(Solution Approach)

3 4 6 5

9 1 2 5 8 3

To understand the soln of this approach we want to revise picking the maximum subsequence in a single array.

9	1	2	5	8	3

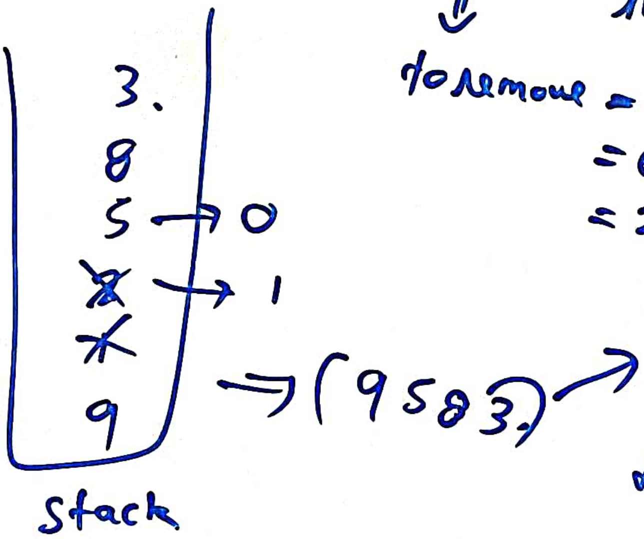
→ Lets say you have only 1 array.

(K = 4) select 4 elements.

