18. Median of Two sorted arrays

Hard Accuracy: 46.21% Submissions: 11054 Points: 8

→ In $O(\log(\min(m,n)))$ & $O(1)$ S.

Given two sorted arrays of sizes **N** and **M** respectively. The task is to find the median of the two arrays when they get merged.

Example 1:

```
Input:
N = 5, M = 6
arr[] = {1,2,3,4,5}
brr[] = {3,4,5,6,7,8}
Output: 4
Explanation: After merging two arrays,
elements will be as 1 2 3 3 4 4 5 5 6 7 8
So, median is 4.
```

Algorithm: →

① We will calculate median of both the arrays and would discard one half of each array.

② Recursive fn. findMedianUtil → with 3 broad corner cases.

```
int idxA = ( N - 1 ) / 2;
int idxB = ( M - 1 ) / 2;

 /* if A[idxA] <= B[idxB], then median must exist in
    A[idxA....] and B[....idxB] */
if (A[idxA] <= B[idxB] )
  return findMedianUtil(A + idxA, N/2 + 1, B, M - idxA );

/* if A[idxA] > B[idxB], then median must exist in
   A[...idxA] and B[idxB....] */
return findMedianUtil(A, N/2 + 1, B + idxA, M - idxA );
}
```

→ coz also taking idxA.

1, 2, 3, 4, 5 6, 7, 8, 9, 10

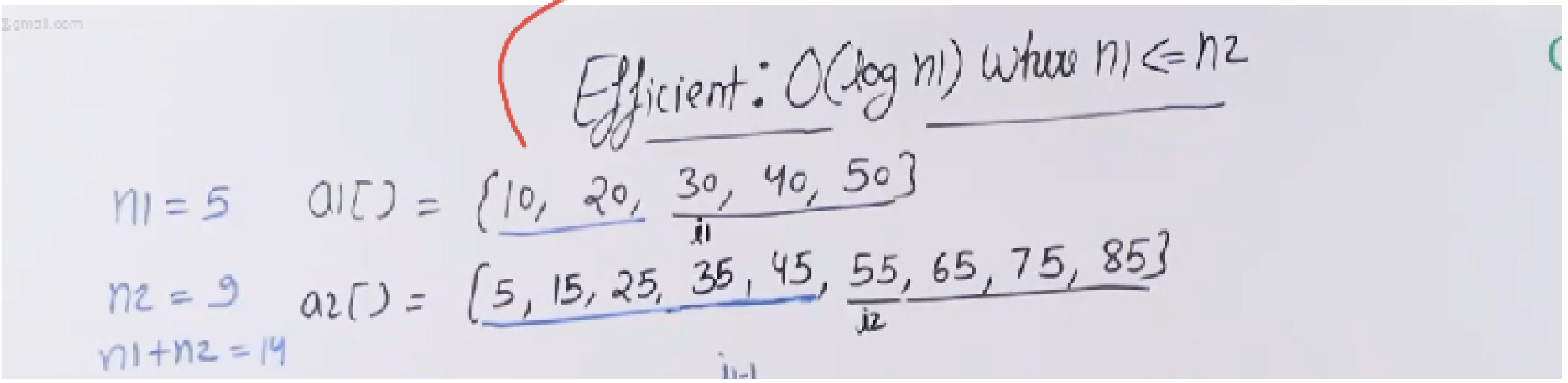
x

merged

median.

Approach 2 :

① The idea is to find those points which collectively divide the array elements $(m+n)$ in halves, and then these point could be used for median calculation



In the blue part of set we want all the elements smaller than current median.

Left and right halves should contain exactly same no of elements, in case of odd no's extra goes to left

