - 'a' ] ++
}
```

2) freq_s using sliding window.

```
ans = ""

l = 0    r = -1          // l — r

while (r < N) {
    if ( check(freq_s, freq_t)) {   // TC = O(26)
        // ∀char freq(s) >= freq(t)
        ans = s.substring(l, r)
        freq_s[s[l] - 'a']--      l++
    } else {
        r++          freq[s[r] - 'a'] ++
    }
} return ans
```

TC = O(N+M) SC = O(2*26) = O(1)

---

t = "abcb" // M

freq-t = [1 2 1 0 0 ...]
0 1 2 3 4

s = " a d b e c d e b a n c " // N
x x x ✓ ✓ ✓ ✓ ✓ ✓
↓ ↓
true

freq-s = [1 1 1 1 2 ...
0 1 2 3 4

for i → 0 to 25 {
    if (freq-s[i] < freq-t[i]) return false
} return true