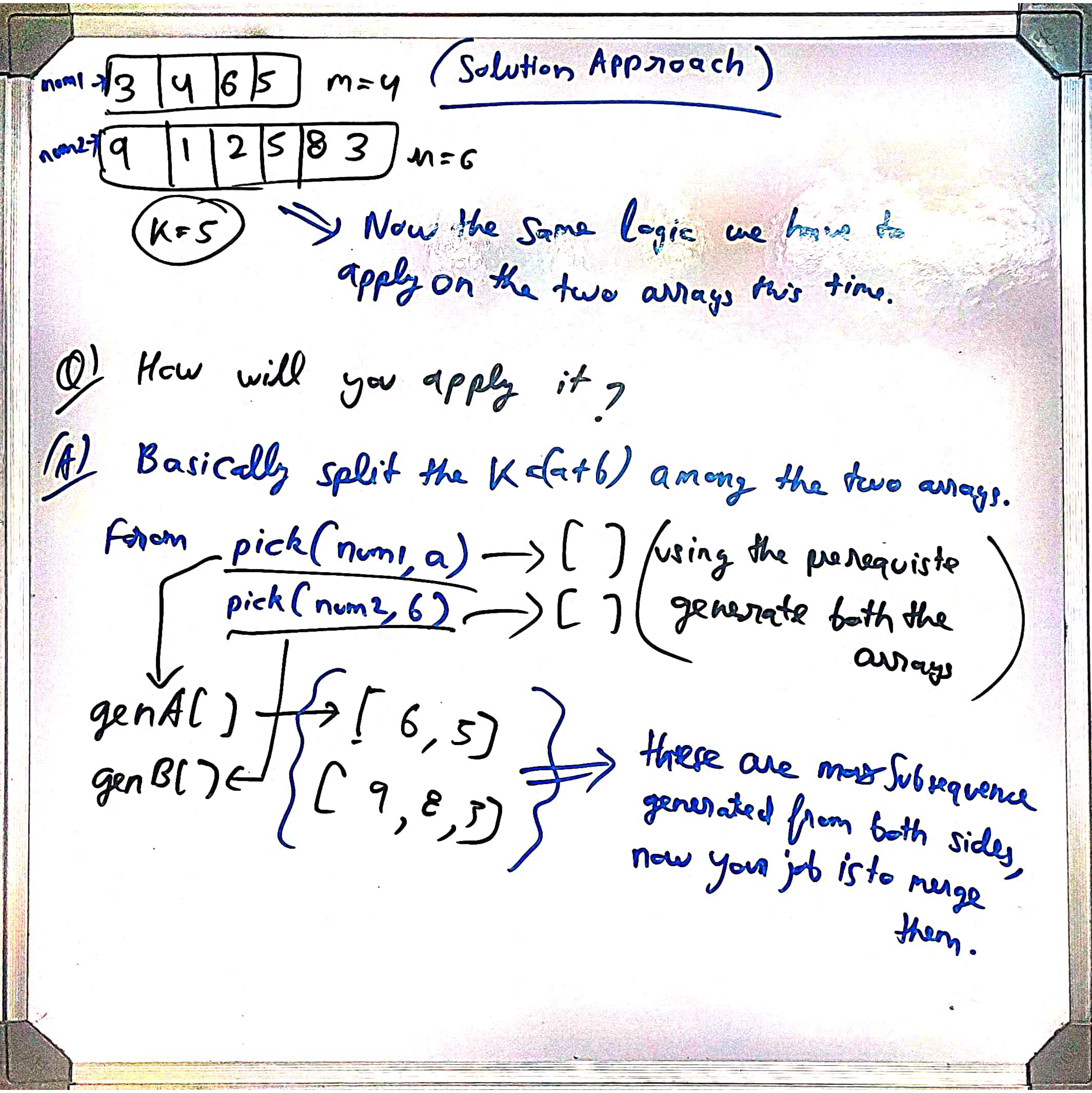
n = 6

⇓

to remove = n − K
= 6 − 4
= 2.



Just did a dry run to show how we did it for a single array



num1 → | 3 | 4 | 6 | 5 | m=4

(Solution Approach)

num2 → | 9 | 1 | 2 | 5 | 8 | 3 | n=6