Efficient: $O(\log n_1)$ where $n_1 \le n_2$

$n_1 = 5$ $\quad a_1[] = \{10, 20, \underset{i_1}{30}, 40, 50\}$

$n_2 = 9$ $\quad a_2[] = \{5, 15, 25, 35, 45, \underset{i_2}{55}, 65, 75, 85\}$

$n_1 + n_2 = 14$

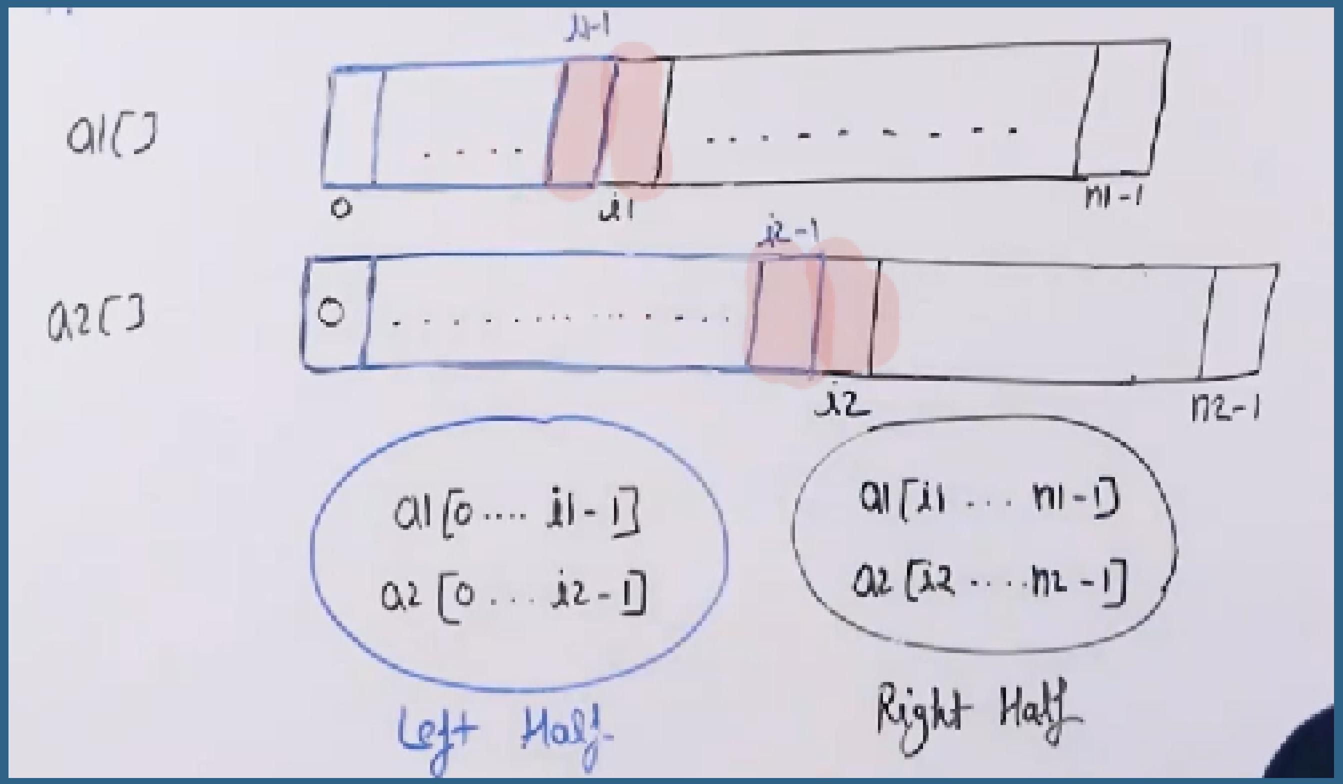
① We know i_1 in this, and we can calculate i_2 such that the whole arr $(n + m)$ is divided into 2 parts using this formula

