We were supposed to choose two questions which do not have correct ratio as green. So we chose 2SUM & Counting Inversions.

1. 2SUM

According to the question given, there are ‘k’ arrays stored in ‘A’ and for each array we have to output the two indices such that $A[p]=-A[q]$.

So in our code, ‘get_indices’ function prints the indices. We store the array elements along with their index location in a vector of pairs called ‘new_arr’. First element of the pair is array element and second element is the index location.
Consider the input as 5 4 -5 6 8 & vector of pairs is as follows

new_arr.first	5	4	-5	6	8
new_arr.second	0	1	2	3	4

After sorting the vector of pairs in ascending order on the first element we get,

new_arr.first	-5	4	5	6	8
new_arr.second	2	1	0	3	4

Then we keep two pointers ‘l’ & ‘r’, left & right pointers. l = 0 & r = n-1 & sum = 0 initially & we want sum of two elements to be ‘0’, because the condition is $A[p]=-A[q]$ & we check as follows:

```
while (l < r && flag == 1
{
        sum = new_arr[l].first + new_arr[r].first;
           if (sum1 < sum)
                 l++;
           else if (sum1 > sum)
                 r--;
           else {
                 Print  indices ‘p’ & ‘q’           }
}

if (l >= r) {
           cout << "Sum not found!\n";
           cout << -1 << endl << endl;
      }
```

2. Counting Inversions:

The inversions of an array indicate; how many changes are required to convert the array into its sorted form. When an array is already sorted, it needs 0 inversions, and in other cases, the number of inversions will be maximum, if the array is reversed.

To solve this problem, we will follow the Merge sort approach to reduce the time complexity, and make it in Divide and Conquer algorithm.

For example,

i = 0

j = 0

a = [1, 3, 5]

b = [2, 4, 6]

1. a[i] is smaller then b[j] so a[i] is appended to ‘c’ and ‘i’ is incremented.

c = [1]

i = 1, j = 0

2. a[i] is greater then b[j] so b[j] is appended to ‘c’ and ‘j’ is incremented.

c = [1, 2]

i = 1, j = 1

3. a[i] is smaller then b[j] so a[i] is appended to ‘c’ and ‘i’ is incremented.

c = [1, 2, 3]

1 = 2, j = 1

and so on…

Merge Sort with inversion counting, just like regular Merge Sort is O(n log(n)) time.