K=5 ⇒ Now the same logic we have to apply on the two arrays this time.

Q) How will you apply it?

A) Basically split the K=(a+b) among the two arrays.

From pick(num1, a) → [ ]

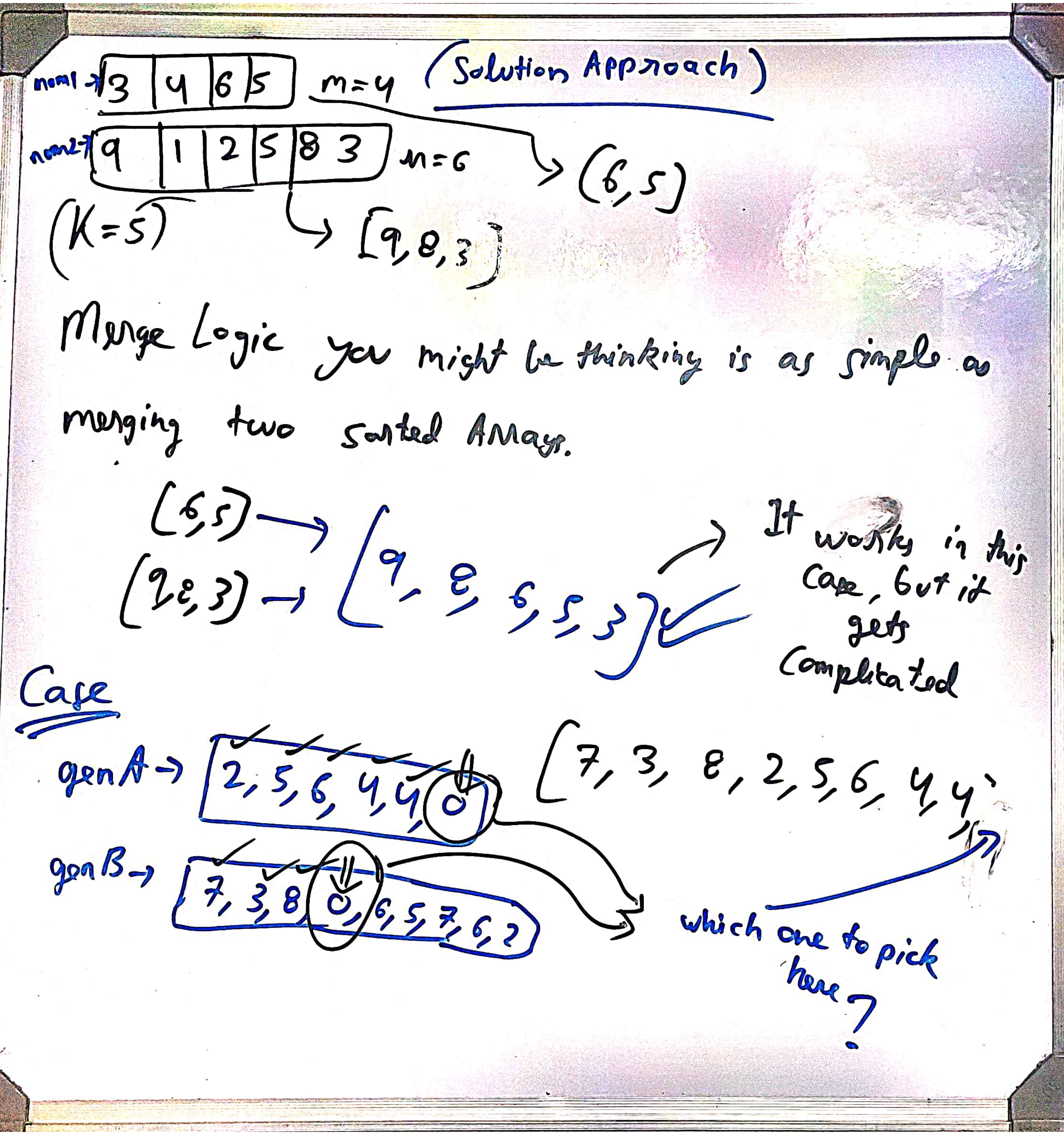
pick(num2, b) → [ ]