$$i_2 = \left[\frac{n_1 + n_2 + 1}{2}\right] - i_1$$

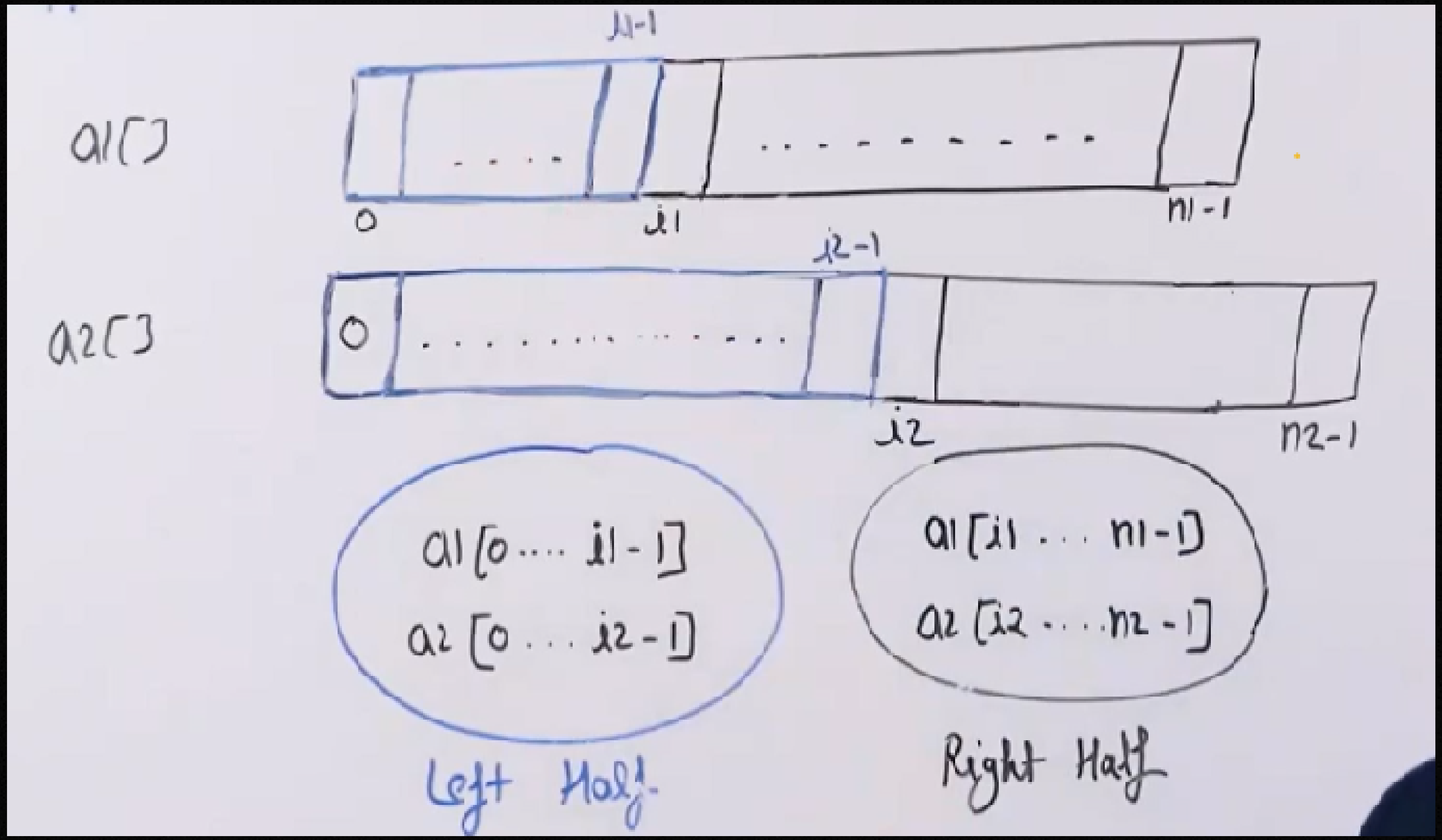
① We've to get to the condition when all the elements in blue set are smaller than the ones in right set.

② At this point we could calculate our median.

③ ↳ And when such sets are obtained, we compute median using avg of the largest element of blue set and smallest element of black set.



→ Based on these i1 values we make decision of our binary search proceeding.



Coding the solution :-

We maintain 4 variables

$$a\begin{cases} \text{min1} & i_1 \\ \text{max1} & i_1-1 \end{cases}$$

$$b\begin{cases} \text{min2} & i_2 \\ \text{max2} & i_2-1 \end{cases}$$

(*) Based on these 4 elements we drive our binary search max → blue set   min → Black set

```
double getMed(int a1[], int a2[], int n1,
                                  int n2)
{  int begin1 = 0, end1 = n1
   while(begin1 <= end1)
   {  int i1 = (begin1 + end1)/2;
      int i2 = (n1 + n2 + 1)/2 - i1;
      int min1 = (i1==n1)? INT_MAX : a1[i1];
      int max1 = (i1==0)? INT_MIN: a1[i1-1];
      int min2 = (i2==n2)? INT_MAX : a2[i2]
      int max2 = (i2 == 0)? INT_MIN : a2[i2-1];
      if(max1 <= min2 && max2 <= min1)
      {  if((n1+n2)%2 == 0)
            return ((double) max(max1, max2) +
                    min(min1, min2))/2
         else
            return (double) max(max1, max2);
      }
      else if(max1 > min2)   end1 = i1-1;
      else    begin1 = i1+1;
   }
}
```