(using the prerequiste generate both the arrays)

genA[ ] → [6, 5]

genB[ ] ← [9, 8, 3]

⇒ these are max Subsequence generated from both sides, now your job is to merge them.



num1 → [3 | 4 | 6 | 5] m=4 (Solution Approach)

num2 → [9 | 1 | 2 | 5 | 8 3] n=6

(K=5)

→ (6,5]

→ [9,8,3]

Merge Logic you might be thinking is as simple as merging two sorted Arrays.

[6,5] → [9, 8, 6, 5, 3]

[9,8,3] →

It works in this case, but it gets Complitated

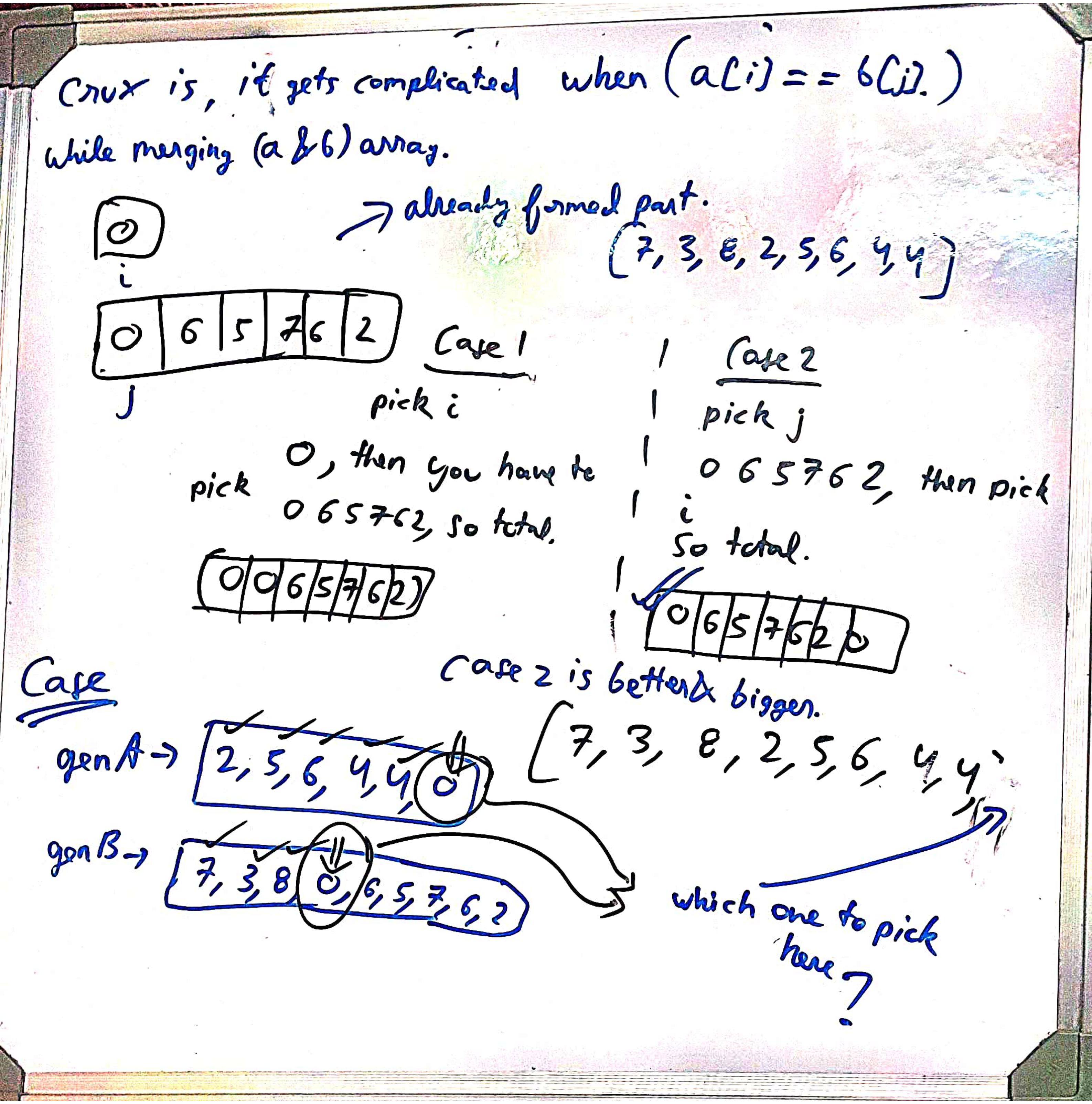
Case

genA → [2, 5, 6, 4, 4, 0]

genB → [7, 3, 8, 0, 6, 5, 7, 6, 2]

[7, 3, 8, 2, 5, 6, 4, 4

which one to pick here?



Crux is, it gets complicated when (a[i] == b[j].)

while merging (a & b) array.

0

i

→ already formed part.

[7, 3, 8, 2, 5, 6, 4, 4]

0	6	5	7	6	2

j

Case 1

pick i

pick 0, then you have to 0 6 5 7 6 2, so total.

0	0	6	5	7	6	2

Case 2

pick j

0 6 5 7 6 2, then pick i

so total.

0	6	5	7	6	2	0

case 2 is better & bigger.

Case

genA → [2, 5, 6, 4, 4, 0]

genB → [7, 3, 8, 0, 6, 5, 7, 6, 2]

[7, 3, 8, 2, 5, 6, 4, 4]

which one to pick here?

0 0 6 5 7 6 2
0 6 5 7 6 2 0

So we have to write a logic inside merge, when we are merging.

①

(merge Function)

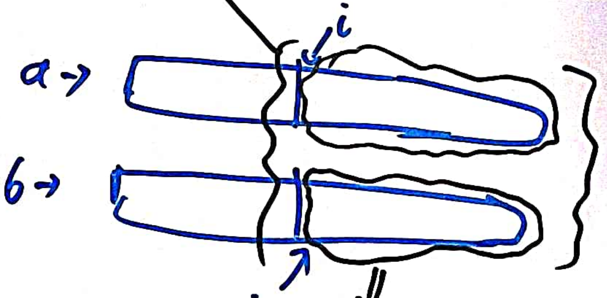
0	6	5	7	6	2

```
int a[], int b[]

while( i < a.len && j < b.len){
        if ( a_i > b_j )  take a_i  i++;
        if ( b_j > a_i )  take b_j, j++;
        if ( a_i == b_j ) → {


                    }

}
```

→ write a greater function for (a, i, b, j)

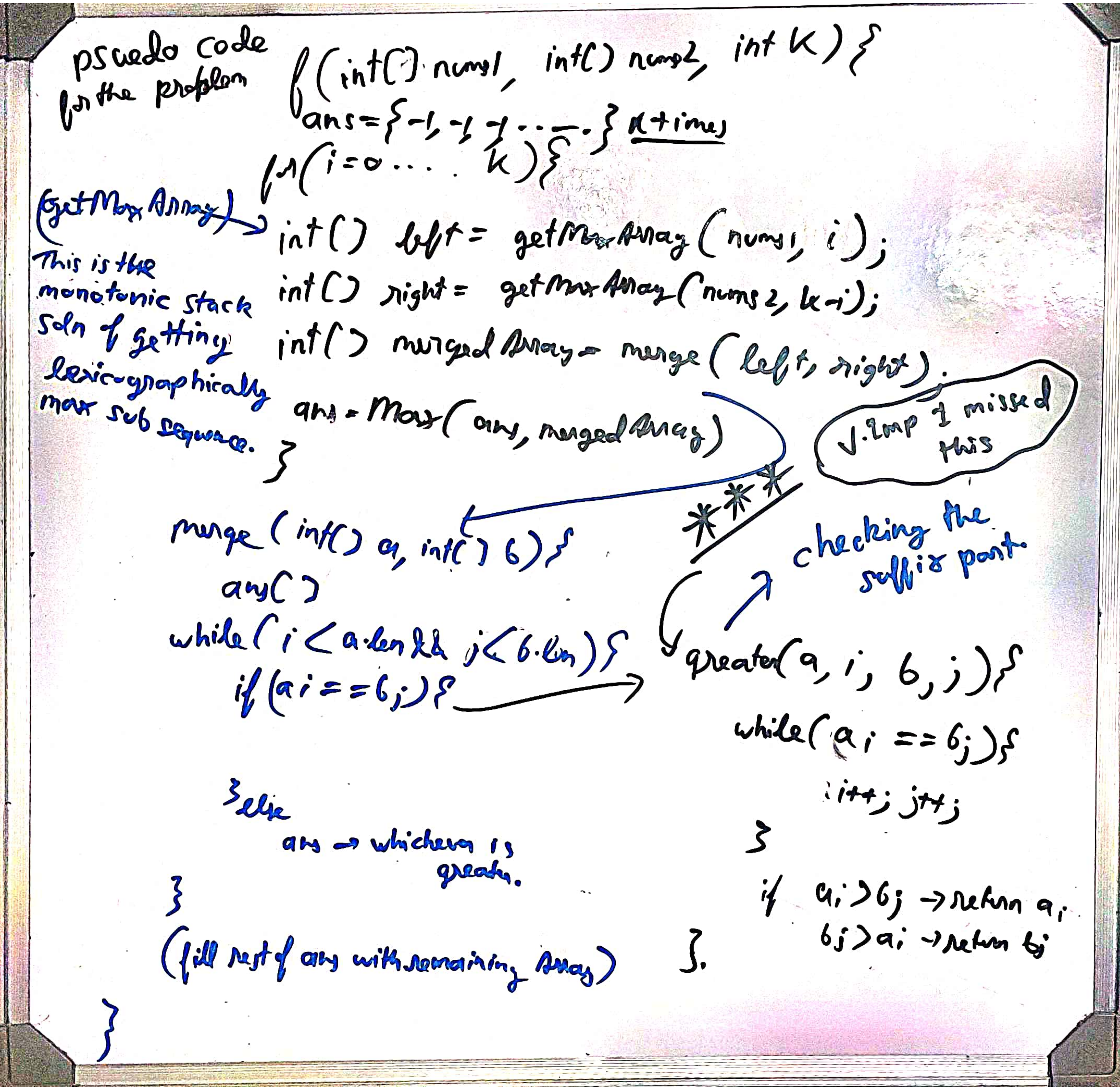
⇓

and return which suffix is greater.



⇓

Compare which suffix is greater, cause ultimately there the difference will happen.



pseudo code
for the problem

```
f(int[] nums1, int[] nums2, int K){
  ans={-1,-1,-1.......} k+times
  for(i=0.... k){
(getMaxArray) → int[] left= getMaxArray(nums1, i);
    int[] right= getMaxArray(nums2, k-i);
    int[] mergedArray= merge(left, right);
    ans= Max(ans, mergedArray)
  }

  merge(int[] a, int[] b){
    ans[]
    while(i<a.len && j<b.len){
      if(ai==bj){

      }else
        ans → whichever is
                greater.
    }
    (fill rest of ans with remaining Array)
}
```

This is the monotonic stack soln of getting lexicographically max sub sequence.

V.imp I missed this

***

checking the suffix part.

```
greater(a, i, b, j){
  while(ai == bj){
    i++; j++;
  }
  if ai>bj → return ai
     bj>ai → return bj
}.
```