computing i_1 & i_2

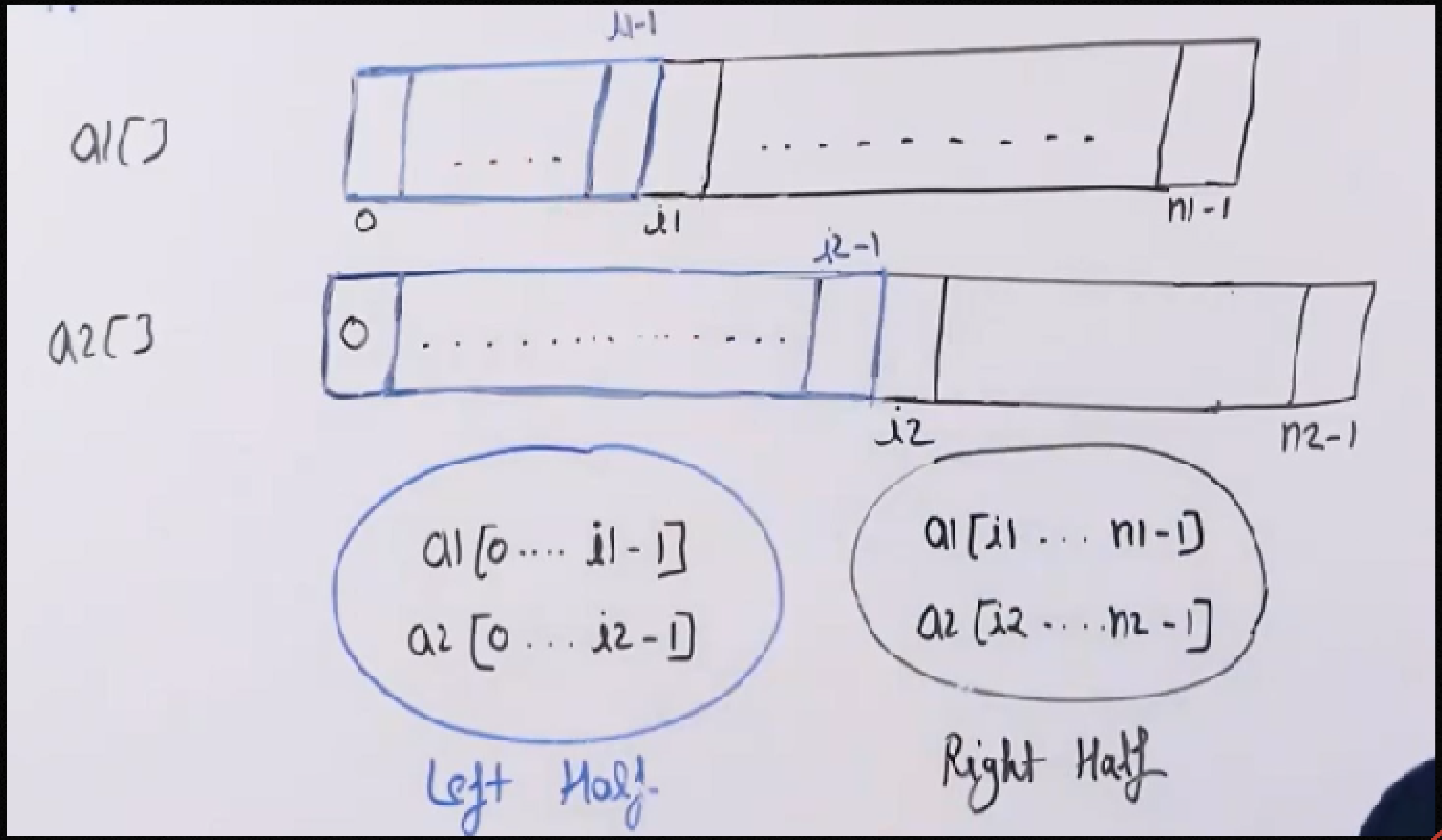
Nothing on the right side

Nothing on left side

otherwise $i_1, i_1-1, \quad i_2, i_2-1$

Since the array is sorted:

① max1 <= min2   max2 <= min1



$max_1 \rightarrow i_1 - 1 \qquad min_1 \rightarrow i_1$

$max_2 \rightarrow i_2 - 1 \qquad min_2 \rightarrow i_2$

Check condition →

$\begin{cases} max_1 <= min_2 \\ max_2 <= min_1 \end{cases}$

violated

↳ end → $i_1 - 1$

all the elements till max1 not smaller than min2. Therefore we eliminate them

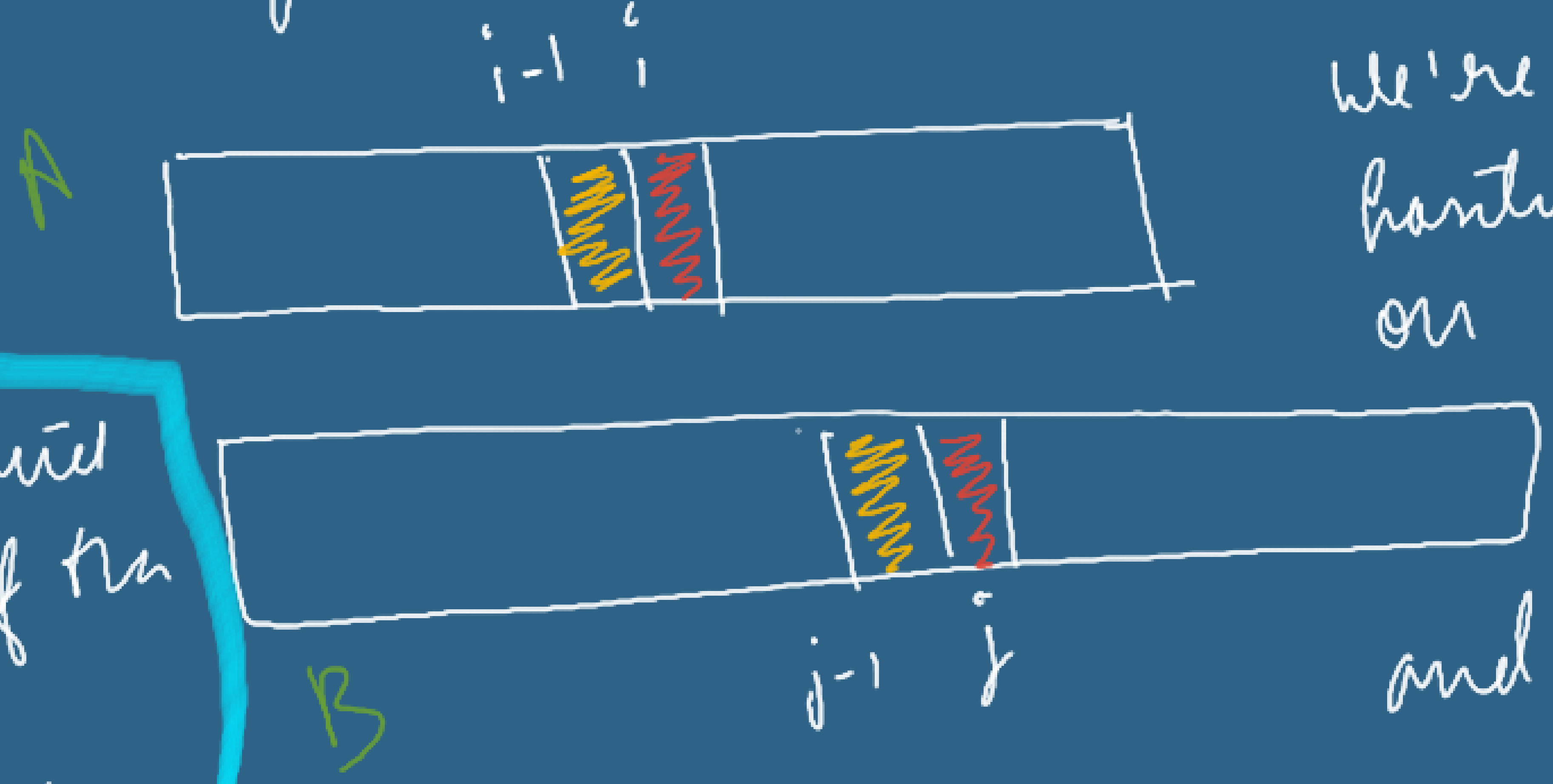
violated ; min1 has shift.

④ Beginning → $i_1 + 1$

① Given 2 arrays, we need to find their median, when merged, given both of these arrays are sorted.

② Approach → To find 2 such points that, could divide combined array elements (m+n) into 2 partition.

③



We're gonna be calibrating position of i & j based on comparison with the bigger array

and do keep in mind with every iteration indices would be recalculated for both the array.

formula used to divide the arrays in half the elements as

$i_2 = \frac{(n_1 + n_2 + 1)}{2} - i_1$

A → smaller array →

① begin = 0   end = n

② while begin < end

↳ i_1 = begin + end / 2 $i_2 = \frac{n_1 + n_2 + 1}{2} - i_1$

③ A → min1 ⟶ INT_MAX ($i_1 == n_1$) else a[i_1]

max1 ⟶ INT_MIN ($i_1 == 0$) else a[$i_1 - 1$]

④ B → min2 ⟶ INT_MAX ($i_2 == n_2$) else B[i_2]

max2 ⟶ INT_MAX ($i_2 == 0$) INT_MIN else B[$i_2 - 1$]

(5) if (max1 <= min2 && max2 <= min1) → perfect partition

{ if ((n1 + n2) % 2 == 0) → even

return (double(max(max1, max2) + min(min1, min2)) / 2)

else
return (double) max(max1, max2)

}

6) Else recalibrate the position of indices

max1 > min2 ← end = i1 - 1

max2 > min1 → min1 begin = i1 + 1